Introduction to Queueing Petri Nets: Modeling Formalism, Tool Support and Case Studies

Samuel Kounev
Karlsruhe Institute of Technology
Am Fasanengarten 5
Karlsruhe, Germany
kounev@kit.edu

Simon Spinner
FZI Research Center for Information Technology
Haid-und-Neu-Str. 10-14
Karlsruhe, Germany
spinner@fzi.de

Philipp Meier
Karlsruhe Institute of Technology
Am Fasanengarten 5
Karlsruhe, Germany
mail@philippmeier.com

ABSTRACT

Queueing Petri nets are a powerful formalism that can be exploited for modeling distributed systems and evaluating 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. This tutorial presents an introduction to queueing Petri nets first introducing the modeling formalism itself and then summarizing the results of several modeling case studies which demonstrate how queueing Petri nets can be used for performance modeling and analysis. As part of the tutorial, we present QPME (Queueing Petri net Modeling Environment), an open-source tool for stochastic modeling and analysis of systems using queueing Petri nets. Finally, we briefly present a model-to-model transformation automatically generating a queueing Petri net model from a higher-level software architecture model annotated with performance relevant information.

Categories and Subject Descriptors

C.4 [**Performance Of Systems**]: Modeling Techniques; I.6.5 [**Simulation and Modeling**]: Model Development—*meta-modeling, modeling methodologies*; D.4.8 [**Operating Systems**]: Performance—*Modeling and prediction*

General Terms

Performance, design

Keywords

Performance, stochastic models, system simulation, modeling tools

1. INTRODUCTION

Introduced in 1993 by Falko Bause [1], queueing Petri nets (QPNs) have 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 software 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 [4]. Last but not least, QPN models have an intuitive graphical representation that facilitates model development. In [9], 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 [7]. 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 [7,8,11,12,15,18,19]. These studies consider different types of systems including distributed component-based systems, service-oriented applications, event-based systems and Grid computing environments.

In this tutorial, we present an introduction to queueing Petri nets (QPNs) first introducing the formalism itself. Then we present QPME (Queueing Petri net Modeling Environment) [17], an open-source tool for stochastic modeling and analysis of systems using QPNs. The tool is developed and maintained by the Descartes Research Group [6] at Karlsruhe Institute of Technology (KIT). The first version of the tool was released in January 2007 and since then it has been distributed to more than 130 organizations worldwide (universities, companies and research institutes). Since May 2011, QPME is distributed under the Eclipse Public License.

Afterwards, we summarize the results of several modeling case studies to demonstrate how QPNs can be used for performance modeling and analysis. Finally, as part of the tutorial, we briefly discuss our latest work on developing an automated model-to-model transformation from

ICPE'12, April 22-25, 2012, Boston, Massachusetts, USA

component-based software architecture models (with performance annotations) to QPN models analyzed using QPME. The transformation allows to specify models of software systems at a higher level of abstraction eliminating the need to build QPN models manually [14].

2. QUEUEING PETRI NETS

The main idea behind the QPN formalism 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. This is depicted in Figure 1.

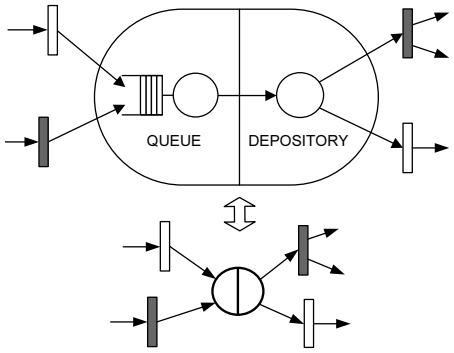


Figure 1: A queueing place and its shorthand notation.

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. $P = \{p_1, p_2, ..., p_n\}$ *is a finite and non-empty set of* places,
2. $T = \{t_1, t_2, ..., t_m\}$ *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. $Q = (\tilde{Q_1}, \tilde{Q_2}, (q_1, ..., q_{|P|}))$ *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*[2] *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. $W = (\tilde{W_1}, \tilde{W_2}, (w_1, ..., w_{|T|}))$ *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*
   - $w_i \in [C(t_i) \longmapsto \mathbb{R}^+]$ *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* $t_i \in \tilde{W_2}$.

For a more detailed introduction to the QPN modeling formalism, the reader is referred to [1, 4, 7].

2.1 Hierarchical Queueing Petri Nets

A major hurdle to the practical application of QPNs is the so-called largeness problem or *state-space explosion problem*: as one increases the number of queues and tokens in a QPN, the size of the model's state space grows exponentially and quickly exceeds the capacity of today's computers. This imposes a limit on the size and complexity of the models that are analytically tractable. An attempt to alleviate this problem was the introduction of *Hierarchically-Combined QPNs (HQPNs)* [2]. 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 that are about an order of magnitude larger than those analyzable with conventional techniques. HQPNs are a natural generalization of the original QPN formalism. In HQPNs, a queueing place may contain a whole QPN instead of a single queue. Such a place is called a *subnet place* and is depicted in Figure 2. A subnet place might contain an ordinary QPN or again a HQPN allowing multiple levels of nesting. For simplicity, we restrict ourselves to two-level hierarchies. We use the term *High-Level QPN (HLQPN)* to refer to the upper level of the HQPN and the term *Low-Level QPN (LLQPN)*

[1]The subscript MS denotes multisets. $C(p)_{MS}$ denotes the set of all finite multisets of $C(p)$.
[2]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 QPME, we use conventional queues as defined in queueing network theory.

to refer to a subnet of the HLQPN. Every subnet of a HQPN has a dedicated input and output place, which are ordinary places of a CPN. Tokens being inserted into a subnet place after a transition firing are added to the input place of the corresponding HQPN subnet. The semantics of the output place of a subnet place is similar to the semantics of the depository of a queueing place: tokens in the output place are available for output transitions of the subnet place. Tokens contained in all other places of the HQPN subnet are not available for output transitions of the subnet place. Every HQPN subnet also contains a *actual-population* place used to keep track of the total number of tokens fired into the subnet place.

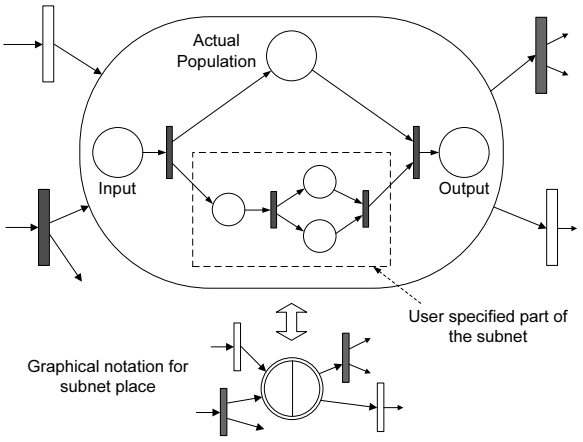


Figure 2: A subnet place and its shorthand notation.

2.2 Departure Disciplines

Departure disciplines are an extension of the QPN modeling formalism introduced in [7] to address a common limitation 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 [7]. 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 [4]. 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.

2.3 Example QPN Model

We now present an example QPN model of a simple Java EE system. The model was taken from [9] and is shown in Figure 3.

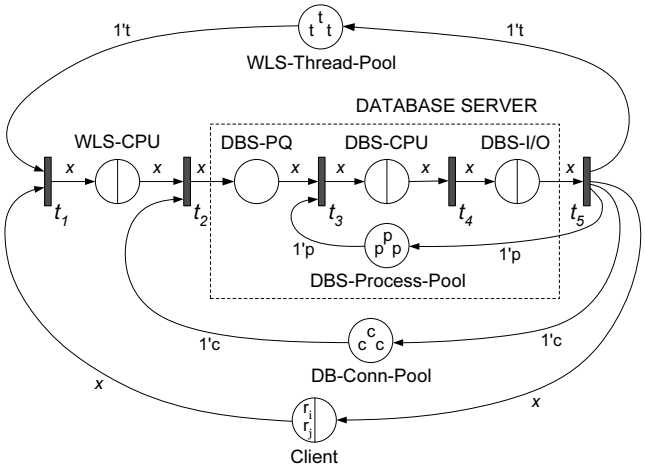


Figure 3: QPN Model of a Java EE System [9].

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 representing 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 representing the CPU of the *WebLogic Server (WLS)*.

`DBS-CPU` Queueing place with PS scheduling strategy representing the CPU of the *database server (DBS)*.

`DBS-I/O` Queueing place with FCFS scheduling strategy representing the disk subsystem of the DBS.

`WLS-Thread-Pool` Ordinary place representing the thread pool of the WLS. Each token in this place represents a WLS thread.

`DB-Conn-Pool` Ordinary place representing the database connection pool of the WLS. Tokens in this place represent database connections to the DBS.

`DBS-Process-Pool` Ordinary place representing 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`.

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_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 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. QUEUEING PETRI NET MODELING ENVIRONMENT (QPME)

QPME (Queueing Petri net Modeling Environment) [17] is an open-source tool for stochastic modeling and analysis of systems using QPNs, distributed under the Eclipse Public License. The tool is developed and maintained by the Descartes Research Group [6] at Karlsruhe Institute of Technology (KIT). QPME consists of two main components: a QPN Editor (QPE) and a Simulator for QPNs (SimQPN). In the following, we briefly describe these components.

3.1 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 4 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 instance, scheduling strategies and service time distributions of queueing places can be specified 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 5 shows the *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. 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.

- *Shared queues.* The user can specify that multiple queueing places share the same underlying physical queue[3]. 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 [18]. Shared queues are not supported in standard QPN models [18].

- *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.

[3]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].

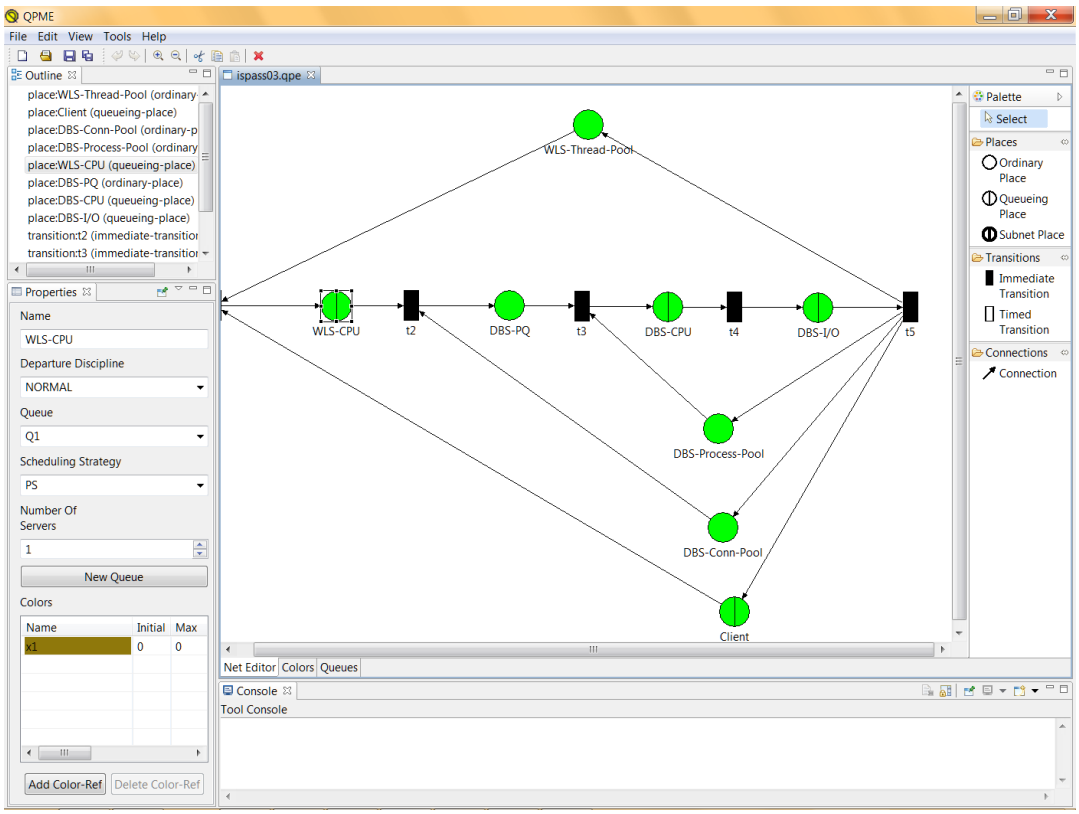


Figure 4: QPE main window.

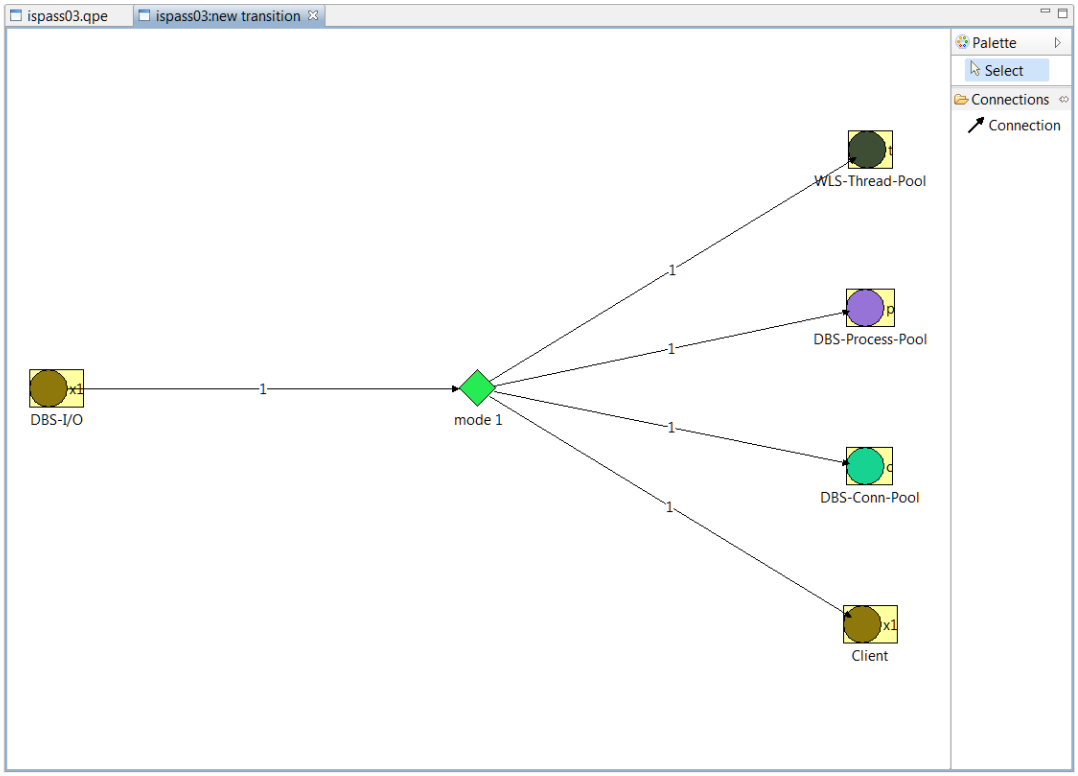


Figure 5: QPE incidence function editor.

- *Departure Disciplines.* Departure disciplines are defined for ordinary places or depositories and determine the order in which arriving tokens become available for output transitions. QPE supports two disciplines: Normal (used by default) and First-In-First-Out (FIFO). The latter implies that tokens become available for output transitions in the order of their arrival whereas the former does not consider the order of arrival of tokens. 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].

3.2 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 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.

3.2.1 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: First-Come-First-Served (FCFS), Processor-Sharing (PS), and Infinite Server (IS). 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. 3.1 are fully supported by the SimQPN simulator. A current limitation of SimQPN is the missing support for timed transitions[4] 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.

3.2.2 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: i) ordinary place, ii) queue of a queueing place (considered from the perspective of the place), iii) depository of a queueing place, iv) queue (considered from the perspective of all places it is part of), and v) probe.

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.

[4]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.

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.
- *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.

3.2.3 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 [13, 16] 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 [10]. 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 [13] for choosing the number of replications, their length and the window size.

3.2.4 Processing and Visualization of 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:

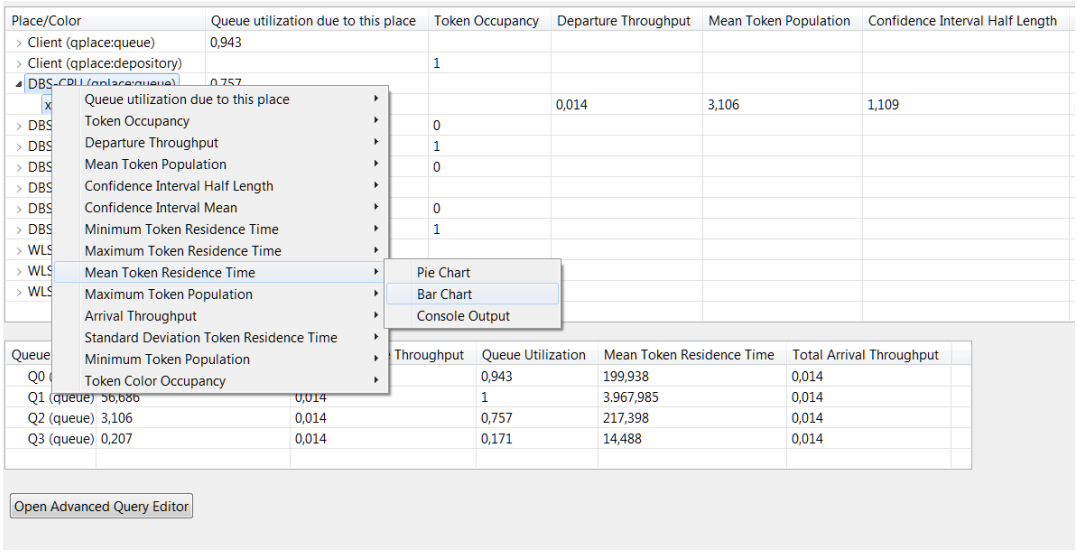


Figure 6: Metrics context menu of the simple query editor.

- *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".

Figure 6 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. When opening the context menu of one of the places, a context menu with a set of possible metrics of interest is made available.

4. MODELING CASE STUDIES

In this section, we summarize the results of several case studies in different application domains showing how QPN models can be exploited for performance analysis of different types of computer systems.

4.1 Distributed Component-Based Systems

In [7], we presented a novel case study of a realistic distributed component-based system, showing how QPN models can be exploited as a powerful performance prediction tool in the software engineering process. A detailed system model was built in a step-by-step fashion, validated, and then used to evaluate the system performance and scalability. The system studied in [7] is a deployment of the SPECjAppServer2004 benchmark for J2EE application servers. It models a complete end-to-end application that is designed to be representative of today's real-life distributed component-based systems. Along with the case study, a practical performance modeling methodology was presented in [7] which helps to construct models that accurately reflect the system performance and scalability characteristics. We showed that by taking advantage of the modeling power and expressiveness of QPNs our approach makes it possible to model the system at a higher degree of accuracy, providing a number of important benefits.

The QPN model of the SPECjAppServer2004 benchmark was validated by comparing the model predictions against measurements from the real system for a number of different transaction mixes. The QPN model we developed according to the presented methodology was able to predict the performance of the system accurately. We then used the validated QPN model to analyze the system with different deployment configurations and workload scenarios in order to analyze its scalability. We were able to correctly identify the load balancer as the bottleneck resource with our model. Furthermore, we showed that the accuracy of the QPN model can be improved further when introducing departure disciplines for places in QPNs. In summary, it was possible to analyze QPN models of realistic size and complexity using SimQPN, taking advantage of the modeling power and expressiveness of the QPN paradigm. Even for the largest and most complex scenarios, the modeling error for transaction response time did not exceed 20.6% and was much lower for transaction throughput and resource utilization.

4.2 Distributed Event-Based Systems

In [12,19], we presented a comprehensive methodology for workload characterization and performance modeling of Distributed Event-Based Systems (DEBS). The methodology helps to identify and eliminate bottlenecks and ensure that systems are designed and sized to meet their QoS requirements. The methodology is based on operational analysis and QPNs. We first used analytical analysis techniques to find the utilization of system components and derive an approximation for the mean event delivery latency. We then showed how more detailed performance models based on QPNs can be built to provide more accurate performance prediction. Modeling DEBS is particularly challenging because of the complete decoupling of communicating parties, on the one hand, and the dynamic changes in the system

structure and behavior, on the other hand. When a request is sent in a traditional request/reply-based distributed system, it is sent directly to a given destination which makes it easy to identify the system components and resources involved in its processing. In contrast to this, when an event is published in a DEBS, it is not addressed to a particular destination, but rather routed along all paths that lead to subscribers with matching subscriptions. In this case study, the DEBS was modeled with QPNs because it simplifies the modeling of forks of asynchronous tasks. In [12, 19], we also proposed the extension of the QPN formalism to support queues shared by several queueing places as it eases the modeling of a DEBS. This extension has been meanwhile integrated into QPME.

The SPECjms2007 benchmark was used in this case study to evaluate the performance of the proposed methodology. The SPECjms2007 benchmark is based on an application scenario modeling the supply chain of a supermarket company where RFID technology is used to track the flow of goods. Given the size and complexity of the modeled system, the resulting performance model was much larger and more complex than existing queueing models of message-oriented event-based systems. Overall, the presented model contained a total of 59 queueing places, 76 token colors and 68 transitions with a total of 285 firing modes. The model was analyzed using SimQPN which took less than 5 minutes in all cases. We considered several different scenarios that represent different types of messaging workloads stressing different aspects of the MOM infrastructure including both workloads focused on point-to-point messaging as well as workloads focused on publish/subscribe. In summary, the model proved to be very accurate in predicting the system performance, especially considering the size and complexity of the system that was modeled [19].

4.3 Enterprise Data Fabrics

Enterprise data fabrics are gaining increasing attention in many industry domains including financial services, telecommunications, transportation and health care. Providing a distributed, operational data platform sitting between application infrastructures and back-end data sources, enterprise data fabrics are designed for high performance and scalability. In [8], we presented a case study of a representative enterprise data fabric, the Gem-Fire EDF, presenting a simulation-based tool that we have developed for automated performance prediction and capacity planning. We implemented a tool that automates resource demand estimation, model generation, model analysis and results processing based on QPN models. Given a system configuration and a workload scenario, the tool generates a report showing the predicted system throughput, server utilization and operation response times. The tool uses SimQPN for solving the generated QPN models.

Both, the modeling approach and the model extraction technique were evaluated to demonstrate their effectiveness and practical applicability. We considered several workload and configuration scenarios and compared performance predictions obtained with SimQPN against measurements on the real system. Overall, our experiments showed that predictions of network and server utilization were quite accurate independent of the load level, while predictions of response times were reasonably accurate in the cases where the average server utilization was lower than 75% [8]. We also considered the analysis overhead and showed that the analysis overhead is acceptable for capacity planning purposes and the proposed approach can be applied for scenarios of reasonable size [8].

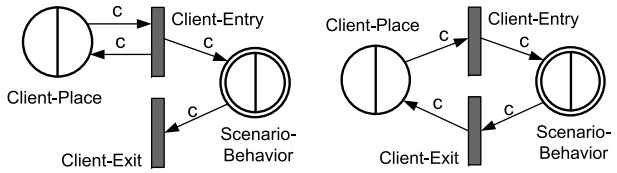
4.4 Enterprise Grid Environments

In [15], we presented a methodology for designing autonomic Quality of Service (QoS) aware resource managers that have the capability to predict the performance of the Grid components they manage and allocate resources in such a way that service level agreements are honored. Support for advanced features such as autonomic workload characterization on-the-fly, dynamic deployment of Grid servers on demand, as well as dynamic system reconfiguration after a server failure is provided. We implemented a resource manager framework that uses QPNs as online performance models for autonomic QoS control. The QPN models are solved using SimQPN to obtain QoS predictions. The approach was validated in two different experimental setups, the first one with only two Grid servers, the second one with up to nine servers running in a virtualized environment. As a basis for the experiments, we used three sample services each with different behavior and service demands. The results of the predictions from SimQPN were compared to measurements from a real system and to predictions obtained with OMNeT++ models. The results showed that both QPN and OMNeT++ models provided very consistent and accurate predictions of performance metrics. There was hardly any difference between confidence intervals provided by SimQPN and OMNeT++. At the same time, while OMNeT++ results were limited to request response times, SimQPN results were more comprehensive and included estimates of request throughputs, server utilization, queue lengths, etc. In all cases, the modeling error was below 15%. For details about the experiment setup and procedure see [15].

Five different scenarios each focusing on selected aspects of the framework were studied. We compared the behavior of the system in two different configurations - "with QoS Control" vs. "without QoS Control". In the first configuration, the resource manager applied admission control using our resource allocation framework to ensure that SLAs are honored. In the second configuration, the resource manager simply load-balanced the incoming requests over the two servers without considering QoS requirements. In all scenarios with QoS Control enabled, the measured response times were stable and nearly 100% of the client SLAs were fulfilled. The results confirmed the effectiveness of our resource manager architecture in ensuring that QoS requirements are continuously met.

5. MODEL-TO-MODEL TRANSFORMATIONS

In [14], we showed how an automated model-to-model transformation can be used to evaluate the performance of component-based systems modeled at the software architectural level by means of automatically generated QPN models. The source language, in this case, the Palladio Component Model (PCM) [5], is automatically transformed into a suitable QPN representation, which is then solved using the tools introduced earlier. In this section, we briefly discuss PCM and then illustrate the transformation by presenting as an example the mapping of open and closed workloads.



(a) OpenWorkload (b) ClosedWorkload

Figure 7: Workload QPN.

Finally, we summarize the results from an in-depth evaluation of the transformation.

5.1 Palladio Component Model (PCM)

The Palladio Component Model (PCM) [5] is a meta-model allowing the specification of performance-relevant information of a component-based architecture. It focuses on the software performance engineering (SPE) and component-based software engineering (CBSE) domains. Four factors essentially determine the performance of a software component: its implementation, the performance of external services it requires, the performance of the execution environment it is deployed on, and the usage profile. Each of these aspects is described using parametric dependencies. The PCM provides a domain-specific language for describing component-based software architectures and the performance characteristics of the employed components.

5.2 PCM2QPN Transformation

To illustrate how elements of PCM are mapped to QPN elements, the mapping of closed and open workloads is presented in the following.

The workload in PCM is represented as a number of usage scenarios, running in parallel. Each scenario has a workload specification and contains a specification of the scenario behavior. Two kinds of workloads are supported. An open workload is characterized by an inter-arrival time distribution, which describes the time that elapses between consecutive requests. Figure 7(a) shows how the *OpenWorkload* usage model entity is represented in the generated QPN to achieve the open workload semantics. The *Client-Place* queueing place generates tokens of a color c which is different for each *UsageScenario*. It references a client queue with Infinite Server (IS) scheduling strategy. An empirical distribution is used for the *Client-Place* resource demand representing *inter-arrival time* distribution of the *OpenWorkload*. The initial number of tokens is set to 1. For each input token the *Client-Entry* transition creates a new token in the subnet representing the *ScenarioBehavior*. Another token is created in the *Client-Place* queue. The *Client-Exit* transition destroys tokens of color c from the *ScenarioBehavior* subnet.

A closed workload is characterized both by an integral population, as well as by a think time distribution. There is a fixed number of requests, each of which has to wait according to the think time after its processing is complete. Figure 7(b) shows the mapping for a *ClosedWorkload*. The difference is that the *Client-Entry* transition does not generate any tokens in the *Client-Place*. Instead, this is done by the *Client-Exit* transition. At the *Client-Place* we use an empirical distribution for the resource demand equal to the *think time* distribution of the *ClosedWorkload*. The initial number of tokens is set to the *population* of the closed workload. A new token will now be available after it has passed through the whole *ScenarioBehavior* and after the residence time in the queue, which equals the *think time*.

Other elements of the PCM meta-model are mapped in a similar manner. Also, as part of the transformation, the individual QPN parts of the mapped PCM elements are connected using ordinary places according to the connections within the PCM model. The different possible token colors for each QPN part are similarly derived from the different usages of a component definition within the PCM system model.

5.3 Evaluation

The approach of using an automated transformation for the analysis of PCM models was evaluated in the context of five representative case studies. The details are omitted here for brevity and can be found in [14]. Compared to existing tools to analyze PCM models, our approach leads to performance predictions of mean value metrics with high accuracy at a significantly reduced analysis overhead in many cases by an order of magnitude. Important limitations of the approach are the inability to model the synchronization of multiple requests at a barrier, as well as the inability to represent stochastic dependencies between PCM expression language variables.

6. CONCLUSIONS

In this tutorial, we presented an introduction to the QPN modeling formalism and showed how it can be used for performance modeling of distributed systems. QPNs have a number of benefits compared to traditional queueing networks and stochastic Petri nets. Exploiting the modeling power and expressiveness of QPNs, QPN models can be used to accurately capture both hardware and software aspects of system behavior. We summarized the results from several modeling case studies in different application domains (including distributed component-based systems, distributed event-based systems, enterprise data fabrics, and enterprise grid environments) showing how QPNs can be used for performance modeling in these domains. Furthermore, we briefly introduced QPME, a tool for stochastic modeling and analysis using QPNs. Finally, we showed how software architecture models (with performance annotations capturing performance relevant aspects) can be automatically transformed to QPNs eliminating the need to build QPN models manually.

Acknowledgements

This work was partially funded by the German Research Foundation (DFG) under grant No. KO 3445/6-1.

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. *Hierarchically combined queueing Petri nets*, volume 199 of *Lecture Notes in Control and Information Sciences*, pages 176–182. Springer Berlin / Heidelberg, 1994.

[3] F. Bause, P. Buchholz, and P. Kemper. 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] F. Bause and F. Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory*. Vieweg Verlag, 2002.

[5] S. Becker, H. Koziolek, and R. Reussner. The palladio component model for model-driven performance prediction. *Journal of Systems and Software*, 82(1):3–22, 2009.

[6] Descartes Research Group. http://www.descartes-research.net, November 2011.

[7] 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.

[8] S. Kounev, K. Bender, F. Brosig, N. Huber, and R. Okamoto. Automated Simulation-Based Capacity Planning for Enterprise Data Fabrics. In *Proceedings of the 4th International ICST Conference on Simulation Tools and Techniques (SIMUTools 2011), Barcelona, Spain*, 2011.

[9] S. Kounev and A. Buchmann. Performance Modelling of Distributed E-Business Applications using Queuing Petri Nets. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2003), Austin, USA, March 20-22*, 2003.

[10] 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.

[11] S. Kounev, R. Nou, and J. Torres. Autonomic QoS-Aware Resource Management in Grid Computing using Online Performance Models. In *Proceedings of the 2nd International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2007), Oct. 23-25, Nantes, France*, 2007.

[12] S. Kounev, K. Sachs, J. Bacon, and A. Buchmann. A Methodology for Performance Modeling of Distributed Event-Based Systems. In *Proceedings of the 11th IEEE International Symposium on Object/Component/Service-oriented Real-time Distributed Computing (ISORC 2008), Orlando, USA*, May 2008.

[13] A. Law and D. W. Kelton. *Simulation Modeling and Analysis*. Mc Graw Hill Companies, Inc., third edition, 2000.

[14] P. Meier, S. Kounev, and H. Koziolek. Automated Transformation of Palladio Component Models to Queueing Petri Nets. In *Proceedings of the 19th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2011)*, Singapore, July 25-27 2011.

[15] R. Nou, S. Kounev, F. Julia, and J. Torres. Autonomic QoS control in enterprise Grid environments using online simulation. *Journal of Systems and Software*, 82(3):486–502, Mar. 2009.

[16] K. Pawlikowski. Steady-State Simulation of Queueing Processes: A Survey of Problems and Solutions. *ACM Computing Surveys*, 22(2):123–170, 1990.

[17] QPME Homepage. http://qpme.sourceforge.net, November 2011.

[18] K. Sachs. *Performance Modeling and Benchmarking of Event-based Systems*. PhD thesis, TU Darmstadt, 2010.

[19] K. Sachs, S. Kounev, and A. Buchmann. Performance Modeling and Analysis of Message-oriented Event-driven Systems. *Journal of Software and Systems Modeling (SoSyM)*, January 2012. To appear.