Stochastic Modeling and Analysis Using QPME: Queueing Petri Net Modeling Environment v2.0

Simon Spinner, Samuel Kounev, and Philipp Meier

Karlsruhe Institute of Technology (KIT), 76131 Karlsruhe, Germany
{simon.spinner,kounev}@kit.edu, mail@philippmeier.com

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 our tool QPME (Queueing Petri net Modeling Environment) for modeling and analysis using queueing Petri nets. QPME provides an Eclipse-based editor for building queueing Petri net models and a powerful simulation engine for analyzing these models. The development of the tool started in 2003 and since then the tool has been distributed to more than 120 organizations worldwide.

Keywords: Queueing Petri nets, stochastic modeling and analysis, simulation.

1 Introduction

Introduced in 1993 by Falko Bause [1], the Queueing Petri Net (QPN) formalism combines the modeling power and expressiveness of queueing networks and stochastic Petri nets. QPNs are commonly used for the performance evaluation of computer systems because they provide a number of benefits compared to traditional queueing networks and stochastic Petri nets. QPNs enable the integration of hardware and software aspects of system behavior in 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. 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 [4]. The main idea behind the QPN formalism was to add queueing and timing aspects to the places of Colored Generalized Stochastic Petri Nets (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. For a detailed description of the QPN formalism see [1].

The major goal of QPME (Queueing Petri net Modeling Environment) is to support the modeling and analysis of QPN models. The presented tool provides user-friendly graphical editors enabling the user to quickly and easily construct QPN models. It offers a highly optimized simulation engine that can be used to analyze QPN models efficiently. The simulation engine enables the analysis of QPN models too large to be analyzable with analytical techniques due to the state space explosion problem [8]. QPME also offers advanced features for processing and visualizing the results of simulating a QPN model. In addition, being implemented in Java, QPME runs on all major platforms and is widely accessible. The QPN formalism can be used for stochastic modeling in many domains. One major area of application is the performance analysis of computer systems. The tool has been successfully used in several performance modeling studies, e.g. in [10, 11, 14, 16].

The development of QPME started in 2003 at the Technische Universität Darmstadt and has been continuously extended since then. Currently, the tool is developed and maintained by the Descartes Research Group[1] at Karlsruhe Institute of Technology (KIT). Since May 2011, QPME is available in version 2.0 under an open-source license (Eclipse Public License 1.0). QPME has been distributed to more than 120 universities and research organizations worldwide so far.

The rest of this paper is organized as follows: Section 2 provides an overview of the functionality provided by QPME. Section 3 gives some technical insights into its implementation. Section 4 describes typical use cases of the tool and Sect. 5 provides a comparison with other tools for QPNs. Finally, the paper is wrapped up with some concluding remarks in Sect. 7.

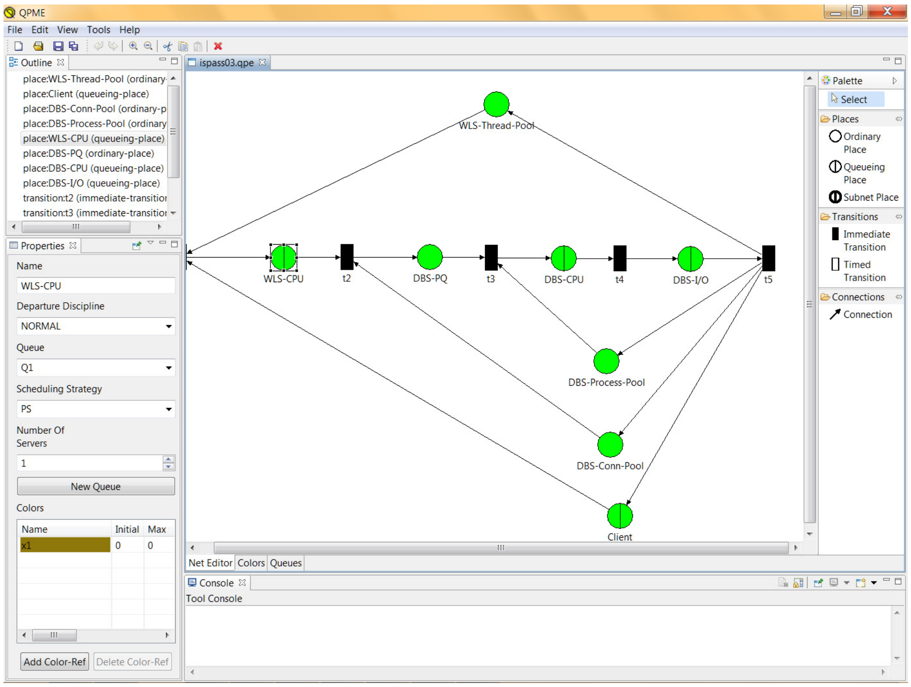
2 Functionality

2.1 Queueing Petri net Editor (QPE)

QPE is a graphical editor for QPNs. The user can create QPN models with a simple drag-and-drop approach. Figure 1 shows the QPE main window which is comprised of four views. The *Main Editor View* displays the graphical representation of the currently edited QPN. The palette contains the set of QPN elements that can be inserted in a QPN model by drag-and-drop, such as places, transitions, and connections. Furthermore, it provides editors for the central definition of colors and queues used in a QPN model. In the *Properties View* the user can edit the properties of the element currently selected in the QPN model. For queueing places, for instance, the user can specify a scheduling strategy and service time distributions for each color in this view. The *Outline View* shows a list of all elements in the QPN model. The *Console View* displays the output when simulating a QPN model.

In a QPN, a transition defines a set of firing modes. An 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. Figure 2 shows the

[1] `http://www.descartes-research.net`



place:WLS-Thread-Pool (ordinary-
place:Client (queueing-place)
place:DBS-Conn-Pool (ordinary-p
place:DBS-Process-Pool (ordinary
place:WLS-CPU (queueing-place)
place:DBS-PQ (ordinary-place)
place:DBS-CPU (queueing-place)
place:DBS-I/O (queueing-place)
transition:t2 (immediate-transitio
transition:t3 (immediate-transitio

Name
WLS-CPU
Departure Discipline
NORMAL
Queue
Q1
Scheduling Strategy
PS
Number Of Servers
1

Colors

Name	Initial	Max
x1	0	0

Fig. 1. QPE Main Window

Incidence Function Editor, which is used to edit the incidence function of a transition. Once opened this editor displays the transition input places on the left, the transition firing 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.

In addition to the basic features described above, QPE has several characterizing features that improve the model expressiveness of QPNs and simplify the creation of complex QPN models. Special mention must be made of the following features:

– *Central color management.* The user can define token colors globally for the whole QPN instead of on a per place basis. 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

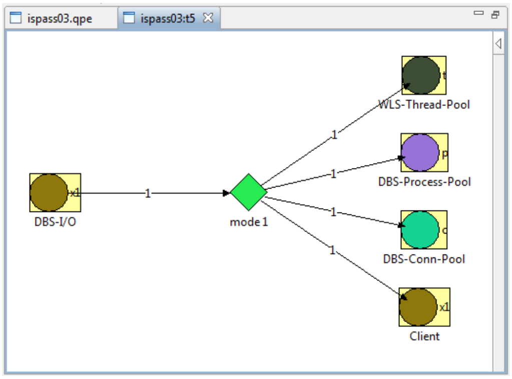


Fig. 2. QPE Incidence Function Editor

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.

– *Shared queues.* The user can specify that multiple queueing places share the same underlying physical queue[2]. In QPE, queues are defined centrally (similar to token colors) and once defined they can be referenced from inside multiple queueing places. This allows to use queueing places to represent software entities, e.g., software components, which can then be mapped to different hardware resources modeled as queues [16]. Shared queues are not supported in standard QPN models [16].
– *Hierarchical QPNs.* Subnet places can contain complete child QPN models. Hierarchical QPNs enable to model layered systems and improve the understandability of huge QPNs. QPE fully supports hierarchical QPNs.
– *Departure Disciplines.* Departure disciplines determine the order in which tokens arriving at an ordinary place or a depository of a queueing place become available for output transitions. QPE supports two disciplines: Normal (used by default) and FIFO. The former implies that tokens become available for output transitions immediately after arriving at an ordinary place or depository whereas in the latter case a token can only leave the place or depository if all tokens that have arrived before it have left. For an example of how this extension of the QPN formalism can be exploited and the benefits it provides we refer the reader to [7].

2.2 Simulation of QPN Model (SimQPN)

SimQPN is a discrete-event simulation engine specialized for QPNs. It simulates QPN models directly and has been designed to exploit the knowledge of the

[2] While the same effect can be achieved by using multiple subnet places mapped to a nested QPN containing a single queueing place, this would require expanding tokens that enter the nested QPN with a *tag* to keep track of their origin as explained in [3].

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.

Model Support. SimQPN implements most, but not all of the QPN elements that can be modeled in QPE. It currently supports three different scheduling strategies for queues: 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 and VonMises as well as deterministic and empirical distributions. All of the characterizing features of QPE described in Sect. 2.1 are fully supported by the SimQPN simulator. A current limitation of SimQPN is the missing support for timed transitions[3] and immediate queueing places. 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 for future releases.

Output Data. 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 on a per *location* basis where location is defined to have one of the following five types: 1. ordinary places, 2. queue of queueing places (considered from the perspective of the place), 3. depository of queueing places, 4. queues (considered from the perspective of all places it is part of), and 5. probes.

A probe enables the user to specify a region of interest for which data should be collected during simulation. The region of a probe includes one or more places and is defined by one start and one end place. The goal is to evaluate the time tokens spend in the region when moving between its begin and end place. The probe starts its measurements for each token entering its region at the start place and updates the statistics when the token leaves at the end place. Probes are realized by attaching timestamps to individual tokens. With probes it is possible to determine statistics for the residence time of tokens in a region of interest.

For each location the user can choose between six 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 progress. Therefore, with data collection modes the user can speed up the simulation by avoiding the collection of data that is not required. The six data collection modes are defined as follows:

– *Mode 0.* No data is collected.
– *Mode 1.* Only token throughput data is collected.

[3] 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.

– *Mode 2.* Additionally, data to compute token population, token occupancy, and queue utilization is collected
– *Mode 3.* Token residence time data is collected (maximum, minimum, mean, standard deviation, steady state mean, and confidence interval of steady state mean).
– *Mode 4.* This mode adds a histogram of observed token residence times.
– *Mode 5.* Additionally token residence times are dumped to a file for further analysis with external tools.

Steady State Analysis. SimQPN supports two methods for the estimation of steady state mean residence times of tokens inside the various locations 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 [12, 15] 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.

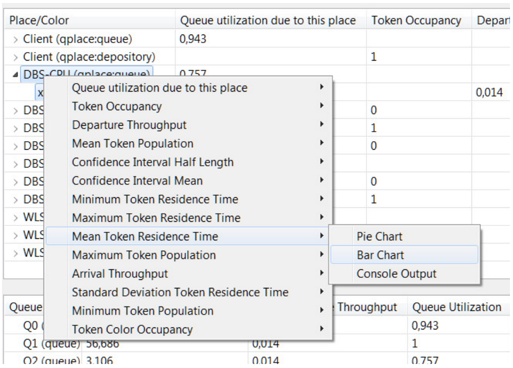
We have validated the analysis algorithms implemented in SimQPN by subjecting them to a rigorous experimental analysis and evaluating the quality of point and interval estimates [9]. Our analysis showed that data reported by SimQPN is very accurate and stable. Even for residence time, the metric with 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).

Furthermore, 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 [12] for choosing the number of replications, their length and the window size.

2.3 Processing and Visualization of Simulation Results

After a successful simulation run, SimQPN saves the results from the simulation in an XML file with a `.simqpn` extension. QPE provides an advanced query engine for the processing and visualization of simulation results. The query engine allows to define queries on the simulation results in order to filter, aggregate and visualize performance data for multiple places, queues and colors of the QPN. The results from the queries can be displayed in textual or graphical form. QPE provides the following two query editors:

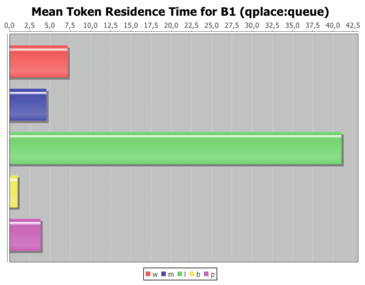
– *Simple Query Editor.* The user can quickly filter and visualize metrics of a *single* location or token color with a few clicks. Currently, three visualization options are available: "Pie Chart", "Bar Chart" and "Console Output".
– *Advanced Query Editor.* The user can create complex queries including the aggregation of metrics over multiple locations and token colors with a powerful user interface. The following two aggregation operators are currently supported: "Average" and "Sum".



Place/Color	Queue utilization due to this place	Token Occupancy	Depart
Client (qplace:queue)	0,943		
Client (qplace:depository)		1	
DBS-CPU (qplace:queue)	0,757		
x			0,014
DBS		0	
DBS		1	
DBS		0	
DBS			
DBS		0	
DBS		1	
WLS			
WLS			
WLS			

Queue			Throughput	Queue Utilization
Q0 (				0,943
Q1 (queue)	56,686	0,014		1
Q2 (queue)	3,106	0,014		0,757

(a) Metrics context menu



(b) Bar Chart

Fig. 3. Simple query editor

Figure 3(a) shows an area of the user interface of the simple query editor. The statistics for all QPN places are presented in the table in the background. By opening the context menu on one of the places a context menu with a set of metrics gets available. When choosing the bar chart for the mean token residence time, the diagram shown in Fig. 3(b) is displayed.

3 Architecture

QPME is based on the Eclipse OSGi framework and comes as a stand-alone *Rich Client Plattform (RCP)* application. QPME is written completely in Java making it widely accessible on different platforms. The architecture of QPME is plugin-based and consists of the following three core plugins: `qpe.base` contains the set of editors for building QPN models (QPE) and for analyzing simulation results (simple and advanced query editor), `qpe.simqpn.kernel` provides the core of the SimQPN simulator, and `qpe.simqpn.ui` contains the graphical wizard for calling the SimQPN simulator from within QPE. The core plugins are highly integrated with each other while at the same time it is possible to also use them separately.

The editors of QPE are based on the Graphical Editing Framework (GEF)[4]. 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. The individual editors (net, incidence function, subnet, queues, and colors) are all realized as different views which work on one single model. Thus it is ensured that all editors working on the same QPN model are always up-to-date.

[4] `http://www.eclipse.org/gef/`

QPME knows two types of files. Files with the extension `.qpe` contain a complete QPN model. They are XML files with an own schema based on the Petri Net Markup Language (PNML) [6] with some changes and extensions to support the additional constructs available in QPN models. Files with the extension `.simqpn` contain the results of a simulation run. They are also XML files and can be opened with the simple or advanced query editor.

SimQPN is realized as a discrete-event simulator. It is very light-weight and optimized to exploit the knowledge of the structure and behavior of QPNs to improve the efficiency of the simulation. A simulation run consists of five phases. First the QPN model is loaded from a `.qpe` file or it is directly imported from an open editor in QPE. If the QPN contains subnet places, a model transformation is applied in phase two which integrates the subnets into the main net recursively resulting in a flat net. This simplifies the simulation of hierarchical QPNs. In phase three, the configuration of the simulation is processed and the simulation engine is configured accordingly. Especially, the simulation engine is set up to collect only the data that is necessary to calculate the requested statistics. By avoiding the collection of data that is not requested by the user, the simulation performance can be significantly sped up in a lot of cases. In phase four, the actual simulation is performed. An event loop advances the simulation clock and in each iteration all events that are scheduled at the current simulation time are processed. SimQPN utilizes the *Colt* open-source library, which is developed at CERN[5], for random number generation. The loop is running until the configured relative or absolute precision is reached or the configured maximum run length is reached. After the simulation, the requested statistics are calculated from the data collected during the simulation and the data is stored in a `.simqpn` file.

4 Use Cases

In [7], we have developed a methodology for performance modeling of distributed component-based systems using QPNs. The methodology has been applied to model a number of systems ranging from simple systems to systems of realistic size and complexity. Here, QPME can be used as a powerful tool for performance and scalability analyses. Some examples of modeling studies using QPME can be found in [10, 11, 14, 16].

Furthermore, QPME can be used at the design time of software systems for solving architecture-level performance models, e.g., the Palladio Component Model (PCM) [5]. In [13], we described how to derive QPN models from PCM models automatically using a formal mapping. In numerous representative case studies, we showed that SimQPN predicted all mean value metrics with high accuracy while the analysis overhead compared to the standard simulator of PCM could be significantly reduced, in many cases by an order of magnitude [13].

[5] `http://acs.lbl.gov/software/colt/`

5 Comparison with Other Tools

While the QPN modeling paradigm provides many important benefits, there are currently few tools that support the modeling and analysis of systems using QPNs. According to [17], 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 demonstrated in [8], QPN models of realistic systems are too large to be analyzable using analytical techniques due to the state space explosion problem. Furthermore, it is only available on Sun-OS 5.5.x / Solaris 2, which significantly limits its accessibility. In contrast, QPME is implemented in Java and runs on all major platforms. It provides a highly optimized simulation engine capable of analyzing models of realistically sized systems.

6 Installation

The binaries of QPME for Windows, Linux and MacOS X and the source code can be obtained from `http://qpme.sourceforge.net` free-of-charge. The QPME binary drops are installed by extracting the zipped archive in an arbitrary location on the hard disk. QPME is open-source and is distributed under the Eclipse Public License (EPL) 1.0.

7 Conclusion

In this paper, we presented QPME 2.0, 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 under an open-source license.

Acknowledgements. This work was funded by the German Research Foundation (DFG) under grant No. KO 3445/6-1. We acknowledge the support of Frederik Zipp from KIT.

References

1. Bause, F.: 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, October 19-22 (1993)

2. Bause, F., Buchholz, P., Kemper, P.: QPN-Tool for the Specification and Analysis of Hierarchically Combined Queueing Petri Nets. In: Beilner, H., Bause, F. (eds.) MMB/TOOLS 1995. LNCS, vol. 977. Springer, Heidelberg (1995)
3. Bause, F., Buchholz, P., Kemper, P.: Integrating Software and Hardware Performance Models Using Hierarchical Queueing Petri Nets. In: Proc. of the 9. ITG / GI - Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, Freiberg, Germany (1997)
4. Bause, F., Kritzinger, F.: Stochastic Petri Nets - An Introduction to the Theory. Vieweg Verlag (2002)
5. Becker, S., Koziolek, H., Reussner, R.: The Palladio Component Model for Model-Driven Performance Prediction: Extended version. Jour. of Sys. and Softw. (2008)
6. Billington, J., Christensen, S., van Hee, K., Kindler, E., Kummer, O., Petrucci, L., Post, R., Stehno, C., Weber, M.: The Petri Net Markup Language: Concepts, Technology, and Tools. In: Proc. of 24th Intl. Conf. on Application and Theory of Petri Nets, Eindhoven, Holland, June 23-27 (2003)
7. Kounev, S.: Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets. IEEE Trans. on Softw. Eng. 32(7), 486–502 (2006)
8. Kounev, S., Buchmann, A.: Performance Modelling of Distributed E-Business Applications using Queuing Petri Nets. In: Proc. of the 2003 IEEE Intl. Symp. on Performance Analysis of Systems and Software, Austin, USA, March 20-22 (2003)
9. Kounev, S., Buchmann, A.: SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation. Performance Evaluation 63(4-5), 364–394 (2006)
10. Kounev, S., Nou, R., Torres, J.: 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, Nantes, France, October 23-25 (2007)
11. Kounev, S., Sachs, K., Bacon, J., Buchmann, A.: 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)
12. Law, A., Kelton, D.W.: Simulation Modeling and Analysis, 3rd edn. Mc Graw Hill Companies, Inc. (2000)
13. Meier, P., Kounev, S., Koziolek, H.: Automated Transformation of Palladio Component Models to Queueing Petri Nets. In: 19th IEEE/ACM Intl. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2011) (July 2011)
14. Nou, R., Kounev, S., Julia, F., Torres, J.: Autonomic QoS control in enterprise Grid environments using online simulation. Jour. of Sys. and Softw. 82(3), 486–502 (2009)
15. Pawlikowski, K.: Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions. ACM Computing Surveys 22(2), 123–170 (1990)
16. Sachs, K.: Performance Modeling and Benchmarking of Event-based Systems. PhD thesis, TU Darmstadt (2010)
17. University of Hamburg. Petri Net Tool Database (2008), `http://www.informatik.uni-hamburg.de/TGI/PetriNets/tools`