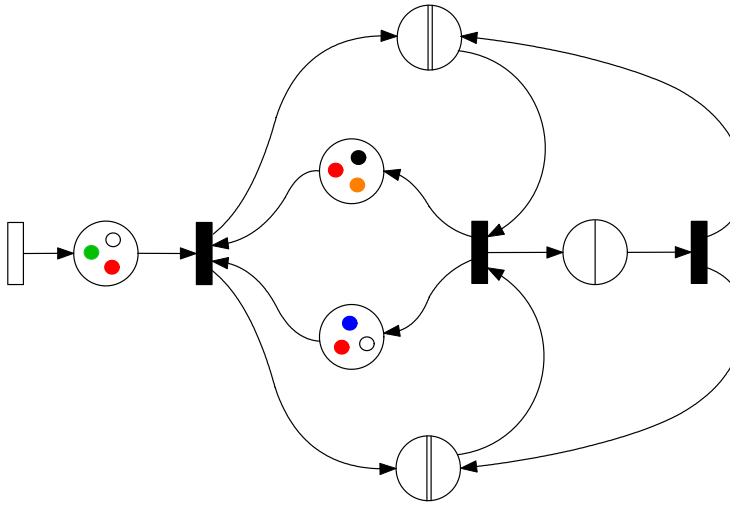
QPME User's Guide v2.1

Queueing Petri net Modeling Environment



A software tool for performance modeling and analysis using queueing Petri nets

Samuel Kounev, Simon Spinner and Jürgen Walter

March 2015

v2.1-150311

Contents

List of Acronyms

Acronym	Meaning
CGSPN	Colored Generalized Stochastic Petri Net
CPN	Colored Petri Net
DBS	Database Server
DCS	Distributed Component-based System/s
DES	Discrete Event Simulation
FCFS	First-Come-First-Serve (scheduling strategy)
FIFO	First-In-First-Out
GSPN	Generalized Stochastic Petri Net
HLQPN	High-Level Queueing Petri Net
HQPN	Hierarchical Queueing Petri Net
IID	Independent and Identically Distributed (random variables)
IS	Infinite Server (scheduling strategy)
LCFS	Last-Come-First-Served (scheduling strategy)
LLQPN	Low-Level Queueing Petri Net
LQN	Layered Queueing Network
NOBM	Method of Non-Overlapping Batch Means
PEPSY-QNS	Perf. Evaluation and Prediction SYstem for Queueing NetworkS
PN	(Ordinary) Petri Net
PS	Processor-Sharing (scheduling strategy)
QN	Queueing Network
QoS	Quality of Service
QPN	Queueing Petri Net
RR	Round-Robin (scheduling strategy)
SLAs	Service Level Agreements
SPN	Stochastic Petri Net

Chapter 1

Introduction

This document describes the software package QPME (Queueing Petri net Modeling Environment), a performance modeling and analysis tool based on the queueing Petri net (QPN) modeling formalism. QPME is made of two components: QPE (QPN Editor) and SimQPN (Simulator for QPNs). QPE provides a user-friendly graphical editor for QPN models based on the Eclipse/GEF framework. SimQPN provides an efficient discrete-event simulation engine for QPNs that can be used for model analysis. QPME runs on a wide range of platforms including Windows, Linux, MacOS and Solaris. The tool is developed and maintained by the Descartes Research Group at Karlsruhe Institute of Technology (KIT).

This document presents QPME discussing its features and usage. The aim is to provide the user with the information needed in order to use the tool effectively without needing to understand its internals. More information on QPME's internal implementation details, including detailed specification of the analysis techniques implemented, can be found in [9, 14]. An overview of the tool is available in [15, 16].

1.1 System Requirements

QPE runs on all platforms supported by Eclipse including Windows, Linux, Solaris, HP-UX, IBM AIX and Apple Mac OS. The only thing required is a Java Runtime Environment (JRE) 6.0 or later. It is recommended to run QPE on Windows since this is the platform it has been tested on.

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 6 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.

Chapter 2

Primer on Queueing Petri Nets

2.1 Basic Queueing Petri Nets

Queueing Petri nets can be seen as a combination of a number of different extensions to conventional Petri nets (PNs) along several different dimensions. In this section, we include some basic definitions and briefly discuss how queueing Petri nets have evolved. A more detailed treatment of the subject can be found in [2, 3, 11, 12]. An ordinary *Petri net* is a bipartite directed graph composed of places, drawn as circles, and transitions, drawn as bars. A formal definition follows [3]:

Definition 2.1 *An ordinary Petri Net (PN) is a 5-tuple* $PN = (P, T, I^-, I^+, M_0)$ *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. $I^-, I^+ : P \times T \to \mathbb{N}_0$ *are called backward and forward* incidence functions*, respectively,*
4. $M_0 : P \to \mathbb{N}_0$ *is called* initial marking*.*

The incidence functions I^- and I^+ specify the interconnections between places and transitions. If $I^-(p,t) > 0$, an arc leads from place p to transition t and place p is called an *input place* of the transition. If $I^+(p,t) > 0$, an arc leads from transition t to place p and place p is called an *output place* of the transition. The incidence functions assign natural numbers to arcs, which we call *weights* of the arcs. When each input place of transition t contains at least as many tokens as the weight of the arc connecting it to t, the transition is said

to be *enabled*. An enabled transition may *fire*, in which case it destroys tokens from its input places and creates tokens in its output places. The amounts of tokens destroyed and created are specified by the arc weights. The initial arrangement of tokens in the net (called *marking*) is given by the function M_0, which specifies how many tokens are contained in each place.

Different extensions to ordinary PNs have been developed in order to increase the modeling convenience and/or the modeling power. *Colored PNs (CPNs)* introduced by K. Jensen [10] are one such extension. The latter allow a type (color) to be attached to a token. A color function C assigns a set of colors to each place, specifying the types of tokens that can reside in the place. In addition to introducing token colors, CPNs also allow transitions to fire in different *modes* (transition colors). The color function C assigns a set of modes to each transition and incidence functions are defined on a per mode basis. A formal definition of a CPN follows [3]:

Definition 2.2 *A Colored PN (CPN) is a 6-tuple* $CPN = (P, T, C, I^-, I^+, M_0)$ *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* $I^-(p,t), I^+(p,t) \in [C(t) \to C(p)_{MS}]$, $\forall (p,t) \in P \times T$[1]
5. M_0 *is a function defined on* P *describing the* initial marking *such that* $M_0(p) \in C(p)_{MS}$.

Other extensions to ordinary PNs allow temporal (timing) aspects to be integrated into the net description [3]. In particular, *Stochastic PNs (SPNs)* attach an exponentially distributed *firing delay* to each transition, which specifies the time the transition waits after being enabled before it fires. *Generalized Stochastic PNs (GSPNs)* allow two types of transitions to be used: immediate and timed. Once enabled, immediate transitions fire in zero time. If several immediate transitions are enabled at the same time, the next transition to fire is chosen based on *firing weights* (probabilities) assigned to the transitions. Timed transitions fire after a random exponentially distributed firing delay as in the case of SPNs. The firing of immediate transitions always has priority over that of timed transitions. A formal definition of a GSPN follows [3]:

Definition 2.3 *A Generalized SPN (GSPN) is a 4-tuple* $GSPN = (PN, T_1, T_2, W)$ *where:*

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

1. $PN = (P, T, I^-, I^+, M_0)$ *is the underlying ordinary PN,*
2. $T_1 \subseteq T$ *is the set of timed transitions,* $T_1 \neq \emptyset$*,*
3. $T_2 \subset T$ *is the set of immediate transitions,* $T_1 \cap T_2 = \emptyset$*,* $T_1 \cup T_2 = T$*,*
4. $W = (w_1, ..., w_{|T|})$ *is an array whose entry* $w_i \in \mathbb{R}^+$ *is a rate of a negative exponential distribution specifying the firing delay, if* $t_i \in T_1$ *or is a firing weight specifying the relative firing frequency, if* $t_i \in T_2$*.*

Combining definitions 2.2 and 2.3, leads to *Colored GSPNs (CGSPNs)* [3]:

Definition 2.4 *A Colored GSPN (CGSPN) is a 4-tuple* $CGSPN = (CPN, T_1, T_2, W)$ *where:*

1. $CPN = (P, T, C, I^-, I^+, M_0)$ *is the underlying CPN,*
2. $T_1 \subseteq T$ *is the set of timed transitions,* $T_1 \neq \emptyset$*,*
3. $T_2 \subset T$ *is the set of immediate transitions,* $T_1 \cap T_2 = \emptyset$*,* $T_1 \cup T_2 = T$*,*
4. $W = (w_1, ..., w_{|T|})$ *is an array with* $w_i \in [C(t_i) \longmapsto \mathbb{R}^+]$ *such that* $\forall c \in C(t_i) : w_i(c) \in \mathbb{R}^+$ *is a rate of a negative exponential distribution specifying the firing delay due to color c, if* $t_i \in T_1$ *or is a firing weight specifying the relative firing frequency due to c, if* $t_i \in T_2$*.*

While CGSPNs have proven to be a very powerful modeling formalism, they do not provide any means for direct representation of queueing disciplines. The attempts to eliminate this disadvantage have led to the emergence of *Queueing PNs (QPNs)*. The main idea behind the QPN modeling paradigm was to add queueing and timing aspects to the places of CGSPNs. This is done by allowing queues (service stations) to be integrated into places of CGSPNs. A place of a CGSPN that has an integrated queue is called a *queueing place* and consists of two components, the *queue* and a *depository* for tokens which have completed their service at the queue. This is depicted in Figure 2.1.

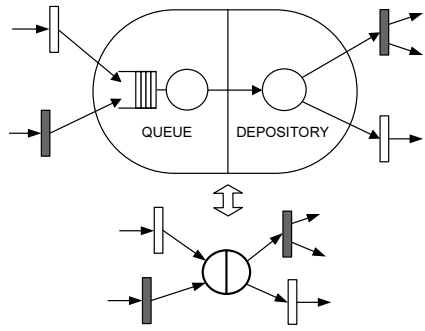


Figure 2.1: A queueing place and its shorthand notation.

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:

Definition 2.5 *A Queueing PN (QPN) is an 8-tuple* $QPN = (P, T, C, I^-, I^+, M_0, Q, W)$ *where:*

1. $CPN = (P, T, C, I^-, I^+, M_0)$ *is the underlying Colored PN*
2. $Q = (\tilde{Q}_1, \tilde{Q}_2, (q_1, ..., q_{|P|}))$ *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.*
3. $W = (\tilde{W}_1, \tilde{W}_2, (w_1, ..., w_{|T|}))$ *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* $t_i \in \tilde{W}_2$.

Example 2.1 (QPN) *Figure 2.2 shows an example of a QPN model of a central server system with memory constraints based on [3]. Place* p_2 *represents several terminals, where users start jobs (modeled with tokens of color 'o') after a certain thinking time. These jobs request service at the CPU (represented by a G/C/1/PS queue, where C stands for Coxian distribution) and two*

[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 [2]. For the purposes of this paper, it is enough to use conventional queues as defined in queueing network theory.

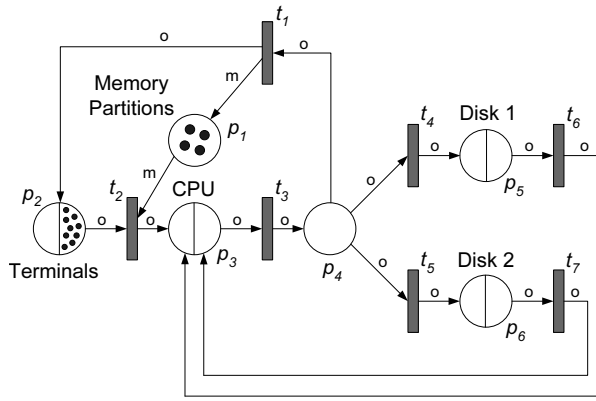


Figure 2.2: A QPN model of a central server with memory constraints (reprinted from [3]).

disk subsystems (represented by G/C/1/FCFS queues). To enter the system each job has to allocate a certain amount of memory. The amount of memory needed by each job is assumed to be the same, which is represented by a token of color 'm' on place p_1*. According to Definition 2.5, we have the following:* $QPN = (P, T, C, I^-, I^+, M_0, Q, W)$ *where*

- $CPN = (P, T, C, I^-, I^+, M_0)$ *is the underlying Colored PN as depicted in Figure 2.2,*
- $Q = (\tilde{Q}_1, \tilde{Q}_2,$
  $(null, G/C/\infty/IS, G/C/1/PS, null, G/C/1/FCFS, G/C/1/FCFS)),$
  $\tilde{Q}_1 = \{p_2, p_3, p_5, p_6\}, \tilde{Q}_2 = \emptyset,$
- $W = (\tilde{W}_1, \tilde{W}_2, (w_1, ..., w_{|T|}))$*, where* $\tilde{W}_1 = \emptyset, \tilde{W}_2 = T$ *and*
  $\forall c \in C(t_i) : w_i(c) := 1$*, so that all transition firings are equally likely.*

In [1] it is shown that QPNs have greater expressive power than QNs, extended QNs and SPNs. In addition to hardware contention and scheduling strategies, using QPNs one can easily model simultaneous resource possession, synchronization, blocking and software contention. This enables the integration of hardware and software aspects of system behavior into the same model [5]. While the above could also be achieved by using Layered QNs (LQNs) (or stochastic rendezvous networks), the latter are defined at a higher-level of abstraction and are usually less detailed and accurate. Another benefit of QPNs is that, since they are based on Petri nets, one can exploit a number of efficient techniques from Petri net theory to verify some important qualitative properties of QPNs, such as ergodicity, boundedness, liveness or existence of home states. The latter not only help to gain insight into the behavior of QPNs, but are also essential preconditions for a successful quantitative analysis [2].

2.2 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)* [4]. 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.

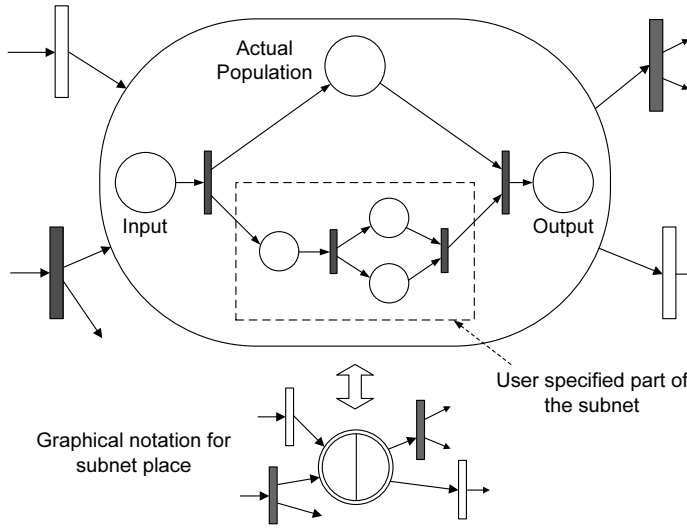


Figure 2.3: A subnet place and its shorthand notation.

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 2.3. 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)* 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 $actual - population$ place used to keep track of the total number of tokens fired into the subnet place.

Chapter 3

Building QPN Models with QPE

3.1 Overview

QPE (Queueing Petri net Editor), the first major component of QPME, provides a graphical tool for modeling using QPNs. It offers a user-friendly interface enabling the user to quickly and easily construct QPN models. QPE is based on GEF (Graphical Editing Framework) [20] - an Eclipse sub-project. GEF 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 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.

Internally, being a GEF application, QPE is based on the model-view-controller 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 PNML [6] with some changes and extensions to support the additional constructs available in QPN models.

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.

Another characterizing feature of QPE, not supported in standard QPN mod-

els, is the ability to have multiple queueing places configured to share the same underlying physical queue[1]. 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. This feature of QPE, combined with the support for hierarchical QPNs, allows to build multi-layered models of software architectures similar to the way this is done in layered queueing networks, however, with the advantage that QPNs enjoy all the benefits of Petri nets for modeling synchronization aspects.

For further details on the way QPE is implemented, the reader is referred to [9].

3.2 QPE User Interface

3.2.1 QPE Main Window

Figure 3.1 shows the QPE main window, which is comprised of four views: "Main Editor View", "Outline View", "Properties View" and "Console View". In the following, we take a brief look at each of these views. After that, we show how QPN models are defined in QPE.

Main Editor View

The "Main Editor View" is made up of "Net Editor", "Color Editor", "Queue Editor" and "Palette". The "Net Editor" displays the graphical representation of the currently edited QPN. It provides multiple document interface allowing to have several models open at the same time. The "Color Editor" is used to define the global list of token colors available for use in the places of the QPN, whereas the "Queue Editor" is used to define the global list of queues used inside the queueing places of the QPN. Finally, the "Palette" displays the set of QPN elements from which QPN models are constructed.

Outline View

The "Outline View" provides a summary of the content of the currently active "Net Editor". It lists the elements of the QPN model displayed in the latter and makes it easy to find an element based on its name. When an element is selected in the "Outline View", it is automatically selected in the "Net Editor" as well, and the canvas is scrolled to its position so that the user can see it. This feature is especially useful for large QPN models.

[1] 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 [5].

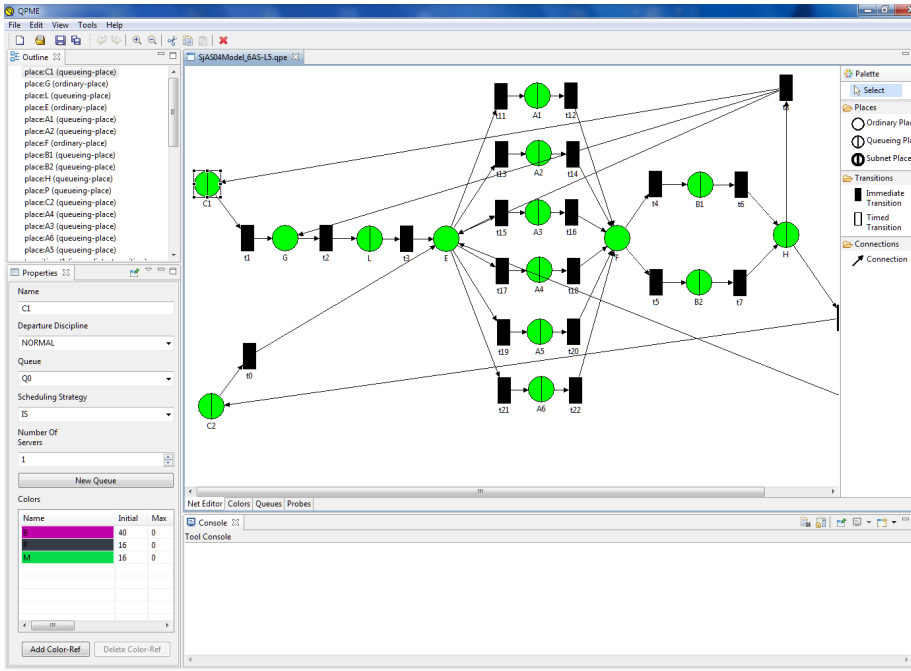


Figure 3.1: QPE Main Window

Properties View

The "Properties View" enables the user to edit the properties of the currently selected element in the "Net Editor". The content of this view depends on the type of the selected element.

Console View

The "Console View" is used to display output from QPE extensions and plug-ins such as SimQPN. For example, SimQPN uses the "Console View" to display progress updates during a simulation run as well as the results from the simulation output data analysis.

3.2.2 Building QPN Models

The first thing that has to be done when building a QPN model is to define the global list of colors that will be available for use in the places of the QPN. As mentioned earlier, colors are defined using the "Color Editor" in the "Main Editor View". The "Color Editor", shown in Figure 3.2, is opened by selecting the "Colors" tab at the bottom of the "Main Editor View". The "Color Editor" consists of a table showing the currently defined colors and two buttons at the

bottom of the table for adding and deleting colors. The delete button is only enabled when a color is selected. Each color has three attributes - "Name", "Real Color" and "Description". These attributes can be edited by clicking inside the table. The "Name" attribute provides a unique identifier of each color that can be used as a reference to the latter inside the places of the QPN. The "Real Color" is used to make it easier to visually distinguish between different colors when referencing them. The "Description" attribute defines the semantics of the entity modeled using the respective token color.

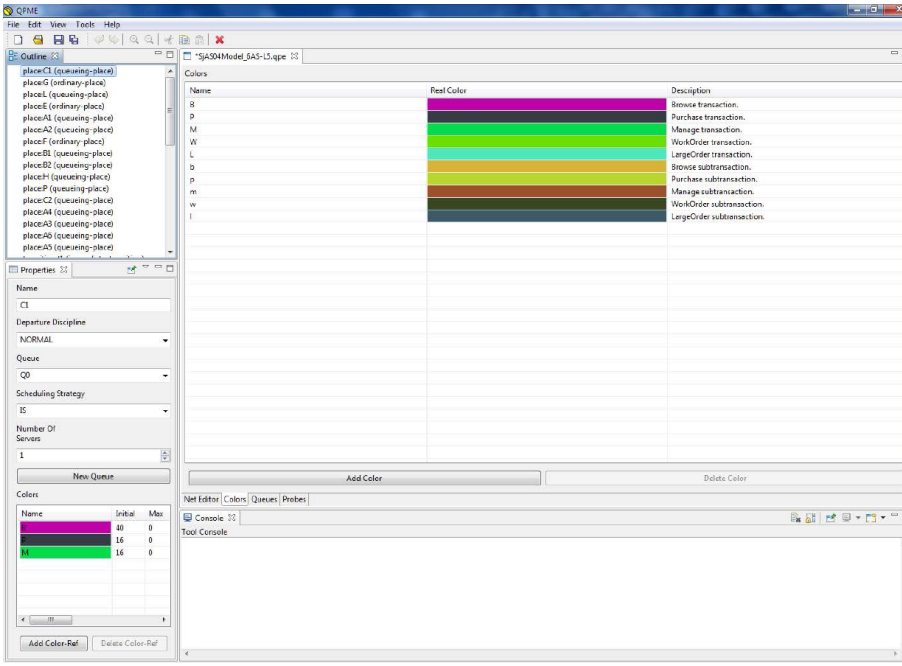


Figure 3.2: Color Editor

Similarly to the "Color Editor", QPE provides a "Queue Editor" where the global list of queues available for use as underlying queues in the queueing places of the QPN can be defined. The "Queue Editor", shown in Figure 3.3, is opened by selecting the "Queues" tab at the bottom of the "Main Editor View". The "Queue Editor" consists of a table showing the currently defined queues and two buttons at the bottom of the table for adding and deleting queues. The delete button is only enabled when a queue is selected. Each queue has four attributes - "Name", "Scheduling Strategy", "Number of Servers" and "Description". These attributes can be edited by clicking inside the table. The "Name" attribute provides a unique identifier of each queue that can be used as a reference to the latter inside the queueing places of the QPN. The "Scheduling Strategy" determines the order in which tokens are served in the queue (the possible settings

are explained later in this section). The "Number of Servers" attribute specifies the number of servers in the queue. Finally, the "Description" attribute defines the semantics of the entity modeled using the respective queue.

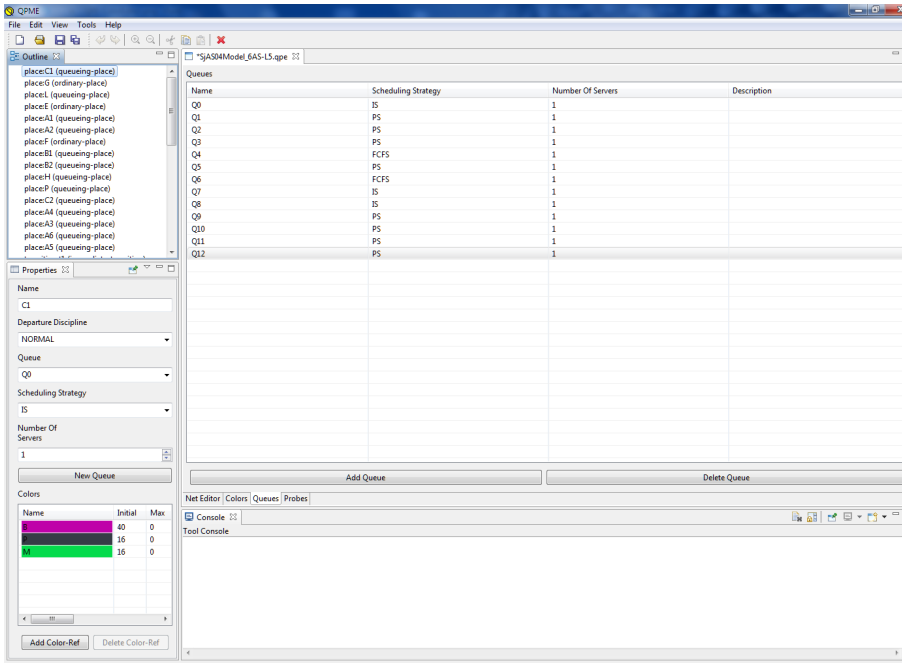


Figure 3.3: Queue Editor

Once the required colors and queues have been defined, the user can start putting together the QPN model[2]. In order to do this the user has to switch back to the "Net Editor" tab of the "Main Editor View". QPN models are built using the set of QPN elements available in the "Palette". In order to add an element to the model the user has to select it in the "Palette" and then click inside the canvas of the "Net Editor". The following QPN elements are currently available in the "Palette": "Ordinary Place", "Queueing Place", "Subnet Place", "Immediate Transition", "Timed Transition" and "Connection". The "Connection" element is used to create connections between places and transitions. A connection is created by selecting the "Connection" element and then dragging the mouse pointer from the input element to the output element.

The attributes of a QPN element (place or transition) can be edited by selecting the element and using the "Properties View". Depending on the type of element selected, different attributes are available as shown in the following:

[2]Note that QPE does not require that colors and queues be defined in the beginning of the model construction. Colors and queues can be added at any time by opening the respective tab of the "Main Editor View".

Attributes of Ordinary Places

- **Name:** Name of the ordinary place.
- **Departure Discipline:** NORMAL or FIFO (First-In-First-Out). 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. Departure disciplines are an extension to the QPN modeling formalism introduced in QPME. For more details refer to [11, 12].
- **Colors:** Token colors allowed in this place. For each token color the following parameters can be configured:
  - **Name:** Name of the color as defined in the "Color Editor".
  - **Initial:** Initial number of tokens of the respective color in the place (initial marking of the QPN).
  - **Max:** Maximum number of tokens of the respective color allowed in the place.

Attributes of Queueing Places

Name: Same as for ordinary place.

Departure Discipline: Same as for ordinary place.

Queue: Underlying queue of the queueing place.[3]

Scheduling Strategy: The scheduling strategy (or queueing discipline) determines the order in which tokens are served in the queue. The following values are currently allowed:

- **FCFS:** First-Come-First-Served.
- **PS:** Processor-Sharing.
- **IS:** Infinite-Server.
- **PRIO:** Priority scheduling.
- **RANDOM:** Random scheduling.

Number of Servers: Number of servers in the queue (queueing station) of the place[4].

[3] The "New Queue" button can be used to create a new queue for the queueing place which is automatically added to the global queue list in the "Queue Editor".

[4] Note that the "Scheduling Strategy" and "Number of Servers" fields are automatically set to the values of the respective fields in the "Queue Editor".

Colors: Token colors allowed in this place. For each token color the following parameters can be configured:

- **Name:** Same as for ordinary place.
- **Initial:** Same as for ordinary place.
- **Max:** Same as for ordinary place.
- **Ranking:** Ranking of the token color.
- **Priority:** Used for "Priority" scheduling.
- **Distribution:** Distribution of the token service time.
- **p1:** 1st parameter of the distribution.
- **p2:** 2nd parameter of the distribution (if applicable).
- **p3:** 3rd parameter of the distribution (if applicable).
- **Input File:** Input file for empirical distribution.

Table 3.1 shows a list of the currently supported distribution functions and their respective input parameters.

Table 3.1: Supported distributions and their input parameters.

Distribution	**p1**	**p2**	**p3**
Beta	alpha	beta	*na*
BreitWigner	mean	gamma	cut
BreitWignerMeanSquare	mean	gamma	cut
ChiSquare	freedom	*na*	*na*
Gamma	alpha	lambda	*na*
Hyperbolic	alpha	beta	*na*
Exponential	lambda	*na*	*na*
ExponentialPower	tau	*na*	*na*
Logarithmic	p	*na*	*na*
Normal	mean	stddev	*na*
StudentT	freedom	*na*	*na*
Uniform	min	max	*na*
VonMises	freedom	*na*	*na*
Empirical	scale	offset	*na*
Deterministic	c	*na*	*na*

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). The array is read from an external text file whose name and location are initialized using the "Input File" parameter.

Successive values in the text file must be delimited using semicolon ';' characters. 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).

Attributes of Subnet Places

- **Name:** Name of the subnet place.
- **Departure Discipline:** NORMAL or FIFO (First-In-First-Out). 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. Departure disciplines are an extension to the QPN modeling formalism introduced in QPME. For more details refer to [11, 12].
- **Colors:** Token colors allowed in this place. For each token color the following parameters can be configured:
  - **Name:** Name of the color as defined in the "Color Editor".
  - **Initial:** Initial number of tokens of the respective color in the place (initial marking of the QPN).
  - **Max:** Maximum number of tokens of the respective color allowed in the place.
  - **Direction:** Specifies if tokens of the respective color are allowed to enter the subnet (**in**), if they are allowed to leave the subnet (**out**) or if both directions are possible (**both**). This setting controls the propagation of color references to the input and the output place of the subnet.

Attributes of Immediate Transitions

- **Name:** Name of the immediate transition.
- **Priority:** Firing priority.
- **Firing Weight:** Relative firing frequency of the transition.
- **Modes:** Modes in which the transition can fire. For each mode the following parameters can be configured:
  - **Name:** Name of the mode.
  - **Real Color:** Used to make it easier to visually distinguish between different modes when defining the incidence functions.
  - **Firing Weight:** Relative firing frequency of the mode.

Attributes of Timed Transitions

- **Name:** Name of the timed transition.
- **Priority:** Firing priority.
- **Modes:** Modes in which the transition can fire. For each mode the following parameters can be configured:
  - **Name:** Name of the mode.
  - **Real Color:** Used to make it easier to visually distinguish between different modes when defining the incidence functions.
  - **Mean Firing Delay:** Firing delay of the mode.

Defining Transition Incidence Functions

Transition incidence functions in QPE are defined using the "Incidence Function Editor" shown in Figure 3.4.

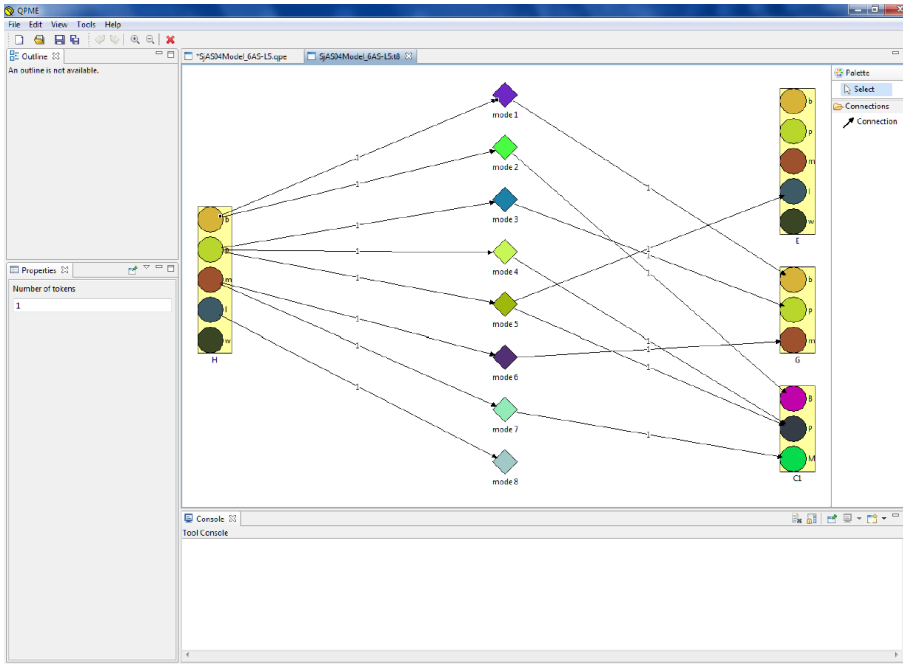


Figure 3.4: Incidence Function Editor

The "Incidence Function Editor" can be opened by double-clicking on the respective transition element in the "Net Editor" or by right-clicking it and using the context menu. 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. Using the "Connection" tool in the "Palette", 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 by clicking on it and using the "Properties" view. The weight, displayed as label next to the connection line, is interpreted as the number of tokens removed/deposited in the place when the transition fires in the respective mode.

Defining Subnets

Subnets of subnet places in QPE are defined using the "Subnet Editor" shown in Figure 3.5.

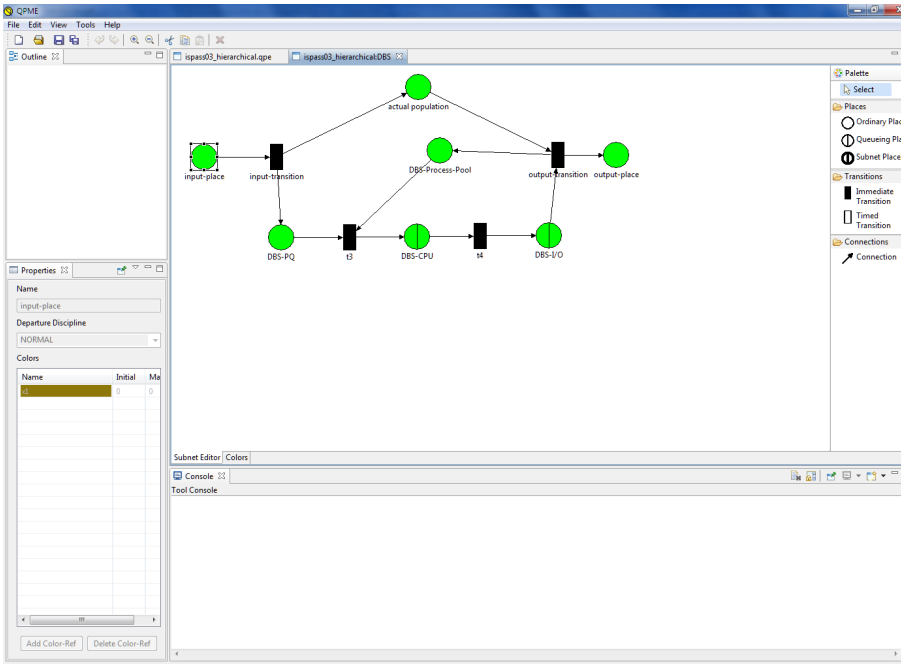


Figure 3.5: Subnet Editor

The "Subnet Editor" can be opened by double-clicking on the respective

subnet place element in the "Net Editor" or by right-clicking it and using the context menu. Once opened the "Subnet Editor" provides two editor pages: "Subnet Editor" and "Color Editor". The former can be used in the same way as the "Net Editor", except that it contains three special places (input-place, actual-population and output-place) and two special transitions (input-transition and output-transition). The semantics of these elements are described in Section 2.2. The properties of the input, output and actual-population places cannot be modified. They are automatically managed by QPE depending on the properties of the subnet place. If more flexibility is required, the actual-population place, the input and the output transitions can be deleted from the subnet.

The "Colors" page of the "Subnet Editor" can be used to define colors local to the subnet. These colors can only be referenced within the same subnet or contained subnets. Colors that are defined globally are greyed out and cannot be edited in the subnet's color editor. In order to edit global color definitions use the color editor of the whole net.

Use of Probes for Data Collection

A probe is a tool 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 start 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. It can be specified whether the measurements start when the token enters the start place or when the token leaves it. The same can be specified for the end place. Each probe references a subset of the colors defined in the QPN. A probe only collects data for tokens of the referenced colors.

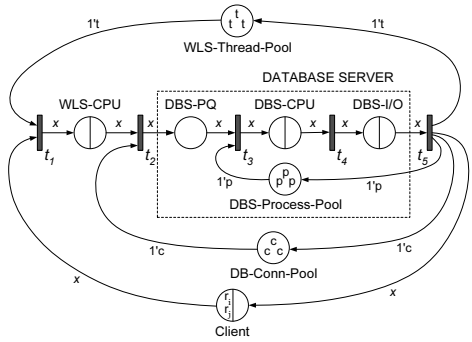


Figure 3.6: QPN Model of a Java EE System [13]

Currently, probes allow to gather statistics for the residence time of a token in a region of interest. For example, in the model shown in Figure 3.6, a probe

can be used to measure the time spent at the database server, which consists of places `DBS-PQ`, `DBS-CPU` and `DBS-I/O`. In this case, the probe starts at place `DBS-PQ` (on entry) and ends at place `DBS-I/O` (on exit). For each transaction of type i for which data should be collected, a reference to color 'r_i' is defined in the probe. As a result, the user is provided with the mean residence time of requests in the database server including the associated confidence interval and distribution.

The probes are realized by attaching timestamps to individual tokens. In the start place a probe adds the current simulation time as a timestamp to all tokens of colors it is interested in. A token can carry timestamps from different probes. Thus intersecting regions of several probes in a QPN are supported. Firing transitions collect all timestamps from input tokens and copy the timestamps to the output tokens. For each output token only the timestamps of probes interested in the token color are passed on. In some models, e.g. with a synchronous fork/join, it is possible that a transition gets tokens with different timestamps from the same probe. In this case, a warning is issued and only the minimal timestamp is passed on. The other timestamps are discarded. In the end place of a probe, its timestamp is removed and its statistics are updated.

Probes in QPE are defined using the "Probe Editor" shown in Figure 3.7. The "Probe Editor" is opened by selecting the "Probes" tab at the bottom of

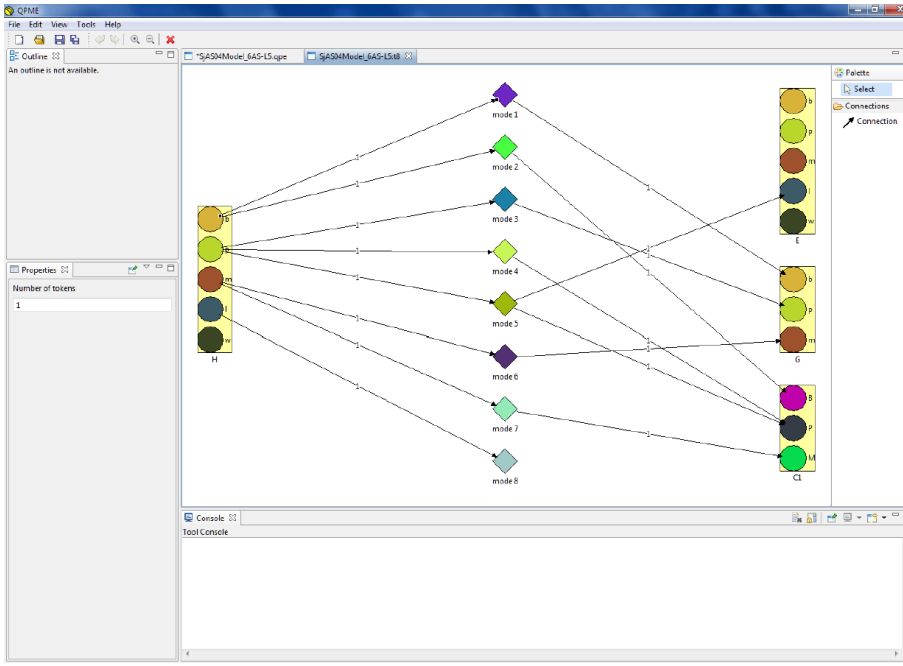


Figure 3.7: Probe Editor

the "Main Editor View". The "Probe Editor" consists of a table showing the currently defined probes and two buttons at the bottom of the table for adding and deleting probes. The delete button is only enabled when a probe is selected. Each probe has five attributes - "Name", "Start Place", "Start Trigger", "End Place" and "End Trigger". These attributes can be edited by clicking inside the table. The "Name" attribute provides a unique identifier of each probe. The "Start Place" attribute determines at which place the data collection of the probe starts. The "Start Trigger" attribute determines whether the data collection starts when a token enters ("On Entry") or leaves ("On Exit") the start place. The "End Place" attribute correspondingly determines at which place the data collection of the probe stops. Similarly, the "End Trigger" specifies whether the data collection stops when the token enters ("On Entry") or leaves ("On Exit") the end place. When a probe is selected, the colors for which a probe collects data can be selected using the "Properties View".

Behavior of Copy & Paste in QPE

The implementation of the standard "Copy" and "Paste" operations might seem obvious in most editors, however, their implementation is more complicated in the case of QPE. This is because elements in QPNs are interdependent and copying an element from one location to another might not make sense without adjusting the element or copying its associated elements along with it. There is a difference in how this is handled when an element is pasted inside the same document or when it is pasted into another document.

If an element is copied and pasted into the same document, a replica of the element is inserted next to source element with a little offset so that the user can distinguish between the two. Any connections of the copied element are replicated as well. If multiple elements are copied, any connections between them are replicated as connections between the replicas of the copied elements. If connections between a copied element and a non-copied element exist, a connection between the replica of the copied element and the non-copied element is created. When transitions are copied, the newly created replicas have identical incidence functions as the source transitions.

The behavior of "Copy" and "Paste" is slightly different when copying elements from one document to another. When a place is copied, it might be that its referenced colors are not defined in the target document. Therefore, any color definitions referenced by a copied element, have to be created in the target document. To avoid name conflicts, the names of copied colors are prefixed with the name of the source QPN model. Another difference is in the way connections are treated. Connections between copied elements and non-copied elements are not copied in the target document, since this does not make sense in this case. Therefore, a transition might lose some connections when copied and its incidence function has to be adjusted accordingly.

Chapter 4

Model Analysis using SimQPN

4.1 Overview

QPME provides a discrete-event simulator, SimQPN, that can be used to analyze QPN models built in QPE. SimQPN is extremely light-weight and has been implemented in Java to provide maximum portability and platform-independence. It 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 for which Java Runtime Environment (JRE) 1.6 or higher 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 simulates QPNs based on the event-scheduling approach to 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.

In this chapter, we present SimQPN from the user's perspective. For information on SimQPN's internal implementation details as well as precise specification of the analysis techniques it supports, we refer the reader to [9, 14]. It should be noted that SimQPN currently supports most but not all of the QPN features that can be configured in QPE. The reason for not limiting QPE to only those features supported by SimQPN is that QPE is intended to be usable as a standalone QPN editor and as such the QPN features it offers should not be limited to any particular analysis technique.

4.1.1 Supported QPN Features

In this section we list the main features SimQPN supports. 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.

Scheduling Strategies

SimQPN currently supports the following scheduling strategies for queues inside queueing places:

- First-Come-First-Served (FCFS)
- Processor-Sharing (PS)
- Infinite Server (IS)
- Random (RANDOM)
- Priority (PRIO)

Service Time Distributions

The following service time distributions are supported (input parameters of distributions are shown in brackets):

- Beta (alpha, beta)
- BreitWigner (mean, gamma, cut)
- BreitWignerMeanSquare (mean, gamma, cut)
- ChiSquare (freedom)
- Gamma (alpha, lambda)
- Hyperbolic (alpha, beta)
- Exponential (lambda)
- ExponentialPower (tau)
- Logarithmic (p)
- Normal (mean, stddev)
- StudentT (freedom)
- Uniform (min, max)
- VonMises (freedom)

- Empirical (scale, offset)
- Deterministic (value)

Empirical distributions are supported in the following way. The user is expected to provide a probability distribution function (PDF), specified as an array of positive real numbers (histogram). A cumulative distribution function (CDF) is constructed from the PDF and inverted using a binary search for the nearest bin boundary and a linear interpolation within the bin (resulting in a constant density within each bin). The next version of SimQPN will also include support for deterministic distributions.

Departure Disciplines

A novel feature of SimQPN is the introduction of the so-called *departure disciplines*. The latter are defined for ordinary places or depositories and determine the order in which arriving tokens become available for output transitions. Three departure disciplines are currently supported:

Normal (used by default) Implies that tokens become available for output transitions immediately upon arrival just like in conventional QPN models.

First-In-First-Out (FIFO) The FIFO discipline 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 the FIFIO feature can be exploited and the benefits it provides we refer the reader to [11, 12].

Random (RANDOM) The RANDOM departure discipline has one big queue, the randomly chosen token is the only one available. After the token has been picked, a new token is randomly chosen.

4.1.2 Simulation Output Data Analysis

Modes of Data Collection

SimQPN offers the ability to configure what data exactly to collect during the simulation and what statistics to provide at the end of the run. This can be specified for each place (ordinary or queueing) of the QPN. The user can choose between six modes of data collection (called `stats-levels`). 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, by configuring data collection modes, the user can speed up the simulation by making sure that no time is wasted collecting unnecessary data. Statistics in SimQPN are provided on a per *location* basis where location is defined to have one of the following four types:

1. Ordinary place.

2. Queue of a queueing place (considered from the perspective of the place).
3. Depository of a queueing place.
4. Queue (considered from the perspective of all places it is part of).

The six data collection modes (stats-levels) are defined as follows:

stats-level 0 In this mode no statistics are collected.

stats-level 1 This mode considers only token throughput data, i.e., for each location the token arrival and departure rates are estimated for each color.

stats-level 2 This mode adds token population, token occupancy and queue utilization data, i.e., for each location the following data is provided:

- Token occupancy (for locations of type 1 or 3): fraction of time in which there is a token inside the location.
- Queue utilization (for locations of type 2 or 4): proportion of the queue's server resources used by tokens arriving through the respective location.
- For each token color of the respective location:
  - Minimum/maximum number of tokens observed in the location.
  - Average number of tokens in the location.
  - Token color occupancy: fraction of time in which there is a token of the respective color inside the location.

stats-level 3 This mode adds token residence time data, i.e., for each location 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.

stats-level 4 This mode adds a histogram of observed token residence times.

stats-level 5 This mode additionally dumps token residence times to a file for further analysis.

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 *method of independent replications* (in its variant referred to as *replication/deletion approach*) and the *method of non-overlapping batch means*. Both of them can be used to provide point and interval estimates of the steady state mean token residence time. The method of Welch is used for determining the length of the initial transient (warm-up period). For users that would like to use different methods for steady state analysis (for example ASAP [18, 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 (for example [8]).

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. 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 [17].

4.2 Working with SimQPN

4.2.1 Run Configuration Wizard

SimQPN can be launched by choosing "SimQPN" from the "Tools" menu in QPE. This opens the "Run Configuration Wizard". The latter consists of three dialog windows:

1. Select Run Configuration
2. Simulation Run Configuration
3. Configuration Parameters for the chosen Analysis Method

Before a QPN model can be simulated, a "configuration" must be created which encapsulates all input parameters required for the simulation. The "Select Run Configuration" dialog window (Figure 4.1) can be used to create new configurations or delete existing ones. All parameters belonging to a configuration are stored as meta-attributes in the model's XML file.

When creating a new configuration, the user is first asked to select the analysis method that will be used for analysis of the output data from the simulation. Three analysis methods are currently supported:

1. **Batch Means:** Steady-state analysis using the method of non-overlapping batch means.

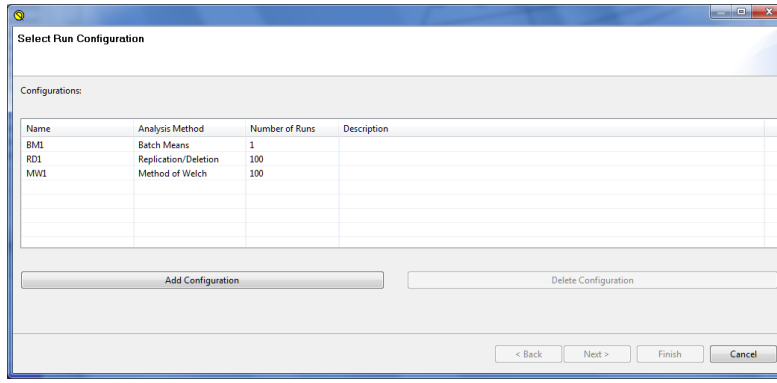


Figure 4.1: Select Run Configuration Dialog Window

2. **Replication/Deletion:** Steady-state analysis using the method of independent replications in its variant referred to as replication/deletion approach.

3. **Method of Welch:** Analysis of the length of the initial transient (warm-up period) using the method of Welch.

Steady-state analysis is applied to the observed token residence times at places, queues and depositories of the QPN.

General Run Configuration Parameters

After a configuration has been created it can be used by selecting it and clicking on the "Next" button in the "Select Run Configuration" dialog window. This opens the "Simulation Run Configuration" dialog window (Figure 4.2) which allows the user to configure the following general simulation parameters:

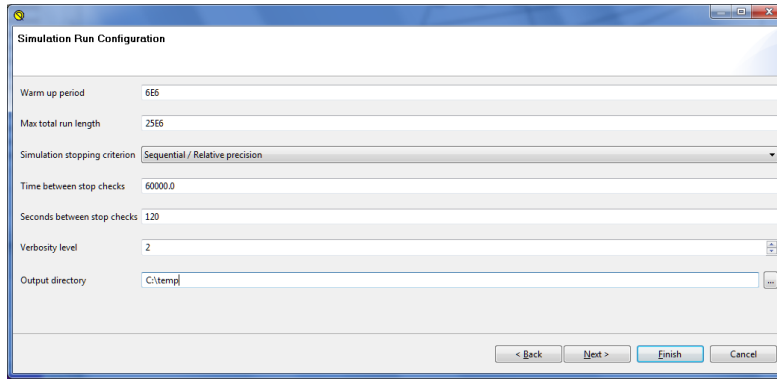


Figure 4.2: Simulation Run Configuration Dialog Window

Warm up period: Length of the warm up period (initial transient) of the simulation run (in model time).

Max total run length: Maximum total length of the simulation run including the warm up period (in model time).

Simulation stopping criterion: Criterion for determining when the simulation run should be stopped. Three values are allowed:

- Fixed sample size
- Sequential / Absolute precision
- Sequential / Relative precision

"Fixed sample size" means that the simulation is run until the "max total run length" has been reached. "Sequential / Absolute precision" or "Sequential / Relative precision" means that the length of the simulation is increased sequentially from one checkpoint to the next, until enough data has been collected to provide estimates of residence times with a given user-specified precision. The precision is defined as an upper bound on the confidence interval half length. It can be specified either as an absolute value ("Sequential / Absolute precision") or as a percentage relative to the mean residence time ("Sequential / Relative precision"). Note that if the "Replication/Deletion" method or the "Method of Welch" has been chosen, the stopping criterion is automatically set to "fixed sample size" because the sequential stopping criteria are not applicable to these methods.

Time between stop checks: Specifies how often (in model time) the simulator should check if the conditions of the stopping criterion have been fulfilled to determine if the simulation run should be stopped.

Seconds between stop checks: Used only when "time between stop checks" is set to 0. In this case, "time between stop checks" is automatically adjusted to correspond roughly to the configured "seconds between stop checks".

Verbosity level: Specifies the amount of details about the progress of the simulation that should be provided during the run. Verbosity level is an integer from 0 to 3.

Output directory: Directory in which the simulation results and auxiliary output files (e.g., raw data) should be stored.

After the user has finished configuring the parameters in the "Simulation Run Configuration" dialog window and clicks on the "Next" button, the next dialog window depends on the chosen analysis method.

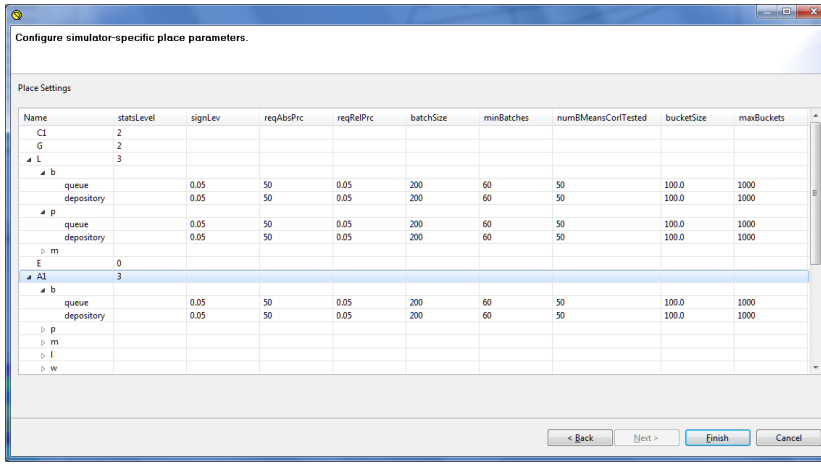


Figure 4.3: Configuration Parameters for Batch Means Method

Configuration Parameters for Batch Means Method

Figure 4.3 shows the dialog window for the batch means method. The following parameters must be configured for every ordinary place, queue or depository:

statsLevel: Specifies the data collection mode - from 0 to 5 (see Section 4.1.2. If set to 0, no data is collected for the respective place and no statistics are provided at the end of the run.

signLev: Specifies the significance level of the confidence intervals to be provided for the mean token residence times.

reqAbsPrc: If "Sequential / Absolute precision" stopping criterion has been chosen, this field specifies the absolute precision required. Simulation is not stopped before enough data has been collected to provide confidence intervals for token residence times at the respective place with half widths not exceeding $reqAbsPrc$.

reqRelPrc: If "Sequential / Relative precision" stopping criterion has been chosen, this field specifies the relative precision required. Simulation is not stopped before enough data has been collected to provide confidence intervals for token residence times at the respective place with half widths not exceeding $(reqRelPrc * 100\%)$ percent of the corresponding mean values.

batchSize: Specifies the batch size used.

minBatches: Minimum number of batches required for steady state statistics. If set to 0, no steady state analysis is performed for the respective token color.

numBMeansCorlTested: If set greater than 0, the first *numBMeansCorlTested* batch means observed from the beginning of the steady state period are tested for autocorrelation to determine if the batch size is sufficient. If the test fails, the batch size is increased repeatedly until the test is passed. If set to 0, no autocorrelation test is performed.

bucketSize: The size of histogram buckets.

maxBuckets: Maximum number of buckets of the histogram.

The above parameters are specified on a per-color basis for every place of the QPN. For queueing places, the parameters are set separately for the queue and depository of the place. Note that the parameters "signLev", "reqAbsPrc", "reqRelPrc", "batchSize", "minBatches" and "numBMeansCorlTested" are only enabled for places where "statsLevel" is set to be greater than or equal to 3. Otherwise, no steady state analysis is performed and these parameters do not make sense. The parameters "bucketSize" and "maxBuckets" are only of interest for place where "statsLevel" is set to be greater than or equal to 4. Otherwise, no histograms are created.

Configuration Parameters for Replication/Deletion Method

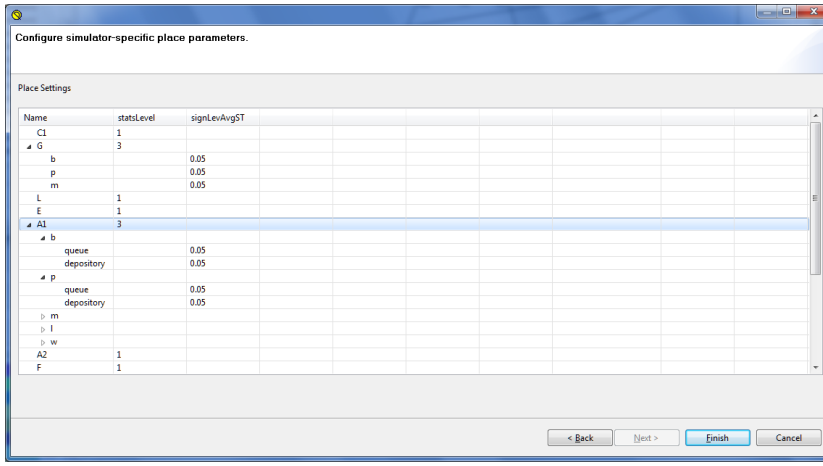


Figure 4.4: Configuration Parameters for Replication/Deletion Method

Figure 4.4 shows the dialog window for replication/deletion method. The following parameters must be configured for every ordinary place, queue or depository:

statsLevel: Specifies the data collection mode - from 0 to 5 (see Section 4.1.2. If set to 0, no data is collected for the respective place and no statistics are provided at the end of the run.

sighLevAvgST: Specifies the significance level of the confidence intervals to be provided for the mean token residence times.

Note that the parameter "sighLevAvgST" is only enabled for places where "statsLevel" is set to be greater than or equal to 3. Otherwise, no statistics are gathered for token residence times. The number of replications performed is specified in the "Select Run Configuration" dialog window (Figure 4.1).

Configuration Parameters for Method of Welch

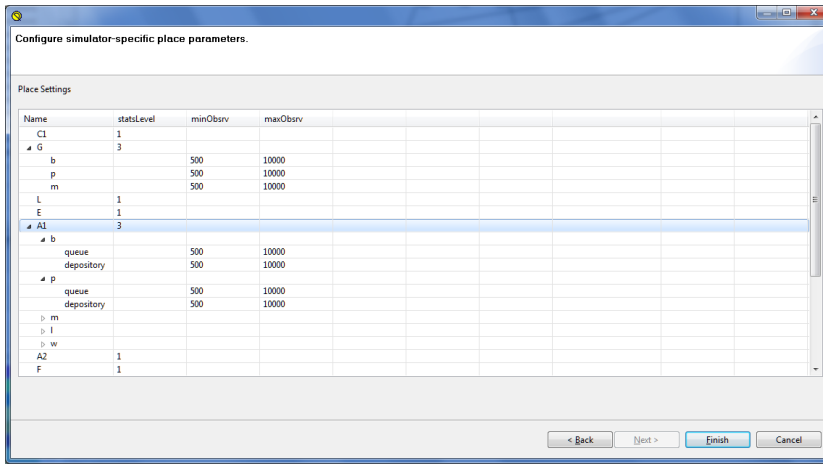


Figure 4.5: Configuration Parameters for Method of Welch

Figure 4.5 shows the dialog window for the method of Welch. The following parameters must be configured for every ordinary place, queue or depository:

statsLevel: Specifies the data collection mode - from 0 to 5 (see Section 4.1.2). If set to 0, no data is collected for the respective place and it is excluded from the analysis.

minObsrv: Minimum number of observations required.

maxObsrv: Maximum number of observations considered. If set to 0, no data is collected for the respective token color and it is excluded from the analysis.

Note that the parameters "minObsrv" and "maxObsrv" are only enabled for places where "statsLevel" is set to be greater than or equal to 3. Otherwise, no statistics are gathered for token residence times. The number of replications performed is specified in the "Select Run Configuration" dialog window (Figure 4.1).

For every token color, SimQPN computes the moving averages of observed token residence times for four different window sizes and stores them in text files

in the "output directory". Output files are named as follows:

```
WelchMovAvgST-<TYPE><NAME>-col<COLOR>-win<SIZE>.txt
```

where `<TYPE>` is `place`, `queue` or `depository`; `<NAME>` is the name of the respective place, queue or depository; and `<SIZE>` is the window size. The window sizes considered are $m/4$, $m/16$, $m/32$ and $m/64$, where m is the actual number of observations.

4.2.2 SimQPN Command-Line Interface

As mentioned earlier, SimQPN can also be run as a standalone Java application outside of QPE. This is done using a shell script, `SimQPN.bat` on Windows or `SimQPN.sh` on Unix/Linux platforms.

On Windows, the script is started as follows:

```
SimQPN.bat [-l] [-r "config"] qpe-file
```

where the command line parameters are interpreted as explained below:

`-l` tells SimQPN to list the simulation configurations defined in the QPE file.

`qpe-file` is the QPE file containing the model to be analyzed.

`-r` tells SimQPN to run the specified simulation configuration.

`config` is the simulation configuration to be run.

On Unix/Linux platforms exactly the same syntax is used with the only difference that the name of the script is `SimQPN.sh`.

4.3 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 which is stored in the configured output directory. In addition, a summary of the results in text format is printed on the console and stored in a separate file with a `.log` extension.

QPE provides an advanced query engine for processing and visualization of the 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 two editors that can be used as a front-end to the query engine: "Simple Query Editor" and "Advanced Query Editor".

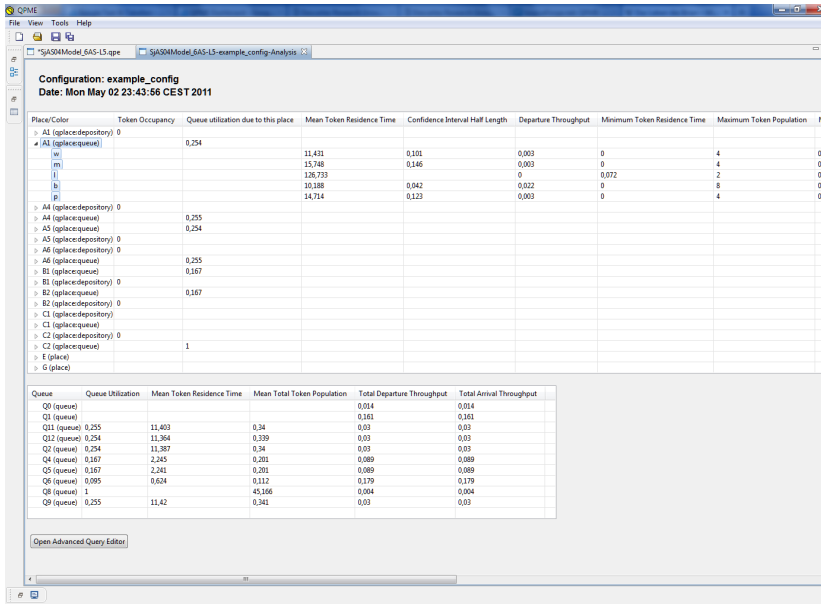


Figure 4.6: Basic Query Editor

4.3.1 Simple Query Editor

The "Simple Query Editor", shown in Figure 4.6, is displayed when opening the `.simqpn` file containing the results from the simulation. The editor displays the collected statistics for the various places and queues of the QPN. Statistics are reported on a per *location* basis where location is defined as in Sect. 4.1.2). The four location types are denoted as follows:

1. "place" - ordinary place.
2. "qplace:queue" - queue of a queueing place considered from the perspective of the place.
3. "qplace:depository" - depository of a queueing place.
4. "queue" - queue considered from the perspective of all places it is part of.

The statistics for the various locations are presented in two tables. The first table contains the statistics for locations of type "place", "qplace:queue" and "qplace:depository", while the second one contains the statistics for locations of type "queue". Depending on the configured data collection modes (see Sect. 4.1.2), the set of available performance metrics for the various locations may vary.

By clicking on multiple locations while holding "Ctrl", the user can select a set of locations and respective token colors. A right click on a selection opens

the context menu (see Figure 4.7) in which the user can choose which metric should be visualized for the selected set of locations and token colors. After choosing a metric, the user can select the form in which the results should be presented. Currently, three options are available: "Pie Chart", "Bar Chart" and "Console Output". Figure 4.8 shows an example of a pie chart and bar chart for the metric mean token residence time.

The "Simple Query Editor" is intended for simple filtering and visualization of the simulation results and does not provide any means to aggregate metrics over multiple locations and token colors. Queries involving aggregation are supported by the "Advanced Query Editor".

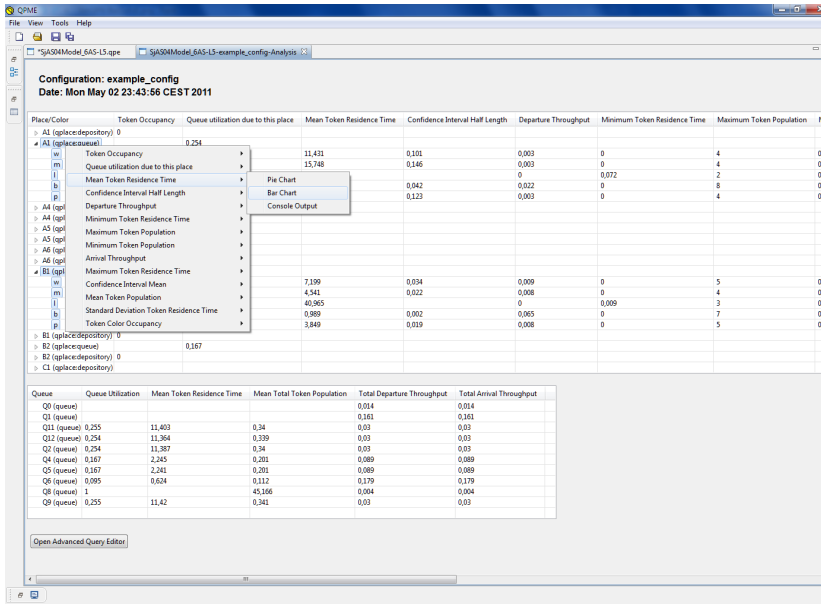
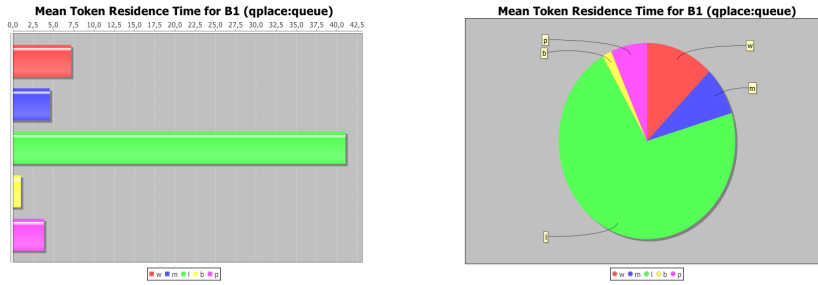


Figure 4.7: Context Menu in Basic Query Editor

4.3.2 Advanced Query Editor

The "Advanced Query Editor" is opened by clicking on the respective button at the bottom of the "Simple Query Editor". Using this editor the user can define complex queries on the simulation results involving both filtering and aggregation of performance metrics from multiple places and queues of the QPN. An example of such a query is shown in Figure 4.9.

A query is defined by first selecting a set of locations and a set of colors using the combo boxes and the +/- buttons at the top of the editor. The selected locations and colors specify a filter on the data that should be considered as part of the query. Using the table at the bottom of the editor, the user can select the specific performance metrics of interest and how data should be aggregated with



(a) Bar Chart (b) Pie Chart

Figure 4.8: Example Diagrams

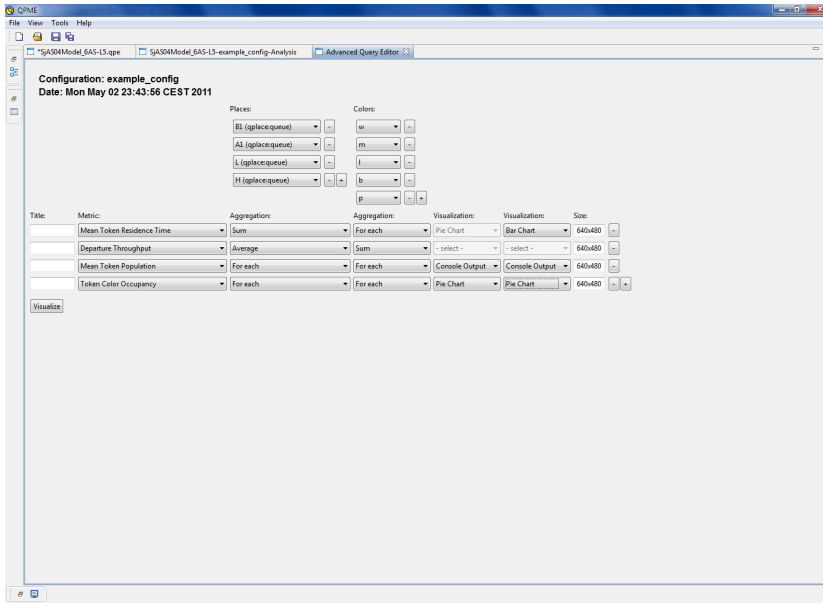
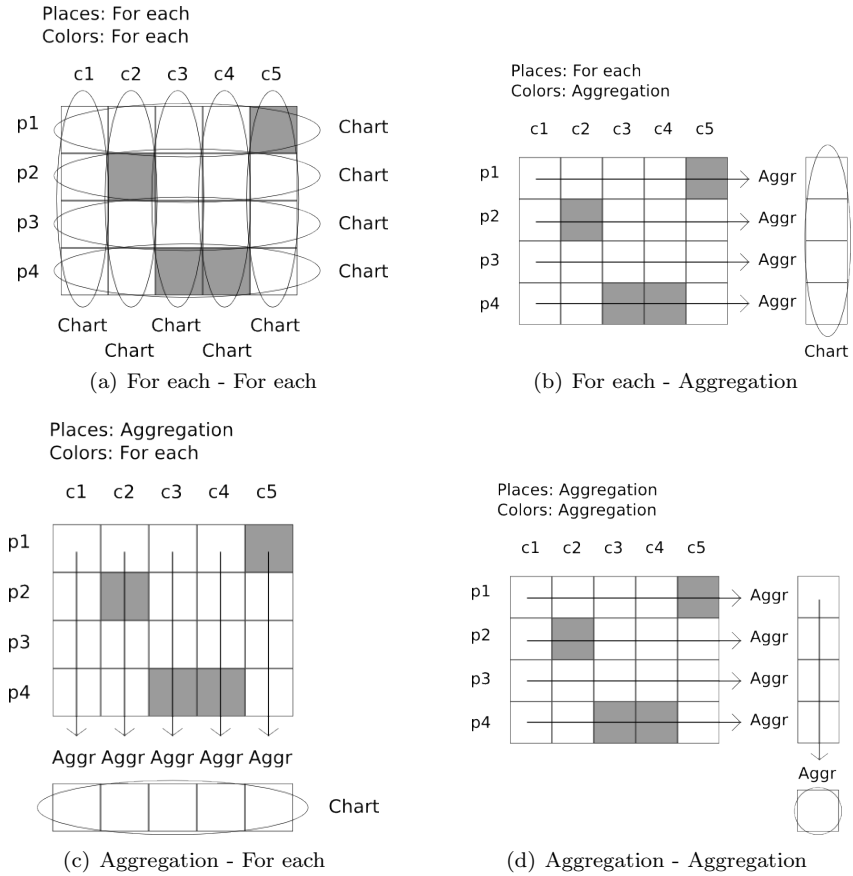


Figure 4.9: Advanced Query Editor

respect to the considered locations and colors. Three options for aggregating data are available:

- "For each" - no aggregation is applied and performance metrics are considered separately for each location/color.
- "Average" - the average over the selected locations/colors is computed.
- "Sum" - the sum over the selected locations/colors is computed.



(a) For each - For each

(b) For each - Aggregation

(c) Aggregation - For each

(d) Aggregation - Aggregation

Figure 4.10: Aggregation Scenarios

Two "Aggregation" fields are available, the left one is applied to the set of locations, while the right one is applied to the set of colors. Similarly, two "Visualization" fields are available, one applied to the set of locations, the other one to the set of colors. QPE currently offers three visualization options: "Bar Chart", "Pie Chart" and "Console Output".

Depending on the selected aggregation options, there are four possible scenarios: a) no aggregation, b) aggregation over colors, c) aggregation over locations, d) aggregation over both colors and locations. The four scenarios are depicted in Figure 4.10 illustrating how performance metrics are aggregated and used to produce a set of charts capturing the results from the respective query. Assuming that the user has selected a set of locations $p_1, p_2, ..., p_m$ and a set of colors $c_1, c_2, ..., c_n$, a matrix is generated that contains the values of the selected performance metric for each combination of location and color. Some of the cells of

the matrix could be empty (denoted in grey in Figure 4.10). This could happen if the metric is not available in the configured data collection mode or if the considered color is not defined for the respective location. The number of charts generated depends on the selected aggregation options. In case no aggregation is selected, $m + n$ charts are generated. In the case of aggregation over the set of colors or locations, one chart is generated. Finally, in the case of aggregation over both colors and locations, the result of the query is a single value.

4.3.3 Format of SimQPN Console Output

As mentioned in Section 4.3, after a successful simulation run, in addition to saving the simulation results in an XML file (`.simqpn`), SimQPN prints a summary of the results on the console and stores it in a separate `.log` file. In this section, the format of the produced results summary is presented in detail.

Results from Batch Means Method

The excerpt below shows the format of results from the method of batch means for one queueing place (queue and depository) and one color.

```
REPORT for Queue of Queueing Place : WLS-CPU---------------------

queueUtilQPl=1.0

------------------ Color = x1 --------------------
arrivCnt=80934 deptCnt=80935
arrivThrPut=0.0141931190313492 deptThrPut=0.014193294397932
meanTkPop=56.817972595483404 tkColOcp=1.0
-----
meanST=4002.072502543596 stDevST=3997.8883004676186

Steady State Statistics:
numBatchesST=202 batchSizeST=400 stDevStdStateMeanST=234.045554
95% c.i. = 4002.120611373594 +/- 32.27548690360934


REPORT for Depository of Queueing Place : WLS-CPU----------------

tkOcp=0.0

------------------ Color = x1 --------------------
arrivCnt=80935 deptCnt=80935
arrivThrPut=0.0141932943979323 deptThrPut=0.0141932943979323
meanTkPop=0.0 tkColOcp=0.0
-----
```

```
meanST=0.0 stDevST=0.0

Steady State Statistics:
numBatchesST=404 batchSizeST=200 stDevStdStateMeanST=0.0
95% c.i. = 0.0 +/- 0.0
```

The various quantities in the results report are defined as follows:

queueUtilQPl: Utilization of the underlying queue due to this place, i.e., proportion of the queue's server resources used by tokens arriving through this place.

tkOcp: Token occupancy of the depository, i.e., fraction of time in which there is a token inside the depository.

arrivCnt: Total number of tokens of the respective color that arrived in the queue/depository during the run.

deptCnt: Total number of tokens of the respective color that departed from the queue/depository during the run.

arrivThrPut: Rate at which tokens of the respective color arrive at the queue/depository.

deptThrPut: Rate at which tokens of the respective color depart from the queue/depository.

meanTkPop: Mean number of tokens of the respective color in the queue/depository.

tkColOcp: The probability that there is a token of the respective color in the queue/depository.

meanST: Mean token residence (sojourn) time, i.e., time that tokens of the respective color spend in the queue/depository.

stDevST: Standard deviation of the token residence time.

numBatchesST: Number of batches of observations collected.

batchSizeST: Batch size used.

stDevStdStateMeanST: Standard deviation of the steady state residence time.

90% c.i.: 90% confidence interval for the steady state mean residence time.

The following excerpt shows the aggregate results for the underlying queue of the respective queueing place.

```
REPORT for Queue : Q1----------------------------------------

totArrivThrPut=0.01419311903134 totDeptThrPut=0.014193294397
meanTotTkPop=56.817972595483404 queueUtil=1.0
meanST=4002.072502543596
```

The various quantities in the results report are defined as follows:

totArrivThrPut: Total arrival throughput over all queueing places this queue is part of.

totDeptThrPut: Total departure throughput over all queueing places this queue is part of.

meanTotTkPop: Mean queue total token population.

queueUtil: Utilization of the queue, i.e., fraction of the available server resources that are used on average.

meanST: Mean token residence (sojourn) time over all tokens visiting this queue.

Results from Replication/Deletion Method

The excerpt below shows the format of results from the replication/deletion method for one queueing place (queue and depository) and one color.

```
REPORT for Queue : DBS-CPU--------------------------------------

numReplicationsUsed = 100 numTooShortRepls = 0
minRunLen=5000000.047045088 maxRunLen=5000175.44340017
avgRunLen=5000020.540000993 stDevRunLen=25.94565026505922
avgWallClockTime=1.18217999999 stDevWallClockTime=0.030668768043

meanQueueUtil=0.7574721018056024 stDevQueueUtil=0.0046913938556502

------------------ Color=0 --------------------
meanArrivThrPut[c]=0.0142910684137 meanDeptThrPut[c]=0.01429092841
stDevArrivThrPut[c]=6.38614705E-5 stDevDeptThrPut[c]=6.3797896E-5
minAvgTkPop[c]=2.876744782905197 maxAvgTkPop[c]=3.4270894141218826
meanAvgTkPop[c]=3.118214443226206 meanColUtil[c]=0.757472101805624
stDevAvgTkPop[c]=0.10659712560 stDevColUtil[c]=0.00469139385565026
-----
meanAvgST[c]=218.18885562939914 stDevAvgST[c]=7.15056639668919
```

```
90% c.i. = 218.18885562939914 +/- 1.1872797998046334


REPORT for Depository : DBS-CPU----------------------------------

numReplicationsUsed = 100 numTooShortRepls = 0
minRunLen=5000000.047045088 maxRunLen=5000175.44340017
avgRunLen=5000020.540000993 stDevRunLen=25.94565026505922
avgWallClockTime=1.1821799999999 stDevWallClockTime=0.030668768043

------------------ Color=0 --------------------
meanArrivThrPut[c]=0.0142909284 meanDeptThrPut[c]=0.01429093507
stDevArrivThrPut[c]=6.379789E-5 stDevDeptThrPut[c]=6.376607356E-5
minAvgTkPop[c]=0.0 maxAvgTkPop[c]=0.0
meanAvgTkPop[c]=0.0 meanColUtil[c]=0.0
stDevAvgTkPop[c]=0.0 stDevColUtil[c]=0.0
-----
meanAvgST[c]=0.0 stDevAvgST[c]=0.0

90% c.i. = 0.0 +/- 0.0
```

The various quantities in the results report are defined as follows:

numReplicationsUsed: Total number of run replications used for steady state analysis.

numTooShortRepls: This variable is currently not used, so it can be ignored.

minRunLen: The minimum length of a run replication (in model time).

maxRunLen: The maximum length of a run replication (in model time).

avgRunLen: The average length of a run replication (in model time).

stDevRunLen: The standard deviation of the run replication length (in model time).

avgWallClockTime: The average duration of a run replication (in wall clock time).

stDevWallClockTime: The standard deviation of the run replication duration (in wall clock time).

meanQueueUtil: The mean queue utilization - probability that there is a token of any color in the queue.

stDevQueueUtil: Standard deviation of the queue utilization measured from the run replications.

meanArrivThrPut: Mean rate at which tokens of the respective color arrive at the queue/depository (arrival rate).

meanDeptThrPut: Mean rate at which tokens of the respective color depart from the queue/depository (departure rate).

stDevArrivThrPut: Standard deviation of the token arrival rate.

stDevDeptThrPut: Standard deviation of the token departure rate.

minAvgTkPop: Minimum average token population measured from the run replications.

maxAvgTkPop: Maximum average token population measured from the run replications.

meanAvgTkPop: Mean average token population measured from the run replications.

meanColUtil: Mean probability that there is a token of the respective color in the queue/depository.

stDevAvgTkPop: Standard deviation of the average token population.

stDevColUtil: Standard deviation of the probability that there is a token of the respective color in the queue/depository.

meanAvgST: Mean of the average residence times measured form the run replications.

stDevAvgST: Standard deviation of the residence times measured form the run replications.

90% c.i.: 90% confidence interval of the mean residence time.

Bibliography

[1] F. Bause. "QN + PN = QPN" - Combining Queueing Networks and Petri Nets. Technical report no.461, Department of CS, University of Dortmund, Germany, 1993. 7

[2] F. Bause. Queueing Petri Nets - A formalism for the combined qualitative and quantitative analysis of systems. In *Proceedings of the 5th International Workshop on Petri Nets and Performance Models, Toulouse, France, October 19-22*, 1993. 3, 6, 7

[3] F. Bause and F. Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory.* Vieweg Verlag, second edition, 2002. 3, 4, 5, 6, 7

[4] F. Bause, P. Buchholz, and P. Kemper. Hierarchically Combined Queueing Petri Nets. In *Proceedings of the 11th International Conference on Analysis and Optimization of Systems, Discrete Event Systems, Sophie-Antipolis (France)*, 1994. 8

[5] F. Bause, P. Buchholz, and P. Kemper. Integrating Software and Hardware Performance Models Using Hierarchical Queueing Petri Nets. In *Proceedings of the 9. ITG / GI - Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, (MMB'97), Freiberg (Germany)*, 1997. 7, 12

[6] 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 *Proceedings of the 24th International Conference on Application and Theory of Petri Nets, June 23-27, Eindhoven, Holland*, June 2003. 11

[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/. 29

[8] Department of Industrial Engineering, North Carolina State University. ASAP3 Software For Steady-State Simulation Output Analysis, 2003. ftp://ftp.ncsu.edu/pub/eos/pub/jwilson/installasap3.exe. 29

[9] C. Dutz. QPE - A Graphical Editor for Modeling using Queueing Petri Nets. Master thesis, Technical University of Darmstadt, Apr. 2006. 1, 12, 25

[10] K. Jensen. *Coloured Petri Nets and the Invariant Method.* Mathematical Foundations on Computer Science, Lecture Notes in Computer Science 118:327-338, 1981. 4

[11] S. Kounev. *Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction.* Shaker Verlag, Dec. 2005. ISBN 3832247130. ISBN: 3832247130. 3, 16, 18, 27

[12] 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. doi:10.1109/TSE.2006.69. 3, 16, 18, 27

[13] 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 - ISPASS2003, Austin, Texas, USA, March 20-22*, 2003. 21

[14] 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. doi:10.1016/j.peva.2005.03.004. 1, 25

[15] S. Kounev and C. Dutz. QPME - A Performance Modeling Tool Based on Queueing Petri Nets. *ACM SIGMETRICS Performance Evaluation Review (PER), Special Issue on Tools for Computer Performa nce Modeling and Reliability Analysis*, 36(4):46–51, March 2009. 1

[16] S. Kounev, C. Dutz, and A. Buchmann. QPME - Queueing Petri Net Modeling Environment. In *Proceedings of the 3rd International Conference on Quantitative Evaluation of SysTems (QEST-2006), Riverside, CA, September 11-14*, 2006. 1

[17] M. Matsumoto and T. Nishimura. Mersenne Twister: A 623-Dimensionally Equidistributed Uniform Pseudo-Random Number Generator. *ACM Transactions on Modeling and Computer Simulation*, 1998. 29

[18] N. Steiger and J. Wilson. Experimental Performance Evaluation of Batch Means Procedures for Simulation Output Analysis. In *Proceedings of the 2000 Winter Simulation Conference, Orlando, FL, USA, December 10-13*, 2000. 29

[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. ftp://ftp.ncsu.edu/pub/eos/pub/jwilson/tomacsv37.pdf. 29

[20] The Eclipse Foundation. Graphical Editing Framework (GEF), 2006. http://www.eclipse.org/gef/. 11