A Petri net based design engine for manufacturing systems*

Armin Zimmermann, Jörn Freiheit, and Alexander Huck
Institut für Technische Informatik
Technische Universität Berlin, Sekr. FR 2-2
Franklinstr. 28/29, 10587 Berlin, Germany
phone: +49 (30) 314 73 112 fax: +49 (30) 314 21 116
email: {azi, freiheit, gaspar}@cs.tu-berlin.de

Abstract

Support for the efficient design and operation of complex manufacturing systems requires an integrated modelling, analysis, and control methodology as well as its implementation in a software tool. In this paper the Petri net based design engine TimeNET is presented for this task. Petri nets are able to capture the characteristic features of manufacturing systems in a concise form. A subclass of coloured Petri nets is used, that has been developed especially for the application area of manufacturing. Structure and work plans are modelled separately. Stochastic as well as deterministic and more general distributions are adopted for the firing times of transitions. Fundamental questions about system properties can be answered using qualitative analysis. For an efficient performance and dependability prediction, different evaluation techniques are proposed: direct numerical analysis, approximate analysis, and simulation. Finally, the model can be used to evaluate different control strategies and to control the manufacturing system directly. There is no need to change the modelling methodology, thus avoiding additional effort e.g. for model conversion. In the paper the necessary steps are described using an application example.

1 Introduction

Design and operation of modern manufacturing systems is a complex task. For its support throughout the life cycle of production systems, an integrated modelling, analysis, and control methodology and its implementation in a software tool is necessary. Petri nets have been successfully used in the application area of manufacturing systems because of their ability to describe and analyse their inherent behaviour, which is characterised by synchronisations, resource sharing conflicts, and concurrent activities.

Manufacturing systems are a classical application area of Petri nets; see (Silva and Teruel 1997) for a survey. Two popular extensions including stochastic timing are stochastic Petri nets (SPNs, (Ajmone Marsan 1990)) and generalised stochastic Petri nets (GSPNs,

*Int. Journal of Production Research, vol. 39, no. 2, pp. 225-253 (Special issue on Modeling, Specification and Analysis of Manufacturing Systems)

(Chiola et al. 1993)). Both have been used for manufacturing system modelling and evaluation (Al-Jaar and Desrochers 1990; Silva and Valette 1989). However, the use of these uncoloured net classes requires that if more than one product is processed by a machine, the machine's model has to be replicated due to the lack of distinguishable tokens. Zurawski and Dillon (1991) proposed a method for constructing those replicated uncoloured subnets in a systematic way. In general, using uncoloured nets leads to models that do not reflect the actual structure, making the models less understandable. Uncoloured net classes therefore do not appear to be adequate for the modelling of more complex production systems (Zimmermann et al. 1996a).

Therefore, coloured Petri nets (CPNs, (Jensen 1992)) have been applied to manufacturing systems. Viswanadham and Narahari (1987) used coloured Petri nets for the modelling of automated manufacturing systems. Based on these models, deadlocks can be found by analysing the invariants. A coloured Petri net model of a manufacturing cell controller is described by Kasturia, DiCesare and Desrochers (1988). After obtaining its invariants, the liveness of the model is checked.

The independent modelling of structural and functional system parts is of high importance for the modelling of complex production systems. Only then it is possible to change parts of the work plans without having to redesign the whole model. Separate models for manufacturing system structure and work plans have also been proposed in other frameworks. Ezpeleta and Colom (1997) generate a model that is used for deadlock prevention control policies. Villaroel, Martìnez and Silva (1989) proposed to model work plans with coloured Petri nets, while for the structural model predefined building blocks are used.

Martìnez, Muro and Silva (1987) show how the coordination subsystem of a flexible manufacturing system can be described by a coloured Petri net. The obtained model is embedded into the surrounding levels of control (local controllers and scheduling subsystem), and a terminology based upon the Petri net colours is used for the interaction. Analysing the model detects deadlocks, decision problems, and gives performance measures that depend on variations in the system being modelled.

In the present paper, a design engine for the integrated support of the different steps of manufacturing design is presented. To the best of the authors knowledge, the presented tool TimeNET is the only modelling and analysis environment supporting coloured stochastic Petri nets including non-markovian firing delays, whose performance evaluation is not only based on simulation. The class of Petri nets used has been introduced especially for the modelling of manufacturing systems. The same model is used from the first rough design until the control interpretation. During this process, more details are added, thus refining the model. The advantage of an integrated design engine is obvious: the process becomes easier, faster, and less error-prone.

2 The architecture of the design engine

Modelling and evaluation of complex systems is only feasible with the support of appropriate software tools. Since the modelling framework of stochastic Petri nets has been proposed, many algorithms and their implementations as software tools have been developed. A powerful and easy-to-use graphical interface is important in addition. The modelling and analysis techniques described throughout this paper have been implemented in the tool TimeNET (**Time**d **N**et **E**valuation **T**ool, (German et al. 1995; Zimmermann et al. 2000)).

Figure 1 sketches the different modules of the design engine together with their interaction. The starting point are information about the system to be modelled, like the descriptions of the resources as well as the work plans. In a first step, structure and work

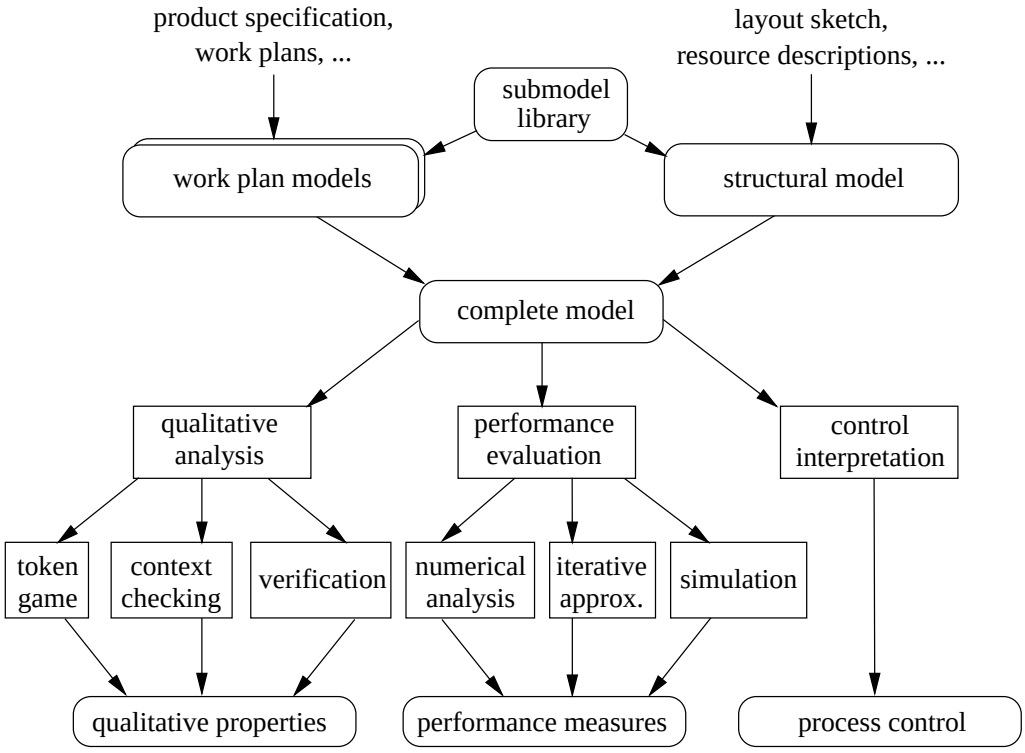

Figure 1: Overview of the design engine

plans are modelled with a dedicated class of coloured hierarchical Petri nets. sections 3.2 and 3.3 demonstrate this using the application example, which is described in section 3.1.

Templates from a library of common submodels can be parameterised and instantiated to ease the description of large systems. They can be used for the hierarchical refinement of substitution transitions. Each of them represents a class of structurally similar resources. It consists of a structural model and a generic work plan model. The latter contains a model of the possible production paths through the submodel structure. Templates usually have a set of parameters such as processing times or buffer capacities. The graphical user interface lets the modeller select the desired template, and then asks for the actual parameter values. The template parameters are set accordingly, and then it is copied from the library into the current model. It is possible to mask out the instantiation of template parts by the use of parameter dependent conditions. Thus it is possible to change the template structure depending on the parameter values.

The separate models of structure and work plans are later on automatically merged to a complete model. Different analysis techniques can then be applied to it. Qualitative analysis techniques developed recently for the dedicated net class use linear algebraic techniques. Information about structural properties from the net are derived directly and efficiently (see section 4). The results can be interpreted in terms of the modelled system to check basic properties, thus helping to understand better its behaviour.

For the evaluation of the system performance different techniques have been developed and implemented within the described design engine. For a realistic description of manufacturing system behaviour, stochastic as well as deterministic and more general distributions are adopted for the firing times of transitions as defined for extended deterministic and stochastic Petri nets (eDSPNs, (German 1994)). Direct numerical analysis (see section 5.1) is applicable to systems up to a certain complexity, e.g. during the first design stages when the model is not yet very detailed. An efficient discrete event

simulator (described in section 5.3) can be used for models of any size, but may require high computational effort depending on the desired accuracy especially in the case of rare events.

An approximate performance evaluation technique has been developed for the dedicated net class recently (Freiheit and Zimmermann 1998). Section 5.2 describes a new variant of it and its application to the example. The approximation method follows the divide-and-conquer principle by analysing the model in parts. The local results are iteratively improved using an abstraction of the global model, until convergence is reached.

Moreover, control strategies for resource allocation, transport strategies and others can be included in the model during a more detailed design. Their influence on the behaviour and performance can therefore be evaluated to select the best variant. Input and output signals can be assigned to transitions of the model, making possible the direct online control of the designed system. This part of the design engine is described in section 6.

Slightly different classes of Petri net models are used during the modelling and the evaluation steps. While the modeller describes the system structure and work plans, no evaluation is possible. The compilation of the complete model has to be done first. The resulting net can then be used for the evaluation. By allowing only the actions that correspond to the current net class (like complete model generation or numerical analysis), some guidance is given for the systematic use of the design engine.

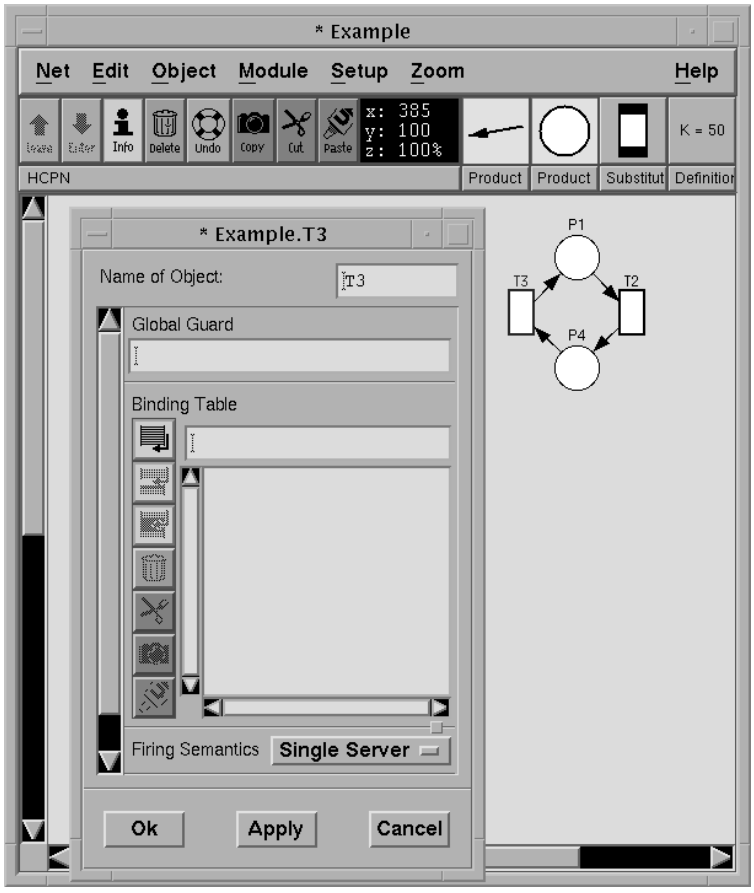

Figure 2: Sample screenshot of the graphical user interface

For the current version 3.0 of TimeNET a new generic graphical user interface has been developed. Figure 2 shows a sample screen shot of the interface during a modelling session with coloured Petri nets. It is implemented in C++ and uses the Motif toolkit.

Two design concepts have been included in the interface to make it applicable for different model classes: A *net class* corresponds to a model type and is defined by a text file. For each node and arc type of the model the corresponding attributes and the graphical appearance is specified. The shape of each node and arc is defined using a set of primitives (e.g. polyline, ellipse, and text). Shapes can depend on the attribute value of an object, making it possible to show tokens as dots inside places. Nodes can correspond to submodels of a different net class. All used net classes and their corresponding analysis algorithms are thus integrated with the same 'look-and-feel' for the user.

More complex functions depend on the net class and require programming. TimeNET offers the possibility to implement *modules* that are compiled and linked to the program. A module has a predefined interface to the main program. It can select its applicable net classes and extend the menu structure by adding new algorithms. An example of an implemented module applicable without restrictions is an export filter to the drawing program xfig. The token game is implemented as a module for the coloured models of manufacturing systems in the user interface. The modules of the design engine depicted in figure 1 are implemented as independent programs. They are started by a user interface module when the user selects the corresponding menu button, and run as background processes. Output messages are shown in windows of the user interface, while analysis results directly become part of the model description. More details about the general tool usage and software architecture can be found in (Zimmermann et al. 2000).

TimeNET is available free of charge for non-commercial use under Solaris and Linux. Further information can be found at `http://pdv.cs.tu-berlin.de/~timenet`.

3 Manufacturing systems modelling

The modelling of complex production systems with uncoloured Petri nets usually leads to large models that are hard to understand and maintain. Coloured Petri nets (Jensen 1992) offer more advanced modelling facilities like distinguishable tokens and hierarchical modelling compared to uncoloured nets. The pure graphical description method of Petri nets is, however, hampered by the need to define colour types and variables comparable to programming languages. The class of coloured stochastic Petri nets used throughout this paper (Zimmermann and Hommel 1999; Zimmermann 1997) has been introduced especially for the modelling of manufacturing systems, trying to overcome this problem.

Two colour types are predefined: *Object tokens* model work pieces inside the manufacturing system, and consist of a name and the current state. *Elementary tokens* can not be distinguished, and are thus equivalent to tokens from uncoloured Petri nets. Places can contain only tokens of one type. Arcs and places are drawn thin (thick), to mark them as corresponding to elementary (object) tokens. Textual descriptions needed in coloured Petri nets for the definition of variables and colour types are omitted, and the specification of the types of places and arcs is implicitly given. The net class is only informally introduced here, during the modelling of the application example. More detailed information can be found in previous papers (Zimmermann et al. 1996a; Zimmermann and Hommel 1999). The models are hierarchically structured, which is necessary to handle complex systems. Furthermore, a library of template models for typical machines and their behaviour is used.

Structure and work plans are modelled independently using this net class. This is important for the evaluation of different work plans, where the structural model is not changed. The structural model describes the abilities and work plan independent properties of the manufacturing system resources, such as machines, buffer capacities, and transport connections. Work plan models specify the work piece dependent features of the manufacturing system. Later on, the different model parts are automatically merged

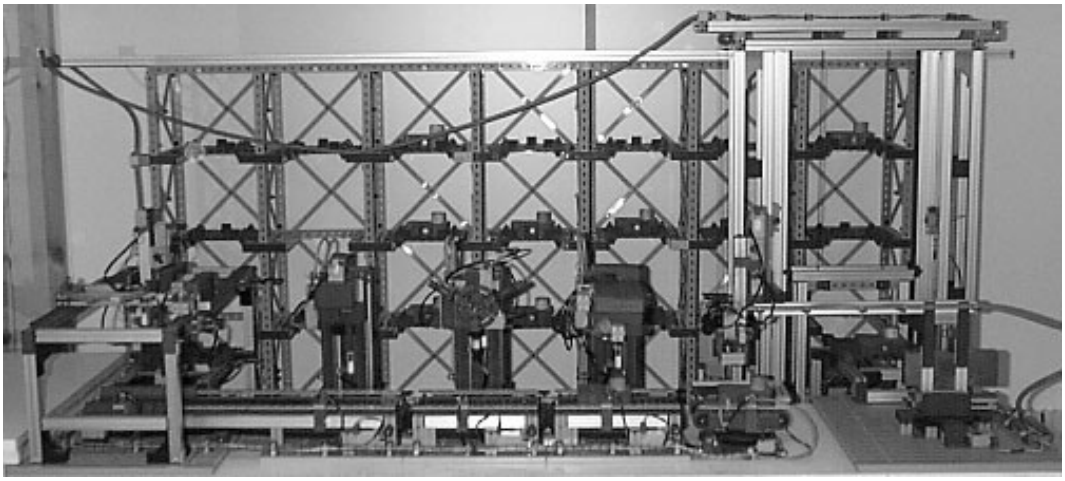

Figure 3: The considered production cell model

resulting in a complete model, which then includes both the resource constraints of the system and the synchronisation of the production steps.

The proposed net class strongly encourages the modeller to describe the system under evaluation as it structurally is. This means that e.g. a buffer should be modelled by exactly one place with the appropriate capacity. One location of parts (one active resource) should not be modelled with more than one place (transition). However, if a complex manufacturing system is modelled, one can also start at a higher level of abstraction, e.g. one production cell is modelled by a transition without further details. Later on, this transition could then be changed into a substitution transition, which is then hierarchically refined by submodels at lower levels of hierarchy. Besides that, the degree of abstraction is left to the modeller.

3.1 An application example

This section briefly describes the application example that is used in the remainder of the paper. It is a manufacturing cell built of parts from the 'Fischertechnik' construction kit, which is used for education and research at the department. Figure 3 shows the system and figure 4 its layout. Please note that the layout sketch as well as all subsequent model figures show the system after a counterclockwise rotation. The application example has been chosen here to demonstrate the proposed integrated technique until the control interpretation.

In the considered production cell, new work pieces are initially stored in the high bay racking on pallets. The rack conveyor can fetch one of them and deliver it to the upper pallet exchange place. A horizontal crane then takes it to the first conveyor belt. Three conveyor belts moves work pieces from one processing station to another. There are two drilling stations, the second having three different interchangeable drilling tools. The last station is a milling machine. Work pieces stay on the conveyor during processing. After leaving the machines, they arrive at a turn table. This table puts them into position for the slewing picker arm, who takes the work piece to the right pallet exchange place. From there it is brought back to a place in the high bay racking by the rack conveyor.

The exchange of crude and finished work pieces takes place via the rack storage. Throughout the paper it is assumed that work pieces have to be machined by the two drilling machines and the milling machine, in this order. The work pieces are moving counterclockwise through the system.

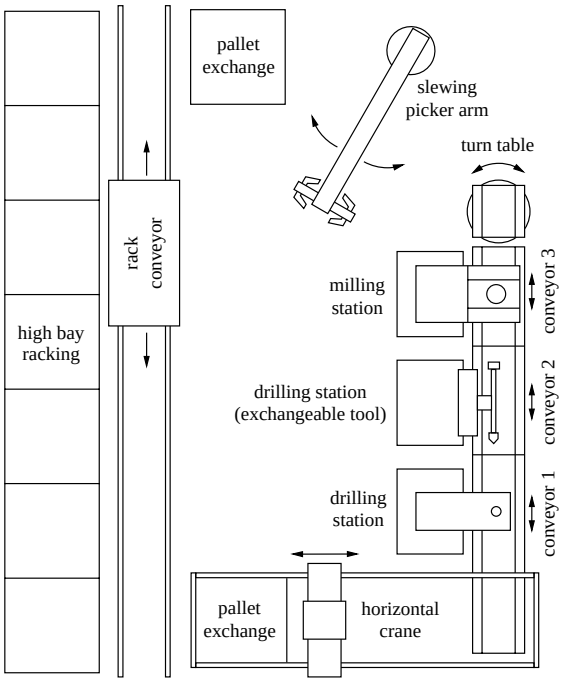

Figure 4: Overview of the modelled system

The 'Fischertechnik' production cell comprises 22 motors and 84 sensing devices altogether. For the online control they can be accessed through a standard RS 232 communication interface.

3.2 Modelling the structure

The described application example is modelled with the proposed class of dedicated coloured Petri nets. A strict separation between the model of the manufacturing system's structure and the sequence of the processing steps for each product is observed.

Figure 5 shows the top level of the structural model. Its composition follows the layout of the modelled system, which makes it easier to understand. Places model buffers and other possible locations of work pieces. The place `rack` corresponds to the rack storage, the places `exchpl1` and `exchpl2` to the pallet exchange places, and place `turnpl` to the turn table. The remaining four places represent the locations of work pieces on the conveyors which are directly in front of the machines or the horizontal crane. As described above, in- and output of work pieces takes place through the rack storage and is modelled with transitions `input` and `output`.

In principle, there are two different operations that can be performed: transport and processing of work pieces. The former corresponds to moving a token to another place, while the latter is modelled by a change in the colour of the token corresponding to the work piece. Transitions modelling machines specify processing steps which only change the token colour. This is emulated by removing the former token from the place and instantly adding a token with the new colour by firing the transition. Therefore many transitions and places are connected by arcs in both directions (loops), which are conveniently drawn on top of each other. The structural model contains all possible actions of the resources, even if they are not used for the processing. The horizontal crane could e.g. move work pieces from the conveyor to the exchange place as well.

Transitions with thick bars depict substitution transitions, which are refined by a

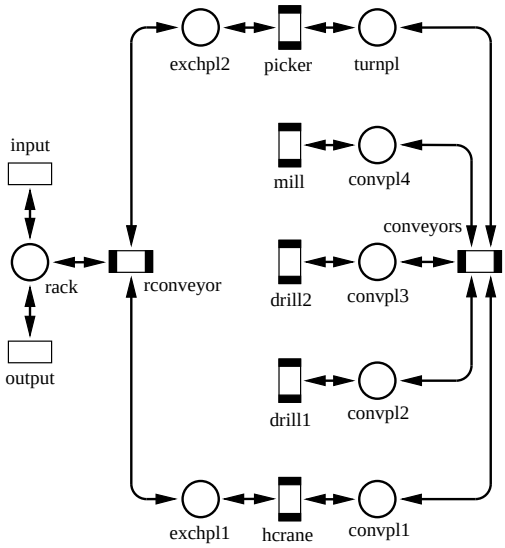

Figure 5: Highest level of the hierarchical coloured model

submodel on a lower level of hierarchy. The whole model consists of 11 submodels. Figure 7 shows such a refining submodel for the slewing picker arm and is described later on. Substitution transitions are e.g. used to describe the behaviour of a machine with more detail during a top-down design. Submodels from a library of standardised building blocks (templates) can be parameterised and instantiated while refining the model. This alleviates the creation of complex manufacturing system models, where many structurally similar parts can be found.

Transition `rconveyor` contains the model of the high bay rack conveyor, while transitions `hcrane` and `picker` correspond to the horizontal crane and slewing picker arm, respectively. For the transport of a work piece from one machine to the next, two of the three conveyor belts have to operate simultaneously. All three conveyors are therefore treated together as one transport facility and are modelled by transition `conveyors`. Thus, their synchronisation is hidden at a lower level and can be specified together.

Alternatively, the modeller can start with function symbols and automatically translate them into a Petri net with separate models of structure and work plans (Zimmermann et al. 1998). The design engine described in this paper can therefore be applied without the need to use Petri nets. However, this technique is not described in detail here. For the translation as well as the modelling of large manufacturing systems with e.g. similar machines, the above mentioned library of Petri net submodels is used.

3.3 Modelling the work plans

In addition to the structural model, for each product a model of the work plans has to be defined. This is done using the same class of coloured Petri nets with some slight differences. Figure 6 shows the first part of the work plan model of one work piece.

This model describes the sequence of operations and transports for a work piece at the highest level of hierarchy. Each step can only be carried out by a resource that is available in the manufacturing system layout. Therefore, only transitions, places, and their connecting arcs from the structural model can be used here. Arc inscriptions show the name (`A`) and processing state (`crude` or `stage1`) of the work piece, separated by a dot.

The model shown in figure 6 corresponds to the structural model in figure 5. Model

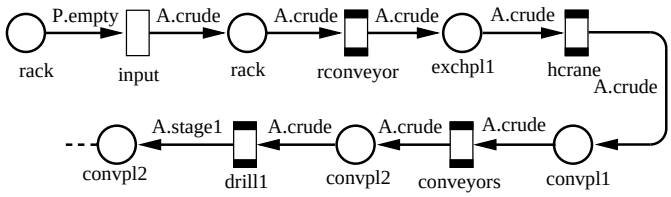

Figure 6: Part of the work plan model

elements in work plan models refer to their structural counterpart through the use of identical names. It is obvious that a substitution transition in a work plan model has to be refined with a submodel. This submodel is then associated to the submodel of the corresponding substitution transition in the structural model. This relationship between both model parts holds for all submodels in the hierarchy. We use the term *associated Petri nets* for this concept of specifying different views of a system in related model parts.

It is possible to model alternatives in production sequences of work pieces. Logical expressions depending on the current marking as well as probabilities can be used to choose a path for a token at such an alternative. A work plan model usually consists of a simple succession of transitions and places. An exception is the modelling of (dis)assembly operations. In this case more than one input or output arc is connected to a transition. Although it can not be immediately seen in figure 6, an assembly operation is also needed for the example work plan. Each work piece is transported and processed while being fixed to one pallet. For an input of a new work piece into the rack storage, there has to be an empty pallet in it (place `rack` contains a token of colour `P.empty`). The input operation (transition `input` fires) removes this token and puts back a token with colour `A.crude`. The inverse operation is carried out by the `output` transition.

A pallet without a mounted work piece has no different states. The transport strategy of empty pallets is described in an additional work plan model.

After the structure and work plans have been modelled with separate coloured Petri nets as described above, a complete model is automatically generated. This is done by adding the information contained in the work plan models to the structural model. Each transition of the structural model is extended with descriptions of its different firing possibilities, called *transition table*. Such a table contains the input/output behaviour, the firing time distribution, and guard functions. Every time a transition appears in a work plan model, a new firing possibility is added to the transition table. Thus, the resource constraints that are imposed by the structure and the synchronisation of the work plans are compiled into one model. The resulting model contains all necessary information from the structural and work plan models. Please refer to (Zimmermann et al. 1996a) for the details of the compilation.

4 Qualitative analysis

As soon as the complete model has been compiled, the design engine (figure 1) offers the possibility to carry out a qualitative analysis of the system. A qualitative analysis answers fundamental questions on the functionality and safety of the system. It considers only time-less properties that are valid independent of time-restrictions. Following Heiner (Heiner 1998), qualitative validation techniques can be classified into two groups after the system properties in question:

- **Context checking** or general analysis deals with general qualitative properties like boundedness or liveness which must be valid in any system independent of its special

semantics.

- **Verification** or special analysis aims at special qualitative properties like safety or robustness, which are determined by the intended special semantics.

The context checking of general properties is a prerequisite for the verification of special properties. For the verification step, the intended system functionality has to be specified. From a technical point of view, three main types of qualitative analysis techniques can be distinguished:

- **Animation** lets the modeller watch the simulated net behaviour. It is useful as an informal debugging technique.
- **Static analysis** techniques avoid the generation of the state space of a system. Static analysis comprises net reduction and various techniques that are related to linear algebra and integer programming. An important static technique is structural analysis. It is independent of the initial marking and uses only structural net information, which is normally expressed by the incidence matrix of the net.
- **Dynamic analysis** techniques generate the state space of a system. The classic approach of constructing the complete reachability graph can be improved by various compression and reduction techniques. Dynamic analysis allows to answer sophisticated queries, but its use is often prevented by state-space explosion.

The remainder of this section concentrates on static analysis techniques, while dynamic analysis is not treated here. However, properties like boundedness and liveness are implicitly checked during the reachability graph generation, which is the first step of the numerical performance evaluation described in section 5.

For its qualitative analysis component, TimeNET makes use of recently developed techniques for uncoloured Petri nets, which are based on convex geometry (Huck et al. 2000). Coloured object places and transitions are treated by unfolding them first to elementary places and transitions. For the output of the results, this unfolding is reversed, so that the process is transparent for the user.

In order to show small examples of qualitative net properties, the refined model of the slewing picker arm is used in this section and will be explained first.

4.1 Model of the slewing picker arm

The Petri net model shown in figure 7 is a hierarchical refinement of the substitution transition `picker`, that is part of the main structural model (shown in figure 5). It specifies the inner behaviour of the slewing picker arm as well as the correlated control of the turn table. The system states of the slewing picker arm are modelled by elementary places and arcs (drawn thin in figure 7). The possible locations of work pieces are modelled by the object places `TurnTable` and `PalletExch` (drawn thick). Because these places are also visible on the upper model level, they are depicted with dashed circles.

The slewing picker arm can execute two useful actions: take a work piece from the turn table to the upper pallet exchange and the reverse. The current state of the picker arm (and of the turn table) corresponds to the location of the elementary token in the model. Figure 7 shows the state after initialisation, where the token is in place `Idle`. Either one of the two immediate transitions `StartF` and `StartB` can fire if the resources are idle, thus starting one of the two transport actions. The decision is made by firing guards (marking dependent boolean expressions) of the transition table entries. It ensures that if the work plan models are correctly specified, the picker arm is only activated for useful transport activities.

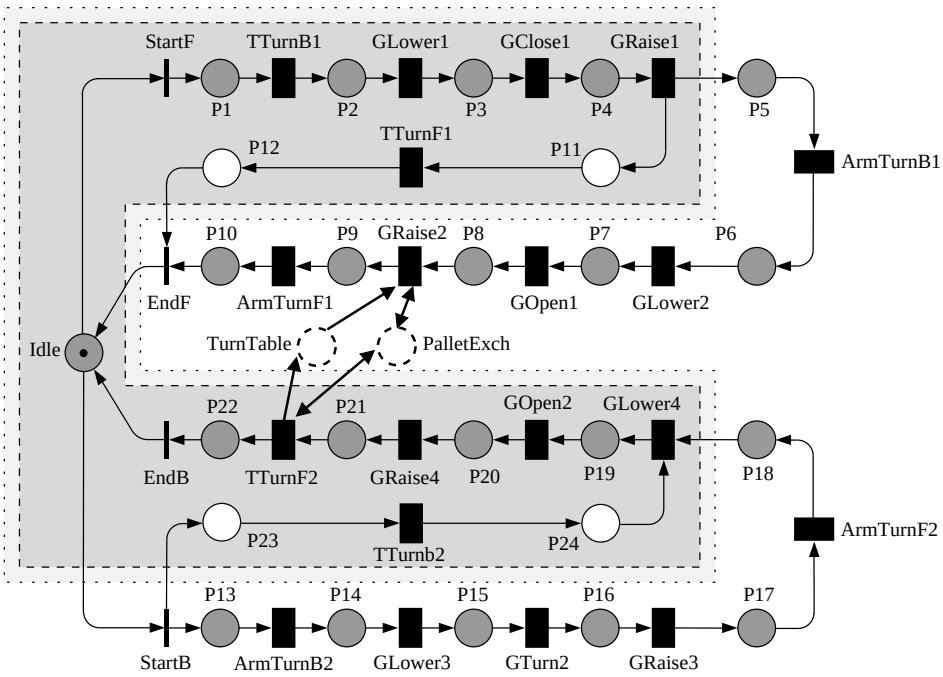

Figure 7: Refined model of the slewing picker arm

Transitions with names beginning with `G` describe actions of the picker arm gripper (lower, close, raise, open). The `ArmTurn` transitions model the turning of the picker arm, and the turn table is described with transitions named `TTurn`.

After the firing of one of the two starting transitions, the corresponding transport action begins. The different transitions become enabled and fire in succession. They describe the individual steps of each transport.

4.2 Token game

Interactive simulation is important for gaining insight into the system behaviour. It can not replace exhaustive analysis methods, but it is a way to build confidence in the model. During a token game, TimeNET displays the state of the model and all possible activities (by flashing all enabled transitions). The modeller chooses one of them to fire. By doing so, it is possible to test the behaviour of the net and to detect errors. Alternatively, an automatic animation can be carried out, letting the modeller watch the changes in the system's state during a slow simulation. Section 6 describes how the token game facility can also be used for the online control of the modelled manufacturing system.

4.3 Boundedness and safety

If a net is *bounded*, then for each place of the net there is an upper bound for the number of tokens it holds. This notion should not be confused with the association of capacities to places, which is possible in the used net class. If the upper bound equals one for all places, then the model is called *safe*. The net properties boundedness and safety are checked by means of *place invariants*: a net is bounded, if it is covered by invariants, i.e. there is a place invariant for every place in the net. A place invariant y is a non-negative integer vector with dimension equal to the number of places P in the net. The value of the p-th element is a weight attached to the marking of the p-th place. The notion

of a place invariant is that the weighted sum of the tokens in a net is constant for any firing sequence, or in other words, the weighted sum of any reachable marking m equals the weighted sum of the initial marking m_0. This is expressed by the token conservation equation:

$$\sum_{p \in P} y(p) \cdot m(p) = \sum_{p \in P} y(p) \cdot m_0(p).$$

Place invariants can be computed independent from the initial marking of the net, their computation therefore belongs to structural analysis. For the details on the computation of place invariants, the reader is referred to (Colom and Silva 1991).

Once that the set of place invariants has been computed, boundedness and safety properties of a net can be easily checked depending on the net's initial marking. The token conservation equation resp. the set of place invariants are also useful for reachability queries, which will be covered in section 4.5.

TimeNET computes the set of place invariants and the bounds for each place. The manufacturing system example is covered by place invariants and is therefore bounded. Moreover it is safe, because all bounds equal one. This ensures that there can be no more than one work piece at a time in one place in the model, as it is the case in reality. In figure 7, the places belonging to one example place invariant are marked grey. For this invariant, TimeNET produces the following textual output:

```
1 =
    P1   + P2  + P3  + P4
  + P5   + P6  + P7  + P8
  + P9   + P10
  + Idle
  + P13  + P14 + P15 + P16
  + P17  + P18 + P19 + P20
  + P21  + P21
```

The set of places belonging to this invariant can hold at most one token, which corresponds to the fact that the picker arm can only execute one action at a time: it is either idle (place `Idle`), moves forward (places `P1` through `P10`) or it moves backward (places `P13` through `P21`).

4.4 Liveness and dead states

A Petri net is live for an initial marking m_0, if, no matter what marking has been reached from m_0, it is possible to fire any transition by progressing through some further firing sequence. A dead state is a state in which no transition is able to fire. A dead state refers to a deadlock in the system, e.g. caused by an incorrect management of shared resources. A Petri net that is not live corresponds to a system which has dead parts.

In order to prove liveness and freedom from dead states, it is necessary to compute the traps and siphons of a net. A trap is a set of places that remains marked once it is marked. A siphon is a set of places that can not be marked again once it is unmarked. Siphons are also named deadlocks, but may not be confused with the term deadlock as used in connection with resource sharing or concurrent processing. The computation of traps and siphons is a structural analysis technique. Details on this topic can be found in (Ezpeleta et al. 1993).

For Petri nets that are at least extended free choice (Heiner 1998), liveness can be proven efficiently by examining the deadlock-trap-property: A Petri net is live iff every siphon contains a marked trap in the initial marking.

For nets that are not extended free choice, only freedom from dead states can be proven with the deadlock-trap-property. In addition to the examination of the deadlock-trap-property, as a necessary (but not sufficient) condition for liveness, it can be checked if the net is covered with transition invariants.

For the model of the manufacturing system, liveness can not be proven because the net is not extended free choice. However, the net is covered by transition invariants and the examination of the deadlock-trap-property shows that there are no dead states.

An example for a siphon and a trap contained in it is shown in figure 7. Siphon and Trap in this example comprise the same set of places, which is marked by a dotted (dashed) box. For this set of places, TimeNET produces the following textual output:

```
1 =
    P1   + P2  + P3  + P4
  + P11  + P12
  + Idle
  + P23  + P24
  + P19  + P20 + P21 + P22
```

In this equation, a place name refers to the number of tokens of any colour in it. The trap is marked (`Idle = 1`) and it is contained in the siphon, therefore the deadlock-trap-property is true for this small example.

4.5 Reachability queries

So far only context checking has been considered, by testing general properties like boundedness and freedom from dead states. An example for the verification of special properties is given in the sequel.

In an early state of the model development, no constraints were made on the assignment of the work pieces to the pallets. It quickly showed that this strategy results in a deadlock where all places on the conveyor belts are occupied by work pieces. This happens when the last empty pallet is occupied by a crude work piece on the first exchange place (`exchpl1`), so that there is no more empty pallet available to pick up the finished work pieces from the second exchange place (`exchpl2`). The assignment strategy for empty pallets had to be changed in such a way that there is always an empty pallet left to pick up finished work pieces. This requirement had to be obeyed when experimenting with different quantities of pallets, and different transport strategies for the rack storage.

To check such requirements, TimeNET offers a (sub-marking) reachability query. The modeller specifies a marking for the places in consideration. The marking of the remaining places is treated by means of don't-care-conditions. If place invariants have already been computed, TimeNET checks the reachability of a marking by testing the token conservation equation. Otherwise, TimeNET tries to solve the state equation, which gives a necessary condition for a marking m to be reachable from the initial marking m_0:

$$m = m_0 + C \cdot x.$$

The state equation uses the incidence matrix C of the net, which describes the net's firing behaviour, and the Parikh vector x, which counts occurrences of transition firings. The state equation is useful in its inversion as sufficient condition for unreachability: a marking m is unreachable, if there is no solution for x.

There are two possible answers to a reachability query, either 'the marking is possibly reachable' or 'the marking is unreachable'. The first answer is not of much use, since it is still unknown if the marking is reachable or not. The second answer verifies the unreachability of a certain (partial) marking.

The critical situation described above is met when there are no more empty pallets (object name `P.empty`) in the rack storage (object place `rack`) and the exchange places (`exchpl1`, `exchpl2`). The corresponding query looks like this:

```
       rack(P.empty) = 0
AND exchpl2(P.empty) = 0 AND exchpl1(P.empty) = 0
```

TimeNET computes that no marking is reachable that satisfies these equations, thus making sure that the assignment strategy for empty pallets was correct.

4.6 Summary of qualitative model properties

As a conclusion of this section we summarise the properties that were checked by qualitative analysis for the manufacturing system (table 1).

Net property	Explanation
ordinary	the multiplicity of every arc equals one
not pure	presence of self-loops
not extended simple	does not belong to this net class, liveness can not be proven by static analysis
strongly connected	for each node of the net, there exists an directed path to every other node
covered by place invariants	there exists a P-invariant which assigns a positive value to each place
bounded	for every place, there is an upper bound for the number of tokens that the place can hold
safe	not more than one token is on a place
covered by transition invariants	there exists a T-invariant which assigns a positive value to each transition (necessary condition for liveness)
satisfies the deadlock-trap-property	every siphon contains a marked trap
net is free of dead states	at least one of the transitions is enabled in all states
possibly live	(a dynamic analysis of the net shows that the net is really life)

Table 1: Qualitative net properties

Qualitative analysis of Petri nets allows to prove certain system properties, which is an advantage over simulation languages. Especially static analysis techniques are computationally inexpensive in many cases, because they generate no state space. During the design process, qualitative analysis naturally precedes a performance evaluation, because it would be pointless to compute performance measures for an incorrect system.

5 Performance evaluation

After the qualitative properties of a manufacturing system have been analysed, its performance and dependability can be evaluated. Questions on the throughput, utilisation, work

in process, and others are answered. Different variations of the system and their resulting performance and dependability measures can thus be computed and compared. The aim of this investigation is to obtain a better understanding of the correlations between details of the manufacturing system (e.g. the buffer capacities) and the main performance measures (e.g. the throughput). Proposals can be derived in order to increase the manufacturing system's productivity.

For the application example, the aim is to evaluate the throughput of work pieces. A performance measure is defined in the model, which gives the throughput of all finished work pieces per hour. This can be done using the throughput `TP[output]` of a transition that each work piece passes exactly once like `output`.

In this paper the focus is on steady-state performance evaluation (and not on transient), which computes the performance of a system in equilibrium (provided that it exists). Direct numerical analysis, approximate analysis, or discrete-event simulation can be used to obtain the desired measures from the model. All three methods are implemented as modules of TimeNET, as depicted in figure 1. The advantages and disadvantages of the three algorithms depend heavily on the actual model and the performance measures of interest (state space size, numerical stiffness, rare events, etc.). Unfortunately thus no automatic decision can be made which method will be the best one. Already the definition of 'best' at least necessitates the specification of a balance between required speed and result exactness. Therefore, the modeller decides which method is applied, or can compare the results of different ones, as it is done in the following subsections for the application example. The throughput of the modelled manufacturing system is computed with the different algorithms, facilitating a comparison of results and computational efforts. The TimeNET tool is used for all evaluations.

Although the used class of coloured Petri nets offers advanced modelling facilities for manufacturing systems, the underlying stochastic process is the same as for a behaviourally equal uncoloured model. The techniques developed for these net types can therefore be adapted to the coloured case. Analysis and simulation methods for eDSPNs (Ciardo et al. 1994; German 1994) are applied to the models. However, it should be noted that the algorithms directly use the coloured model, and not an unfolded one as some other approaches/tools do.

5.1 Numerical analysis

In order to evaluate the performance of a manufacturing system, delays are associated with the transition firings. In stochastic Petri nets, exponential distributions are used for the firing delays due to analytical simplicity. However, the fixed processing time of a certain workpiece or a transport delay should better be modelled using deterministic times. It has been shown that the results obtained from models with different distributions may vary significantly (German 1994). To obtain more realistic results, analysis methods for models incorporating non-exponentially distributed firing times have to be used.

The firing delay of transitions considered in the techniques used here can either be zero (immediate), exponentially distributed, deterministic, or belong to a class of general distributions called expolynomial. Such a distribution function can be piecewise defined by exponential polynomials and has finite support. It can even contain jumps, making it possible to mix discrete and continuous components. Many known distributions (uniform, triangular, truncated exponential, finite discrete) belong to this class.

If no more than one general or deterministic transition is enabled in each marking, a semi-regenerative stochastic process underlies the Petri net model (Ciardo et al. 1994). The type of process depends on the types of allowed firing delays and whether certain transitions are enabled together in one marking or not.

Techniques for uncoloured nets (German 1994; German et al. 1995) are adopted here for the analysis of the dedicated Petri net models. This is possible because both model types have the same underlying stochastic process. At the level of the (reduced) reachability graph they can both be analysed in the same way. The reachability graph of the coloured model is computed directly without a prior unfolding.

5.2 Approximate analysis

For many systems of real-life size the state space is too large to be handled. This is called the state explosion problem and necessitates advanced techniques to overcome this limitation. The idea of decomposition methods is to divide the whole system into smaller subsystems, so that the computation of the whole state space can be avoided.

An algorithm for the approximate analysis of GSPNs (Ajmone Marsan et al. 1984) based on decomposition has been presented in (Pérez-Jiménez et al. 1998; Campos et al. 1994). This section describes a new variant of an approximate evaluation method (Freiheit and Zimmermann 1998) for the special class of coloured stochastic Petri nets described in section 3.

The main advantage of decomposition approaches is that the analysis of small subsystems needs less memory. Despite the need to iteratively repeat the algorithm, the approximate results are computed faster than with standard methods. The disadvantage is that only approximate performance results are computed, although experiences show that the error is acceptable in most cases.

In the method presented here, a net called *basic skeleton* contains a simplification of the whole original net. Additionally, for every subsystem a *low-level system* is derived, in which only the corresponding subsystem is simplified. The set of low-level systems and the basic skeleton are then used by an iterative algorithm to compute approximate performance measures of the original model.

The approximation technique presented in this section is applied to the throughput computation of the manufacturing system example described before.

5.2.1 Partition and aggregation

Low-level systems and a basic skeleton are derived from the whole system first. The state space of the low-level systems and the basic skeleton is usually smaller than the original one by more than one order of magnitude. In each low-level system one or more subsystems are kept and the other subsystems are aggregated. In the basic skeleton all subsystems are aggregated.

For each substitution transition (and thus each submodel) one subsystem is generated. A path-wise aggregation method is employed. In (Pérez-Jiménez et al. 1998) the aggregation method is based on implicit places. In this approach not only a complex structural analysis of the system is necessary, but also a computation to avoid spurious states. The path-wise aggregation presented here is much simpler, although experiences show that the approximation results are the same. The idea is that each path corresponds to a possible token flow through the system, which must not be destroyed during the aggregation. In the following some important terms are introduced.

$N = (P, T, F)$ is a net if P and T are disjoint sets of places and transitions and $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs. The places connected to a subsystem are called *buffers*. The set of buffers is denoted with $B \subseteq P$ and the set of subsystems $S \subseteq N$ ($S_i = (P_i, T_i, F_i)$ with $S = S_1 \cup S_2 \cup \ldots \cup S_n$ where $S_1 \cap S_2 \cap \ldots \cap S_n = \emptyset$. The *preset* of B is denoted by ${}^\bullet B$, and the *postset* $B^\bullet$, respectively. These definitions denote the set of transitions having buffer places as output or input, respectively.

The set of transitions $IT_i \subseteq T_i$ ($OT_i \subseteq T_i$), where for each $t \in IT_i$ ($t \in OT_i$) is $t \in B^\bullet$ ($t \in {}^\bullet B$) is called *input (output) transitions*[1] *of the subsystem* S_i. There is a *path* between an input transition $t_1 \in IT_i$ and an output transition $t_2 \in OT_i$ if and only if $(t_1, t_2) \in F_i^*$, where F_i^* is the transitive closure of F_i. A path between (t_1, t_2), where $t_1 \in IT_i$ and $t_2 \in OT_i$ is called *longest path* if there is no path between $t_x, t_y \subset T_i$ with $(t_1, t_2) \subset (t_x, t_y)$.

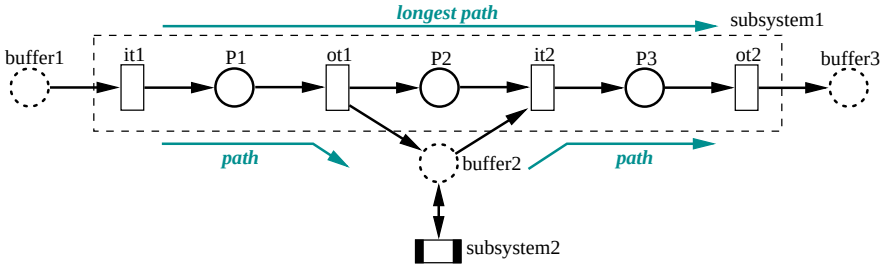

Figure 8: Paths and a longest path

Figure 8 shows an example to explain the term longest path. There are three paths in `subsystem1`: (it_1, ot_1), (it_2, ot_2), and (it_1, ot_2), but only the last one is a longest path, because it includes the former ones.

There are two different aggregation rules, which are used both for the structural model and the work plan models. The first aggregation rule is called *general aggregation rule*. The rule is partitioned into two steps. The first step adds a new place for each longest path of a subsystem, resulting in the *extended subsystem*. In the second step the longest path is deleted, while the associated new place is kept. Additionally each timed input transition is changed to an immediate one. The obtained model is called *aggregated subsystem*.

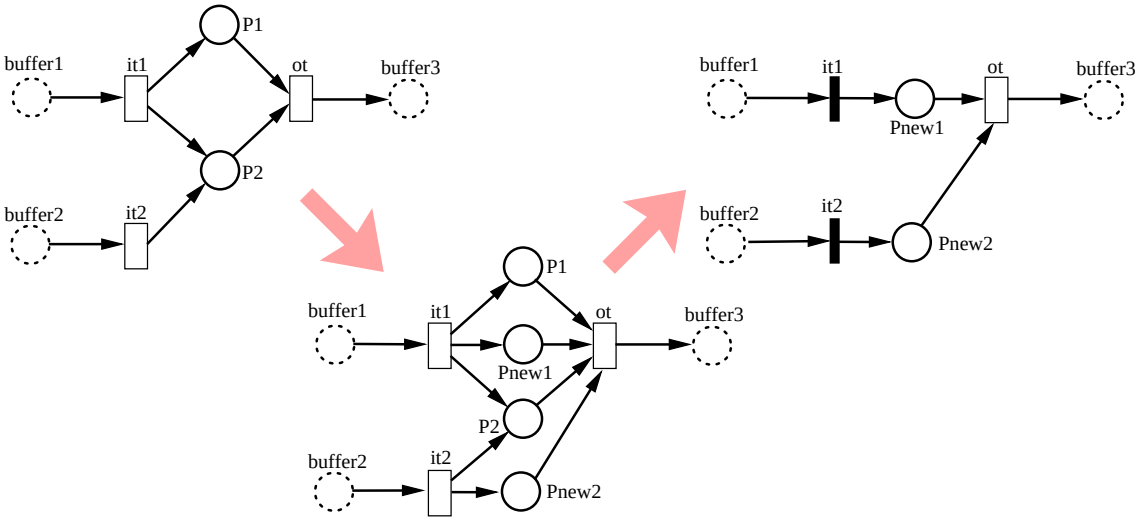

Figure 9: General aggregation rule

Figure 9 shows an example for the application of this aggregation rule. If there is at least one longest path between two different transitions $t_1 \in IT_i$ and $t_2 \in OT_i$, add a new place s with $(t_1, s) \cup F_i$ and $(s, t_2) \cup F_i$. Thus, from the original subsystem S_i the extended

[1]Please note that $IT_i \cap OT_i = \emptyset$ does not necessarily hold

subsystem S_i^+ is derived. After this first step the places and transitions in the longest path(s) are deleted while the new place(s) are kept. The resulting subsystem S_i^- is the aggregated subsystem.

In figure 9, `it1` and `it2` are input transitions, and `ot` is an output transition. There are two paths between `it1` and `ot`: `P1` and `P2`. That is why a new place `Pnew1` is added. The path between `it2` and `ot` is handled in the same way by adding the new place `Pnew2`. In the next step the original paths `P1` and `P2` are deleted and the new places are kept, resulting in the aggregated subsystem.

The second aggregation rule is called *one-transition rule*. It is used in the special case when a transition t is connected either with one buffer (by a loop) or with two or more buffers both by an input and an output arc. The aggregation rule is simple: transition t is kept and no place is added.

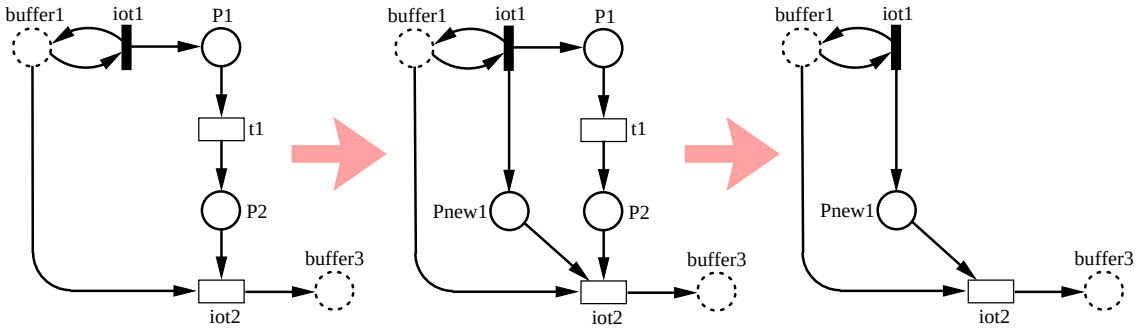

Figure 10: Combination of general and one-transition rule

In the example shown in figure 10 both aggregation rules are applied. The one-transition rule is applied twice, because both input and output transitions `iot1` and `iot2` are connected with buffers both by an input and an output arc. The transitions are kept following the one-transition rule. However, there is also a path between these transitions. The general aggregation rule requires a new place `Pnew1` to be added. During the second step of the rule, `P1, T1, P2` are deleted to aggregate the subnet.

After generating the low-level subsystems and the basic skeleton for the structural model, the same has to be done for each work plan model. The aggregation rules for both model parts are the same. The strong relationship between net elements in the work plan models and the structural model is kept after the aggregation.

Low-level systems and basic skeleton

In the following step the aggregated subsystems are derived. They are called *low-level systems* and denoted by LS_i ($i \in \{1, \ldots, k\}$). Each LS_i is obtained from the net N by aggregating one or more extended subsystems S_j^+ and by keeping the other subsystems $S_{l \neq j}^+$ such that $S_j^+ \cup S_l^+ = S^+$. In the example presented here two low-level systems LS_1 and LS_2 are derived. In LS_1 the subsystem `rconveyor` is aggregated and the others are extended (see figure 11).

In the original model there were five combinations of paths between the buffers `exchpl1`, `exchpl2`, and `rack`. New work pieces (`A.crude`) are moved from `rack` to `exchpl1` while finished work pieces are moved from `exchpl2` to `rack`. Additionally it is possible that empty pallets are going both from `rack` and from `exchpl1` to `exchpl2`. The last possible way of empty pallets goes from `exchpl1` to the `rack`. Thus, the transportation of both crude work pieces and empty pallets between `rack` and `exchpl1` is possible. Therefore

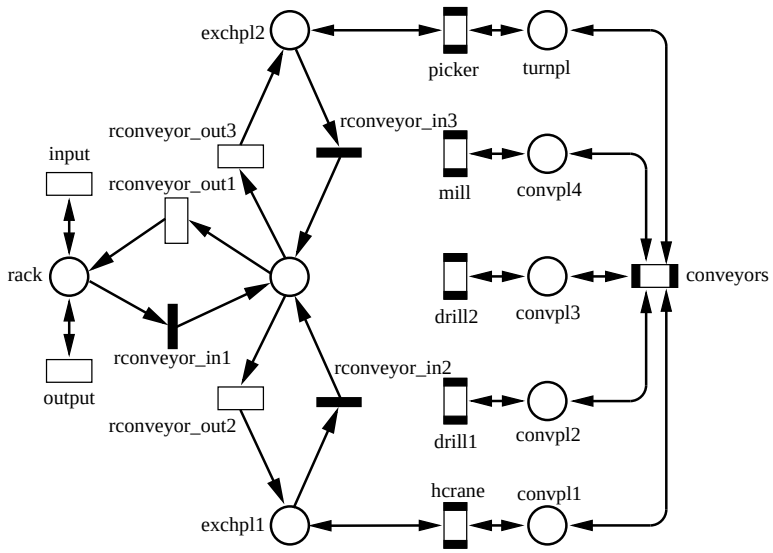

Figure 11: Low-level system example

it is important to distinguish between different colours in the `rconveyor` subsystem (see table 3) during the throughput computation.

In LS_2 of the manufacturing system example the subsystem `rconveyor` is extended and the others are aggregated. Due to the aggregation of the subsystem `conveyors` the subsystems `drill1`, `drill2`, and `mill` are deleted. The reason for this is that there is a longest path in subsystem `conveyors` which includes all transitions connected with the buffers `convpl2`, `convpl3`, and `convpl4`, while these buffers are not connected to other subsystems.

Figure 12 shows the basic skeleton of the example. In the basic skeleton all subsystems are aggregated. For the aggregation of both the `hcrane` and the `picker` subsystems the combination of the two aggregation rule is applied (see figure 10), while the `conveyors` subsystem is aggregated only using the general aggregation rule once.

Only the transitions `input` and `output` are kept from the original model. In the following subsection the iterative computation of the manufacturing system example throughput is explained.

5.2.2 Iterative throughput approximation algorithm

In this section the low-level systems and the basic skeleton are used to iteratively compute an approximation of the model throughput. The applied technique is based on the response time approximation method presented in (Pérez-Jiménez et al. 1998; Campos et al. 1994; Pérez-Jiménez et al. 1996).

During the algorithm, the service rates of the output transitions are adjusted in order to achieve a balance between the throughput values of all extended subsystems and the basic skeleton. The results of the iterative approximation algorithm are independent from the initial service rates of the output transitions of the aggregated model parts. In the non-aggregated parts the transitions always keep the initial service rates of the original model. For each LS_i the sojourn time of the added new places in the extended subsystems for each work piece (colour) are computed. Then the service rates of the corresponding transitions of the basic skeleton are changed for each work piece, such that the sojourn time of the same tokens are equal. The obtained service rates of the output transitions $t \in OT_i$ in all LS_j with $j \neq i$ is then set according to the values in the basic skeleton.

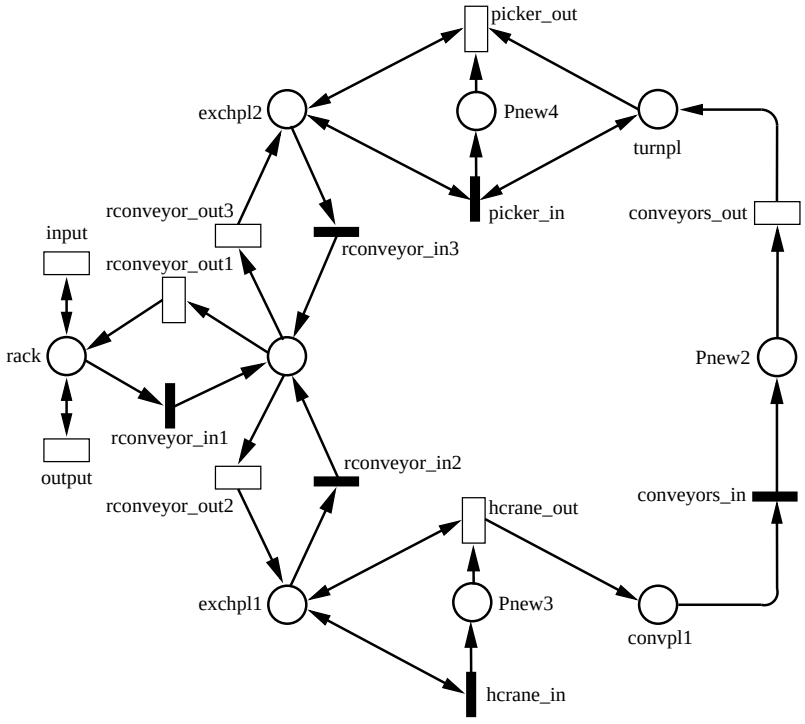

Figure 12: Basic skeleton of the manufacturing system example

The procedure is repeated until convergence is reached.

Algorithm

derive LS_i ($i \in \{1, \ldots, n\}$) and BS
initialise all $t \in (IT_i \cup OT_i)$ in LS_j ($j \neq i$)
and in BS with any service rate μ_t
keep the original service rates for all
$t \in (IT_i \cup OT_i)$ in LS_i
repeat
for $k := 1$ **to** n **do**
compute sojourn time of all different work-
pieces in the new places
repeat
adjust the service rates μ_t in BS
compute sojourn time of the same different work
pieces in the same new places in BS
until sojourn time of the new places are equal
change the new service rates for the corres-
ponding transitions t in all LS_i $i \neq k$
end for
until convergence

In the manufacturing system example first the sojourn time of tokens in the new places `Pnew2`, `Pnew3`, and `Pnew4` in the non-aggregated parts of the low-level system LS_1 is computed while the service rates of the output transitions `rconveyor_out1..3` is set to 1. Changing the service rates of the output transitions `hcrane_out`, `conveyors_out` and

picker_out of the basic skeleton the same sojourn time of the places Pnew2, Pnew3, and Pnew4 has to be reached. The obtained service rates of the output transitions hcrane_out, conveyors_out and picker_out are set in the low-level system LS_2. The sojourn time of place Pnew1 is computed distinguishing between the sojourn times of crude and finished work pieces, and empty pallets. If the token sojourn times of Pnew1 for the different colours is the same in the basic skeleton, the obtained service rates of rconveyor_out1..3 are set in the low-level system LS_1 and the procedure starts again. This pattern is repeated until convergence is reached.

5.3 Simulation

For many models the restriction of not more than one enabled non-exponential transition per marking is violated. A substitution of the non-exponential firing delays with exponential ones makes an analysis possible, but might result in significant errors. Another problem of all analysis methods is the size of the reachability graph. Not only the computational complexity grows, the analysis may become impossible for some models of realistic size due to memory space restrictions. Approximate methods like the one proposed in section 5.2 reduce this problem, but there is a limit on the manageable complexity as well.

Discrete-event simulation is still applicable for the performance evaluation in these cases. However, other problems arise with the statistical evaluation of the samples and the accuracy of the results. The reason for this is that the simulation is a stochastic experiment. All samples drawn during the simulation run are random variables. The user-specified performance measures can only be obtained by estimating the mean value of the sampled data. The precision of this estimate has to be calculated as well, based on the confidence interval derived from the sample variance. Standard methods require that the input data satisfies certain properties like independence or normal distribution. However, simulating the Petri net behaviour does not fulfill them, thus necessitating special algorithms for a valid computation of the accuracy. One of them is the efficient simulation component (Kelling 1995) of the tool TimeNET, that has been adapted to the special class of coloured Petri nets used in this paper.

Before a simulation run is started in TimeNET, the user specifies the desired accuracy of the performance measures in terms of the confidence interval and the maximum relative error of the final result. The initial transient phase of the simulation run of a steady-state evaluation is detected and ignored. Variance estimation of the samples is performed by spectral variance analysis, allowing a robust estimation even for correlated samples as they are common if only one replication of the simulation process is running.

The length of a simulation run is decreased with a parallelisation of simulation processes. Each of them simulates the whole net and sends sample packets to a central process. As long as the model can be handled on one workstation, this approach is simpler to implement and more efficient than parallel simulation with a distributed model. The central process monitors the accuracy and stops the simulation after reaching the specified threshold.

The simulation of the system can e.g. be used to evaluate the expected number of tokens in the different places of the model. This corresponds to checking at which places of the system the work pieces spend most of the time. Bottlenecks are thus detected, because they are resource(s) that delay the work pieces more than the others.

5.4 Performance evaluation of the example

The three performance evaluation methods described above have been used for the direct numerical throughput computation of the application example. Table 2 compares

computational effort and achieved accuracy of the three methods. The CPU times were measured on a Sun Ultra 5 workstation running at 333 megahertz.

Technique	Numerical analysis	Iterative approximation	Simulation		
			99%, 1%	98%, 2%	90%, 10%
Throughput	19.67	19.12	19.59	19.48	20.75
Error	0%	2.8%	0.4%	1.0%	5.5%
CPU Time (sec)	1625	186	4176	1104	56

Table 2: Accuracy and computational effort of different evaluation techniques

The numerical analysis method (see section 5.1) yields the most exact results. For the modelled system it calculates a throughput of 19.6692 work pieces per hour. The computation took approx. 27 minutes and generated an CTMC of 41279 states for the original model. The approximation algorithm could deliver its result after approx. 3 minutes, because the sizes of the CTMCs it had to cope with were about one magnitude smaller than for the original model: it generated two low level systems with 1128 and 306 states, and a basic skeleton with only 15 states. For the initialization of the algorithm, service rates of all output transitions of the aggregated parts in the low-level systems and the basic skeleton were set to 1. Convergence was reached after only three iterations. Table 3 shows the computed sojourn time of tokens in the new places. With an error of less than 3%, the approximation algorithm shows a good relation between result quality and computational effort.

Subsystem S_k	New place p (Colour)	Sojourn time for p in LS	μ_t in BS $t \in OT_k$ in LS	Sojourn time for p in BS
rconveyor	Pnew1 (A.finished)	0.095063	0.0227272	0.095605
	Pnew1 (A.crude)	0.117001	0.0555555	0.116851
	Pnew1 (Pallet.empty)	0.095063	0.0243902	0.095605
hcrane	Pnew3	0.183017	0.0294117	0.183017
conveyors	Pnew2	0.252994	0.0217666	0.252995
picker	Pnew4	0.139954	0.0384615	0.139954

Table 3: Results after the final iteration of the approximation algorithm

For the simulation (see section 5.3), three runs with different accuracies have been carried out. Table 2 shows confidence level and relative error in percent. It can be seen that the computational effort increases dramatically with the desired accuracy. For a desired relative error of 1%, the computation took about 70 minutes.

The simulation facility was used to evaluate the rack conveyor by using the experiment feature of TimeNET. This feature allows the automatic execution of a series of evaluations for the same model, but with a changing parameter. For the application example, a factor W was introduced as a parameter that is taken as a multiplier for all firing delays associated to elements of the rack conveyor. In the range from 0.4 up to 5.0 for W, the throughput was analysed by simulation. Figure 13 shows the results graphically, making evident the high influence of the speed of the rack conveyor.

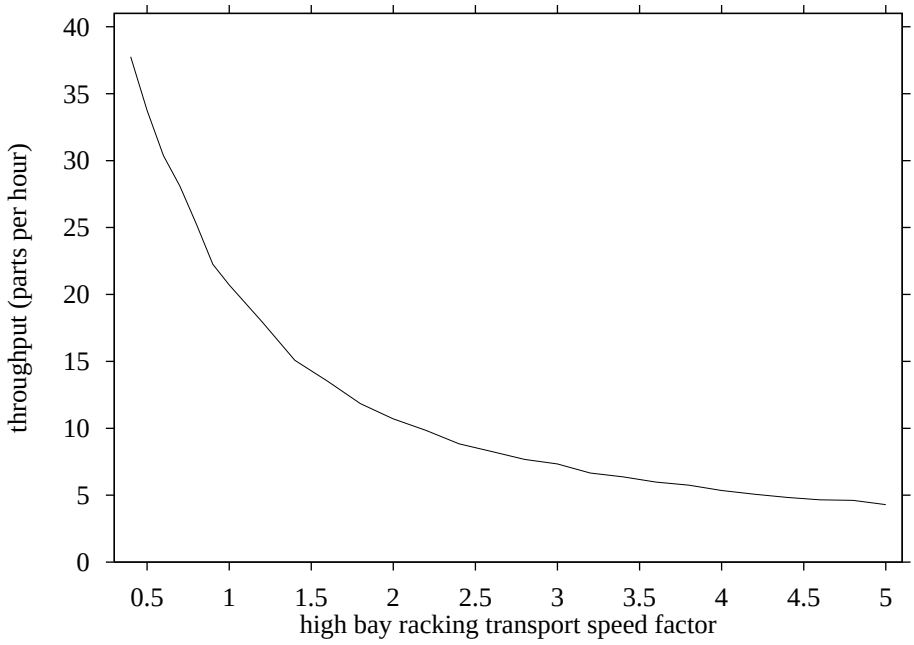

Figure 13: Throughput of the system

For the application example, the results showed that (in accordance with the observation of the real system) the rack conveyor is the main bottleneck of the system. For a planned real life production system, this would surely be not acceptable, because the usually expensive processing stations should be utilised more than a transport facility. For this reason, the increase of throughput was evaluated, that would be possible with a hypothetical speedup of the rack conveyor. In reality, during the system design, this would have led to selecting a faster version of it. From the resulting throughput values and the associated profit of work pieces as well as additional costs for a faster rack conveyor, the system can be optimised.

6 Online control of the application example

After the planned system has been designed and its performance evaluated and possibly improved, the final step is to bring the specified behaviour into reality. Input and output signals are assigned to transitions of the model, making possible the direct online control of the designed system. Thus only one model is used throughout the whole design process. It should however be noted that the aim of this paper is not synthesis, PLC program generation, nor verification of control strategies. The purpose is rather to show that the used class of Petri nets is applicable for the different life cycle phases without major changes in the model. It is easily possible to integrate control rules in the model. Afterwards, the influence on the system behaviour (liveness, performance) can be analysed. Finally the behaviour specified in the model is directly executed.

To allow the control of a real-world process using the Petri net model, possibilities for its interaction with the outside world have to be added to the otherwise autonomous model. From the model's point of view, input and output signals are necessary. They should be added in a simple way, following the meaning of the performance model elements in a natural way. This is possible without problems, because the used kind of coloured Petri net models strongly observe the modelled system's structure.

Only 'active resources' like machines, transport facilities etc. are controllable. Their activities are modelled by transitions, who can either move tokens (transport) or change token colour (processing). A transition becomes enabled, when its guard function evaluates to true, the necessary input tokens are available, and enough space for added tokens is free in the subsequent places (if they have a restricted capacity). This is the point of time at which the firing time of the transition starts running. The actual firing with the corresponding marking change takes places when the firing delay has elapsed. It then appears natural to assign single controllable activities to transitions. When the transition becomes enabled, a start (output) signal is sent from the model to the process (e.g. a motor is switched on). After termination of the activity (e.g. a sensor detects the stop position), an (input) signal from the process is sent to the model, which then initiates the instantaneous transition firing. This model interpretation only changes the model behaviour by adopting the unknown delays of external activities. Therefore, results of the qualitative analysis still hold.

Transitions with associated input signals are called *external*, all others *internal*. Associating output signals to a transition does not change its firing semantic in the model. Opposed to that, external transitions fire if and only if they are enabled and receive their input signal. The firing delay being specified in the model for external transition is ignored during the online control. The modeller should be careful with cases in which external transitions can be disabled by the firing of other transitions, because their input signal could then be lost. Conflicts of this type can be automatically detected from the model structure. However, they are not generally forbidden, because there are cases in which this behaviour is useful.

It is possible to assign any number of input and output signals to a transition. All output signals are being sent when the transition becomes enabled, while the arrival of any one of the input signals triggers the firing of the transition. Transitions with output signals but without input signals (or vice versa) are allowed as special cases. An example of an activity which can be finished at any time without having been started before is e.g. the failure of a machine. A sensor which detects this failure can trigger the firing of an associated transition in the model. The firing time of internal transitions keeps its semantics from the autonomous model. During the online control of a production process, the usually unit-less numbers have to be interpreted e.g. as seconds. They can be used to control the timing of activities without stop positions and to detect deadline violations.

Petri nets have been often considered for control of manufacturing systems (Holloway et al. 1997). Some approaches to interpret a Petri net model for control use different methods to exchange information between model and process. Sometimes the token contents of places (the marking) is used to generated the model output. This is e.g. the case in GRAFCET (David 1995; David and Alla 1992) and related methods. The enabling of transitions that depend on external information can also be done with *control places* (Holloway and Krogh 1994); the models are then called controlled Petri nets. Another possibility is the association of control procedures with transitions (Martínez and Silva 1984). Coloured Petri nets are used for the control of a manufacturing system e.g. in (Kasturia et al. 1988; Martínez et al. 1987). In (Feldmann et al. 1995), transitions modelling processing steps are hierarchically refined and input/output signals are associated with the subtransitions. Generally, control design based on a Petri net model is advantageous with respect to a state machine description, because they are able to capture parallelism in a much clearer way.

The application of the control technique is explained using the submodel of the slewing picker arm as shown in figure 7. The different steps for the two possible transport actions are described by sequences of elementary transitions and places. Each transition corresponds to one controllable activity.

After the inclusion of the work plan information in the transition tables of the model, transitions `StartF` and `StartB` can only fire if tokens that correspond to work pieces to be transported are located in places `TurnTable` and `PalletExch`. Firing guards (marking dependent boolean expressions) of the transition table entries are used for this. It ensures that if the work plan models are being specified correctly, the picker arm is only activated for useful transport orders. Modelling errors or wrong transport strategies leading e.g. to deadlocks or bottlenecks can be detected by the different analysis techniques.

After the firing of one of the two starting transitions, the corresponding transport action begins. The transitions describe the individual steps of each transport, and have input and output signals associated to them. As an example, transition `TTurnB1` models the movement of the turn table into the backward position. When the transition becomes enabled, the turn table motor is switched on by the associated output signal. When the final position is reached, the corresponding sensor is activated and the input signal leads to the firing of the transition.

Motivated by the use of separate models for structure and work plans of a production system, the task of designing the control system can be divided in two steps. The specification of the work plans ensures that the manufacturing system produces (in the model) the desired products. However, the afterwards automatically generated main model does not necessarily have to be free of deadlocks nor optimal with respect to some performance measures like the throughput. The model can be analysed and the problems detected, leading to control strategies that improve the system behaviour. In any state of the production system, the controller can only forbid activities that would otherwise be possible. This corresponds to disabling transitions inside the model. Using marking dependent guard functions, this can be done easily with the used class of Petri nets. The guard function of a transition (or of one firing possibility) is a boolean function of the net marking, which has to be true to allow the enabling. The second step of the control design thus corrects and optimises the production process. Finally, the model based online control ensures the execution of the specified behaviour.

With the control module of the design engine, production processes can be controlled by activating the control during the token game facility. This is especially important for debugging and demonstration purposes. Two variations of the token game are available: interactive (the user controls the firing of enabled transitions) and animation (simply a simulation). Both show the marking and state changes and can thus be used for online control and visualisation of the production process. The animation can be stopped at any time for manual control. Therefore the control of the modelled production process is seamlessly integrated in the modelling and evaluation tool.

The online control is implemented in TimeNET in the following way: during a token game with activated control, an additional process monitors the model state. If a state change enables a transition, its possible output signals are sent to the production process. The sensor states are checked additionally by this process. If a sensor state change is detected, which is associated to an input signal, the corresponding transition(s) are fired if they are enabled in the model and the displayed model state is updated. Internal transitions may fire independently from input signals, only depending on their associated firing time. The correspondence between transitions and signals for models of controllable systems is described in a file.

7 Conclusion

This paper describes the recently developed design engine TimeNET for the modelling, analysis, and control of manufacturing systems. There is no need to change model or description method between the stages of design and operation. This avoids additional

effort for model conversion, making the design process more efficient. A dedicated class of coloured Petri nets is used to describe structure as well as work plans of the system.

Different analysis techniques can be applied to the automatically generated composed model. The paper presents adapted and new techniques to compute qualitative and quantitative properties of the analysed production process. For the evaluation of the system performance three different algorithms are offered: direct numerical analysis, iterative approximation, and simulation. Finally, the model can be used to evaluate control strategies and to directly control the manufacturing system based on the model.

The different parts of the design engine are described in the paper using an application example, showing the usefulness of an integrated technique for the different phases in the life cycle. In a previous work (Zimmermann et al. 1996b), parts of the tool presented here have been successfully applied to a real life industrial example of complex size.

References

Ajmone Marsan, M., 1990, Stochastic Petri Nets: An Elementary Introduction. In G. Rozenberg (ed), *Advances in Petri Nets 1989, Lecture Notes in Computer Science*, Vol. 424, (Springer Verlag), pp. 1–29

Ajmone Marsan, M., Balbo, G., and Conte, G., 1984, A class of Generalized Stochastic Petri Nets for the performance evaluation of multiprocessor systems. *ACM Transactions on Computer Systems*, **2** (2), 93–122.

Al-Jaar, R. Y., and Desrochers, A. A., 1990, Petri Nets in Automation and Manufacturing. In G.N. Saridis (ed), *Advances in Automation and Robotics*, Vol. 2, (JAI Press)

Campos, J., Colom, J. M., and Silva, M., 1994, Approximate Throughput Computation of Stochastic Marked Graphs. *IEEE Transactions on Software Engineering*, **20** (7), 526–535.

Chiola, G., Ajmone Marsan, M., Balbo, G., and Conte, G., 1993, Generalized Stochastic Petri Nets: A Definition at the Net Level and Its Implications. *IEEE Transactions on Software Engineering*, **19** (2), 89–107.

Ciardo, G., German, R., and Lindemann, C., 1994, A Characterization of the Stochastic Process Underlying a Stochastic Petri Net. *IEEE Transactions on Software Engineering*, **20**, 506–515.

Colom, J.M., and Silva, M., 1991, Convex Geometry and Semiflows in P/T Nets. A Comparative Study of Algorithms for Computation of Minimal P-Semiflows. In G. Rozenberg (ed), *Advances in Petri Nets 1990, Lecture Notes in Computer Science*, Vol. 483, (Springer-Verlag), pp. 79–112

David, R., 1995, GRAFCET – a powerful tool for specification of logic controllers. *IEEE Trans. on Control Systems Technology*, **3** (3), 253–268.

David, R., and Alla, H., 1992, *Petri Nets and Grafcet (Tools for modelling discrete event systems)* (Prentice Hall).

Ezpeleta, J., and Colom, J. M., 1997, Automatic Synthesis of Colored Petri Nets for the Control of FMS. *IEEE Transactions on Robotics and Automation*, **13** (3), 327–337.

Ezpeleta, J., Couvreur, J. M., Silva, M., 1993, A new technique for finding a generating family of siphons, traps and st-components. Application to colored Petri Nets. In G. Rozenberg (ed), *Advances in Petri Nets 1993, Lecture Notes in Computer Science*, Vol. 674, (Springer-Verlag), pp. 126–147

Feldmann, K., Colombo, W., and Schnur, C., 1995, An Approach for Modelling, Analysis and Real-Time Control of Flexible Manufacturing Systems Using Petri Nets. *Proc. European Simulation Symposium*, (Erlangen-Nürnberg), pp. 661–665

Freiheit, J., and Zimmermann, A., 1998, Extending a Response Time Approximation Technique to Colored Stochastic Petri Nets. *Proc. 4th Int. Workshop on Performability Modelling of Computer and Communication Systems (PMCCS)*, (College of William and Mary, Williamsburg, VA, USA), pp. 67–71

German, R., 1994, Analysis of Stochastic Petri Nets with Non-Exponentially Distributed Firing Times. Dissertation, Technische Universität Berlin.

German, R., Kelling, C., Zimmermann, A., and Hommel, G., 1995, TimeNET – A Toolkit for Evaluating Non-Markovian Stochastic Petri Nets. *Performance Evaluation*, **24**, 69–87.

Heiner, M., 1998, Petri Net Based System Analysis without State Explosion. *Proc. High Performance Computing '98*, (Boston, San Diego: SCS), pp. 394-403.

Holloway, L. E., Krogh, B. H., and Giua, A., 1997, A Survey of Petri Net Methods for Controlled Discrete Event Systems. *Discrete Event Dynamic Systems: Theory and Applications*, **7**, 151–190.

Holloway L. E., and Krogh, B. H., 1994, Controlled Petri Nets: A Tutorial Overview. In G. Cohen and J.-P. Quadrat (ed), *11th Int. Conf. on Analysis and Optimization of Systems*, *Lecture Notes in Control and Information Sciences* (Sophia-Antipolis, Springer-Verlag), Vol. 199, pp. 158–168

Huck, A., Freiheit, J., Zimmermann, A., 2000, Convex Geometry Applied to Petri Nets: State Space Size Estimation and Calculation of Traps, Siphons, and Invariants. Technical Report 2000-6, Technische Universität Berlin.

Jensen, K., 1992, *Colored Petri Nets: Basic Concepts, Analysis Methods and Practical Use*, EATCS Monographs on Theoretical Computer Science (Springer Verlag).

Kasturia, E., DiCesare, F., and Desrochers, A. A., 1988, Real Time Control of Multilevel Manufacturing Systems Using Colored Petri Nets. *Proc. Int. Conf. on Robotics and Automation*, pp. 1114–1119

Kelling, C., 1995, Simulationsverfahren für zeiterweiterte Petri-Netze. Dissertation, Technische Universität Berlin, *Advances in Simulation*, (SCS International).

Martínez, J., Muro, P., and Silva, M., 1987, Modelling, Validation and Software Implementation of Production Systems Using High Level Petri Nets. In *Proc. Int. Conf. on Robotics and Automation* (Raleigh, North Carolina), pp. 1180–1185

Martínez, J., and Silva, M., 1984, A language for the description of concurrent systems modelled by coloured Petri nets: Application to the control of flexible manufacturing systems. In *Proc. of the 1984 IEEE Workshop on Languages for Automation* (New Orleans), pp. 72–77

Pérez-Jiménez, C. J., Campos, J., and Silva, M., 1996, State machine reduction for the approximate performance evaluation of manufacturing systems modelled with cooperating sequential processes. In *1996 IEEE International Conference on Robotics and Automation* (Minneapolis, Minnesota, USA), pp 1159–1165

Pérez-Jiménez, C. J., and Campos, J., 1998, A response time approximation technique for stochastic general P/T systems. In *Proc. Symp. Industrial and Manufacturing Systems, 2nd IMACS Int. Multiconf. on Computational Engineering in Systems Applications (CESA '98)* (Nabeul-Hammamet, Tunisia)

Silva, M., and Teruel, E., 1997, Petri Nets for the Design and Operation of Manufacturing Systems. *European Journal of Control*, **3** (3), 182–199.

Silva, M., and Valette, R., 1989, Petri Nets and Flexible Manufacturing. In G. Rozenberg (ed), *Advances in Petri Nets 1989*, *Lecture Notes in Computer Science* (Springer Verlag, 1989), Vol. 424, pp. 374–417

Villaroel, J. L., Martínez, J., and Silva, M., 1989, GRAMAN: A Graphic System for Manufacturing System Design. In: S. Tzafestas (ed), *IMACS Symposium on System Modelling and Simulation* (Elsevier Science Publ.), pp. 311–316

Viswanadham, N., and Narahari, Y., 1987, Coloured Petri Net Models for Automated Manufacturing Systems. In *Proc. Int. Conf. on Robotics and Automation* (North Carolina, Raleigh), pp. 1985–1990

Zimmermann, A., Bode, S., and Hommel, G., 1996, Performance and Dependability Evaluation of Manufacturing Systems Using Petri Nets. In *1st Workshop on Manufacturing Systems and Petri Nets, 17th Int. Conf. on Application and Theory of Petri Nets* (Osaka, Japan), pp. 235–250

Zimmermann, A., Dalkowski, K., and Hommel, G., 1996, A Case Study In Modelling And Performance Evaluation Of Manufacturing Systems Using Colored Petri Nets. In *Proc. of the 8th European Simulation Symposium* (Genoa, Italy), pp. 282–286

Zimmermann, A., and Freiheit, J., 1998, TimeNETms — An Integrated Modelling and Performance Evaluation Tool for Manufacturing Systems. In *IEEE Int. Conf. on Systems, Man, and Cybernetics* (San Diego, USA), pp. 535–540

Zimmermann, A., Kühnel, A., and Hommel, G., 1998, A Modelling and Analysis Method for Manufacturing Systems Based on Petri Nets. In *Computational Engineering in Systems Applications (CESA '98)* (Nabeul-Hammamet, Tunisia), pp. 276–281

Zimmermann, A., 1997, Modellierung und Bewertung von Fertigungssystemen mit Petri-Netzen. Dissertation, Technische Universität Berlin.

Zimmermann, A., Freiheit, J., German, R., and Hommel, G., 2000, Petri Net Modelling and Performability Evaluation with TimeNET 3.0. In B. Haverkort et.al. (eds.), *11th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation* (Schaumburg, Illinois, USA), *Lecture Notes in Computer Science*, Vol. 1786, pp. 188–202

Zimmermann, A., and Hommel, G., 1999, Modelling and Evaluation of Manufacturing Systems Using Dedicated Petri Nets. *Int. Journal of Advanced Manufacturing Technology*, **15**, pp. 132–137.

Zurawski, R. Z., and Dillon, T. S., 1991, Systematic Construction of Functional Abstractions of Petri Net Models of Typical Components of Flexible Manufacturing Systems. In *Proc. 4th Int. Workshop on Petri Nets and Performance Models* (Melbourne), pp. 248–257