

# QPME - A Performance Modeling Tool Based on Queueing Petri Nets

Samuel Kounev  
Karlsruhe Institute of Technology  
Department of Computer Science  
76131 Karlsruhe, Germany  
skounev@acm.org

Christofer Dutz  
Technische Universität Darmstadt  
Department of Computer Science  
64289 Darmstadt, Germany  
dutz@c-ware.de

## ABSTRACT

Queueing Petri nets are a powerful formalism that can be exploited for modeling distributed systems and analyzing their performance and scalability. By combining the modeling power and expressiveness of queueing networks and stochastic Petri nets, queueing Petri nets provide a number of advantages. In this paper, we present QPME (Queueing Petri net Modeling Environment) - a tool that supports the modeling and analysis of systems using queueing Petri nets. QPME provides an Eclipse-based editor for designing queueing Petri net models and a powerful simulation engine for analyzing the models. After presenting the tool, we discuss the ongoing work on the QPME project and the planned future enhancements of the tool.

## 1. INTRODUCTION

QPME (Queueing Petri net Modeling Environment) is a performance modeling tool based on the Queueing Petri Net (QPN) modeling formalism. Introduced in 1993 by Falko Bause [1], the QPN formalism has a number of advantages over conventional modeling formalisms such as queueing networks and stochastic Petri nets. By combining the modeling power and expressiveness of queueing networks and stochastic Petri nets, QPNs enable the integration of hardware and software aspects of system behavior into the same model. In addition to hardware contention and scheduling strategies, QPNs make it easy to model simultaneous resource possession, synchronization, asynchronous processing and software contention. These aspects have significant impact on the performance of modern enterprise systems. Another advantage of QPNs is that they can be used to combine qualitative and quantitative system analysis. A number of efficient techniques from Petri net theory can be exploited to verify some important qualitative properties of QPNs. The latter not only help to gain insight into the behavior of the system, but are also essential preconditions for a successful quantitative analysis [3]. Last but not least, QPN models have an intuitive graphical representation that facilitates model development. In [11], we showed how QPNs can be used for modeling distributed e-business applications. Building on this work, we have developed a methodology for performance modeling of distributed component-based systems using QPNs [9]. The methodology has been applied to model a number of systems ranging from simple systems to systems of realistic size and complexity. It can be used as a powerful tool for performance and scalability analysis. Some examples of modeling studies based on QPNs can be

found in [13, 14, 17]. These studies consider different types of systems including distributed component-based systems, event-based systems and Grid computing environments.

While the QPN modeling paradigm provides many important benefits, there are currently few tools that support the modeling and analysis of systems using QPNs. Based on [21], apart from the QPME tool presented in this paper, the only tool that is available is the HiQPN-Tool [2] developed at the University of Dortmund. HiQPN can be used to build and analyze QPN models, however, it only supports analytical solution techniques. As we demonstrated in [11], due to the state space explosion problem, QPN models of realistic systems are too large to be analyzable using analytical techniques. Another problem with HiQPN is that it is only available on Sun-OS 5.5.x / Solaris 2, which significantly limits its accessibility.

Recognizing the need for a tool to support the modeling and analysis of realistically-sized systems using QPNs, we have developed QPME - a QPN modeling environment with a user-friendly graphical user interface. In this paper, we present QPME, discussing its features and benefits. QPME is made of two major components, a QPN Editor (QPE) and a Simulator for QPNs (SimQPN). In the following sections, we present an overview of these components. Further details on their internal architecture and implementation can be found in [8, 12]. QPME is available free-of-charge for non-profit use (see [10]) and has been distributed to more than 70 universities and research organizations worldwide. The current license is closed-source, however, there are plans to make the tool open-source in the near future.

The rest of this paper is organized as follows: We start with a brief introduction to QPNs in Section 2. Sections 3 and 4 provide an overview of the QPN editor and the simulation engine, respectively. Section 5 summarizes the ongoing and future work on QPME. Finally, some concluding remarks are presented in Section 6.

## 2. QUEUEING PETRI NETS

The main idea behind the QPN modeling paradigm was to add queueing and timing aspects to the places of Colored Generalized Stochastic Petri Nets (CGSPNs) [1]. 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. 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. A formal definition of a QPN follows [1]:

DEFINITION 1. A QPN is an 8-tuple  $QPN = (P, T, C, I^-, I^+, M_0, Q, W)$  where:

1.  $P = \{p_1, p_2, \dots, p_n\}$  is a finite and non-empty set of places,
2.  $T = \{t_1, t_2, \dots, t_m\}$  is a finite and non-empty set of transitions,  $P \cap T = \emptyset$ ,
3.  $C$  is a color function that assigns a finite and non-empty set of colors to each place and a finite and non-empty set of modes to each transition.
4.  $I^-$  and  $I^+$  are the backward and forward incidence functions defined on  $P \times T$ , such that  $I^-(p, t), I^+(p, t) \in [C(t) \rightarrow C(p)_{MS}]$ ,  $\forall (p, t) \in P \times T^1$
5.  $M_0$  is a function defined on  $P$  describing the initial marking such that  $M_0(p) \in C(p)_{MS}$ .
6.  $Q = (\tilde{Q}_1, \tilde{Q}_2, (q_1, \dots, q_{|P|}))$  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<sup>2</sup> 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.
7.  $W = (\tilde{W}_1, \tilde{W}_2, (w_1, \dots, w_{|T|}))$  where
  - $\tilde{W}_1 \subseteq T$  is the set of timed transitions,
  - $\tilde{W}_2 \subseteq T$  is the set of immediate transitions,  $\tilde{W}_1 \cap \tilde{W}_2 = \emptyset$ ,  $\tilde{W}_1 \cup \tilde{W}_2 = T$  and
  - $w_i \in [C(t_i) \mapsto \mathbb{R}^+]$  such that  $\forall c \in C(t_i)$ :  $w_i(c) \in \mathbb{R}^+$  is interpreted as a rate of a negative exponential distribution specifying the firing delay due to color  $c$ , if  $t_i \in \tilde{W}_1$  or a firing weight specifying the relative firing frequency due to color  $c$ , if  $t_i \in \tilde{W}_2$ .

<sup>1</sup>The subscript MS denotes multisets.  $C(p)_{MS}$  denotes the set of all finite multisets of  $C(p)$ .

<sup>2</sup>In the most general definition of QPNs, queues are defined in a very generic way allowing the specification of arbitrarily complex scheduling strategies taking into account the state of both the queue and the depository of the queueing place [1]. In QPMÉ, we use conventional queues as defined in queueing network theory.

For a more detailed introduction to the QPN modeling formalism, the reader is referred to [1, 3]. To illustrate the above definition, we present an example QPN model of a simple Java EE system. The model was taken from [11] and is shown in Figure 1.

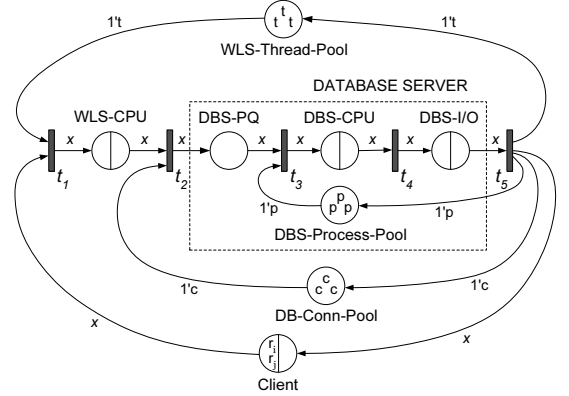


Figure 1: QPN Model of a Java EE System [11]

The system modeled is an e-business application running in a Java EE environment consisting of a WebLogic Server (Java EE application server) hosting the application components and a backend database server used for persisting business data. In the following, we describe the places of the model:

- 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 the *WebLogic Server (WLS)*.
- DBS-CPU** Queueing place with PS scheduling strategy used to represent the CPU of the *database server (DBS)*.
- 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 database connection pool of the WLS. Tokens in this place represent database 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<sub>i</sub>'** 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<sub>1</sub>', 'r<sub>2</sub>', 'r<sub>3</sub>',...). 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**.

**Token 'c'** represents a database connection to the DBS. Tokens of this color can be contained only in place **DB-Conn-Pool**.

We now take a look at the life-cycle of a client request in our system model. Every request (modeled by a token of color 'r<sub>i</sub>' 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 database connection to be allocated to it. The database 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 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, database connection and WLS thread.

### 3. QPE - QUEUEING PETRI NET EDITOR

QPE (Queueing Petri net Editor), the first major component of QPME, provides a graphical tool for building QPN models [8]. It offers a user-friendly interface enabling the user to quickly and easily construct QPN models. QPE is based on the Eclipse Rich Content Platform (RCP) and the Graphical Editing Framework (GEF) [20]. The latter is an open source framework dedicated to providing a rich, consistent graphical editing environment for applications on the Eclipse platform. As a GEF application, QPE is written in pure Java and runs as a standalone RCP application on all operating systems officially supported by the Eclipse platform. This includes Windows, Linux, Solaris, HP-UX, IBM AIX and Apple Mac OS among others, making QPE widely accessible. The only thing required is a Java Runtime Environment (JRE) 5.0. It is recommended to run QPE on Windows since this is the platform it has been tested on.

Internally, being a GEF application, QPE is based on the model-view-controller (MVC) architecture. The model in our case is the QPN being defined, the views provide graphical representations of the QPN, and finally the controller connects the model with the views, managing the interactions among them. QPN models created with QPE can be stored on disk as XML documents. QPE uses its own XML schema based on the Petri Net Markup Language

(PNML) [5] with some changes and extensions to support the additional constructs available in QPN models. Figure 2 shows the QPE main window which is comprised of four views: Main Editor View, Outline View, Properties View and Console View. The Main Editor View contains a Net Editor, Palette and Color Editor. The Net Editor displays the graphical representation of the currently edited QPN, the Palette displays the set of QPN elements that are used to build QPN models and the Color Editor, shown in Figure 3, is used to define the token colors available for use in the places of the QPN. The Properties View enables the user to edit the properties of the currently selected element in the Net Editor. Finally, the Console View is used to display output from QPE extensions and plug-ins.

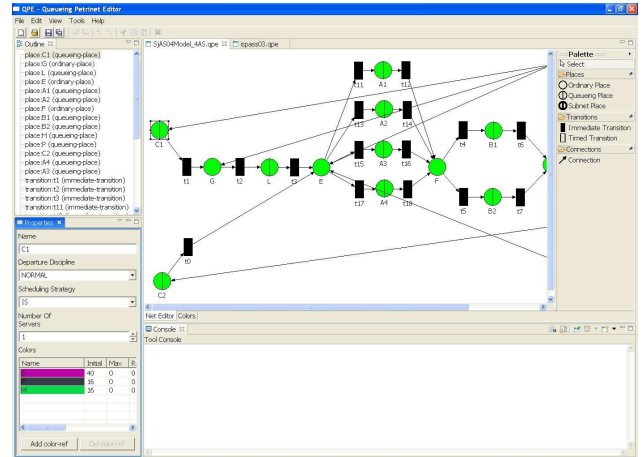


Figure 2: QPE Main Window

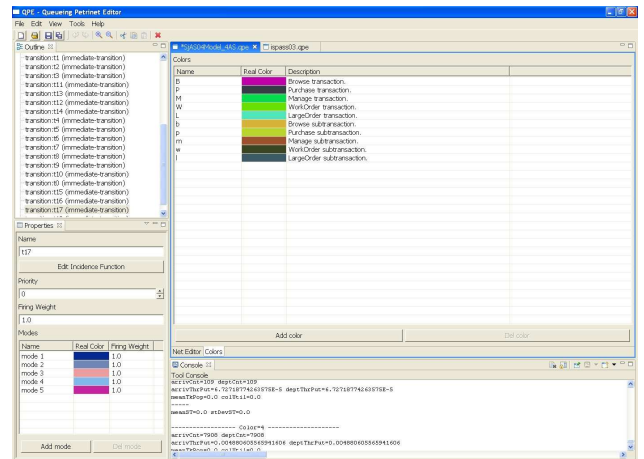


Figure 3: QPE Color Editor

A characterizing feature of QPE is that it allows token colors to be defined globally for the whole QPN instead of on a per place basis. This feature was motivated by the fact that in QPNs typically the same token color (type) is used in multiple places. Instead of having to define the color multiple times, the user can define it one time and then reference it in all places where it is used. This saves time, makes the model definition more compact, and last but not

least, it makes the modeling process less error-prone since references to the same token color are specified explicitly.

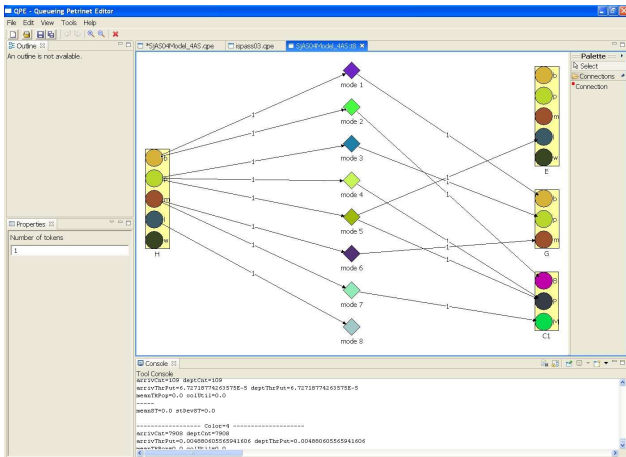


Figure 4: QPE Incidence Function Editor

Figure 4 shows the Incidence Function Editor. The incidence function specifies the behavior of the transition for each of its firing modes in terms of tokens destroyed and/or created in the places of the QPN. Once opened the Incidence Function Editor displays the transition input places on the left, the transition modes in the middle and the transition output places on the right. Each place (input or output) is displayed as a rectangle containing a separate circle for each token color allowed in the place. The user can create connections from token colors of input places to modes or from modes to token colors of output places. If a connection is created between a token color of a place and a mode, this means that when the transition fires in this mode, tokens of the respective color are removed from the place. Similarly, if a connection is created between a mode and a token color of an output place, this means that when the transition fires in this mode, tokens of the respective color are deposited in the place. Each connection can be assigned a weight interpreted as the number of tokens removed/deposited in the place when the transition fires in the respective mode.

## 4. SIMQPN - SIMULATOR FOR QUEUEING PETRI NETS

The second major component of QPME is SimQPN - a discrete-event simulation engine specialized for QPNs. It is very light-weight and has been implemented 100% in Java to provide maximum portability and platform-independence. SimQPN can be run either as Eclipse plugin in QPE or as a standalone Java application. Thus, even though QPE is limited to Eclipse-supported platforms, SimQPN can be run on any platform on which Java SE 5.0 is available. This makes it possible to design a model on one platform (e.g., Windows) using QPE and then analyze it on another platform (e.g., Solaris) using SimQPN. SimQPN configuration parameters are stored as metadata inside the XML file containing the QPN model.

SimQPN simulates QPNs using a sequential algorithm based on the event-scheduling approach for simulation modeling. Being specialized for QPNs, it simulates 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 provides much better performance than a general purpose simulator would provide, both in terms of the speed of simulation and the quality of output data provided.

SimQPN currently supports most, but not all of the QPN features that are supported in QPE. The reason for not limiting QPE to only those features supported by SimQPN is that QPE is meant as a general purpose QPN editor and as such the QPN features it offers should not be limited to any particular analysis method. SimQPN currently supports three different scheduling strategies for queues inside queueing places: Processor-Sharing (PS), Infinite Server (IS) and First-Come-First-Served (FCFS). A wide range of service time distributions are supported including Beta, BreitWigner, ChiSquare, Gamma, Hyperbolic, Exponential, ExponentialPower, Logarithmic, Normal, StudentT, Uniform, VonMises and Empirical. 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) read from an external text file. 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). The next version of SimQPN will also include support for deterministic distributions.

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. 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.

### 4.1 Data Collection Modes

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. The modes are defined as follows:

**Mode 1** This mode which is used by default 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** This mode 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** This mode 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 (c.i.) for the steady state mean token residence time at a user-specified significance level.

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

## 4.2 Steady State Analysis

SimQPN supports two 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* (in its variant referred to as *replication/deletion approach*) and the classical *method of non-overlapping batch means (NOMB)*. We refer the reader to [15, 18] 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. Details on the way these methods were implemented in SimQPN can be found in [12]. For users that would like to use different methods for steady state analysis (for example ASAP [19]), 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. SimQPN utilizes the *Colt* open source library for high performance scientific and technical computing in Java, developed at CERN [7]. In SimQPN, Colt is primarily used for random number generation and, in particular, its implementation of the Mersenne Twister random number generator is employed [16].

We have validated the analysis algorithms implemented in SimQPN by subjecting them to a rigorous experimental analysis and evaluating the quality of point and interval estimates [12]. In particular, the variability of point estimates provided by SimQPN and the coverage of confidence intervals reported were quantified. A number of different models of realistic size and complexity were considered. Our analysis showed that data reported by SimQPN is very 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). For FCFS queues, SimQPN also supports *indirect estimation* of the steady state token residence times according to the variance-reduction technique in [6].

SimQPN includes an implementation of the method of Welch for determining the length of the initial transient (warm-up period). We have followed the rules in [15] 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. 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.

## 4.3 Departure Disciplines

A novel feature of SimQPN is the introduction of the so-called *departure disciplines*. This is an extension of the QPN modeling formalism introduced to address a common drawback of QPN models (and of Petri nets in general), i.e., tokens inside ordinary places and depositories are not distinguished in terms of their order of arrival. Departure disciplines are defined for ordinary places or depositories and determine the order in which arriving tokens become available for output transitions. We define two departure disciplines, Normal (used by default) and First-In-First-Out (FIFO). The former implies that tokens become available for output transitions immediately upon arrival just like in conventional QPN models. The latter implies that tokens become available for output transitions in the order of their arrival, i.e., a token can leave the place/depository only after all tokens that have arrived before it have left, hence the term FIFO. For an example of how this feature can be exploited and the benefits it provides we refer the reader to [9]. An alternative approach to introduce token ordering in an ordinary place is to replace the place with an immediate queueing place containing a FCFS queue. The generalized queue definition from [1] can be exploited to define the scheduling strategy of the queue in such a way that tokens are served immediately according to FCFS, but only if the depository is empty [3]. If there is a token in the depository, all tokens are blocked in their current position until the depository becomes free. However, the generalized queue definition from [1], while theoretically powerful, is impractical to implement, so, in practice, it is rarely used and queues in QPNs are usually treated as conventional queues from queueing network theory.

## 5. ONGOING AND FUTURE WORK

QPME's development is going to be continued as an open-source project which will be started soon. Enhancements along three different dimensions are envisioned: i) user friendliness, ii) model expressiveness and iii) model analysis methods. In the following, we outline the major enhancements that have been planned.

### 5.1 Improve User Friendliness

- Improve the presentation of analysis results providing graphical views of data.
- Introduce modeling templates (e.g., for modeling common types of resources and workloads) to facilitate model reuse.
- Introduce a wizard-like interface.
- Enhance the help system.

In addition to the above, we are currently working on a model-to-model transformation from the Palladio Component Model (PCM) [4] to QPN models. This will make it

possible to specify models at a higher level of abstraction and will facilitate model reuse.

## 5.2 Improve Model Expressiveness

Support for the following features will be added:

- Tokens containing data properties.
- Timed transitions.
- Deterministic distributions.
- Further scheduling strategies for queues, e.g., priority scheduling.
- Transition priorities and inhibitor arcs.
- Load-dependent resource demands.

## 5.3 Improve Model Analysis Methods

- Support for simulating hierarchical models.
- Support for parallel/distributed simulation to take advantage of multi-code processors.
- Support for deriving distributions of performance metrics in addition to the mean and standard deviation.
- Support for analytical model solution techniques.
- Further methods for simulation output data analysis.
- Further methods for determining the length of the simulation warm-up period.

## 6. SUMMARY

In this paper, we presented QPME, our tool for modeling and analysis using queueing Petri nets. QPME provides a user-friendly graphical interface enabling the user to quickly and easily construct QPN models. It offers a highly optimized simulation engine that can be used to analyze models of realistically-sized systems. In addition, being implemented in Java, QPME runs on all major platforms and is widely accessible. QPME provides a robust and powerful tool for performance analysis making it possible to exploit the modeling power and expressiveness of queueing Petri nets to their full potential. The tool is available free-of-charge for non-profit use. Further information can be found at the QPME homepage [10].

## 7. REFERENCES

- [1] F. Bause. Queueing Petri Nets - A formalism for the combined qualitative and quantitative analysis of systems. In *Proc. of 5th Intl. Workshop on Petri Nets and Perf. Models, Toulouse, France, Oct. 19-22, 1993*.
- [2] F. Bause, P. Buchholz, and P. Kemper. QPN-Tool for the Specification and Analysis of Hierarchically Combined Queueing Petri Nets. In *Quantitative Evaluation of Computing and Communication Systems*, volume 977 of *LNCIS*. Springer, 1995.
- [3] F. Bause and F. Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory*. Vieweg Verlag, 2002.
- [4] S. Becker, H. Koziolok, and R. Reussner. The Palladio Component Model for Model-Driven Performance Prediction: Extended version. *Journal of Systems and Software*, 2008. In Press, Accepted Manuscript.
- [5] J. Billington, S. Christensen, K. van Hee, E. Kindler, O. Kummer, L. Petrucci, R. Post, C. Stehno, and M. Weber. The Petri Net Markup Language: Concepts, Technology, and Tools. In *Proc. of 24th Intl. Conf. on Application and Theory of Petri Nets, June 23-27, Eindhoven, Holland, 2003*.
- [6] J. Carson and A. Law. Conservation Equations and Variance Reduction in Queueing Simulations. *Operations Research*, 28, 1980.
- [7] CERN - European Organisation for Nuclear Research. The Colt Distribution - Open Source Libraries for High Performance Scientific and Technical Computing in Java, 2004. <http://dsd.lbl.gov/~hoschek/colt/>.
- [8] C. Dutz. QPE - A Graphical Editor for Modeling using Queueing Petri Nets. Master thesis, Technische Universität Darmstadt, Apr. 2006.
- [9] S. Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets. *IEEE Transactions on Software Engineering*, 32(7):486–502, July 2006.
- [10] S. Kounev. QPME Homepage, 2008. [http://sdq.ipd.uka.de/people/samuel\\_kounev/projects/QPME](http://sdq.ipd.uka.de/people/samuel_kounev/projects/QPME).
- [11] S. Kounev and A. Buchmann. Performance Modelling of Distributed E-Business Applications using Queueing Petri Nets. In *Proc. of the 2003 IEEE Intl. Symposium on Performance Analysis of Systems and Software, Austin, USA, March 20-22, 2003*.
- [12] S. Kounev and A. Buchmann. SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation. *Performance Evaluation*, 63(4-5):364–394, May 2006.
- [13] S. Kounev, R. Nou, and J. Torres. Autonomic QoS-Aware Resource Management in Grid Computing using Online Performance Models. In *Proc. of 2nd Intl. Conf. on Perf. Evaluation Methodologies and Tools - VALUETOOLS, Oct. 23-25, Nantes, France, 2007*.
- [14] S. Kounev, K. Sachs, J. Bacon, and A. Buchmann. A Methodology for Performance Modeling of Distributed Event-Based Systems. In *Proc. of 11th IEEE Intl. Symp. on Object/Comp./Service-oriented Real-time Distr. Computing (ISORC), Orlando, USA, May 2008*.
- [15] A. Law and D. W. Kelton. *Simulation Modeling and Analysis*. Mc Graw Hill, Inc., 3rd edition, 2000.
- [16] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Trans. on Modeling and Comp. Simulation*, 8(1):3–30, 1998.
- [17] R. Nou, S. Kounev, F. Julia, and J. Torres. Autonomic QoS control in enterprise Grid environments using online simulation. *Journal of Systems and Software*, to appear, 2008.
- [18] K. Pawlikowski. Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions. *ACM Computing Surveys*, 22(2):123–170, 1990.
- [19] N. Steiger, E. Lada, J. Wilson, J. Joines, C. Alexopoulos, and D. Goldsman. ASAP3: a batch means procedure for steady-state simulation analysis. *ACM Transactions on Modeling and Computer Simulation*, 15(1):39–73, 2005.
- [20] The Eclipse Foundation. Graphical Editing Framework (GEF). <http://www.eclipse.org/gef/>, 2006.
- [21] University of Hamburg. Petri Net Tool Database. <http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools>, 2008.