Available online at www.sciencedirect.com

Performance Evaluation 63 (2006) 364–394

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

SimQPN—A tool and methodology for analyzing queueing Petri net models by means of simulation[☆]

Samuel Kounev[*], Alejandro Buchmann

Department of Computer Science, Darmstadt University of Technology, Germany

Received 17 June 2004; received in revised form 2 February 2005
Available online 4 May 2005

Abstract

The queueing Petri net (QPN) paradigm provides a number of benefits over conventional modeling paradigms such as queueing networks and generalized stochastic Petri nets. Using queueing Petri nets (QPNs), one can integrate both hardware and software aspects of system behavior into the same model. This lends itself very well to modeling distributed component-based systems, such as modern e-business applications. However, currently available tools and techniques for QPN analysis suffer the state space explosion problem, imposing a limit on the size of the models that are tractable. In this paper, we present SimQPN—a simulation tool for QPNs that provides an alternative approach to analyze QPN models, circumventing the state space explosion problem. In doing this, we propose a methodology for analyzing QPN models by means of discrete event simulation. The methodology shows how to simulate QPN models and analyze the output data from simulation runs. We validate our approach by applying it to study several different QPN models, ranging from simple models to models of realistic systems. The performance of point and interval estimators implemented in SimQPN is subjected to a rigorous experimental analysis.
© 2005 Elsevier B.V. All rights reserved.

Keywords: Queueing Petri nets; Simulation modeling; Performance prediction; Capacity planning; Distributed component-based systems

[☆] This work was partially funded by BEA Systems, Inc., as part of the project "Performance modeling and evaluation of large-scale J2EE applications" and the Deutsche Forschungsgemeinschaft (DFG) as part of the PhD program "Enabling technologies for E-commerce" at Darmstadt University of Technology.

[*] Corresponding author. Tel.: +49 6151 166231; fax: +49 6151 166229.
E-mail addresses: skounev@informatik.tu-darmstadt.de, skounev@acm.org, skounev@spec.org (S. Kounev), buchmann@informatik.tu-darmstadt.de (A. Buchmann).

0166-5316/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2005.03.004

1. Introduction

The queueing Petri net (QPN) modeling formalism was introduced in 1993 by Bause [5]. It allows queues (queueing stations) to be integrated into the places of Petri nets (more specifically colored generalized stochastic Petri nets) and thus brings the benefits of queueing networks into the world of Petri nets. In [4] it is shown that QPNs have greater expressive power than queueing networks, extended queueing networks and stochastic Petri nets. In addition to hardware contention and scheduling strategies, using QPNs one can easily model simultaneous resource possession, synchronization, blocking and software contention. This enables the integration of both hardware and software aspects of system behavior into the same model [8,22]. While the above could also be achieved by using layered queueing networks (LQNs) (or stochastic rendezvous networks) [45,38,29], the latter are defined at a higher-level of abstraction and are usually less detailed and accurate. Another benefit of QPNs is that, since they are based on Petri nets, one can exploit a number of efficient techniques from Petri net theory to verify some important qualitative properties of QPNs, such as ergodicity, boundedness, liveness or existence of home states. The latter not only help to gain insight into the behavior of QPNs, but are also essential preconditions for a successful quantitative analysis [5].

In [22], we showed that QPNs lend themselves very well to modeling distributed e-business applications with software contention and demonstrated how this can be exploited for performance prediction in the capacity planning process. However, we also showed that modeling a realistic e-business application using QPNs often leads to a model that is way too large to be analyzable using currently available tools and techniques. This is the reason why QPNs have hardly been exploited in the past decade and very few, if any, practical applications have been reported. The problem is that, until now, available tools and solution techniques for QPN models have all been based on Markov chain analysis, which suffers the well known *state space explosion problem* and limits the size of the models that can be analyzed.

This paper shows how the above problem can be approached by exploiting discrete event simulation for model analysis. We present SimQPN—a Java-based simulation tool for QPNs that can be used to analyze QPN models of realistic size and complexity. While doing this, we propose a methodology for simulating QPN models and analyzing the output data from simulation runs. SimQPN can be seen as an implementation of this methodology. We validate our approach by studying several different QPN models by means of SimQPN. Models of different size and complexity are considered, including models of realistic e-business applications such as the SPECjAppServer[1] set of benchmarks. We evaluate the quality of data provided by SimQPN, conducting an exhaustive experimental analysis of the variation of point estimates and coverage of confidence intervals reported.

An alternative approach to simulate QPN models would be to use a general purpose simulation package. However, this approach has some disadvantages. First, general purpose simulation packages do not provide means to represent QPN constructs directly. Instead, they require that simulation models are described using a general purpose simulation language. Mapping a QPN model to a description in the terms of a general purpose simulation language is a complex, time-consuming and error-prone task. Moreover, not all simulation languages provide the expressiveness needed to describe complex QPN models. Some simplifications might be required, which could lead to less accurate results. Another disadvantage is that general purpose simulators are generally not as fast and efficient as specialized simulators, since they are usually not optimized for any particular type of models. Being specialized for QPNs, SimQPN simulates

[1] SPECjAppServer is a trademark of the Standard Performance Evaluation Corp. (SPEC).

QPN models directly and has been designed to exploit the knowledge of the structure and behavior of QPNs to improve the efficiency of the simulation. Therefore, SimQPN is expected to provide much better performance than a general purpose simulator, both in terms of the speed of simulation and the quality of output data provided. Last but not least, SimQPN has the advantage that it is extremely light-weight and being implemented in Java it is platform independent.

The rest of the paper is organized as follows. We start with a brief introduction to queueing Petri nets in Section 2. In Section 3, we present SimQPN—our simulation tool for QPNs, and discuss its features, design and architecture. In parallel to this, we discuss our methodology for simulating QPN models, based on which SimQPN was developed. We look at the methods for output data analysis employed, and discuss the specifics of their implementation. Following this, in Section 4, we study several different QPN models using SimQPN and validate the results with respect to correctness and accuracy. We evaluate the performance of point and interval estimators implemented in SimQPN. Finally, we discuss our ongoing/future work and present some concluding remarks.

2. Queueing Petri nets

Queueing Petri nets can be seen as a combination of a number of different extensions to conventional Petri nets along several different dimensions. In this section, we include some basic definitions and briefly discuss how queueing Petri nets have evolved. A deeper and more detailed treatment of the subject can be found in [9,5,4]. An ordinary *Petri net* (also called *place-transition net*) is a bipartite directed graph composed of places, drawn as circles, and transitions, drawn as bars. A formal definition is given below [9]:

Definition 1 *(PN).* An ordinary Petri net (PN) is a five-tuple $PN = (P, T, I^-, I^+, M_0)$ where

1. P is a finite and non-empty set of places $(p_1, p_2, \ldots, p_{|P|})$,
2. T is a finite and non-empty set of transitions $(t_1, t_2, \ldots, t_{|T|})$,
3. $P \cap T = \emptyset$,
4. $I^-, I^+: P \times \mathrm{T} \to \mathbb{N}_0$ are called backward and forward incidence functions, respectively,
5. $M_0: P \to \mathbb{N}_0$ is called initial marking.

The incidence functions I^- and I^+ specify the interconnections between places and transitions. If $I^-(p, t) > 0$, an arc leads from place p to transition t and place p is called an *input place* of the transition. If $I^+(p, t) > 0$, an arc leads from transition t to place p and place p is called an *output place* of the transition. The incidence functions assign natural numbers to arcs, which we call *weights* of the arcs. When each input place of transition t contains at least as many tokens as the weight of the arc connecting it to t, the transition is said to be *enabled*. An enabled transition may *fire*, in which case it destroys tokens from its input places and creates tokens in its output places. The amounts of tokens destroyed and created are specified by the arc weights. The initial arrangement of tokens in the net (called *marking*) is given by the function M_0, which specifies how many tokens are contained in each place.

Different extensions to ordinary Petri nets have been developed in order to increase the modeling convenience and/or the modeling power. *Colored Petri nets* (CPNs) introduced by Jensen [19] are one such extension. The latter allow a *type* (*color*) to be attached to a token. A color function C assigns a set of colors to each place, specifying the types of tokens that can reside in the place. In addition to introducing token colors, CPNs also allow transitions to fire in different *modes* (*transition colors*). The color function C assigns a set of modes to each transition and incidence functions are defined on a per mode basis.

A formal definition of a CPN follows [9]:

Definition 2 *(CPN).* A colored Petri net (CPN) is a six-tuple CPN = (P, T, C, I^-, I^+, M_0) where

1. P is a finite and non-empty set of places $(p_1, p_2, \ldots, p_{|P|})$,
2. T is a finite and non-empty set of transitions $(t_1, t_2, \ldots, t_{|T|})$,
3. $P \cap T = \emptyset$,
4. C is a color function defined from $P \cup T$ into non-empty sets,
5. I^- and I^+ are the backward and forward incidence functions defined on $P \times T$, such that $I^-(p, t)$, $I^+(p, t)$: $C(t) \to C(p)_{\mathrm{MS}}$, $\forall (p, t) \in P \times T$,[2]
6. M_0 is a function defined on P describing the initial marking such that $M_0(p) \in C(p)_{\mathrm{MS}}$, $\forall p \in P$.

Other extensions to ordinary Petri nets allow temporal (timing) aspects to be integrated into the net description [9]. In particular, *stochastic Petri nets* (SPNs) attach an exponentially distributed *firing delay* to each transition, which specifies the time the transition waits after being enabled before it fires. *Generalized stochastic Petri nets* (GSPNs) allow two types of transitions to be used: immediate and timed. Once enabled, immediate transitions fire in zero time. If several immediate transitions are enabled at the same time, the next transition to fire is chosen based on *firing weights* (probabilities) assigned to the transitions. Timed transitions fire after a random exponentially distributed firing delay as in the case of SPNs. The firing of immediate transitions always has priority over that of timed transitions. Combining CPNs and GSPNs leads to *colored generalized stochastic Petri nets* (CGSPNs) [9]. While CGSPNs have proven to be a very powerful modeling formalism, they do not provide any means for direct representation of queueing disciplines. The attempts to eliminate this disadvantage have led to the emergence of queueing Petri nets.

The main idea behind the QPN modeling paradigm was to add queueing and timing aspects to the places of CGSPNs. This is done by allowing queues (service stations) to be integrated into places of CGSPNs. A place of a CGSPN that has an integrated queue is called a *queueing place* and consists of two components, the *queue* and a *depository* for tokens which have completed their service at the queue. This is depicted in Fig. 1. The behavior of the net is as follows: tokens, when fired into a queueing place by any of its input transitions are inserted into the queue according to the queue's scheduling strategy. Tokens in the queue are not available for output transitions of the place. After completion of its service, a token is immediately moved to the depository, where it becomes available for output transitions of the place. This type of queueing place is called *timed queueing place*. In addition to timed queueing places, QPNs also introduce *immediate queueing places*, which allow pure scheduling aspects to be described. Tokens in immediate queueing places can be viewed as being served immediately. Scheduling in such places has priority over scheduling/service in timed queueing places and firing of timed transitions. The rest of the net behaves like a normal CGSPN. An enabled timed transition fires after an exponentially distributed delay according to a race policy. Enabled immediate transitions fire according to relative firing frequencies and their firing has priority over that of timed transitions. We now give a formal definition of a QPN and then present an example of a QPN model.

[2] The subscript MS denotes multisets. $C(P)_{\mathrm{MS}}$ denotes the set of all finite multisets of $C(p)$.

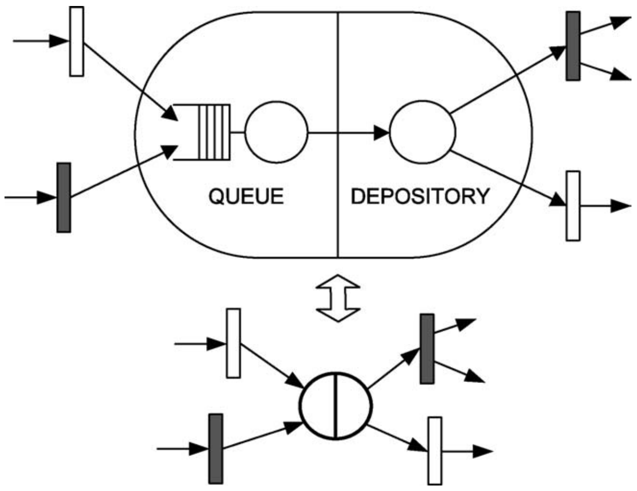

Fig. 1. A queueing place and its shorthand notation.

Definition 3 *(QPN).* A queueing Petri net (QPN) is an eight-tuple QPN = $(P, T, C, I^-, I^+, M_0, Q, W)$ where

1. CPN = (P, T, C, I^-, I^+, M_0) is the underlying colored Petri net.
2. $Q = (\tilde{Q}_1, \tilde{Q}_2, (q_1, \ldots, q_{|P|}))$ where
   - $\tilde{Q}_1 \subseteq P$ is the set of timed queueing places,
   - $\tilde{Q}_2 \subseteq P$ is the set of immediate queueing places, $\tilde{Q}_1 \cap \tilde{Q}_2 = \emptyset$ and
   - q_i denotes the description of a queue taking all colors of $C(p_i)$ into consideration if p_i is a queueing place or equals the keyword 'null' if p_i is an ordinary place.
3. $W = (\tilde{W}_1, \tilde{W}_2, (w_1, \ldots, w_{|T|}))$ where
   - $\tilde{W}_1 \subseteq T$ is the set of timed transitions,
   - $\tilde{W}_2 \subseteq T$ is the set of immediate transitions, $T = \tilde{W}_1 \cup \tilde{W}_2$, $\tilde{W}_1 \cap \tilde{W}_2 = \emptyset$ and
   - $w_i \in [C(t_i) \mapsto \mathbb{R}^+]$ such that $\forall c \in C(t_i)$: $w_i(c)$ is interpreted as the rate of a negative exponential distribution specifying the firing delay due to color c if t_i is a timed transition or a weight specifying the relative firing frequency due to color c if t_i is an immediate transition.

Example 1 *(QPN).* Fig. 2 shows an example of a QPN model of a central server system with memory constraints based on [9]. Place p_2 represents several terminals, where users start jobs (modeled with tokens of color 'o') after a certain thinking time. These jobs request service at the CPU (represented by a $-/C/1-$PS queue, where C stands for Coxian distribution) and two disk subsystems (represented by $-/C/1-$FCFS queues). To enter the system each job has to allocate a certain amount of memory. The amount of memory needed by each job is assumed to be the same, which is represented by a token of color 'm' on place p_1. Note that, for readability, token cardinalities have been omitted from the arc weights in Fig. 2, i.e. symbol o stands for 1'o and symbol m for 1'm. According to Definition 3, we have the following:

QPN = $(P, T, C, I^-, I^+, M_0, Q, W)$ where

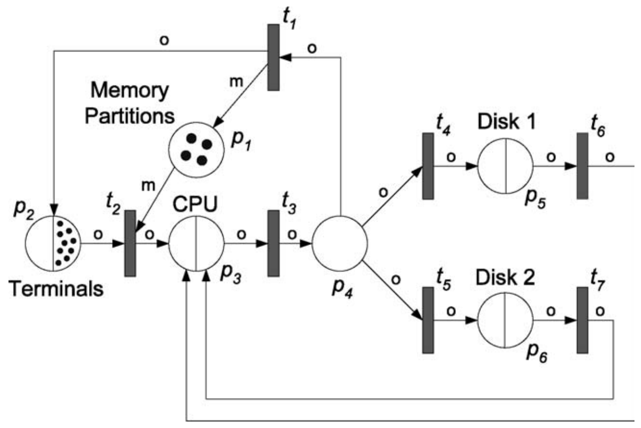

Fig. 2. A QPN model of a central server with memory constraints (based on [9]).

- CPN = (P, T, C, I^-, I^+, M_0) is the underlying colored Petri net as depicted in Fig. 2,
- $Q = (\tilde{Q}_1, \tilde{Q}_2, (\text{null}, -/C/\infty - \text{IS}, -/C/1 - \text{PS}, \text{null}, -/C/1 - \text{FCFS}, -/C/1 - \text{FCFS}))$,
- $\tilde{Q}_1 = \{p_2, p_3, p_5, p_6\}$, $\tilde{Q}_2 = \emptyset$,
- $W = (\tilde{W}_1, \tilde{W}_2, (w_1, \ldots, w_{|T|}))$, where $\tilde{W}_1 = \emptyset$, $\tilde{W}_2 = T$ and $\forall c \in C(t_i)$: $w_i(c) := 1$, so that all transition firings are equally likely.

As already mentioned, the main hurdle to the quantitative analysis of QPNs is the fact that most analysis techniques available are based on Markov chains and are therefore susceptible to the state space explosion problem. More specifically, as one increases the number of queues and tokens in a QPN, the size of the state space of the underlying Markov chain grows exponentially and quickly exceeds the capacity of today's computers. An attempt to alleviate this problem was the introduction of *hierarchically combined queueing Petri nets* (HQPNs) [6]. The main idea is to allow hierarchical model specification and then exploit the hierarchical structure for efficient numerical analysis. This type of analysis is termed *structured analysis* and it allows models to be solved, which are about an order of magnitude larger than those analyzable with conventional techniques. However, while this alleviates the problem, it does not eliminate it, since as shown in [22], models of realistic e-business applications are still too large to be analyzable using this approach.

To the best of our knowledge, there is currently only one tool available that supports modeling and analysis using QPNs [32]. This is the HQPN-Tool presented in [7]. The latter supports a number of structured analysis methods for HQPNs, such as Structured Power, Structured SOR, Structured JOR among others. These methods provide exact solution of the model's underlying Markov chain, however, all of them suffer the state space explosion problem.

3. SimQPN—simulator for queueing Petri nets

In this section, we present SimQPN—our simulator for QPNs, that provides an alternative approach to analyze QPNs, circumventing the state space explosion problem. We take a detailed look at SimQPN's features, design and architecture. In parallel to this, we present our methodology for simulating QPN models, based on which SimQPN was developed. The methodology shows how to simulate QPN models and analyze the output data from simulation runs. We look at the methods for output data analysis employed, and discuss the specifics of their implementation.

3.1. SimQPN features

SimQPN is a discrete-event simulation (DES) engine specialized for queueing Petri nets. It is implemented 100% in Java to provide maximum portability and platform-independence. It is extremely light-weight (less than 1 MB) and requires only an installed Java Runtime Environment (JRE).[3] SimQPN simulates QPNs using a sequential algorithm based on the event-scheduling approach for simulation modeling.

In the first version of SimQPN, the most important features typically used in QPN models have been implemented. As of the time of writing, QPN models with the following restrictions are supported:

- Scheduling strategies for queues are limited to First-Come-First-Served (FCFS), Processor-Sharing (PS) and Infinite Server (IS).
- The following service time distributions are supported for FCFS and IS queues: Beta, BreitWigner, ChiSquare, Gamma, Hyperbolic, Exponential, ExponentialPower, Logarithmic, Normal, StudentT, Uniform and VonMises. For PS queues, currently only exponential service time distributions are supported, which makes it easier to handle residual service times. For the next version of SimQPN it is planned to relax this restriction.
- Empirical distributions are supported in the following way. The user is expected to provide a probability distribution function (PDF), specified as an array of positive real numbers (histogram). A cumulative distribution function (CDF) is constructed from the PDF and inverted using a binary search for the nearest bin boundary and a linear interpolation within the bin (resulting in a constant density within each bin).
- Timed transitions are currently not supported. However, in most cases, a timed transition can be approximated by a serial network consisting of an immediate transition, a queueing place and a second immediate transition.
- Since in practice immediate queueing places are very rarely used, they have been left out from the first version of SimQPN.

The spectrum of scheduling strategies and service time distributions supported by SimQPN will be extended. Support for timed transitions and immediate queueing places is also planned and will be included in a future release.

3.2. Design and architecture

SimQPN has an object-oriented architecture. Every element (e.g. place, transition or token) of the simulated QPN is internally represented as object. Communication between objects is mostly implemented through method calls, with exception of some cases, where object data is accessed directly (bypassing accessor methods) to provide better performance. Although the latter is a deviation from the object-oriented paradigm, we have made this compromise because it speeds up the simulation significantly.

[3] JRE version 1.1 or higher is required.

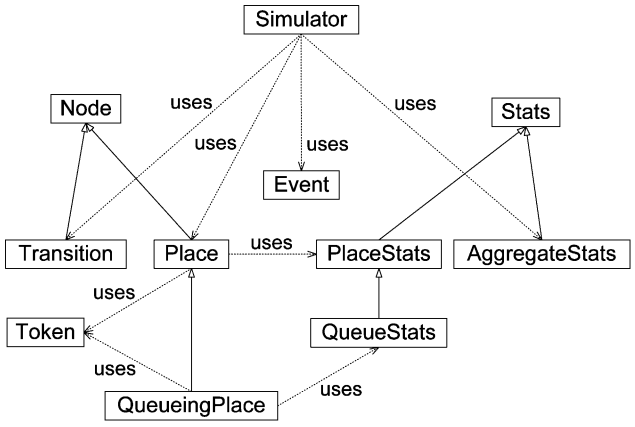

Fig. 3. SimQPN's object model.

Fig. 3 shows the major types of objects (classes) used in SimQPN and the relationships among them. At the top level is the Simulator class, which contains the main simulation routine. The PlaceStats and QueueStats objects are used to manage statistics gathered during the simulation. The AggregateStats object is used to manage statistics gathered from multiple simulation runs.

Fig. 4 outlines the main simulation routine which drives each simulation run. As already mentioned, SimQPN's internal simulation procedure is based on the event-scheduling approach [23,18]. To explain what is understood by event here, we need to look at the way the simulated QPN transitions from one state to another with respect to time. Since only immediate transitions are supported, the only place in the QPN where time is involved is inside the queues of queueing places. Tokens arriving at the queues wait until there is a free server available and are then served. A token's service time distribution determines how long its service continues. After a token has been served it is moved to the depository of the queueing place, which may enable some transitions and trigger their firing. This leads to a change in the marking of the QPN. Once all enabled transitions have fired, the next change of the marking will occur after another service completion at some queue. In this sense, it is the completion of service that initiates each change of the marking. Therefore, we define *event* to be a completion of a token's service at a queue.

For FCFS queues, a token's service completion event is scheduled (added to the event list) as soon as there is a free server available to serve the token. For IS queues, a token's service completion event is scheduled immediately upon arrival of the token at the queue. Finally, for PS queues, in contrast to FCFS and IS queues, service completion events are only scheduled after all enabled transitions have fired. This is because service rates at PS queues depend on the token population, which may change when transitions fire. By deferring the scheduling of service completion events until after all enabled transitions have fired, it is avoided to have to reschedule the events after each change in the token population as transitions fire. Thus, the knowledge of the behavior of the simulated QPN is exploited to save CPU time and improve the efficiency of the simulation. A scheduled service completion event at a PS queue might still need to be rescheduled, however, only in the case where, before the time has come for it to be executed, events in other queues cause new transitions to be enabled and their firing triggers a change in the queue's token population. If this happens, the next service completion event of the PS queue is rescheduled according to its new token population. Elapsed service times can be safely ignored, since PS queues are assumed

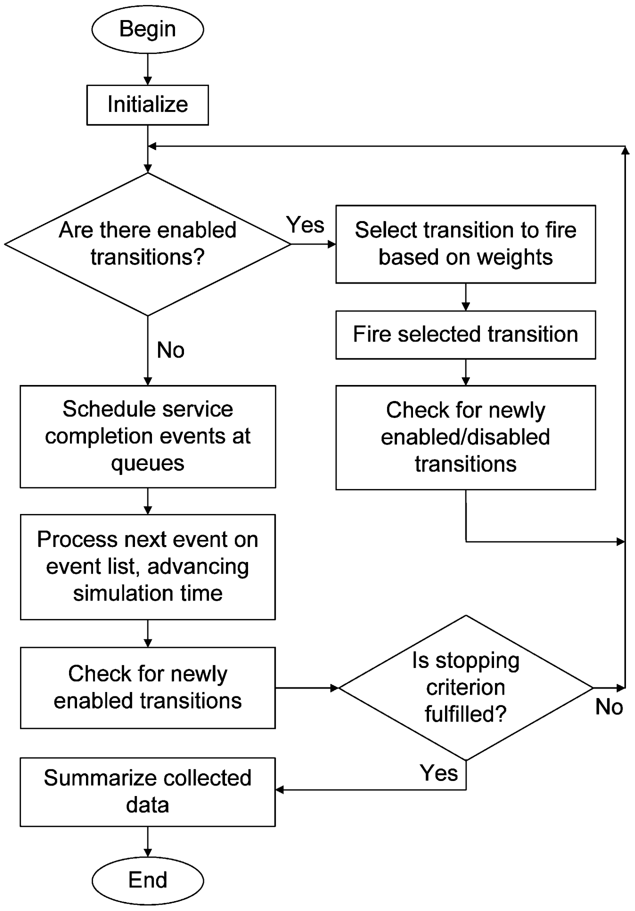
Begin
Initialize
Are there enabled transitions?
Yes
Select transition to fire based on weights
Fire selected transition
Check for newly enabled/disabled transitions
No
Schedule service completion events at queues
Process next event on event list, advancing simulation time
Check for newly enabled transitions
Is stopping criterion fulfilled?
No
Yes
Summarize collected data
End

Fig. 4. SimQPN's main simulation routine.

to have exponentially distributed service times. The next version of SimQPN is planned to also deal with the more complicated case of PS queues with non-exponential service time distributions.

Another way in which SimQPN exploits the knowledge of the structure and behavior of QPNs to improve the efficiency of the simulation is by using an optimized algorithm for keeping track of the enabling status of transitions. Generally, Petri net simulators need to check for enabled transitions after each change in the marking caused by a transition firing. The exact way they do this, is one of the major factors determining the efficiency of the simulation [15]. In [30], it is shown how the *locality principle* of colored Petri nets can be exploited to minimize the overhead of checking for enabled transitions. The locality principle states that an occurring transition will only affect the marking on immediate neighbor places, and hence the enabling status of a limited set of neighbor transitions. SimQPN exploits an adaptation of this principle to QPNs, taking into account that tokens deposited into queueing places do not become available for output transitions immediately upon arrival and hence cannot affect the enabling status of the latter. Since checking the enabling status of a transition is a computationally expensive operation, our goal is to make sure that this is done as seldom as possible, i.e. only when there is a real possibility that the status has changed. This translates into the following two cases when the enabling status of a transition needs to be checked:

(1) After a change in the token population of an ordinary input place of the transition, as a result of firing of the same or another transition. Three subcases are distinguished:
   (a) Some tokens were added. In this case, it is checked for *newly enabled modes* by considering all modes that are currently marked as disabled and that require tokens of the respective colors added.
   (b) Some tokens were removed. In this case, it is checked for *newly disabled modes* by considering all modes that are currently marked as enabled and that require tokens of the respective colors removed.
   (c) Some tokens were added and at the same time others were removed. In this case, both of the checks above are performed.
(2) After a service completion event at a queueing input place of the transition. The service completion event results in adding a token to the depository of the queueing place. Therefore, in this case, it is only checked for *newly enabled modes* by considering all modes that are currently marked as disabled and that require tokens of the respective color added.

SimQPN maintains a global list of currently enabled transitions and for each transition a list of currently enabled modes. The latter are initialized at the beginning of the simulation by checking the enabling status of all transitions. As the simulation progresses, a transition's enabling status is checked only in the above mentioned cases. This reduces CPU costs and speeds up the simulation substantially.

3.3. Random number generation

SimQPN utilizes the *Colt* open source library for high performance scientific and technical computing in Java, developed at CERN [14]. In SimQPN, Colt is primarily used for random number generation and, in particular, its implementation of the Mersenne Twister random number generator is employed [27]. The latter is one of the strongest uniform pseudo-random number generators known, and passes many stringent statistical tests, including the diehard test of G. Marsaglia and the load test of P. Hellekalek and S. Wegenkittl. It has an astronomically large period of $2^{19937} - 1 (=10^{6001})$ and 623-dimensional equidistribution with up to 32 bit accuracy. By default, SimQPN uses Mersenne Twister for all of its random number streams. However, for situations where one is willing to trade off quality for performance, it offers an alternative medium quality uniform pseudo-random number generator that is a bit faster. In addition to Mersenne Twister, SimQPN also employs Colt's random seed generator to ensure that there is no correlation between seeds used to initialize random number generators.

3.4. Output data analysis

3.4.1. Modes of data collection

SimQPN offers the ability to configure what data exactly to collect during the simulation and what statistics to provide at the end of the run. This can be specified for each place (ordinary or queueing) of the QPN. The user can choose one of four modes of data collection. The higher the mode, the more information is collected and the more statistics are provided. Since collecting data costs CPU time, the more data is collected, the slower the simulation would run. Therefore, by configuring data collection modes, the user can make sure that no time is wasted collecting unnecessary data and, in this way, speed up the simulation.

Mode 1 considers only token throughput data, i.e. for each queue, place or depository the token arrival and departure rates are estimated for each color.

Mode 2 adds token population and utilization data, i.e. for each queue, place and depository the following data is provided on a per-color basis:

- Minimum/maximum number of tokens.
- Average number of tokens.
- Mean color utilization, i.e. the fraction of time that there is a token of the respective color inside the queue/place/depository.

For queues, in addition to the above, the overall queue utilization is reported (i.e. the fraction of time that there is a token of any color inside the queue).

Mode 3 adds residence time data, i.e. for each queue, place and depository the following additional data is provided on a per-color basis:

- Minimum/maximum observed token residence time.
- Mean and standard deviation of observed token residence times.
- Estimated steady state mean token residence time.
- Confidence interval (CI) for the steady state mean token residence time at a user-specified significance level.

Mode 4 provides all of the above and additionally dumps observed token residence times to files.

3.4.2. Steady state analysis

SimQPN supports two basic methods for estimation of the steady state mean residence times of tokens inside the queues, places and depositories of the QPN. These are the well-known *method of independent replications* (IR) (in its variant referred to as *replication/deletion approach*) and the classical *method of non-overlapping batch means* (NOBM). We refer the reader to [33,23,2,35] for an introduction to these methods. Both of them can be used to provide point and interval estimates of the steady state mean token residence time. In cases where one wants to apply a more sophisticated technique for steady state analysis (for example ASAP [42,41]), SimQPN can be configured to output observed token residence times to files (mode 4), which can then be used as input to external analysis tools (for example [13]).

Both the replication/deletion approach and the method of non-overlapping batch means have different variants [42,33]. Below we discuss some details on the way they were implemented in SimQPN.

3.4.2.1. Elimination of initialization bias. Since we are interested in estimating steady-state parameters of the simulated queueing process, we need to somehow address the well-known *problem of the initial transient* [31,33]. Both of the above mentioned methods require that the analyzed sequence of observations is stationary, and therefore when using them, the effects of transient system behavior need to be accounted for. A number of different approaches have been proposed in the literature for dealing with this problem, including heuristic, statistical, graphical and hybrid approaches. For a survey of methods refer to [26,37,33]. Most methods attempt to estimate the length of the warm-up period and then discard all data collected during it to eliminate initialization bias. One of the simplest and most popular methods is the graphical method of Welch [44,16], which has met some success [26,1]. The latter is appealing because it is simple, practical and does not make any assumptions about the type of system modeled. For these reasons, we decided as a start to implement the method of Welch in SimQPN. We have followed the rules in [23] for choosing the number of replications, their length and the window size. SimQPN allows the user to configure the first two parameters and then automatically plots the moving averages for different window sizes. Thus, simulation experiments with SimQPN usually comprise two stages: stage 1 during which the length of the initial transient is determined, and stage 2 during which the steady-state behavior of the system is simulated and analyzed. Again, if the user prefers to use another method for elimination of the initialization bias, this can be achieved by dumping collected data to files (mode 4) and feeding it into respective analysis tools.

3.4.2.2. Replication/deletion approach. We briefly discuss the way the replication/deletion approach is implemented in SimQPN. Suppose that we want to estimate the steady state mean residence time ν of tokens of given color at a given place, queue or depository. As discussed in [2], in the replication/deletion approach multiple replications of the simulation are made and the average residence times observed are used to derive steady state estimates. Specifically, suppose that n replications of the simulation are made, each of them generating m residence time observations $Y_{i1}, Y_{i2}, \ldots, Y_{im}$. We delete l observations from the beginning of each set to eliminate the initialization bias. The number of observations deleted is determined through the method of Welch as discussed in Section 3.4.2.1. Let X_i be given by

$$X_i = \frac{\sum_{j=l+1}^{m} Y_{ij}}{m-l} \quad i = 1, 2, \ldots, n \tag{1}$$

and

$$\bar{X}(n) = \frac{\sum_{i=1}^{n} X_i}{n} \qquad S^2(n) = \frac{\sum_{i=1}^{n}[X_i - \bar{X}(n)]^2}{n-1} \tag{2}$$

Then the X_i's are independent and identically distributed (IID) random variables with $E(X_i) \approx \nu$ and $\bar{X}(n)$ is an approximately unbiased point estimator for ν. According to the central limit theorem [43], if m is large, the X_i's are going to be approximately normally distributed and therefore the random variable:

$$t_n = \frac{[\bar{X}(n) - \nu]}{\sqrt{\frac{S^2(n)}{n}}}$$

will have t distribution with $(n-1)$ degrees of freedom (df) [17] and an approximate $100(1-\alpha)$ percent confidence interval for ν is then given by

$$\bar{X}(n) \pm t_{n-1,1-\alpha/2}\sqrt{\frac{S^2(n)}{n}} \tag{3}$$

where $t_{n-1,1-\alpha/2}$ is the upper $(1-\alpha/2)$ critical point for the t distribution with $(n-1)$ df [33,43].

3.4.2.3. Method of non-overlapping batch means. Unlike the replication/deletion approach, the method of non-overlapping batch means seeks to obtain independent observations from a single simulation run rather than from multiple replications. Thus, it has the advantage that it must go through the warm-up period only once and is therefore less sensitive to bias from the initial transient. Suppose that we make a simulation run of length m and then divide the resulting observations $Y_1, Y_2, \ldots, Y_m$ into n batches of length q. Assume that $m = nq$ and let X_i be the sample (or batch) mean of the q observations in the ith batch, i.e.

$$X_i(q) = \frac{\sum_{j=(i-1)q+1}^{(i-1)q+q} Y_j}{q} \quad i = 1, 2, \ldots, n \tag{4}$$

The mean ν is estimated by $\bar{X}(n) = \left(\sum_{i=1}^{n} X_i(q)\right)/n$ and it can be shown (see for example [23]) that an approximate $100(1-\alpha)$ percent confidence interval for ν is given by substituting $X_i(q)$ for X_i in Eqs. (2) and (3) above.

SimQPN offers two different *stopping criteria* for determining how long the simulation should continue. In the first one, the simulation continues until the QPN has been simulated for a user-specified amount of model time (*fixed-sample-size procedure*). In the second one, the length of the simulation is increased sequentially from one checkpoint to the next, until enough data has been collected to provide estimates of residence times with user-specified precision (*sequential procedure*). The precision is defined as an upper bound for the confidence interval half length. It can be specified either as an absolute value (absolute precision) or as a percentage relative to the mean residence time (relative precision). The sequential approach for controlling the length of the simulation is usually regarded as the only efficient way for ensuring representativeness of the samples of collected observations [24,16,34]. Therefore, hereafter we assume that the sequential procedure is used.

The main problem with the method of non-overlapping batch means is to select the batch size q, such that successive batch means are approximately uncorrelated. Different approaches have been proposed in the literature to address this problem (see for example [11,1,33]). In SimQPN, we start with a user-configurable initial batch size (by default 200) and then increase it sequentially until the correlation between successive batch means becomes negligible. Thus, the simulation goes through two stages: the first sequentially testing for an acceptable batch size and the second sequentially testing for adequate precision of the residence time estimates (see Fig. 5). The parameters n and p, specifying how often checkpoints are made, can be configured by the user.

We use the *jackknife estimators* [36,33] of the autocorrelation coefficients to measure the correlation between batch means. A jackknife estimator $\hat{J}(k, q)$ of the autocorrelation coefficient of lag k for the sequence of batch means $X_1(q), X_2(q), \ldots, X_n(q)$ of size q is calculated as follows:

$$\hat{J}(k, q) = 2\hat{r}(k, q) - \frac{\hat{r}'(k, q) + \hat{r}''(k, q)}{2} \tag{5}$$

where $\hat{r}(k, q)$ is the ordinary estimator of the autocorrelation coefficient of lag k, calculated from the formula [33]:

$$\hat{r}(k, q) = \frac{\frac{1}{n-k}\sum_{i=k+1}^{n}[X_i(q) - \bar{X}(n)][X_{i-k}(q) - \bar{X}(n)]}{\frac{1}{n}\sum_{i=1}^{n}[X_i(q) - \bar{X}(n)]^2} \tag{6}$$

and $\hat{r}'(k, q)$ and $\hat{r}''(k, q)$ are calculated like $\hat{r}(k, q)$, except that $\hat{r}(k, q)$ is the estimator over all n batch means, whereas $\hat{r}'(k, q)$ and $\hat{r}''(k, q)$ are estimators over the first and the second half of the analyzed sequence of n batch means, respectively.

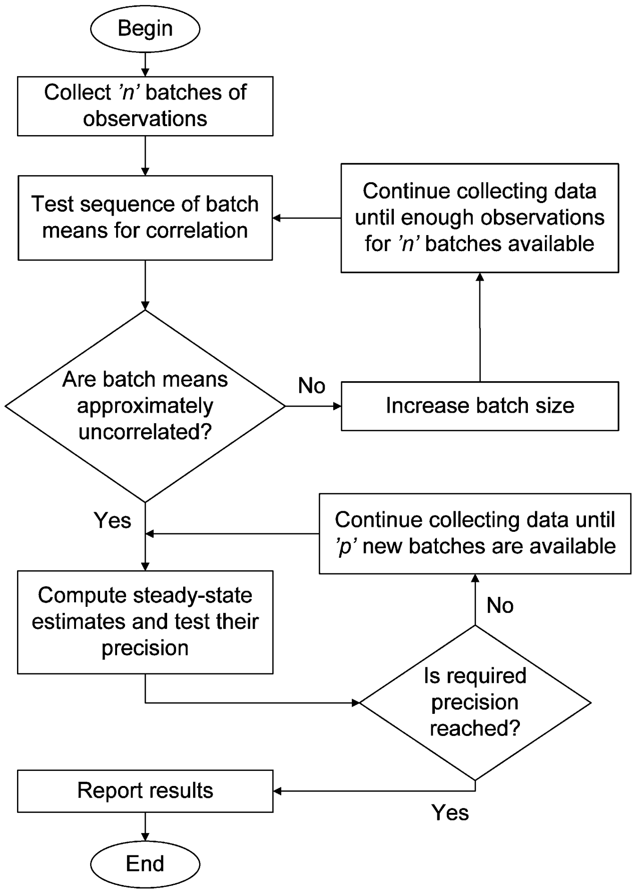
Begin
Collect 'n' batches of observations
Test sequence of batch means for correlation
Continue collecting data until enough observations for 'n' batches available
Are batch means approximately uncorrelated?
No
Increase batch size
Yes
Continue collecting data until 'p' new batches are available
Compute steady-state estimates and test their precision
No
Is required precision reached?
Report results
Yes
End

Fig. 5. SimQPN's batch means procedure.

We use the algorithm proposed in [33] to determine when to consider the sequence of batch means for approximately uncorrelated: a given batch size is accepted to yield approximately uncorrelated batch means if all autocorrelation coefficients of lag k ($k = 1, 2, \ldots, L$; where $L = 0.1n$) are statistically negligible at a given significance level β_k, $0 < \beta_k < 1$. To get an acceptable overall significance level β we assume that

$$\beta < \sum_{k=1}^{L} \beta_k \tag{7}$$

As recommended in [33], in order to get reasonable estimators of the autocorrelation coefficients, we apply the above batch means correlation test only after at least 100 batch means have been recorded (i.e. $n >= 100$). In fact, by default n is set to 200 in SimQPN. Also to ensure approximate normality of the batch means, the initial batch size (i.e. the minimal batch size) is configured to 200.

For FCFS queues, SimQPN also supports *indirect estimation* of the steady state token residence times according to the variance-reduction technique in [12]. The latter suggests, first estimating delay times in the waiting areas of the queues, and then adding them to the mean service times to obtain indirect estimates of the total residence times with reduced variance. SimQPN allows the user to configure for each FCFS queue whether direct or indirect estimates should be used (the default is indirect).

4. SimQPN validation and performance analysis

In this section, we analyze several different QPN models by means of SimQPN, and then validate the results with respect to correctness and accuracy. We follow the guidelines in [3,20,39,24,23] and consider models of different size and complexity, in each case, examining the simulation output under a variety of settings for the input parameters. We compare the simulation results with results obtained using other methods, i.e. analytical techniques, approximation techniques or measurements on the system modeled. In some cases, we consider QPN models that may be mapped to equivalent queueing network models which are analytically tractable. The latter enables us to easily validate simulation results by comparing them against results obtained using analytical techniques applied to the equivalent queueing network models. This allows us to consider large QPN models that are not analyzable using currently available QPN analysis techniques, but whose equivalent queueing network models may be analyzed using conventional techniques.

In addition to validating results for reasonableness, we also study the performance of the point and interval estimators implemented in SimQPN. We conduct an exhaustive experimental analysis of the variation of point estimates and coverage of confidence intervals. Before we begin with the presentation of the results, we briefly discuss the method of coverage analysis we employ.

4.1. Method of coverage analysis used

Let us consider multiple independent replications (runs) of a simulation experiment. *Coverage* of confidence interval is defined as the probability c that the confidence interval (obtained from a replication) covers the true value ν of the respective parameter being estimated. As usual, if n replications of the experiment have been executed, the coverage c can be estimated by the proportion:

$$\hat{c} = \frac{s}{n} \tag{8}$$

where s is the number of replications in which the reported confidence interval contains the true value. The accuracy with which $\hat{c}$ estimates c is usually assessed by the following confidence interval based on the normal distribution:

$$\left(\hat{c} - z_{1-\alpha/2}\sqrt{\frac{\hat{c}(1-\hat{c})}{n}}, \hat{c} + z_{1-\alpha/2}\sqrt{\frac{\hat{c}(1-\hat{c})}{n}}\right) \tag{9}$$

where $z_{1-\alpha/2}$ is the $(1-\alpha/2)$ quantile of the standard normal distribution. This is based on the fact that, while the number of confidence intervals containing the true value ν has a binomial distribution with mean nc, $(\hat{c} - c)\sqrt{\hat{c}(1-\hat{c})/n}$ tends to the standard normal distribution as $n \to \infty$ [17,34].

In [34,28] it is argued that, unless certain rules are adhered to, the above point and interval estimators for coverage cannot be relied upon to produce reliable results. It is advocated to conduct coverage analysis sequentially and several rules are formulated for obtaining reliable and statistically accurate results. In [25] the rules are revised and extended, proposing to use an interval estimator of coverage based on the F distribution. We now briefly summarize these rules indicating how they were implemented in our coverage analysis for SimQPN:

- Rule 1: Coverage should be analyzed sequentially, i.e. analysis of coverage should be stopped when the *absolute precision* of the estimated coverage satisfies a specified level which is sufficiently small. In our case, we stopped when reaching absolute precision of ± 0.05.
- Rule 2: An estimate of coverage has to be calculated from a representative sample of data, so the coverage analysis can start only after a minimum number of "bad" confidence intervals have been recorded. In our case, we required at least 200 "bad" confidence intervals (as suggested in [25]) to be recorded before sequential analysis commences.
- Rule 3: Results from simulation runs that are clearly too short should not be taken into account. To implement this rule, after recording 200 "bad" confidence intervals, we calculated the average run length and then discarded all runs shorter by more than one standard deviation than the average run length. As justified in [34,28], this filters out statistical "noise" and removes significant bias in the results.
- Rule 4: An interval estimator which is based on the F distribution of coverage should be used to ensure that the sequential analysis of coverage produces realistic estimates. A $100(1-\alpha)$ lower limit $\hat{c}_l$ and upper limit $\hat{c}_u$ of the confidence interval for the true coverage are given by

$$\hat{c}_l = \hat{c} - \Delta_1 = \frac{n\hat{c}}{n\hat{c} + (n - n\hat{c} + 1) f_{1-\alpha/2}(r1, r2)},$$
$$\hat{c}_u = \hat{c} + \Delta_2 = \frac{(n\hat{c} + 1) f_{1-\alpha/2}(r3, r4)}{(n - n\hat{c}) + (n\hat{c} + 1) f_{1-\alpha/2}(r3, r4)} \qquad (10)$$

  where $f_{1-\alpha/2}(r1, r2)$ and $f_{1-\alpha/2}(r3, r4)$ are the $(1-\alpha/2)$ quantiles of the F distribution with $(r1, r2)$ and $(r3, r4)$ degrees of freedom, $r1 = 2(n - n\hat{c} + 1)$, $r2 = 2n\hat{c}$, $r3 = 2(n\hat{c} + 1)$ and $r4 = 2(n - n\hat{c})$ [25].

In our analysis of coverage, we consider both the interval estimator in (10) based on the F distribution, as well as the traditional interval estimator (9) based on the normal distribution.

4.2. QPN model of SPECjAppServer2001's customer domain

SPECjAppServer2001 is an industry-standard benchmark for measuring the performance and scalability of J2EE technology-based application servers. The SPECjAppServer2001 workload is based on a large distributed application divided into four domains: *customer domain* dealing with customer orders and interactions, *manufacturing domain* performing "just in time" manufacturing operations, *supplier domain* handling supply-chain management and *corporate domain* managing all corporate information. This is a huge application, claimed to be complex enough to represent a real-world e-business system [40]. In [22], we built a QPN model of SPECjAppServer2001's customer domain, analyzed it using analytical techniques and validated it against measurements on the real system. We now consider the same model again, but this time we analyze it through simulation using SimQPN. We then compare results obtained from the simulation with the analytical solution presented in [22]. The model we consider is depicted in Fig. 6.

In the following we describe the places of the model:

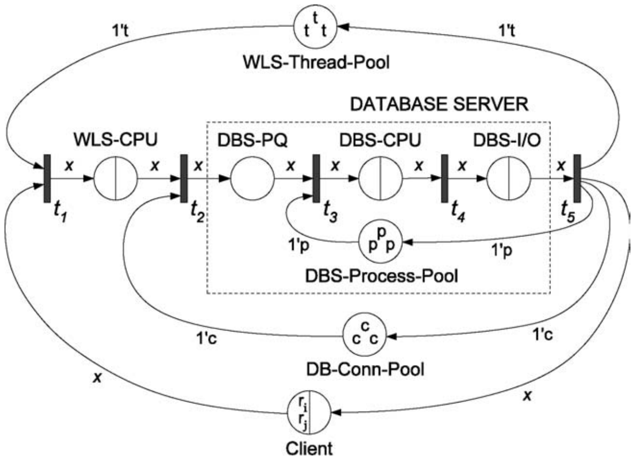

Fig. 6. QPN model of SPECjAppServer2001's customer domain.

- Client: Queueing place with IS scheduling strategy used to represent clients sending requests to the system. Time spent at the queue of this place corresponds to the client think time, i.e. the service time of the queue is equal to the average client think time.
- WLS-CPU: Queueing place with PS scheduling strategy used to represent the CPU of a *WebLogic server* (WLS)[4] hosting the SPECjAppServer2001 J2EE application.
- DBS-CPU: Queueing place with PS scheduling strategy used to represent the CPU of a *database server* (DBS) used for persisting business data.
- DBS-I/O: Queueing place with FCFS scheduling strategy used to represent the disk subsystem of the DBS.
- WLS-Thread-Pool: Ordinary place used to represent the thread pool of the WLS. Each token in this place represents a WLS thread.
- DB-Conn-Pool: Ordinary place used to represent the JDBC connection pool of the WLS. Tokens in this place represent JDBC connections to the DBS.
- DBS-Process-Pool: Ordinary place used to represent the process pool of the DBS. Tokens in this place represent database processes.
- DBS-PQ: Ordinary place used to hold incoming requests at the DBS while they wait for a server process to be allocated to them.

The following types of tokens (token colors) are used in the model:

- Token 'r_i' represents a request sent by a client for execution of a transaction of class i. For each request class a separate token color is used (e.g. 'r_1', 'r_2', 'r_3', ...). Tokens of these colors can be contained only in places Client, WLS-CPU, DBS-PQ, DBS-CPU and DBS-I/O.
- Token 't' represents a WLS thread. Tokens of this color can be contained only in place WLS-Thread-Pool.
- Token 'p' represents a DBS process. Tokens of this color can be contained only in place DBS-Process-Pool.

[4] WebLogic is a trademark of BEA Systems, Inc.

Table 1
NewOrder request mean service times

Request class	WLS-CPU	DBS-CPU	DBS-I/O
NewOrder	70 ms	53 ms	12 ms

- Token ‘c’ represents a JDBC connection to the DBS. Tokens of this color can be contained only in place DB-Conn-Pool.

We now take a look at the lifecycle of a client request in our system model. Every request (modeled by a token of color ‘r_i’ for some i) is initially at the queue of place Client where it waits for a user-specified think time. After the think time elapses, the request moves to the Client depository where it waits for a WLS thread to be allocated to it before its processing can begin. Once a thread is allocated (modeled by taking a token of color ‘t’ from place WLS-Thread-Pool), the request moves to the queue of place WLS-CPU, where it receives service from the CPU of the WLS. It then moves to the depository of the place and waits for a JDBC connection to be allocated to it. The JDBC connection (modeled by token ‘c’) is used to connect to the database and make any updates required by the respective transaction. A request sent to the database server arrives at the place DBS-PQ (DBS Process Queue) where it waits for a server process (modeled by token ‘p’) to be allocated to it. Once this is done, the request receives service first at the CPU and then at the disk subsystem of the database server. This completes the processing of the request, which is then sent back to place Client releasing the held DBS process, JDBC connection and WLS thread.

The following input parameters need to be supplied before the model can be analyzed:

- Number of requests (i.e. clients) of each request class in the initial marking.
- Service times of request classes at the queues of places WLS-CPU, DBS-CPU and DBS-I/O.
- Average client think time (service time at the queue of place Client).
- Number of WLS threads (tokens ‘t’), JDBC connections (tokens ‘c’) and Oracle[5] server processes (tokens ‘p’) in the initial marking.

In [22], we studied three different scenarios (instances of the model) varying the above parameters. For lack of space, in this paper we only consider the first one, which is the one with the highest number of tokens. The conclusions we draw apply equally well to the other two scenarios. We assume that there are 80 NewOrder clients in the system with average think time of 200 ms, and there are 60 WLS threads, 40 JDBC connections and 30 DBS processes available.

The mean service times of NewOrder requests at the various queues are shown in Table 1. They are assumed to be exponentially distributed.

Tables 2 and 3 report the results from simulating the model using SimQPN. The method of batch means was used for steady-state analysis and the simulation was stopped as soon as the half widths of all 90% confidence intervals for residence times dropped below 5% of the respective point estimates (relative precision stopping criterion). The length of the warm-up period (determined through the method of Welch) was 6×10^6 ms (model time) and the total run duration was 9 s (wall clock time) on a machine with a 2 GHz CPU. The results are compared with the exact results from the analytical solution in [22], obtained

[5] Oracle is a trademark of Oracle Corporation.

Table 2
Token population (N) and utilization (U) results for the QPN model of SPECjAppServer2001's customer domain from a single simulation run

Place	N		U	
	Analytical	Simulation	Analytical	Simulation
Client_Q	2.857	2.864	0.942	0.943
Client_D	17.143	17.136	1.000	1.000
WLS-CPU_Q	56.676	56.658	1.000	1.000
DBS-CPU_Q	3.116	3.134	0.757	0.761
DBS-I/O_Q	0.207	0.208	0.171	0.172
WLS-Thread-Pool	0.000	0.000	0.000	0.000
DB-Conn-Pool	36.676	36.658	1.000	1.000
DBS-Process-Pool	26.676	26.658	1.000	1.000

using the *structured SOR* analysis method. For each queue and depository, the estimated steady state token population (N), utilization (U), throughput (X) and residence time (R) are reported. For residence times, in addition, 90% confidence intervals are given. We have used subscripts 'Q' and 'D' to distinguish between queues and depositories of queueing places.

Tables 2 and 3 show the results from a single simulation run. However, as in any simulation, results vary from run to run. To measure this variation and evaluate the confidence interval coverage, we made multiple replications of the above simulation run as described in Section 4.1. We stopped as soon as enough data was available in order to provide, for each confidence interval in the simulation results, a 95% confidence interval for the true coverage with absolute half width less than 0.05. Tables 4 and 5 present the results from our analysis. We skip the results for throughput and utilization, since the analysis showed that their variation was negligible. As discussed in Section 4.1, we include two interval estimates of the true coverage (95% confidence intervals)—the first one based on the normal distribution and the second one based on the F distribution. As expected, the confidence intervals based on the F distribution are slightly wider (i.e. more conservative) than the traditional ones based on the normal distribution. Repeating the above analysis for different variations of the model (with modified input parameters) led to similar results in terms of the precision of the point and interval estimates.

Table 3
Throughput (X) and residence time (R) results for the QPN model of SPECjAppServer2001's customer domain from a single simulation run

Place	X (requests/s)		R (ms)	
	Analytical	Simulation	Analytical	Simulation (90% CI)
Client_Q	14.286	14.309	200.00	200.11 (±00.84)
Client_D	14.286	14.309	1199.97	1197.46 (±05.73)
WLS-CPU_Q	14.286	14.309	3967.25	3958.87 (±19.04)
DBS-CPU_Q	14.286	14.309	218.15	218.97 (±05.25)
DBS-I/O_Q	14.286	14.309	14.48	14.51 (±00.05)

Table 4
Experimental analysis of residence time variation and coverage of 90% confidence intervals for the QPN model of SPEC-jAppServer2001's customer domain from 1430 runs

Place	Variation		Coverage point/interval (95% CI) estimates		
	Mean	S.D.	Pt.Est.	Int.Est. (*N*-dist.)	Int.Est. (*F*-dist.)
Client_Q	199.99	0.44	0.901	0.901 ± 0.015	[0.885, 0.916]
Client_D	1199.98	3.18	0.902	0.902 ± 0.015	[0.886, 0.917]
WLS-CPU_Q	3967.14	11.30	0.896	0.896 ± 0.016	[0.879, 0.912]
DBS-CPU_Q	218.08	3.50	0.889	0.889 ± 0.017	[0.871, 0.906]
DBS-I/O_Q	14.48	0.03	0.898	0.898 ± 0.015	[0.881, 0.913]

Table 5
Experimental analysis of residence time variation and coverage of 95% confidence intervals for the QPN model of SPEC-jAppServer2001's customer domain from 3820 runs

Place	Variation		Coverage point/interval (95% CI) estimates		
	Mean	S.D.	Pt.Est.	Int.Est. (*N*-dist.)	Int.Est. (*F*-dist.)
Client_Q	199.99	0.58	0.948	0.948 ± 0.007	[0.941, 0.955]
Client_D	1200.04	4.10	0.947	0.947 ± 0.007	[0.940, 0.954]
WLS-CPU_Q	3967.53	14.67	0.939	0.939 ± 0.007	[0.931, 0.947]
DBS-CPU_Q	218.05	4.60	0.933	0.933 ± 0.008	[0.925, 0.941]
DBS-I/O_Q	14.48	0.03	0.945	0.945 ± 0.007	[0.937, 0.952]

4.3. Product-form queueing network

The next model we consider is a queueing network model, taken from the examples shipped with the Performance Evaluation and Prediction SYstem for Queueing NetworkS (PEPSY-QNS) tool [10]. The example we consider is called *e_bcmp2* and is shown in Fig. 7. It is a closed product-form queueing network with two request classes. We first translate the queueing network into a QPN and then analyze

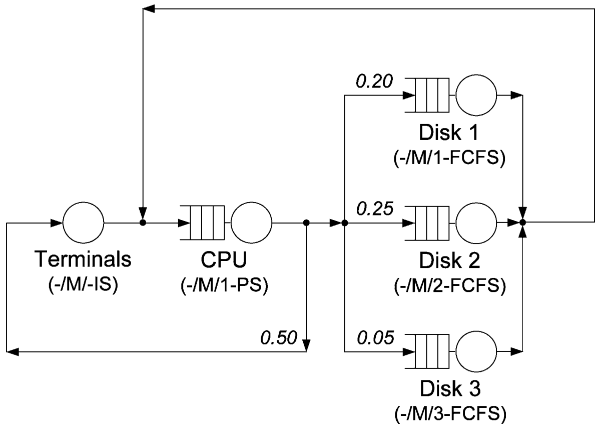

Fig. 7. Product-form queueing network.

Table 6
Mean service times of requests at the queues of the product-form queueing network

Request class	CPU	Disk1	Disk2	Disk3	Terminals
Class 1	200	1000	500	20000	10000
Class 2	250	1000	500	20000	10000

the latter through simulation using SimQPN. We compare results obtained from the simulation with the analytical solution provided by PEPSY. Finally, as in the previous case, we analyze the variation of point estimates and the confidence interval coverage.

The mean service times of requests at the various queues of the model are given in Table 6 (all times are in milliseconds). Service times are exponentially distributed. There are 10 requests of class 1 and 12 of class 2. Mapping the queueing network to an equivalent queueing Petri net is straightforward and the resulting QPN is shown in Fig. 8. Basically, every queue is mapped to a queueing place and request classes are mapped to token colors. Connected queues in the queueing network have their respective queueing places connected through transitions in the QPN.

Tables 7 and 8 show the results from simulating the product-form queueing network (more precisely its equivalent QPN) using SimQPN. Again, the method of batch means was used for steady-state analysis and the simulation was stopped as soon as the half widths of all 90% confidence intervals for residence times dropped below 5% of the respective point estimates (relative precision stopping criterion). The length of the warm-up period (determined through the method of Welch) was 16×10^6 ms (model time) and the total run duration was 65 s (wall clock time) on a machine with a 2 GHz CPU. The results are compared with the exact results from the analytical solution provided by PEPSY. For each queue, the estimated steady state population (N), throughput (X) and residence time (R) are reported. For residence times, in addition, 90% confidence intervals are given.

As in the previous case, to evaluate the variation of point estimates and the confidence interval coverage, we made multiple replications of the above simulation run and applied the coverage analysis method in Section 4.1. The stopping criterion was the same as for the previous model. Tables 9 and 10 present the results from our analysis. Repeating the evaluation for different variations of the model led to similar

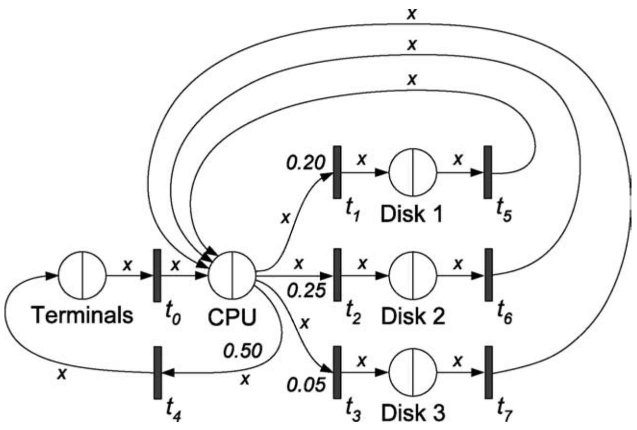

Fig. 8. QPN equivalent to the product-form queueing network.

Table 7
Queue population (N), throughput (X) and residence time (R) results for the product-form queueing network from a single simulation run

Place	***N***		*X* (requests/s)		*R* (ms)	
	Analytical	Simulation	Analytical	Simulation	Analytical	Simulation (90% CI)
Request class 1						
CPU	0.592	0.594	1.241	1.243	476.7	477.8 (±001.9)
Disk1	0.510	0.511	0.248	0.249	2055.0	2056.1 (±012.1)
Disk2	0.159	0.160	0.310	0.311	513.6	513.8 (±000.3)
Disk3	2.535	2.535	0.062	0.062	40857.0	40926.5 (±435.3)
Terminals	6.204	6.200	0.620	0.621	10000.0	9985.4 (±018.2)
Request class 2						
CPU	0.864	0.866	1.468	1.468	588.7	589.6 (±002.4)
Disk1	0.604	0.603	0.294	0.293	2056.0	2054.2 (±012.5)
Disk2	0.188	0.189	0.367	0.367	513.6	513.6 (±000.3)
Disk3	3.005	3.013	0.073	0.073	40941.0	40947.1 (±417.3)
Terminals	7.339	7.330	0.734	0.734	10000.0	9989.2 (±016.5)

results with no degradation in the precision of the point and interval estimates. In Table 11 we present the results for one such variation, in which the service times of requests at the "Terminals" queue (i.e. the client think times) were reduced from 10 000 to 5000 ms, leading to *overloading* Disk3. As expected, most affected by this change were the residence times at Disk3, which increased by over 177%. This is because, at this load, Disk3 is completely saturated (its utilization is over 99%), leading to long waiting times in the queue. Residence times at the other queues were not as much affected by the change, since requests have much lower service demands for them (see Table 6) and in spite of the heavier load, they were still under 70% utilized (the CPU was about 67% utilized, Disk1 about 60% and Disk2 still less than 20%). In all cases, the estimated coverage of 90% and 95% confidence intervals did not drop below 88% and 93%, respectively.

4.4. QPN model of SPECjAppServer2002

SPECjAppServer2002 is a J2EE 1.3 port of the SPECjAppServer2001 benchmark [40]. In [21], we built a queueing network model of SPECjAppServer2002 that spanned the whole benchmark application,

Table 8
Utilization (U) results for the product-form queueing network from a single simulation run

Place	*U*	
	Analytical	Simulation
CPU	0.615	0.616
Disk1	0.542	0.541
Disk2	0.169	0.170
Disk3	0.903	0.904
Terminals	1.000	1.000

Table 9
Experimental analysis of residence time variation and coverage of 90% confidence intervals for the product-form queueing network from 2398 runs

Place	Variation		Coverage point/interval (95% CI) estimates		
	Mean	S.D.	Pt.Est.	Int.Est. (*N*-dist.)	Int.Est. (*F*-dist.)
Request class 1					
CPU	476.7	1.1	0.888	0.888 ± 0.012	[0.875, 0.901]
Disk1	2054.4	6.4	0.902	0.902 ± 0.012	[0.890, 0.914]
Disk2	513.6	0.2	0.903	0.903 ± 0.012	[0.890, 0.914]
Disk3	40854.1	226.9	0.911	0.911 ± 0.012	[0.898, 0.923]
Terminals	9999.6	10.1	0.904	0.904 ± 0.012	[0.891, 0.915]
Request class 2					
CPU	588.8	1.3	0.887	0.887 ± 0.013	[0.873, 0.899]
Disk1	2056.4	6.3	0.909	0.909 ± 0.011	[0.897, 0.920]
Disk2	513.6	0.2	0.894	0.894 ± 0.012	[0.881, 0.906]
Disk3	40937.1	226.1	0.907	0.907 ± 0.012	[0.894, 0.919]
Terminals	10000.3	9.4	0.896	0.896 ± 0.012	[0.883, 0.908]

i.e. all four domains and not just the customer domain as in [22]. This was a non-product-form model and we were only able to analyze it using analytical approximation methods (more specifically, we used the *multisum method* [10]). However, when increasing the number of customers interacting with the system, even approximation methods started to fail. We now consider the same model again, but this time we analyze it through simulation using SimQPN. We compare results obtained from the simulation with the approximate results presented in [21]. We also consider the cases where approximation methods were failing, and in these cases, we compare the simulation results with measurements on the real system modeled.

Table 10
Experimental analysis of residence time variation and coverage of 95% confidence intervals for the product-form queueing network from 4665 runs

Place	Variation		Coverage point/interval (95% CI) estimates		
	Mean	S.D.	Pt.Est.	Int.Est. (*N*-dist.)	Int.Est. (*F*-dist.)
Request class 1					
CPU	476.7	1.2	0.936	0.936 ± 0.007	[0.928, 0.943]
Disk1	2054.6	6.9	0.947	0.947 ± 0.006	[0.941, 0.953]
Disk2	513.6	0.2	0.948	0.948 ± 0.006	[0.941, 0.954]
Disk3	40857.8	240.6	0.950	0.950 ± 0.006	[0.943, 0.957]
Terminals	10000.1	10.8	0.944	0.944 ± 0.006	[0.937, 0.951]
Request class 2					
CPU	588.7	1.4	0.932	0.932 ± 0.007	[0.924, 0.939]
Disk1	2056.5	6.7	0.951	0.951 ± 0.006	[0.945, 0.957]
Disk2	513.6	0.2	0.944	0.944 ± 0.006	[0.937, 0.951]
Disk3	40942.9	238.6	0.946	0.946 ± 0.006	[0.939, 0.953]
Terminals	10000.1	9.6	0.957	0.957 ± 0.005	[0.951, 0.963]

Table 11
Experimental analysis of residence time variation and coverage of 95% confidence intervals for the product-form queueing network under heavy load from 4300 runs

Place	Variation		Coverage point/interval (95% CI) estimates		
	Mean	S.D.	Pt.Est.	Int.Est. (N-dist.)	Int.Est. (F-dist.)
Request class 1					
CPU	591.1	3.2	0.933	0.933 ± 0.007	[0.925, 0.941]
Disk1	2403.1	13.2	0.943	0.943 ± 0.007	[0.936, 0.950]
Disk2	517.5	0.2	0.953	0.953 ± 0.006	[0.947, 0.960]
Disk3	72561.5	401.1	0.952	0.952 ± 0.006	[0.945, 0.958]
Terminals	5000.1	5.3	0.948	0.948 ± 0.006	[0.941, 0.955]
Request class 2					
CPU	726.0	3.9	0.933	0.933 ± 0.007	[0.925, 0.941]
Disk1	2405.5	13.1	0.946	0.946 ± 0.007	[0.938, 0.953]
Disk2	517.6	0.2	0.948	0.948 ± 0.007	[0.940, 0.954]
Disk3	72857.6	398.7	0.950	0.950 ± 0.007	[0.943, 0.956]
Terminals	4999.9	4.8	0.951	0.951 ± 0.006	[0.944, 0.958]

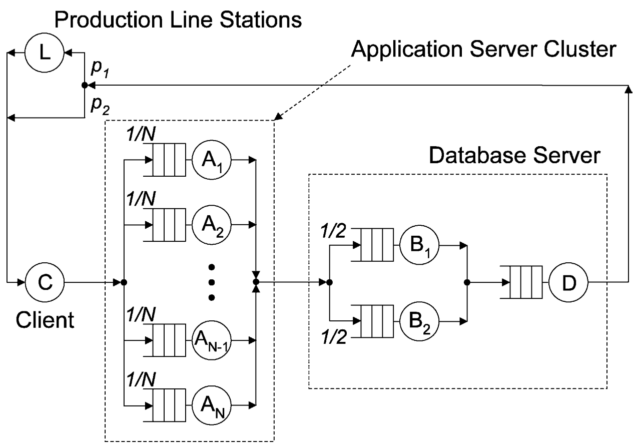

Fig. 9. Queueing network model of SPECjAppServer2002.

The queueing network model from [21] is depicted in Fig. 9. The system it models, i.e. our SPECjAppServer2002 deployment, is depicted in Fig. 10.[6] It is a closed model with five request classes: NewOrder (NO), ChangeOrder (CO), OrderStatus (OS), CustStatus (CS) and WorkOrder (WO). The first four represent order-entry transactions run in the customer domain. The last one represents work orders processed at the production lines in the manufacturing domain.

Following is a brief description of the queues used:

- C: “Infinite Server” (IS) queue used to model the client machine which runs the SPECjAppServer driver and emulates virtual clients sending requests to the system. The service time of order entry requests

[6] SuSE is a trademark of SuSE Linux AG and AMD is a trademark of Advanced Micro Devices, Inc.

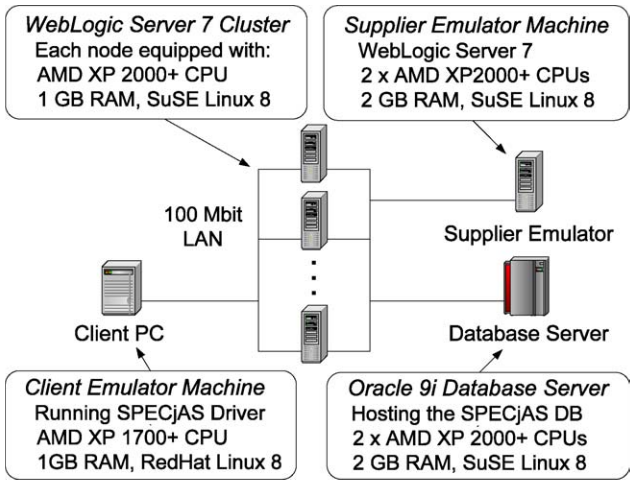

Fig. 10. Our SPECjAppServer2002 deployment.

at this queue is equal to the average client think time, while the service time of WorkOrder requests is equal to the average time a production line waits after processing a work order before starting a new one. Note that time spent on this queue is not part of the system response time.

- $A_1, \ldots, A_N$: "Processor Sharing" (PS) queues used to model the CPUs of the WebLogic servers (WLS) in the cluster.
- B_1, B_2: "Processor Sharing" (PS) queues used to model the two CPUs of the database server (DBS).
- D: "First-Come-First-Served" (FCFS) queue used to model the disk subsystem of the database server.
- L: "Infinite Server" (IS) queue used to model the virtual production line stations in the manufacturing domain. Only WorkOrder requests ever visit this queue (for them $p_1 = 1$ and $p_2 = 0$; for the rest $p_1 = 0$ and $p_2 = 1$). Their service time at the queue corresponds to the average time spent at the production line stations during work order processing.

The mean service times of requests at the various queues in the network are shown in Table 12. Service times are assumed to be exponentially distributed.

The following input parameters need to be supplied before the model can be analyzed:

- Number of WebLogic servers N.
- Number of order entry clients (NewOrder, ChangeOrder, OrderStatus and CustStatus).
- Average think time of order entry clients—*Customer Think Time*.

Table 12
Mean service times of requests at the queues of the model

Request class	A_i for $1 \le i \le N$ (ms)	B_j for $1 \le j \le 2$ (ms)	D (ms)	L (ms)
NewOrder	12.98	10.64	1.12	–
ChangeOrder	13.64	10.36	1.27	–
OrderStatus	2.64	2.48	0.58	–
CustStatus	2.54	2.08	0.3	–
WorkOrder	24.22	34.14	1.68	1000

Table 13
Model input parameters for the three scenarios considered in [21]

Parameter	Low	Moderate	Heavy
NewOrder clients	30	50	100
ChangeOrder clients	10	40	50
OrderStatus clients	50	100	150
CustStatus clients	40	70	50
Planned lines	50	100	200
Customer think time (s)	2	2	3
Mfg think time (s)	3	3	5

- Number of planned production lines generating WorkOrder requests.
- Average time production lines wait after processing a work order before starting a new one—*Manufacturing* (Mfg) *Think Time*.

Each set of values for these parameters generates a different instance of the model. In [21], we considered three scenarios (see Table 13) representing low, moderate and heavy load, respectively. The number of WebLogic servers was ranging from 1 to 9. In this paper, we only consider the moderate- and heavy-load scenarios, since they are the largest and most problematic ones as far as analysis is concerned. Again, we translate the queueing network into an equivalent QPN by mapping queues to queueing places and connecting them through transitions. The resulting QPN is shown in Fig. 11. Note that, compared to the QPN in [22], this is a huge QPN (considering token population and colors) and even for the simplest scenario (low load) trying to analyze it by means of conventional techniques results in explosion of the underlying state space.

Table 14 summarizes the results from 500 simulation runs of the moderate-load scenario with six WebLogic servers. Each run took approximately 5 min on a machine with a 2 GHz CPU. For every request class, the mean and standard deviation of observed throughputs (in requests/s) and residence times at queues A_i, B_j and D (in ms) are reported. The simulation results are compared against the approximate results presented in [21]. The latter were obtained using the *multisum* analytical approximation method supported by the PEPSY tool. As we can see, results from the simulation are consistent with the

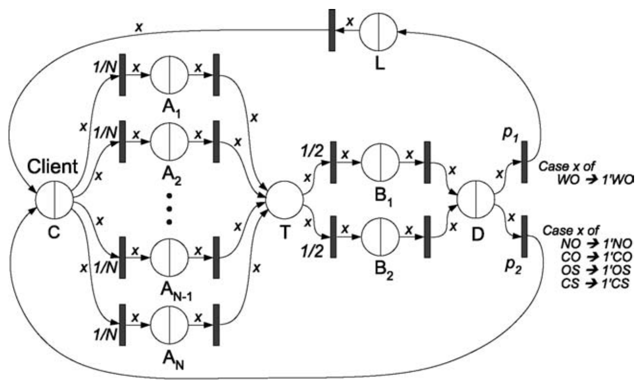

Fig. 11. QPN model of SPECjAppServer2002.

Table 14
Residence time (R), throughput (X) and utilization (U) results for scenario 2 with six WebLogic servers from 500 simulation runs

Metric	A_i (WLS-CPU)			B_j (DBS-CPU)			D (DBS-Disk)		
	Analytical	Simulation	S.D.	Analytical	Simulation	S.D.	Analytical	Simulation	S.D.
R_{NO}	17	16.8	0.22	40	39.2	0.73	1	1.3	≤0.01
R_{CO}	18	17.7	0.24	39	38.1	0.73	1	1.4	≤0.01
R_{OS}	3	3.4	0.03	9	9.2	0.16	1	0.8	≤0.01
R_{CS}	3	3.3	0.03	8	7.7	0.13	0	0.4	≤0.01
R_{WO}	31	31.4	0.42	129	124.9	2.69	2	1.9	≤0.01
X_{NO}	4.05	4.06	≤0.01	12.15	12.15	≤0.01	24.29	24.31	≤0.01
X_{CO}	3.24	3.24	≤0.01	9.72	9.72	≤0.01	19.43	19.45	≤0.01
X_{OS}	8.28	8.28	≤0.01	24.83	24.83	≤0.01	49.67	49.66	≤0.01
X_{CS}	5.80	5.80	≤0.01	17.40	17.40	≤0.01	34.80	34.80	≤0.01
X_{WO}	4.00	4.01	≤0.01	12.01	12.03	≤0.01	24.02	24.05	≤0.01
U	0.23	0.23	≤0.01	0.74	0.74	≤0.01	0.13	0.13	≤0.01

approximate results and have very low variation. Since the exact values of the estimated parameters are not known (exact analytical solution of the model is not available), coverage analysis for confidence intervals does not make sense in this case.

We now repeat the same analysis for the heavy load scenario. The average run duration was 12 min. Results are summarized in Table 15. For each request class, we consider its total response time and throughput. Note that by *response time* we mean the total amount of time needed for processing a request, i.e. the sum of its residence times at queues A_i (WLS-CPU), B_j (DBS-CPU) and D (DBS-Disk). Unfortunately, available approximation methods fail to provide reliable response time estimates for models of this size. Therefore, this time the analytical results only include throughput and utilization.

Table 15
Response time (R), throughput (X) and utilization (U) results for scenario 3 with six WebLogic servers from 500 simulation runs

Metric	6 App. servers		
	Analytical	Simulation	Measured
R_{NO}	–	98	94
R_{CO}	–	97	98
R_{OS}	–	23	27
R_{CS}	–	20	27
R_{WO}	–	286	251
X_{NO}	32.22	32.28	32.66
X_{CO}	16.11	16.15	16.19
X_{OS}	49.60	49.62	49.21
X_{CS}	16.55	16.56	16.24
X_{WO}	31.72	31.82	32.08
$U_{\text{WLS-CPU}}$ (%)	26.5	26.4	29
$U_{\text{DBS-CPU}}$ (%)	86.1	87.7	91

To validate response time results, we compare them against measurements taken on the real system that the model represents. Note that the expected deviation here is higher, since the model is only an approximation of the system, and as discussed in [21], it has some inherent limitations. Nevertheless, we see that results obtained from the simulation are close to the actual values measured on the system modeled. Repeating the analysis for the other configurations considered in [21] led to results of similar accuracy.

5. Summary and conclusions

This paper showed how the problem of analyzing large QPN models can be approached by exploiting discrete event simulation for model analysis. We presented SimQPN—our simulation tool for QPNs, and discussed its features, design and architecture. In parallel to this, we presented our methodology for simulating QPN models, based on which SimQPN was developed. The methods for output data analysis used in SimQPN were presented and the specifics of their implementation were discussed. It was shown how SimQPN exploits the knowledge of the structure and behavior of QPNs to improve the efficiency of the simulation. After that, we validated our approach by applying it to study several different QPN models. In each case, we validated the simulation results by comparing them with results obtained using other methods, i.e. analytical methods, approximation methods or measurements on the system modeled. Models of different size and complexity, ranging from simple models to large and complex models of realistic e-business systems, were considered. Each model was analyzed for different variations of its input parameters. The variability of output data provided by SimQPN, as well as the coverage of confidence intervals reported, were subjected to a rigorous experimental analysis. Results showed that data reported by SimQPN is pretty accurate and stable. Even for residence time, the metric with highest variation, the standard deviation of point estimates did not exceed 2.5% of the mean value. In all cases the estimated coverage of confidence intervals was less than 2% below the nominal value (higher than 88% for 90% confidence intervals and higher than 93% for 95% confidence intervals). In addition to this, SimQPN proved to be pretty fast in terms of measured CPU running times. A simulation run for the simple models in Sections 4.2 and 4.3 on a 2 GHz CPU took less than one minute on average. Even for the much larger models in Section 4.4, CPU running times did not exceed 12 min.

SimQPN provides a portable simulation engine for queueing Petri nets. Being specialized for QPNs, it is extremely light-weight and fast. It can be used to analyze QPN models of realistic size and complexity, making it possible to exploit the modeling power and expressiveness of the QPN paradigm to its full potential as a performance prediction tool. The latter can be taken advantage of in the capacity planning process for large distributed systems.

6. Ongoing and future work

We are currently developing a Java-based Graphical User Interface Environment (GUI) for modeling with queueing Petri nets. This will provide a graphical frontend to SimQPN and make model development and analysis more user-friendly. Furthermore, we are planning to add the following features:

- Support for timed transitions and immediate queueing places.
- Support for load-dependent service demands.

- Support for deterministic distributions.
- Support for more scheduling strategies.
- Support for hierarchical queueing Petri nets (HQPNs).
- Support for parallel simulation.
- Support for additional methods for determining the length of the initial transient and for output data analysis.

Acknowledgments

We gratefully acknowledge the many fruitful discussions with Dr. Falko Bause from the University of Dortmund. We also would like to thank the anonymous reviewers for their valuable comments and suggestions which helped to improve the quality of the paper.

References

[1] C. Alexopoulos, D. Goldsman, To batch or not to batch, ACM Trans Model. Comput. Simul. 14 (1) (2004) 76–114.
[2] C. Alexopoulos, A. Seila, Output data analysis for simulations, in: Proceedings of the 2001 Winter Simulation Conference, 2001.
[3] O. Balci, Validation, verification and testing techniques throughout the life cycle of a simulation study, Ann. Oper. Res. 53 (1994) 121–174.
[4] F. Bause, "QN + PN = QPN"—combining queueing networks and petri nets, Technical Report no. 461, Dept. of CS, University of Dortmund, Germany, 1993.
[5] F. Bause, Queueing Petri nets—a formalism for the combined qualitative and quantitative analysis of systems, in: Proceedings of the Fifth International Workshop on Petri Nets and Performance Models, 1993.
[6] F. Bause, P. Buchholz, P. Kemper, Hierarchically combined queueing Petri nets, in: Proceedings of 11th International Conference on Analysis and Optimization of Systems, 1994.
[7] F. Bause, P. Buchholz, P. Kemper, QPN-tool for the specification and analysis of hierarchically combined queueing Petri nets, Quantitative Evaluation of Computing and Communication Systems, LNCS No. 977, 1995.
[8] F. Bause, P. Buchholz, P. Kemper, Integrating software and hardware performance models using hierarchical queueing Petri nets, in: Proceedings of the ninth ITG/GI on Fachtagung Messung, Modellierung und Bewertung von Rechen-und Kommunikationssystemen (MMB'97), Freiberg, Germany, 1997.
[9] F. Bause, F. Kritzinger, Stochastic Petri Nets—An Introduction to the Theory, Vieweg Verlag, 2002.
[10] G. Bolch, M. Kirschnick, PEPSY-QNS, Technical Report TR-I4-94-18, University of Erlangen-Nuremberg, Germany, June 1994. http://www4.informatik.uni-erlangen.de/Projects/PEPSY/en/pepsy.html.
[11] C. Chien, Batch size selection for the batch means method, in: Proceedings of the 1994 Winter Simulation Conference, 1994.
[12] J.S. Carson, A.M. Law, Conservation equations and variance reduction in queueing simulations, Oper. Res. 28 (1980).
[13] North Carolina State University Department of Industrial Engineering, ASAP3 Software For Steady-State Simulation Output Analysis, 2003. ftp://ftp.ncsu.edu/pub/eos/pub/jwilson/installasap3.exe.
[14] CERN-European Organisation for Nuclear Research, The Colt Distribution, http://hoschek.home.cern.ch/hoschek/colt/.
[15] R. Gaeta, Efficient discrete-event simulation of colored Petri nets, IEEE Trans. Software Eng. 22 (9) (1996).
[16] P. Heidelberger, P.D. Welch, Simulation run length control in the presence of an initial transient, Oper. Res. 31 (1983) 1109–1145.
[17] R.V. Hogg, A.F. Craig, Introduction to Mathematical Statistics, 5th ed., Prentice-Hall, Upper Saddle River, NJ, 1995.

[18] G. Iazeolla, A. Lehmann, H.J. Van Den Herik (Eds.), Simulation Methodologies, Languages, Architectures, AI and Graphics for Simulation, The Society for Computer Simulation International, La Jolla, USA, 1989.
[19] K. Jensen, Coloured Petri Nets and the Invariant Method, Mathematical Foundations on Computer Science, Lecture Notes in Computer Science, vol. 118, 1981, pp. 327–338.
[20] J.P.C. Kleijnen, Theory and methodology: verification and validation of simulation models, Eur. J. Oper. Res. 82 (1) (1995) 145–162.
[21] S. Kounev, A. Buchmann, Performance modeling and evaluation of large-scale J2EE applications, in: Proceedings of 29th International Conference on Resource Management and Performance Evaluation of Enterprise Computing Systems, CMG2003, 2003.
[22] S. Kounev, A. Buchmann, Performance modeling of distributed e-business applications using queueing Petri nets, in: Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS2003, 2003.
[23] A. Law, W. David Kelton, Simulation Modeling and Analysis, 3rd ed., Mc Graw-Hill Companies, Inc., 2000.
[24] A. Law, W.D. Kelton, Confidence intervals for steady-state simulations. II. A survey of sequential procedures, Manage. Sci. 28 (5) (1982) 550–562.
[25] J.-S.R. Lee, D. McNickle, K. Pawlikowski, Confidence interval estimators for coverage analysis in sequential steady-state simulation, in: Proceedings of the 22nd Australian Computer Science Conference, Auckland, New Zealand, 1999.
[26] J.R. Linton, C.M. Harmonosky, A comparison of selective initialization bias elimination methods, in: Proceedings of the 2002 Winter Simulation Conference, 2002.
[27] M. Matsumoto, T. Nishimura, Mersenne twister: a 623-dimensionally equidistributed uniform pseudo-random number generator, ACM Trans. Model. Comput. Simul. (1998).
[28] D. McNickle, K. Pawlikowski, G. Ewing, Experimental evaluation of confidence interval procedures in sequential steady-state simulation, in: Proceedings of the 1996 Winter Simulation Conference, 1996.
[29] D. Menasce, Two-level iterative queueing model of software contention, in: Proceedings of 10th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS 2002), 2002.
[30] K.H. Mortensen, Efficient data-structures and algorithms for a coloured Petri nets simulator, in: Proceedings of the Third Workshop and Tutorial on Practical Use of Coloured Petri Nets and the CPN Tools, 2001.
[31] D. Ockerman, D. Goldsman, The impact of transients on simulation variance estimators, in: Proceedings of the 1997 Winter Simulation Conference, 1997.
[32] University of Aarhus, Petri Net Tool Database, Department of Computer Science (DIAMI). http://www.daimi.au.dk/PetriNets/tools/.
[33] K. Pawlikowski, Steady-state simulation of queueing processes: a survey of problems and solutions, ACM Comput. Surv. 22 (2) (1990) 123–170.
[34] K. Pawlikowski, D. McNickle, G. Ewing, Coverage of confidence intervals in sequential steady-state simulation, J. Simul. Pract. Theory 6 (3) (1998) 255–267 (Plus: "Erratum" 7(1):1, 1999).
[35] H. Perros, Computer simulation techniques—the definitive introduction, E-Book, NC State University, 2003. http://www.csc.ncsu.edu/faculty/perros/simulation.pdf.
[36] R.G. Miller, The Jackknife: a review, Biometrika 61 (1974) 1–15.
[37] S. Robinson, A Statistical process control approach for estimating the warm-up period, in: Proceedings of the 2002 Winter Simulation Conference, 2002.
[38] J. Rolia, K. Sevcik, The method of layers, IEEE Trans. Software Eng. 21 (8) (1995) 689–700.
[39] R.G. Sargent, Verification and validation of simulation models, in: A. Seila, S. Manivannan, J. Tew, D. Sadowski (Eds.), Proceedings of the Winter Simulation Conference, 1994.
[40] Standard Performance Evaluation Corporation (SPEC), SPECjAppServer2001 and SPECjAppServer2002 Benchmark Documentation, 2002. http://www.spec.org/osg/jAppServer/.
[41] N. Steiger, E. Lada, J. Wilson, J. Joines, C. Alexopoulos, D. Goldsman, ASAP3: a batch means procedure for steady-state simulation output analysis, ACM Trans. Model. Comput. Simul. 82 (2003) ftp://ftp.ncsu.edu/pub/eos/pub/jwilson/tomacsv37.pdf.
[42] N. Steiger, J. Wilson, Experimental performance evaluation of batch means procedures for simulation output analysis, in: Proceedings of the 2000 Winter Simulation Conference, 2000.

[43] K.S. Trivedi, Probability Statistics with Reliability, Queuing and Computer Science Applications, 2nd ed., John Wiley & Sons, 2002.
[44] P.D. Welch, The statistical analysis of simulation results, in: S. Lavenberg (Ed.), The Computer Performance Modeling Handbook, Academic Press, NJ, 1983.
[45] M. Woodside, J. Neilson, D. Petriu, S. Majumdar, The stochastic rendezvous network model for performance of synchronous client–server-like distributed software, IEEE Trans. Comput. 44 (1) (1995) 20–34.