Hierarchically Combined Queueing Petri Nets

Falko Bause, Peter Buchholz and Peter Kemper

University of Dortmund, Informatik IV, D-44221 Dortmund, Germany
e-mail: {bause,buchholz,kemper}@ls4.informatik.uni-dortmund.de

1 Introduction

Combining Petri Nets with Queueing Networks yields a modelling formalism which allows a convenient description of queues and scheduling strategies as well as conflicts and synchronisation aspects. In this paper we combine Petri Nets and Queueing Networks to a two-level hierarchy, hence called hierarchically combined Queueing Petri Nets. This hierarchical modelling formalism supports an efficient technique for quantitative analysis based on Markov chains. The analysis technique allows analysis of models with a state space about 10 times larger than conventional algorithms do. If the model description reveals symmetries this factor grows towards several orders of magnitude. The benefits of the approach are illustrated by a non-trivial example.

2 The QPN modelling formalism

Queueing Petri Nets (QPNs) [BB1, Ba1] combine Coloured Generalized Stochastic Petri Nets (CGSPNs) [AB1] with Queueing Networks (QNs) by hiding stations in special places of the CGSPN which are called timed queueing places. Here we regard a natural generalisation of QPNs that allows to integrate a QN into a place of the CGSPN. This specification can be regarded as a two-level description of a system where the QNs of all timed queueing places form the low level part of the model and the Petri Net structure is seen as the top level.

Figure 1 i) depicts such a timed queueing place and its pictorial representation. Here a QN with two stations is integrated into the place of a CGSPN. Such a timed queueing place consists of two components, the QN and a depository for tokens having completed their service at this QN. Tokens, when fired onto a timed queueing place by any of its input transitions, are inserted into one of the queues of the embedded QN. Tokens within a QN are not available for the QPN transitions. After completion of its service at a QN station, a token moves to a different one or leaves the QN part. Tokens that leave the QN part move to the depository of the timed queueing place. Here tokens are available for all output transitions of the timed queueing place. An enabled timed transition will fire

Fig. 1. i) Example timed queueing place of a QPN, ii) its pictorial representation, and iii) its simplified interpretation from a Petri Net point of view

after a certain exponentially distributed delay according to a race policy like in GSPNs. Enabled immediate transitions will fire according to relative firing frequencies. The firings of immediate transitions have priority over those of timed transitions. Every QPN describes a stochastic process which can be analysed by Markovian analysis techniques [Ba1].

3 Generation of the Markov Chain for a QPN

Apart from a more convenient specification the hierarchical structure of a QPN can be exploited for solution purposes. The solution is based on the analysis of the Markov chain (MC) underlying a QPN. The MC has a structure which is strongly related to the model structure and therefore allows a very efficient computation of state space and generator matrix. Furthermore the structure can be exploited for various analysis techniques as shown below. However, due to space limitations we can only give a very brief description of the approach, for a more comprehensive introduction the reader is referred to [Bu1, Bu2].

On the PN level of the QPN each timed queueing place behaves comparable to two places connected by a timed transition. The first place includes tokens actually in the QN underlying the timed place, the second place includes the tokens on the depository, which are ready to leave, see Fig. 1 iii. Let P_{QN} be the set of all places including QNs and T_{QN} the set of newly introduced transitions specifying the output behaviour of the embedded QNs. We impose the following restrictions onto the embedded QNs to simplify the description of the analysis technique: Tokens do not change their colour inside a QN and whenever there is at least one colour c token inside a QN this token can be the next to leave.

These restrictions ensure that transitions from T_{QN} fire at a given time one token from the first to the second place without changing the token colour. A transition $t \in T_{QN}$ does not describe an exponentially distributed delay, however, it behaves qualitatively like an ordinary (exponential) timed transition due to the infinite support of exponential distributions. Note that this point of view just considers qualitative aspects concerning the state space of the PN part. The state space of the PN can be computed using standard methods for GSPNs (see [AB1]) without taking care of the embedded QNs. Vanishing markings can be eliminated locally in the PN part of the model using known techniques from GSPN analysis

[AB1] without effecting any embedded QN. Let M_0 be the marking space of tangible markings resulting from that net. M_0 is, of course, not isomorphic to the state space of the underlying MC, but describes an aggregated view on this state space by hiding the detailed states of all included QNs. Let Q_0 be the transition matrix of the PN part. Each non-zero in Q_0 is originated by a timed transition in the PN or an entity leaving a QN. If it is originated by a timed transition, then $Q_0(n,m) = \lambda_{n,m}$, where $\lambda_{n,m}$ is given by the firing rate of the timed transition possibly multiplied with the probabilities of subsequent firing immediate transition to yield marking m. If it is originated by a transition from T_{QN}, then $Q_0(n,m) = (p_{n,m}, i)$, where $p_{n,m}$ is given by the probabilities of firing immediate transitions to yield marking m and i is the number of the QN where the customer (token) leaves. Furthermore define $Q_0(n,n) = \lambda_n = -\sum_{m \neq n} \lambda_{n,m}$.

The markings from M_0 determine the possible population vectors in the embedded QNs. Let QN_j be the QN embedded in timed queueing place p_j and N_j be the set of all possible population vectors in QN_j. By interpreting the tokens of the PN as customers in the embedded QNs and colours of tokens as customer chains we have $n \in N_j$ if $n(k) = m_j(k)$ for some $m \in M_0$.

The set N_j defines the possible interaction of QN_j with its environment realised by the rest of the model. Thus it is possible to specify QN_j in combination with a finite capacity source and a sink to compute transition matrices and state space. Let the population in QN_j be n, the finite capacity source possibly generates a chain k entity, if $n + e_k \in N_j$. If customers of chain k can be generated, then the interarrival time is exponentially distributed with a fictive rate $\lambda > 0$. Inside QN_j customers of chain k can have an additional class identity from a set of classes $\mathcal{K}_k$ ($\mathcal{K}_k \cap \mathcal{K}_l = \emptyset$ for $k \neq l$). A customer of chain k leaving QN_j looses his class identity and is absorbed by the sink in the isolated specification.

In combination with the source/sink pair QN_j can be interpreted as an isolated model which is mapped on the underlying MC. Let M_j be the state space and Q_j be the corresponding generator matrix. M_j is decomposed into disjoint subsets $M_j(n)$ ($n \in N_j$) according to the actual population in QN_j. In the same way Q_j is decomposed into submatrices $Q_j^{n,m}$ including all transitions between states from $M_j(n)$ and $M_j(m)$. The following interpretation can be used for the different submatrices:

- If $n = m$, then $Q_j^{n,n} = Q_j^n + \sum_{k \in K} \delta(n(k))\lambda I$ and Q_j^n includes all rates of transitions locally in QN_j and in the main diagonal the negative sum of transition rates of possible transitions originated in QN_j.
- If $n - e_k = m$, then $Q_j^{n,m} = S_j^{n,m}$ includes the departure rates of chain k customers from QN_j.
- If $n + e_k = m$, then $Q_j^{n,m} = \lambda U_j^{n,k}$ and $U_j^{n,k}$ includes conditional arrival probabilities of chain k customers into QN_j.
- In all other cases $Q_j^{n,m} = 0$.

Let M be the state space of the complete model, M can be decomposed into disjoint subset $M(n)$ according to the state $n \in M_0$. Thus each state is described by a state of the PN and a state for each embedded QN. M is composed from the isolated state spaces as follows.

$$M = \cup_{m \in M_0} M(m) = \cup_{m \in M_0} \times_{j \in P_{QN}} M_j(m_j) \tag{1}$$

The size of the overall state space grows with the product over the sizes of the subspaces of the included QNs which yields the well known state explosion. Nevertheless M has a very regular structure and can be generated efficiently from the isolated parts. The next step is to compute the generator matrix Q of the MC underlying the QPN. This matrix can be computed from the isolated matrices following the ideas of state space generation. Q is decomposed into submatrices $Q^{n,m}$ according to the state of the PN part. Before we describe the computation of submatrices some additional functions have to be defined. The following two functions are used to map the state of a QN into the state of the complete model.

$$l_j(n) = \prod_{i<j, p_i \in P_{QN}} \|M_i(n_i)\|, \qquad u_j(n) = \prod_{i>j, p_i \in P_{QN}} \|M_i(n_i)\| \tag{2}$$

where $\|\ldots\|$ denotes the cardinality of a set

To describe the behaviour of QNs embedded in the PN we have to specify the behaviour after the arrival of a batch of customers which is possible from the PN but has not been considered in the isolated specification. Define for $n, m \in N_j$ with $n(k) \leq m(k)$ an ordered set $\Delta_j(n, m) = \{(n^1, k^1), \ldots, (n^X, k^X)\}$ such that $n^x = n^{x-1} + e_k$ for some $k \in K$, $0 \leq x \leq X$, where $n^0 = n$ and $n^X = m$. The difference $m - n$ defines the arriving batch of customers, $\Delta_j(n, m)$ is used to define an ordering on this set, thus the batch is handled like a sequence of customers arriving at the same time.

$$U_j^{n,m} = \prod_{(n',k) \in \Delta_j(n,m)} U_j^{n',k} \tag{3}$$

Furthermore let $A(n, m)$ for $Q_0(n, m) \neq 0$ $(n, m \in M_0)$ be the set of QNs which receive new customers when the marking of the PN changes from n to m. Elements from $A(n, m)$ are ordered according to the numbering of the involved QNs. Now we can compute the submatrices $Q^{n,m}$ as follows (see [Bu1, Bu2] for further details).

$$Q^{n,m} = \begin{cases} \lambda_n I_{\|M(n)\|} + \sum_{j \in P_{QN}} I_{l_j(n)} \otimes Q_j^{n_j} \otimes I_{u_j(n)} & \text{if } n = m \\ \lambda_{n,m} \prod_{j \in A(n,m)} I_{l_j(m)} \otimes U_j^{n_j,m_j} \otimes I_{u_j(n)} & \text{if } Q_0(n,m) = \lambda_{n,m} \\ p_{n,m} I_{l_i(n)} \otimes S_i^{n_j,m_j} \otimes I_{u_i(n)} \prod_{j \in A(n,m)} I_{l_j(m)} \otimes U_j^{n_j,m_j} \otimes I_{u_j(n')} & \\ & \text{if } Q_0(n,m) = p_{n,m}, i \\ 0 & \text{if } Q_0(n,m) = 0 \end{cases}$$

where I_x is the identity matrix of order x and $\otimes$ is the tensor product
$n'_j = n_j$ if $j \neq i$ and $n'_i = m_i$

(4)

With the above formula the generator matrix can be computed efficiently, but the size of the matrix for more complex models is often too large to be stored completely using todays computers. In the next section we consider very briefly analysis methods exploiting the structure of the MC to deal with large models.

4 Analysis of QPNs

Analysis of complex MCs faces two problems: time and space requirements. The state space of MCs underlying realistic models often contains more than 10^6 states. Although generator matrices are very sparse, they can exceed storage capacities and even the generation of the state space is not possible. Thus memory requirements are often the most important problem. The use of virtual memory to increase space is often of no help since access time to virtual memory is slow increasing the overall solution time dramatically. Our main goal here is to manage the space problem which often implicitly attacks time complexity.

State space generation as defined above is very efficient since only small subspaces are generated using the conventional algorithms which need to compare a newly computed state with the states already generated. The complete state space and generator is computed using well defined operations which are very efficient. However, this does not help for the solution. The solution requires the determination of the steady state distribution π given as the solution $\pi Q = 0$ normalised to 1, or the transient distribution at time t $\pi(t)$ given as the solution of $\pi(t) = \pi(0)e^{Qt}$. For both solutions iterative solvers are used, e.g. the iteration

$$p^{k+1} = p^k + 1/q_{max} p^k Q \text{ where } q_{max} = 1.001 \max(|Q(i,i)|) \tag{5}$$

converges for irreducible MC and $k \rightarrow \infty$ against π. The vector $\pi(t)$ is given by $\sum_{k=0}^{\infty} p^k e^{q_{max} t}(q_{max} t)^k / k!$, where $p^0 = \pi(0)$ and the vectors p^k are computed as shown above. Other iterative approaches, in particular for stationary analysis, exist, but they all follow similar ideas.

To use the structure of the MC in the solution we decompose all vectors into subvectors according to the state of the PN, i.e. π_n is a $||Z(n)||$-dimensional vector including the steady state probabilities of the states from $Z(n)$. With this decomposition the above iteration is equivalent to:

$$p_n^{k+1} = p_n^k + 1/q_{max} \sum p_m^k Q^{m,n} \tag{6}$$

Each product $p_m^k Q^{m,n}$ is the sum or product of matrices of the form $I \otimes A \otimes I$. Such a matrix has a very regular structure and the multiplication of a vector with a matrix of this type can be performed using only A and the dimension of the identity matrix. A procedure performing the multiplication is given in [Bu2]. Thus it is possible to iterate using only Q_0 and the QN matrices, the huge matrix Q never needs to be computed and stored as a whole. Compared with conventional techniques, computing first Q, MCs can be solved which are approximately larger by the order of a magnitude. With the new approach the iteration and solution vector becomes the limiting factor of an analysis and not the generator matrix, thus we reach some natural limit for analysis approaches which compute the state vector as a whole. However, the use of tensor products only reduces the space complexity, the number of operations is similar to the conventional approach. Nevertheless, the additional effort for some address transformations is rather small for non-trivial QNs due to the large amount of data locality given by the QN matrices.

To decrease also the time complexity of a solution, a special structure of the model is required or an approximation error has to be introduced. A special structure is given, if the PN is symmetric and the embedded QNs in symmetric parts are identical, then an aggregated MC resulting from lumpability can be generated and used for the analysis (see [Bu3]).

5 Example

This example illustrates the modelling and analysis of distributed systems using the QPN formalism. The system architecture is given by $n{+}1$ processors ($proc_j$,manager) with local memory (LM). One processor (manager) is dedicated to a special task, collecting results and informing each processor of the new status. This processor is a multiprocessor system consisting of two processors with separate queues and arriving jobs are inserted into the shortest queue. All processors are connected via a bidirectional bus which must be used exclusively. This architecture is employed for the following iteration problem:

$$f(x^k) = x^{k+1} \text{ where } f(x) = (f_1(x), \ldots, f_t(x)); x, x^k, x^{k+1} \in \mathbb{R}^m \tag{7}$$

f is a vectorial function and each processor $proc_j$ calculates the part $f_i(x^k)$ of f for a given x^k where $i = j + np$ for some $p \in \mathbb{N}_0$. We assume that calculation of the fixed point takes a huge number of iterations, and that the complexity of computing $f_i(x^k)$ requires several local memory accesses at each local processor. Furthermore we assume that $t > n$, i.e. the size of the problem is greater than the number of available processors. The whole system behaves as follows:

After computation of $f_i(x^k)$ processor j transmits the result (x_i^{k+1}) to the manager who updates its local copy of x^k. If the manager has received the results of all processors, i.e. the value $f_i(x^k)$ for all i, it distributes the updated vector x^{k+1} to all processors via broadcasting. Figure 2 shows the corresponding QPN model using the expression representation for Coloured Petri Nets [Je1]. Each processor is modelled by a timed queueing place whose QN part is as in Fig. 1 i). The components x_i^k and $f_i(x^k)$ are represented by a token of colour f_i. After computation $proc_j$ transmits $f_i(x^n)$ to the manager provided the bus is available. The manager updates its local copy. The synchronisation aspect of the iteration is represented by t_2 which consumes t tokens from manager if the bus is available. Broadcasting is modelled by t_1 firing tokens of colour f_i.

Fig. 2. QPN with $k_i = (t \text{ div } n) + g(i, n, t)$ and $g(i, n, t) := 1$ if $i \le t \bmod n$ and 0 else.

		conventional		hierarchical		reduced		
t	$\|M\|$	non-zeros	time	non-zeros	time	$\|M_{red}\|$	non-zeros	time
9	6.42e+4	5.41e+5	3.80e+3	4.64e+3	7.45e+2	6.00e+3	6.33e+2	1.89e+2
13	1.18e+6	–	–	2.37e+4	2.14e+4	7.76e+4	2.02e+3	2.32e+3
20	–	–	–	–	–	1.71e+6	7.62e+3	2.05e+5

Table 1. Sizes and solver performance for different configurations

The example model has been analyzed for $n = 5$ processors and an increasing number of tasks. Table 1 summarises the performance of the analysis approaches. All experiments have been performed on a workstation with $24MB$ main memory and $42MB$ virtual memory. The first column of the table includes the value of t, the second column the number of states of the MC, column 3 and 4 show the number of non-zero elements and the elapsed time in seconds which is needed to compute the solution vector with an estimated accuracy of 0.01% including the times to generate state spaces and matrices from the model specification for the conventional approach. Columns 5 and 6 include the corresponding quantities for the hierarchical approach. For the conventional analysis immediate transitions are eliminated on net level. The last three columns of the table show the number of states, non-zero elements in the submatrices and analysis time for a reduced MC resulting from an exploitation of the identities of processors. The technique used for reduction has been introduced in [Bu3]. The largest MC which can be generated and analyzed with the conventional approach results from the model with $t = 9$, for $t = 10$ we were not even able to generate the state space in an acceptable amount of time (i.e. less than 24 hours). With the new approach we can analyze the model for $t = 13$ which results in a MC with a state space twenty times larger than the one to be analyzable with conventional techniques. Additionally the new approach is faster for large models since paging is not necessary. Exploitation of symmetries yields a significant increase in the size of models which are solvable. Combining this reduction with the new analysis technique allows us to analyze models with t up to 20 which results in a MC about 3 orders of a magnitude larger than the MC solvable with the conventional approach.

References

[AB1] Ajmone-Marsan, M., Balbo, G., Conte, A.: Performance models of multiprocessor systems. MIT Press, London, 1986.

[BB1] Bause, F., Beilner, H.: Eine Modellwelt zur Integration von Warteschlangen- und Petri-Netz-Modellen. Springer IFB 218, 1989, pp. 190-204.

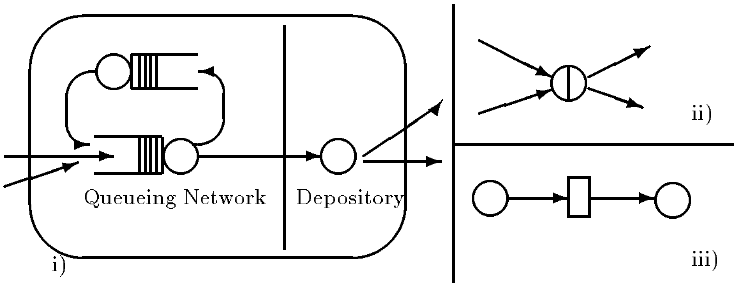
[Ba1] Bause, F.: Queueing Petri nets: a Formalism for the Combined Qualitative and Quantitative Analysis of Systems. 5th International Workshop on Petri Nets and Performance Models, Toulouse (France), 1993.

[Bu1] Buchholz, P.: Hierarchies in Colored GSPNs. Springer LNCS 691, 1993.

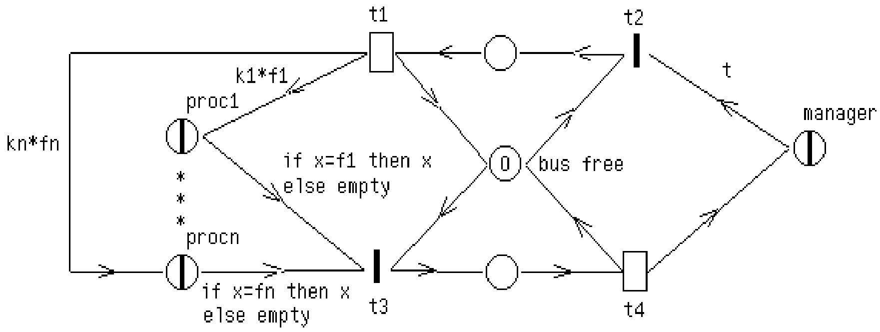
[Bu2] Buchholz, P.: A Class of Hierarchical Queueing Networks and Their Analysis. to appear in QUESTA

[Bu3] Buchholz, P.: Hierarchical Markovian Models -Symmetries and Reduction-. Computer Performance Evaluation '92, Edinburgh University Press 1993

[Je1] Jensen, K.: Coloured Petri Nets. EATCS, Springer, 1992.



Queueing Network
Depository
i)
ii)
iii)


t1
t2
k1*f1
t
proc1
manager
kn*fn
if x=f1 then x
else empty
0
bus free
*
*
*
procn
if x=fn then x
else empty
t3
t4