Integrating Software and Hardware Performance Models Using Hierarchical Queueing Petri Nets

Falko Bause, Peter Buchholz and Peter Kemper

Informatik IV, Universität Dortmund
D-44221 Dortmund, Germany

e-mail: {bause,buchholz,kemper}@ls4.informatik.uni-dortmund.de

Abstract

This paper introduces a new approach for the construction of performance models of complex systems integrating software and hardware. Software components are specified using hierarchical coloured GSPNs which extend the well established coloured GSPNs. Hardware components are composed of basic queues taken from queueing networks. Integration of queues into hierarchical GSPNs facilitates the specification of a virtual machine which provides services for software components. Virtual machines are integrated in the coloured GSPN description of the software component via subnet places. This simplifies the description of the mapping of software onto hardware models. Apart from specification convenience, certain analysis techniques also profit from the hierarchical structure, among others we discuss exact numerical techniques, approximation techniques, simulation and hybrid approaches combining different techniques.

1 Introduction

Well known concepts apply to structure complex software (SW) and hardware (HW) systems, these include modularisation, layers, and virtual machines. SW systems are typically structured hierarchically via refinement and modularisation. This results in a top-down design, where during the design process, the amount of information increases by successive addition of details. For HW resources, basic building blocks are reused and combined to build virtual machines in a bottom-up manner. Clearly, both methods must match for a complete system. This requires to map a SW hierarchy onto virtual machines where the latter have to provide an appropriate set of services. Separating a SW hierarchy

from a virtual machine depends on the point of view, since using any kind of subsystem can be interpreted as employing a virtual machine.

In modeling complex systems, basically the same concepts as for design purposes are at hand to handle complexity. Modeling SW systems at a relatively high level typically results in a tree type structure, because using a SW subsystem (e.g., a function) multiple times usually means that for each time an independent replica is used[1]. At a HW level, multiple use of a component can either mean to create independent replicas of resources or to share a single resource. So a tree type structure becomes a directed acyclic graph. A suitable analysis formalism should provide a hierarchical description, where components can be used either as independent replicas or as a shared resource. Furthermore higher levels should allow for a simple notation of concurrent programs, while lower levels need to support simple modeling of HW resources on a sufficiently abstract level.

Queueing networks (QNs) are useful at the lowest level, since they allow for a simple, abstract description of scheduling. However, they are less useful for describing the logical behaviour of a hierarchical SW system with fork/join effects and refinement. Petri nets are a more prominent formalism for this purpose, they are suitable to model typical language constructs of imperative, concurrent programming languages [3]. Incorporating a stochastic notion of time into Petri nets yields the well known generalised stochastic Petri nets (GSPNs). However GSPNs do not show sufficient support for a structured way of modeling, such that design methodologies have been developed to demonstrate how to use GSPNs for a combined modeling of HW and SW systems:

Botti and de Cindio [7, 8] analyze the effect of program placement by mapping Petri net models of Occam programs on a HW description modeled by a Petri net as well. They consider stochastic timing and use GSPNs with a preselection policy restricted to local conflicts. Preselection implies random scheduling without preemption. The PSR-methodology by Donatelli and Franceschinis [16] extends this approach. SW processes and HW resources are described independently by GSPNs and the assignment of resources to tasks takes place via a special GSPN. Typical problems are the "min-match" problem, i.e. that a transition performing a join of subtasks might match subtasks which do not belong together. Both approaches do not consider refinement, which could be introduced as in [18] to gain more structure, but this does not automatically yield a semantics adequately supporting the concept of virtual machines.

The PRM-methodology by Ferscha [17] describes how to compose untimed SW processes modeled by so-called P-nets with HW resources modeled by so-called R-nets. P-nets are uncoloured Petri nets enriched by transition refinement and resource requirements. A preselection policy is used to accommodate for conflicts between refined transitions. Arcs between places in R-nets and transitions in P-nets describe resource allocation and preselection establishes random scheduling without preemption. The PRM methodology is intended to map parallel programs onto rather simple processing units. It is of limited use to describe complex scheduling and resource allocation patterns.

[1] We do not consider recursion.

In summary all these approaches do not sufficiently support a structured description of hierarchical SW systems and shared complex resources with other scheduling strategies than random without preemption. Furthermore an execution of a top-level net without knowing the details of all refined transitions is not possible, since a refinement does not automatically imply an abstract description which is executable. This would be a desired property in an early stage of the design phase, where models need to be analyzed which are not completely refined and where only a clear specification of the interface and the required services exists yet. Hierarchically combined Queueing Petri nets (HQPNs[1]) have this property. They allow to model SW architectures of possibly concurrent programs in a hierarchical manner: the top level net either describes the communication flow between a set of components which interact via message passing or it describes the control flow through a set of modules which can but need not act concurrently. Refinement takes place via a well defined interface concept such that by definition an abstract behaviour as experienced within the higher level is specified. As a desired result, a high level model can be evaluated without considering the details of a refinement.

HQPNs extend the timing aspects of coloured GSPNs [15] by the ability to integrate queues into places. Such a timed place consists of a queue and a depository for served tokens. If a transition firing adds tokens onto a timed place, these tokens are inserted into a queue according to the queue's scheduling strategy. The service time distribution depends on the colour of the token. After service the token moves to a depository, where it is available to the place's output transitions. Tokens in a queue are not available for firing. Furthermore a place of a HQPN might even contain a whole HQPN subnet (cf. Fig. 1). It is then called a subnet place. A HQPN subnet itself might contain subnet places yielding a multilevel hierarchy. The interface between a net and a corresponding subnet is predefined to a certain extent. It consists of places *input, output, actual_population* and transitions *t_input, t_output.* The colour sets of the interface places are all equal to the colour set of the subnet place. The idea is that a subnet performs a complex, internal transaction on tokens fired at the subnet place. *t_input* denotes the beginning of this transaction, *t_output* its termination. Within the subnet we distinguish unprocessed tokens at *input* from those tokens being processed at *actual_population. t_input* is an identity function with respect to places *input* and *actual_population* (starting a transaction does not modify its input information). Hence this distinction is irrelevant as seen from the subnet place and we distinguish only tokens at the places *input* and *actual_population* from tokens at *output. t_output* describes the various effects the transaction can show on its input. Note that we consider coloured nets, such that *t_output* can fire with respect to a set of colours/modes, e.g. it is possible to describe that a subnet returns either with a result or an error message for a certain kind of input. The concept extends the concept of function calls where calling a function with parameters results in a return of results by the called function; it resembles a similar approach as in HIT [6].

We will elaborate the following example of a distributed computation all through the paper to demonstrate our approach.

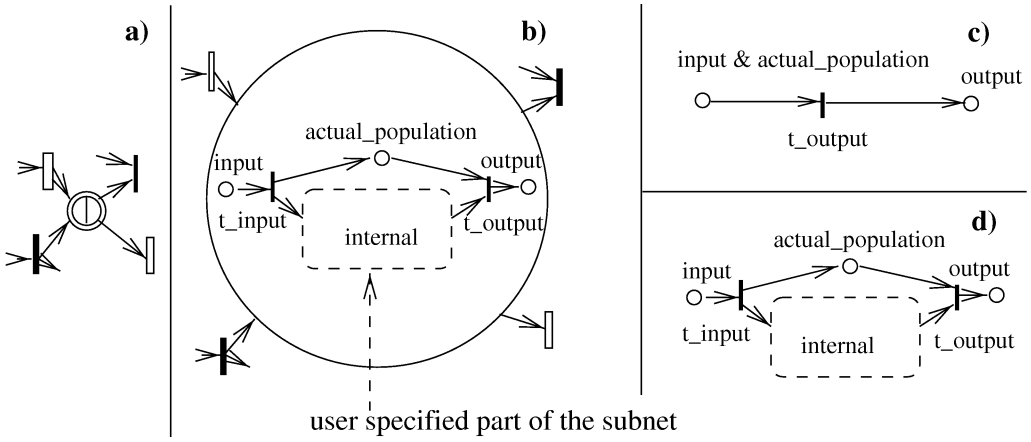

Figure 1: Example subnet place of a HQPN b), its shorthand representation a), the interface as seen from subnet place c) and from the subnet itself d)

Example:
A user wants to perform a transient simulation using several workstations in a local area network. During the simulation he keeps himself busy with e.g. editing documents, answering emails etc., and once in a while he tests whether the simulation has terminated. In that case he starts a statistical analysis package and determines new parameters for the next simulation run. It might also happen that the simulation stops with an error, e.g. the detection of a deadlock situation, such that the user skips the statistical analysis and modifies the model for a new simulation run.

The paper is structured as follows: Sect. 2 describes the formulation of hierarchical SW models within HQPNs. Sect. 3 illustrates the suitability of HQPNs to model HW resources, and Sect. 4 shows how these components can be neatly composed into a hierarchical structure. Advantages of a hierarchical description for validation and performance analysis are briefly sketched in Sect. 5. A summary is given in Sect. 6.

2 Specification of Software with HQPNs

Modeling program flow with Petri nets is well-known [3] and with the introduction of colours also data-dependent behaviour can be encoded with moderate effort [19, 20]. For small programs the modeling process is thus straight forward. The development of larger programs, or SW systems in general, is normally a top-down process and requires additional modeling support. From an abstract point of view the modular structure of SW systems can be regarded as a hierarchy of function calls (typical for sequential programs) or as a set of interacting components (typical for parallel programs). In the latter case the top level net describes the flow of communication, whereas lower levels behave as

sequential programs with refinement via function calls.

In order to accompany a top-down development process, hierarchies have also been integrated into the Petri net formalism (e.g. [18]), where subnets are used to structure a Petri net. For analysis purposes a (structural) hierarchical refinement should be additionally accompanied by a behaviour description for the different hierarchy levels, which offers the possibility to analyze the system for different levels of granularity.

HQPNs offer both possibilities. A structural refinement of the net is supported by place refinements of so-called subnet places. Each subnet has a special interface which specifies its possible behaviour. A subnet place can be viewed as an abbreviation of a place-transition-place subnet (cf. Fig. 1 c)), which might have an internal state and whose behaviour is completely defined by transition *t_output*.

This implies the following procedure for top-down development. At each level the input/output behaviour of a to be refined component (a subnet place) is specified by a transition. Using the interface for subnets (see Fig. 1) the same transition description is defined for *t_output*. So consistency between the aggregated view of a component and its refinement is automatically guaranteed. The mapping between function calls and functions as well as SW and HW (cf. Sect. 3) is very intuitive, because each subnet place belongs to exactly one subnet.

Using place refinements gives the opportunity to use the wide-spread race policy for transition firings. In [17] transition refinements enforce a preselection policy for modeling resource sharing, which is more suitable for systems where resources are used exclusively, implying a very detailed model description of the whole system. However, preselection can easily be modeled in race models via random switches using immediate transitions.

Quantitative (performance) analysis requires to integrate timing considerations into a model. Typically, no timing information is available at the level of the SW components. The specific time consumption of a function is in most cases dependent on other SW operations and the provided HW. Thus functions, whose timing information can only be specified in the context of the complete SW / HW model, should be modeled by subnet places. So normally only ordinary and subnet places as well as immediate transitions will be used for SW description. In rare cases the modeler might use timed places or timed transitions (cf. Figs. 3-5) when specific delays are known a priori or resource sharing is considered negligible.

Example (continued):
The HQPN in Fig. 2 models the process schemata of the user, it gives the top level of our SW hierarchy. A user process first prepares data for a simulation and afterwards starts the simulation calling function simulate *(the second parameter indicates a terminated simulation* (OK) *or the detection of an error). Function* edit_doc *models a user activity executed in parallel to the simulation. Refinements of activities are depicted in Figs. 3, 4 and 5, which together form the second level of our SW hierarchy (cf. Fig. 7). All subnets show the same skeleton structure where* t_input *acts as an identity function just forking tokens/customers and* t_output *completely determines the behaviour of the subnet as seen*

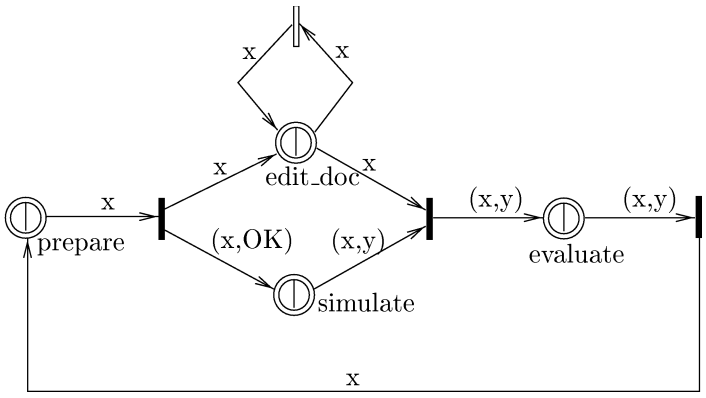

Figure 2: User process schemata

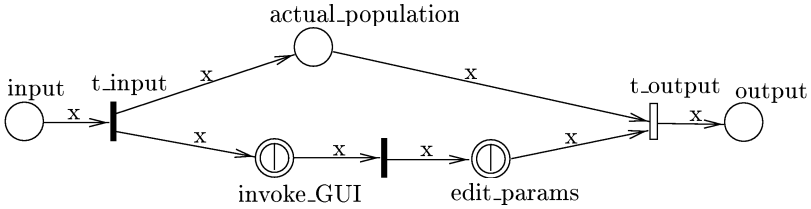

Figure 3: Activities for preparing a simulation run

by the next higher level.

3 Hierarchical Specification of HW Components

Modeling HW components with plain Petri net elements leads to very complex net structures and/or rate definitions when one wants to integrate scheduling information, e.g. priority service strategies, into the model description. At the HW level scheduling is very important, since it specifies the rules how to receive service from shared resources. HQPNs support timed places which simplify the modeling as in QNs.

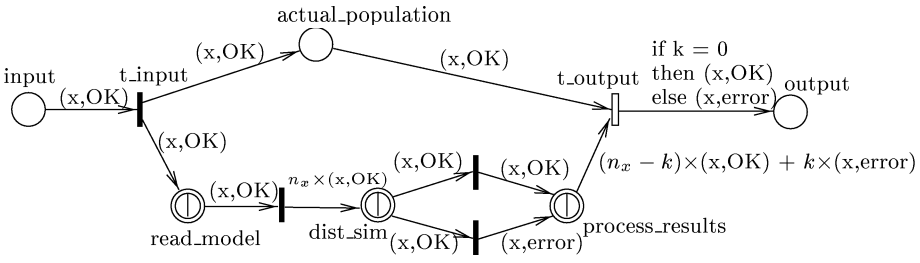

Figure 4: Simulation

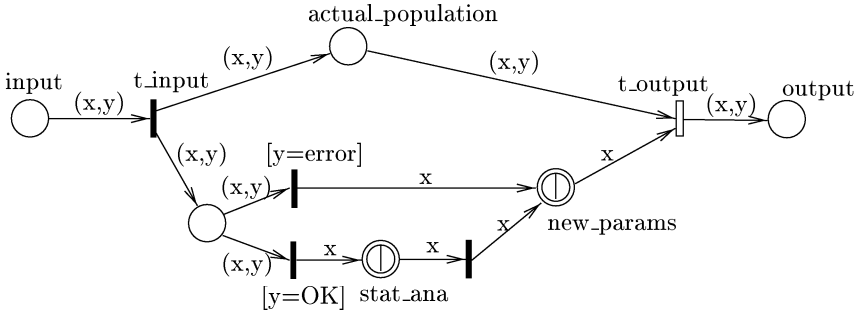

Figure 5: Evaluation of simulation results

Design of the HW level is usually a bottom-up process. The designer is faced with a fixed set of HW components. He defines certain patterns of behaviour for groups of these components, i.e. he builds virtual machines, with the intention to narrow the gap between low-level HW description and the later needs for SW developers.

In HQPNs components of the HW level can be described by (subnets including) timed transitions and/or timed places. Subnets and timed places can be used for a bottom-up development process of a more complex HW facility yielding virtual machines. Mapping the SW onto the HW level is simple in HQPNs, since it is the same as on every level: the modeler has to link a subnet place to a specific subnet. At the HW level we will typically encounter situations where several subnet places are linked to (i.e. are using) the same (virtual) machine component. In [6] such a component is called *enclosed*.[2] Different tokens/customers have to be identified for the dynamics of the complete model in the sense that we have to remember from which subnet place a token entered the subnet. Therefore the colour sets of an enclosed component are expanded implicitly by a tag to keep track of a token's origin.

In [18] this is resembled by the semantics of so-called substitution places similarly to the subnet places of HQPNs: for each substitution place the whole net acts as if an individual copy of the corresponding subnet has been inserted. The modeling of resource sharing, like shared memory, is supported in [18] by a so-called fusion of elements (places or transitions). If two elements belong to a fusion set then they represent the same model element. E.g. if place *read_model* of Fig. 4 and place *stat_ana* of Fig. 5 belong to one fusion set then both represent the same(!) subnet. Therefore all elements of a fusion set must have the same colour set. Timed places cause more complex situations: on the one hand we want to keep track of the token's origin, which implies an individual copy for such a place, on the other hand we want to define sharing of the resource "queue" which implies that all individual copies belong to the same fusion set. Thus within the concept of [18] the modeler is forced to model the timed place with an additional colour component for storing the origins of tokens, which implies that he knows the whole SW/HW hierarchy

[2]Surely such situations may also occur at the SW level, but we find them more typical for the HW level.

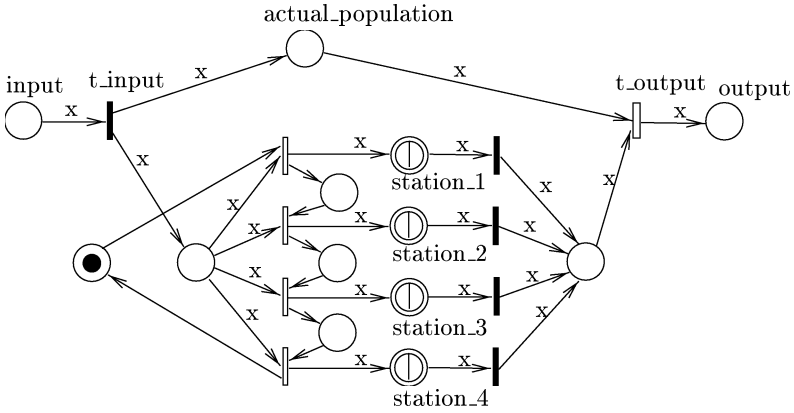

Figure 6: Virtual machine

beforehand.[3]

A very simple solution for this problem is to allow a modeler of an element, a subnet for example, to label elements and rate/timing definitions to be expandable by tags or not. An expandable element thus corresponds to an individual copy of a substitution place in[18] and a non-expandable element corresponds to an element of a fusion set. A shared timed place is now simply defined as an expandable element with a non-expandable timing definition. E.g., this means that for a processor sharing queue all customers/tokens are taken into account regardless of their tags. Thus it is generally possible to keep track of a token's origin even in the presence of shared resources. Note that the notion of an expandable element of a subnet does not support the construction of general fusion sets like, e.g. global ones. Avoiding global fusion sets retains a certain degree of locality which is essential for an efficient analysis exploiting the hierarchical structure.

Example (continued):

At the lowest HW level, there are three workstations $W1, W2, W3$ *and a PC, which are all represented by timed places for simplicity (cf. Fig.7). Since several users of this cluster, like our one, want to distribute jobs more or less equally amongst the workstations we build a virtual machine which distributes incoming tasks in a cyclic manner (see Fig. 6). We have fixed the number of these workstations to* 4 *for simplicity. Assuming that our user does not want to be bothered about simulation processes on his own PC, at least two of our places* station_i *have to be linked to the same subnet, which in our case only consists of a single timed place. Since we want all processes to compete for the workstation, we define service rates to be non-expandable. On the other hand we also want to send a served token back to the* station_i *from where it entered the queue, so we define the timed places to be expandable.*

[3]The reader should note that several enclosed components might occur at different hierarchy levels, so that more than one additional colour component has to be augmented at the lowest levels.

4 Composition of SW/HW Models

We now consider the hierarchical structure of a complete model. Modeling SW is in fact a top-down process starting with an abstract view of the complete system which is refined by filling subnet places with detailed nets until basic hardware components offering the required services are reached. On the other hand, the specification of hardware components is a bottom-up approach starting with queues and simple Petri nets realising basic hardware components. By building virtual machines more complex hardware components are specified which, nevertheless, can be used just like basic queues in the upper level nets.

No strict separation between HW and SW is given in the approach. A virtual machine can be interpreted as a piece of hardware providing the required services to some upper level component. In this case, it does not matter whether the virtual machine is realised by simple queues or a complex hierarchical model. At each separated level two different parts are combined to build the performance model, namely the SW or load and the hardware or machine providing the services required by the SW. This approach is completely different to other approaches in the Petri net area like [16, 17], where a single SW module, a single hardware module and the mapping exist. Our approach is a generalisation of the hierarchical characterisation of workloads as described in [5] and realised in the performance modeling tool HIT [6].

The idea of top-down refinement and bottom-up abstraction offers different possibilities to reuse components. Virtual machines can be reused by different SW components since they provide a clearly defined interface of available services. SW components can also be reused since they define their service requirements to be provided by some underlying (virtual) machine. In this way it is straightforward to define reusable libraries of components.

A model is completely specified in our framework, if for all subnet places detailed specifications exist. Subnets of a model can be numbered consecutively from 0 through $J-1$. We define a relation $i \rightarrow j$ if j is the detailed specification of a subnet place that is part of i. Furthermore we use $\Rightarrow$ for the transitive closure of $\rightarrow$, i.e., $i \Rightarrow j$ implies the existence of submodels $i_1, \ldots, i_n$ such that $i \rightarrow i_1 \rightarrow \ldots \rightarrow i_n \rightarrow j$. According to relation $\rightarrow$ we can define a directed graph with the submodels building the vertices and a directed edge from i to j if $i \rightarrow j$. We denote this graph as the hierarchy graph of the model. Hierarchy graphs have to be acyclic, because otherwise $i \Rightarrow j \Rightarrow i$ has to hold for some i, j which implies a recursion of some function calls. Cyclic hierarchy graphs are beyond the scope of the paper, they complicate structured analysis significantly. Hierarchy graphs need not be trees and for many realistic models they will not be trees. The tree structure of a hierarchy graph is destroyed if for two submodels i and j, which are not related via $\Rightarrow$, a submodel k exists such that $i \Rightarrow k$ and $j \Rightarrow k$. In this case, components with more than one parent exist. These components are enclosed components. The hierarchy graph of many realistic models has a diamond like structure. Starting with a single net, first a number of independent subnets exist which realise different functions of the SW (cf. Sect.

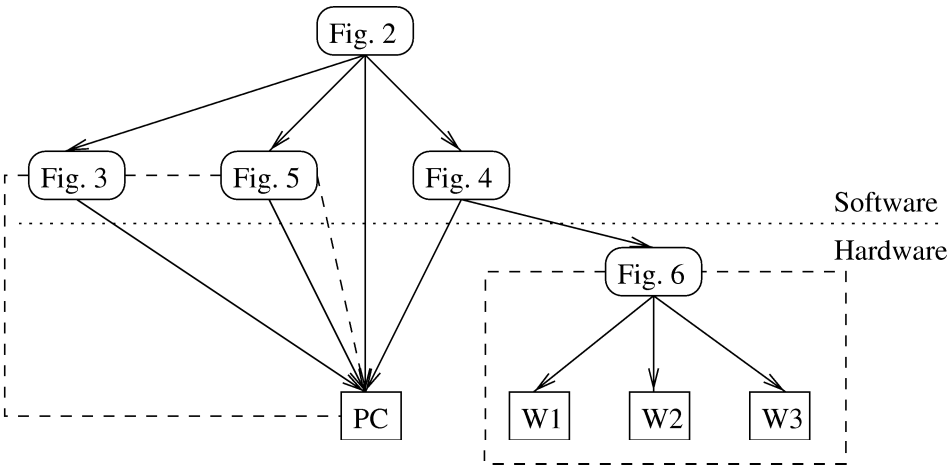

Figure 7: Hierarchy graph of the example model

2). At the lower levels the functions use services provided by a few hardware components, such that the number of vertices in lower levels decreases (cf. Sect. 3).

Finally we define an independent subgraph of a hierarchy graph as a subset of nodes N such that all children of a node $n \in N$ are also elements of N. Independent subgraphs support several efficient solution techniques as shown in the following section. An independent subgraph can be generated by cutting the hierarchy graph into parts such that only vertices and no edges are cut. The interfaces of the vertices that are cut, describe the interface of the independent subgraph.

Example (continued):
We consider in our example a hardware configuration with 3 workstations running the simulations and one PC *per user. On the* PC *the user does his editing and the preparation and evaluation of the simulation run. Since we have only three workstations, but the virtual machine in Fig. 6 contains four stations, we map stations 3 and 4 onto the same physical station. Fig. 7 gives the hierarchy graph. Nodes Fig. 2-6 describe the subnets shown in the Figures 2-6, respectively. The* 4 *bottom level nodes describe the* PC *and the 3 workstations which are all modeled by queues. In the graph, a directed arc points to a subnet which belongs to a subnet place in the net the arc starts from. The graph shows a boundary between SW and HW, which depends on the interpretation of the modeler. We could as well assume that Fig. 6 belongs to the SW and the hardware is realised by the queues only. Alternatively one could assume that the nets in Figs. 3-5 belong to a virtual machine with a very powerful instruction set.*

Two independent subgraphs are described by the dashed lines in the graph. The interface of the first independent subgraph, including the PC and all processes running on the PC*, is realised by the interfaces of the nets in Fig. 3, 5 and by the services* edit_doc, read_model *and* process_results *provided by the* PC*. The interface of the second indepen-*

dent subgraph corresponds to the interface of the net in Fig. 6. Existence of the workstations *is completely hidden from the upper levels.*

5 Hierarchical Analysis Approaches

Although we focus on performance analysis, obviously functional aspects have to be considered in system analysis as well. Particularly if complex HW/SW models are analyzed, where we cannot a priori assume that the model behaves functionally correct. An integrated analysis approach is required, which analyzes functional and quantitative behaviour from one base model. The goal of functional analysis is to assure a "correct" behaviour, while quantitative analysis is used to determine results according to the performance and/or reliability.

A rich variety of analysis techniques exists, see [19, 23] for functional and [21, 24] for quantitative analysis. Since a HQPN can be mapped on a flat model simply by representing a subnet place by its most detailed representation, every analysis approach for flat models can be applied to the hierarchical models. The most detailed methods are state based, which enumerate the state space of the complete model and analyze the corresponding state transition system. For quantitative analysis this means to analyze the resulting Markov chain according to its stationary and transient distribution. Although state based analysis is well understood, practical problems arise due to the enormous complexity of realistic models. Usually the state space grows exponentially with the size of the model in terms of places, queues and tokens. Thus even harmless looking models generate state spaces with several millions or billions of states. Alternatives are non-state space techniques which avoid the state space generation, e.g. invariant analysis for functional analysis of Petri nets [23]. Invariants can be computed for many nets with a state space too large to be analyzable. However, invariants usually give only necessary or sufficient conditions for certain functional results like liveness or boundedness, but no characterisation. In performance analysis, product form models allow efficient computation of performance results. In fact, a class of product form QPNs has been defined [2]. Unfortunately, product form imposes very restrictive conditions on the model structure which are rarely observed in realistic models. Apart from product form, a large number of approximate techniques have been proposed for SPNs or QNs in literature. These techniques are usually based on decomposition and aggregation and are introduced at most in a semi-formal way for specific models. Below we outline that most of the published decomposition and aggregation techniques can be formally integrated in our hierarchical model structure. The last choice for model analysis is always simulation.

The central idea of hierarchy exploitation for analysis purposes is divide and conquer. For exponentially growing problems, it is always better to analyze several smaller problems instead of one large. We will start with state space and reachability graph generation. This can be done efficiently in a top-down way. Observe that each subnet in the hierarchy is executable if we know the incoming transitions from the environment.

The input/output behaviour of subnets, as relevant in the higher level net, is completely defined by transition *t_output* and the marking of place *actual_population*. State spaces can be generated top-down starting from the top-level component i. Since i has no environment its dynamic behaviour is completely specified, so the state space and transition matrix can be generated. Knowledge of the transition system of component i defines completely the environment for all components j with $i \rightarrow j$. Thus, in a subsequent step we can generate state spaces for all j and compute in this way all local state spaces in a top-down manner. A state of the complete model is defined by refining a state of the top level component down to the bottom of the hierarchy. Reachability of compositionally generated states still has to be proved since *t_output* defines all possible input/output behaviours of a component which may be further restricted by the detailed specification.

For performance analysis, we have to assure that the parts for which state spaces and transition matrices are generated are independent subgraphs. An independent subgraph is handled as a flat model by substituting all subnet places by their most detailed realisation. For the resulting flat QPN subnet, state space and transition matrices are generated. Possible arrival sequences of tokens from the environment into the subnet are known since state spaces and transition matrices in the upper levels have already been generated in the top-down approach. The resulting matrices for the independent subgraphs can be combined in a hierarchical way via tensor products to represent the generator matrix of the underlying Markov chain. This extremely compact representation of the generator matrix allows a space efficient numerical analysis of large models [9]. This technique extend the size of exactly solvable models on contemporary workstations to some million states.

Larger models are analyzed approximately by decomposition and aggregation. Independent subgraphs are essential for this kind of analysis as well. Flat models underlying independent subgraphs are analyzed in isolation by assuming a known behaviour for the environment. Local functional analysis can be based on the local reachability graph or local invariant analysis. For quantitative analysis the idea is to assume, usually exponentially distributed, interarrival times for calls to the subsystem and analyze the component under this arrival stream. This results in mean sojourn times for calls in the component, which can be obtained by state based analysis or simulation. These sojourn times usually depend on the number of pending function calls (i.e., the marking of place *actual_population*). In the parent component a subgraph is subsequently represented by the place *actual_population* and transition *t_output* assuming an exponential firing delay with a rate equal to the inverse of the mean sojourn time and depending on the marking of *actual_population*. This strongly reduces the state space of the parent component. If the parent component is not the top-level component of the hierarchy, analysis proceeds as described before. In this way the whole model is analyzed from bottom to top. From the analysis of the top-level component we get the mean interarrival times of tokens to the different subnet places. Using the assumption of exponentially distributed interval times with the computed mean values, the components in the second level of the hierarchy

can be analyzed again. Thus the second analysis phase goes from top to bottom using interarrival times computed in the higher level for the analysis of lower level models. The whole approach is iterated until values do not change too much from one iteration step to the next. Analysis becomes a non-linear fixed point problem, for which we usually cannot assure existence and uniqueness of the fixed point and we can also not assure convergence. However, our experience and also the experience of many other authors using similar techniques [13, 14, 22] show fast convergence and acceptable results for many models.

Even simulation of the complete model profits from the hierarchical structure. It is very convenient to use an object oriented style for HQPN simulators. Independent subgraphs yield objects with a clearly defined interface to their environment. This object structure is also useful to distribute objects in a distributed simulation on a workstation cluster.

Example (continued):
We consider the hierarchical top-down generation of state spaces and present some ideas how to use aggregation bottom-up.

State space generation starts with user process schemata *shown in Fig. 2. For a single user the state space contains* 5 *tangible states and* 8 *transitions. A N-user system has* 5^N *states and* 8^N *transitions. Knowledge of the state space and transitions for the* user process schemata *determines possible arrivals to the included subnets. As an example we consider subnet* prepare *as shown in Fig. 3. From the transition system of the* user process *we know that at most one token per user can be in* prepare *and that single tokens arrive. With this information the state space and transition system for* prepare *can be generated. For one user we obtain* 4 *tangible markings and* 4 *transitions including the situation that no token is in the subnet (i.e., the user does not prepare a simulation run). For the N-user case, the number of states equals* $\sum_{i=0}^{N} \binom{N}{i} 3^i$. *The next step would be to generate the state spaces for subnets* invoke_GUI *and* edit_params. *This step is more crucial since both subnets and some additional subnets are mapped on the* PC. *In general the population of the places* invoke_GUI, edit_param, read_model, process_results, edit_doc, stat_ana *and* new_params, *determine the state space of* PC. *From top-down state space generation it is known that each user performs at most one operation at a time on his* PC. *Thus, if for example a user edits a document he will not prepare a new simulation run. Nevertheless, for generation of the complete state space underlying a model, the state space of the first independent subgraph shown in Fig. 7 has to be computed as a whole in one step.*

As an example for bottom-up aggregation of components we consider the aggregation of the virtual machine *shown in Fig. 6. From top-down state space generation it is known that tokens from user x arrive in a batch of size* n_x. *The next batch for this user cannot arrive before all tokens of the former batch left the subnet. For a single user and batches of size* 4 *the number of states is* 80, *increasing the batch size to* 8 *increases the number of states to* 956. *For two users the state space contains* 6400 *and* 913936 *states, respectively.*

For efficient analysis it is usually necessary to represent the whole component by some aggregate. Usually an aggregate is defined as a single station with exponential service times and state dependent service rates. Since the subnet serves single tokens also the aggregate serves single tokens. Service rates of the aggregate are by short circuit analysis or by analysing the subnet under a Poisson arrival stream of batches, where the rate of the Poisson process for batch arrivals results from the analysis of the remaining parts of the model which form the environment.

6 Conclusions

In this paper we have shown how HQPNs can be used to model complex SW systems in a hierarchical manner and map these onto models of HW systems. Other approaches intended to model parallel programs restricted themselves to exclusive access for a processor by using a preselection policy [16, 17]. Clearly for modeling concurrent programs in computer networks, less simple scheduling strategies need to be described. Our approach simplifies modeling scheduling strategies by incorporating queueing networks into the modeling formalism. Furthermore, the notion of colours and tags are powerful means to use subnets either independently or shared. Clearly these concepts are less powerful than general fusion sets [18], but for good reasons, since they ensure a clear model structure prohibiting arbitrary interconnections across subnets. The effect is similar to prohibiting global variables or arbitrary side-effects of local functions.

Although we focus on the descriptive power of HQPNs, it shall be mentioned as well, that the chosen kind of hierarchy reveals significant advantages for analysis, e.g. numerical analysis [9, 11] and simulation [10]. This desired side effect follows from a careful interface definition between levels which implies a definition of a subnet behaviour as experienced from an upper level. This also allows analysis of models which are not refined up to the lowest level, which is especially useful in early design phases, when information of most details is not available yet.
So far the approach is rudimentarily supported in HiQPN-Tool[1].

References

[1] F. Bause, P. Buchholz, P. Kemper; QPN-tool for the specification and analysis of hierarchically combined queueing Petri nets; In: *H. Beilner, F. Bause (eds.); Quantitative Evaluation of Computing and Communication Systems; Springer LNCS 977 (1995), 224-238.*

[2] F. Bause, P. Buchholz; Aggregation and disaggregation in product form queueing Petri nets; In: *Proc. of the Int. Workshop on Petri Nets and Performance Models 1997 (PNPM'97), IEEE Press, 16-25.*

[3] G. Balbo, S. Donatelli, J. Franceschinis; Understanding parallel program behavior through Petri net models; *J. Parallel and Distributed Comp. 15 (3) 1992, 171-187.*

[4] F. Bause; Queueing Petri Nets — a formalism for the combined qualitative and quantitative analysis of systems. In: *PNPM'93, IEEE Press (1993), 14-23.*
[5] H. Beilner; Workload characterisation and performance modelling tools; In: *Proc. of the Int. Workshop on Workload Characterization of Computer Systems, 1985.*
[6] H. Beilner, J. Mäter, Weissenberg; Towards a performance modelling environment: news on HIT; In: *R. Puigjanger (ed.), Modelling Techniques and Tools for Computer Performance Evaluation, Plenum Pub. (1988), 69-88.*
[7] A. Botti, F. De Cindio; From basic to timed net models of Occam: an application to program placement; In: *PNPM'91, IEEE Press (1991).*
[8] A. Botti, F. De Cindio; Process and resource boxes: an integrated PN performance model for applications and architectures; *Proc. Int. Conf. Systems, Man and Cybernetics, Le Touqet, France 1993.*
[9] P. Buchholz; Structured analysis approaches for large Markov chains - a tutorial; *Performance 96 Tutorials, Ecole Polytechnique Federal de Lausanne (1996).*
[10] P. Buchholz; A distributed numerical/simulative algorithm for the analysis of large continuous time Markov chains; *Proc. of the 11th Workshop on Parallel and Distributed Simulation (1997), PADS'97, 4-11.*
[11] P. Buchholz, P. Kemper; Numerical analysis of stochastic marked graph nets; In: *PNPM'95, IEEE Press (1995), 32-41.*
[12] G. Chiola, M. Ajmone Marsan, G. Balbo, and G. Conte; Generalized stochastic Petri nets: a definition at the net level and its implications; In: *IEEE Trans. Software Engineering; 19 (2) 1993, 89-107.*
[13] J. Campos, J. M. Colom, H. Jungnitz, M. Silva; A general iterative technique for approximate throughput computation of stochastic marked graphs; In: *Proc. of the 5th Int. Work. on Petri Nets and Performance Models, IEEE Press (1993), 138-147.*
[14] G. Ciardo, K. Trivedi; A decomposition approach for stochastic reward net models; *Performance Evaluation 18 (1994), 37-59.*
[15] T. Demaria, G. Chiola, G. Bruno; Introducing a color formalism into generalized stochastic Petri nets. In: *Proc. 9th Int. Work. Application and Theory of Petri Nets (1988); 202-215.*
[16] S. Donatelli, J. Franceschinis; The PSR methodology: integrating hardware and software models; In: *J. Billington, W. Reisig (eds.), Application and Theory of Petri Nets 1996, Springer LNCS 1091 (1996), 133-152.*
[17] A. Ferscha; A Petri net approach for performance oriented parallel program design; *Journal of Parallel and Distributed Computing 15 (3) 1992, 188-206.*
[18] P. Huber, K. Jensen, R.M. Shapiro; Hierarchies in coloured Petri nets; *Advances in Petri Nets, Springer 1990, LNCS 483, 313-341.*
[19] K. Jensen; Coloured Petri Nets Vol. 1; Springer EATCS Monographs on Theoretical Computer Science (1992).
[20] J.B. Jørgensen, K.H. Mortensen; Modeling and analysis of distributed program execution in BETA using coloured Petri nets; In: *J. Billington, W. Reisig (eds.), Application and Theory of Petri Nets 1996, Springer LNCS 1091 (1996), 249-268.*
[21] K. Kant; Introduction to computer system performance evaluation; *Mc Graw Hill (1992).*
[22] Y. Li, C. M. Woodside; Complete decomposition of stochastic Petri nets representing generalized service networks; *IEEE Trans. on Comp. 44 (1995), 577-592.*
[23] T. Murata; Petri nets: properties, analysis and applications; *Proc. of the IEEE 77 (1989), 541-580.*
[24] W. J. Stewart; Introduction to the numerical solution of Markov chains; *Princeton University Press (1994).*