Universität Karlsruhe (TH)
Forschungsuniversität · gegründet 1825

Filterung, Aggregierung und Visualisierung von QPN Analyseergebnissen

Studienarbeit am
Institut für Programmstrukturen und Datenorganisation
Lehrstuhl Software-Entwurf und -Qualität
Prof. Dr. R. Reussner
Fakultät für Informatik
Universität Karlsruhe (TH)

von

cand. inform.
Frederik Zipp

Betreuer:
Prof. Dr. R. Reussner
Dr.-Ing. Samuel Kounev

Tag der Anmeldung:	18. Februar 2009
Tag der Abgabe:	18. Mai 2009

Ich erkläre hiermit, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

Karlsruhe, den 18. Mai 2009

QPME (engl., Queueing Petrinet Modeling Environment) ist ein Modellierungswerkzeug zur Performanzvorhersage verteilter Systeme mittels auf Queueing-Petri-Netzen (QPN) basierender Performanz-Modelle. Im Rahmen dieser Studienarbeit wurde QPME um eine flexible Visualisierung der Simulationsergebnisse für den Anwender erweitert. Bisher wurden die Simulationsergebnisse in einer unübersichtlichen Textform präsentiert und waren daher nur schwer auszuwerten und zu interpretieren. Dem Anwender wird nun über die grafische Benutzeroberfläche des Werkzeugs ermöglicht, komplexe Anfragen mittels verschiedener Kriterien an das System zu formulieren, nach denen die Analyse-Daten der Simulation gefiltert, aggregiert und danach visuell aufbereitet werden, um so eine bessere Übersicht über die Daten zu bekommen und die Auswertung zu erleichtern.

Inhaltsverzeichnis

1. Einleitung

Ein aktueller Trend bei der Entwicklung betrieblicher Anwendungssysteme ist die Verwendung von Performanz-Modellen und entsprechenden Modellierungswerkzeugen, die es ermöglichen die Performanz und Skalierbarkeit des Systems zur Entwurfszeit vorherzusagen. QPME (engl., Queueing Petrinet Modeling Environment) [Koun08] ist ein Modellierungswerkzeug zur Performanzvorhersage verteilter Systeme mittels auf Queueing-Petri-Netzen (QPN) basierender Performanz-Modelle. Diese Modelle stellen eine Kombination von klassischen Warteschlangennetzen und stochastischen Petrinetzen dar, die eine höhere Ausdrucksstärke bietet. Das QPME-Werkzeug besteht aus zwei Teilen: i) einer auf Eclipse basierenden grafischen Oberfläche zum Erstellen von QPN-Modellen und ii) einem Simulator zur Analyse der Modelle. QPME ist inzwischen unter mehr als 80 Forschungseinrichtungen weltweit verbreitet worden und findet vor allem Anwendung im Bereich Performance Engineering [NKJT09], [KSBB08], [NoKT07], [NoKo07], [KoNT07b], [KoBu07], [KoNT07a], [Koun06], [KoBu06], [KoBu03], [Heim07].

1.1 Zielsetzung der Arbeit

Die Hauptaufgabe der hier vorgestellten Studienarbeit ist die Erweiterung von QPME um eine reichhaltige Präsentation der Simulationsergebnisse für den Anwender. Bisher wurden die Simulationsergebnisse in einer unübersichtlichen Textform präsentiert und waren daher nur schwer auszuwerten und zu interpretieren. Dem Anwender soll nun über die grafische Benutzeroberfläche des Werkzeugs ermöglicht werden, komplexe Anfragen mittels verschiedener Kriterien an das System zu formulieren, nach denen die Analyse-Daten der Simulation gefiltert, aggregiert und danach visuell aufbereitet werden, um so eine bessere Übersicht über die Daten zu bekommen und die Auswertung zu erleichtern.

Der Umfang der vorliegenden Studienarbeit umfasst folgende konkreten Ziele:

- Portierung von QPME auf die neueste Eclipse-Version
- Zentrale Verwaltung von Warteschlangen

- Geeignete Speicherung der Simulationsergebnisse
- Erarbeitung und Implementierung von geeigneten Benutzeroberflächen zur Formulierung von Visualisierungsanfragen
- Aggregation und Visualisierung der Daten
- Integration mit der Statistiksoftware R
- Validierung
- Dokumentation

1.2 Gliederung der Arbeit

In dem ersten Abschnitt wird zunächst *QPME* vorgestellt, das Modellierungs- und Simulations-Werkzeug für Queueing-Petri-Netze, welches Basis dieser Studienarbeit ist. Dabei wird sowohl auf die theoretischen Grundlagen von Queueing-Petri-Netzen eingegangen als auch auf die Möglichkeiten, die QPME zur Modellierung und Simulation dieser Netze bietet. Desweiteren wird die *Eclipse Rich Client Platform*, das Java-Anwendungsframework auf dem QPME basiert, vorgestellt sowie *JFreeChart*, eine Java-Bibliothek zur Darstellung von Diagrammen, und die Statistiksoftware *R*.

Im darauf folgenden Abschnitt werden die Anforderungen beschrieben, welche die neuen Funktionen, um die QPME im Rahmen dieser Studienarbeit erweitert wurde, erfüllen sollen. Kern davon sind die Anforderungen an die Speicherung der Simulationsergebnisse, an die Visualisierung der dieser Daten und an die Benutzeroberfläche mit der die entsprechenden Visualisierungsanfragen durch einen Anwender erstellt werden können.

Danach werden der Entwurf uns die Implementierung beschrieben, mit der diese Anforderungen erzielt werden sollen. Dabei wird insbesondere auf die Vorgehensweise zur Aggregierung der Daten eingegangen.

Im letzten Teil wird beschrieben mit welcher Methode überprüft wurde, dass durch die Weiterentwicklung von QPME keine Regressionen entstanden sind und auf welche Weise die neuen Funktionalitäten zur Aggregierung und Visualisierung validiert wurden.

2. Grundlagen

In diesem Kapitel werden die Grundlagen der Studienarbeit vorgestellt: *Queueing-Petri-Netze* und deren Einsatzgebiet sowie das Modellierungs- und Simulations-Werkzeug für Queueing-Petri-Netze Namens *QPME*, welches im Rahmen der Studienarbeit um die Möglichkeit der Filterung, Aggregierung und Visualisierung von Simulationsergebnissen erweitert werden soll.

2.1 Queueing-Petri-Netze

Ein *Petrinetz* ist ein bipartiter, gerichteter Graph, der aus sogenannten *Stellen* und *Transitionen* besteht, die über gerichtete Kanten miteinander verbunden sind, wobei auf eine Stelle nur eine Transition folgen darf und auf eine Transition nur eine Stelle.

Stellen können mehrere *Markierungen* besitzen. Ist in jeder Eingangs-Stelle einer Transition mindestens eine Markierung vorhanden, so kann diese Transition *schalten*. Das bedeutet, dass aus allen Eingangs-Stellen der Transition eine Markierung entfernt wird und an allen Ausgangs-Stellen eine Markierung hinzugefügt wird.

Definition 1 (Petri-Netz) *Ein einfaches Petri-Netz (PN) ist ein 5-Tupel* $PN = (P, T, I^-, I^+, M_0)$*, wobei gilt:*

1. $P = \{p_1, p_2, \ldots, p_n\}$ *ist eine endliche, nicht-leere Menge an Stellen*
2. $T = \{t_1, t_2, \ldots, t_n\}$ *ist eine endliche, nicht-leere Menge an Transitionen,* $P \cap T = \emptyset$
3. $I^-, I^+ : P \times T \to \mathbf{N}_0$ *sind die Inzidenzfunktionen*
4. $M_0 : P \to \mathbf{N}_0$ *ist die Anfangsbelegung der Markierungen*

Eine Erweiterung von Petri-Netzen sind *gefärbte* Petri-Netze, bei denen Markierungen mit einer Farbe versehen sind, um unterschiedliche Typen zu repräsentieren.

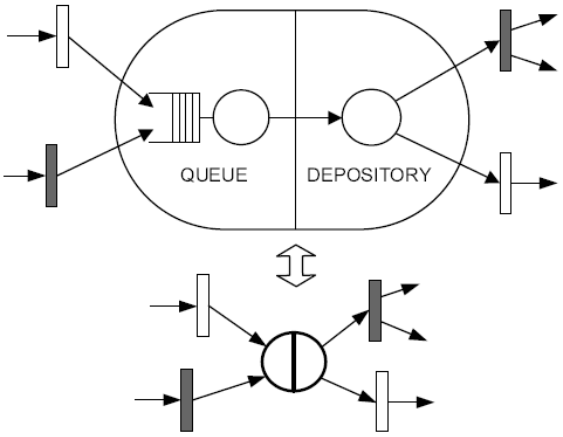

Abbildung 2.1: Eine Warteschlangen-Stelle und ihre verkürzte Darstellung

Hierbei können Transitionen mit unterschiedlichen *Modi* versehen werden, die beschreiben unter welcher Konfiguration der Eingangs-Stellen sie schalten und mit welchen Farben die Markierungen versehen werden sollen, die in die Ausgangs-Stellen der Transition gelangen. [KoDu07]

Eine weitere Form der Petri-Netze sind *Generalized Stochastic Petri Nets* (GSPNs). Bei diesen können Transitionen nicht nur direkt schalten, sondern auch verzögert, wobei die Verzögerung jeder Stelle durch eine Exponentialverteilung beschrieben wird. Wenn mehrere direkte Transitionen gleichzeitig schalten können, wird die Transition, die als nächstes schalten soll, anhand von den Transitionen zugewiesenen *Schaltgewichtungen* (Wahrscheinlichkeiten) ausgewählt. Eine Kombination aus gefärbten Petri-Netzen und GSPNs sind *gefärbte GSPNs*.

Queueing-Petrinetze (QPN) sind wiederum eine Erweiterung von gefärbten GSPNs um sogenannte *Warteschlangen-Stellen*. Eine Warteschlangen-Stelle besteht aus zwei Teilen: der Warteschlange und dem *Depository* [KoBu07], [BaKr02], [Baus93]. Abbildung 2.1 zeigt die graphische Repräsentation eines Warteschlangen-Stelle sowie deren verkürzte Darstellung. Eine Markierung, die in eine solche Stelle gelangt, wird zunächst in die Warteschlange eingereiht, in der sie von einer bestimmten Anzahl von *Servern* nach einer festgelegten Scheduling-Strategie bedient wird, bevor sie in das Depository kommt, wo die Markierung den ausgehenden Transitionen zum Schalten bereitsteht. Mögliche Scheduling-Strategien sind beispielsweise *First-Come First-Served* (FCFS), *Processor-Sharing* (PS), *Infinite-Server* (IS), *Priority-Scheduling* (PRIO) oder *Random-Scheduling* (RANDOM).

Diese Queueing-Petrinetze sind besonders gut geeignet zur Modellierung von verteilten Systemen sowie zur Analyse deren Performanz und Skalierbarkeit. Hierbei können sowohl Hardware- als auch Software-Aspekte im gleichen Modell integriert werden [KoDB06], [KoDu09].

2.2 QPME

QPME (engl. Queueing Petrinet Modeling Environment) ist ein auf Eclipse RCP basiertes Modellierungs- und Simulations-Werkzeug für Queueing-Petrinetze. Es wurde an der TU Darmstadt entwickelt [Koun05], [KoBu06]. Das Werkzeug besteht aus

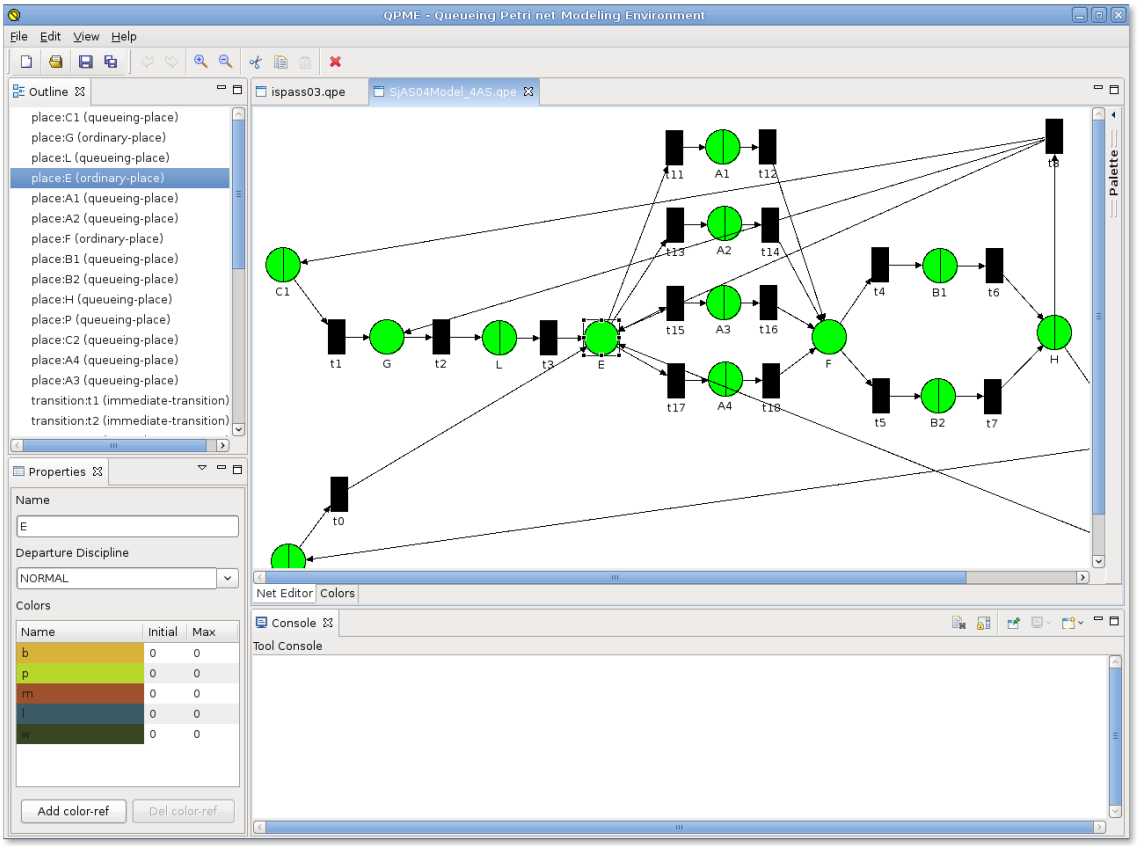

Abbildung 2.2: Der Netzeditor von QPME

zwei Teilen: einem grafischen Editor zur Modellierung der Queueing-Petrinetze und einem Simulator zur Analyse dieser Modelle.

Der grafische Editor bietet dem Anwender eine Sicht, in der er Stellen, Transitionen und Warteschlangen anlegen, platzieren und miteinander zu einem QPN verknüpfen kann. Jeder Stelle kann eine *departure discipline* zugeordnet werden. *NORMAL* oder *FIFO* Den Warteschlangen können zusätzlich noch unterschiedliche *Scheduling-Strategien* zugeordnet werden. Diese sind: *First-Come First-Served* (FCFS), *Processor-Sharing* (PS), *Infinite-Server* (IS), *Priority-Scheduling* (PRIO) und *Random-Scheduling* (RANDOM).

Die Visualisierung des Netzes wurde mit Eclipse GEF (Graphical Editing Framework) umgesetzt, einem Eclipse-basierten Framework zum Darstellen und Bearbeiten von Graphen und Diagrammen. GEF arbeitet nach dem Model-View-Controller-Prinzip (MVC) und bietet dem Entwickler vorgefertigte Komponenten wie beispielsweise eine Werkzeugpalette.

Neben dem Netzeditor bietet QPME auch einen Editor, in dem die möglichen Farben der Markierungen verwaltet werden. QPME verwaltet diese Farben zentral. Jeder Farbe kann ein Name zugeordnet werden, über den sie dann innerhalb des Netzes referenziert wird. Der Vorteil besteht darin, dass auf diese Weise Farben wiederverwendet werden können und nicht einzeln für jede Stelle neu definiert werden müssen.

Das Modell des Netzes wird in einem XML-Dokumentenbaum vorgehalten und als solches auch gespeichert. Das XML-Format speichert sowohl die logischen Beziehun-

gen zwischen den Stellen und Transitionen, aber auch das Layout des Netzes, also die Positionen der Knoten, welche in speziellen Meta-Tags enthalten sind.

Der Simulator von QPME, genannt SimQPN, kann aus QPME heraus oder von der Konsole aus aufgerufen werden und nimmt das XML-Dokument des QPN-Modells als Eingabe entgegen. SimQPN ist ein ereignisdiskreter Siulator und verwendet einen sequenziellen Algorithmus zur Simulation des QPNs. SimQPN nutzt das Wissen um die Struktur und das Verhalten von QPNs aus um die Effizienz der Simulation zu verbessern [KoBu06].

Für einen Simulationslauf muss zunächst eine Simulationis-Konfiguration für das zu simulierende Petrinetz erstellt werden. In dieser Konfiguration wird unter anderem die zu verwendende Simulations-Methode festgelegt. Mögliche Simulations-Methoden sind *Batch Means*, *Replication/Deletion* und *Method of Welch*. Letztere analysiert die *Initial-transient*-Periode (*Warm-Up*) des Netzes Desweiteren gibt der Anwender hier an, welche Stellen des Netzes beobachtet und analysiert werden sollen und in welchem Umfang dies erfolgt. Dazu kann für jede Stelle ein Level von 0 bis 5 angegeben werden, welcher den Detailgrad der Simulationsergebnisse bestimmt:

- Level 0: keine Statistiken
- Level 1: Durchsatz-Daten
- Level 2: zusätzlich Markierungs-Population, Markierungs-Belegung und Warteschlangenauslastung
- Level 3: zusätzlich Verweildauer von Markierungen an einer Stelle: Mittelwert, Standardabweichung und Konfidenzintervall
- Level 4: zusätzlich Histogramm der Verweildauern
- Level 5: Beobachtete Verweildauern (Rohdaten) werden zusätzlich zur weiteren Analyse in einer Datei geloggt

Die Simulations-Konfiguration eines Netzes wird ebenfalls in dessen XML-Datei in speziellen Meta-Tags festgehalten. Die Ergebnisse des Simulationslaufs wurden bisher auf der Konsole als reiner Text ausgegeben.

2.3 Eclipse RCP

Eclipse RCP (Rich Client Platform) [ecli] ist ein Open Source Java-Framework auf dessen Basis sich Anwendungen mit komplexer grafischer Benutzeroberfläche entwickeln lassen. Die bekannteste darauf basierende Anwendung ist die integrierte Java-Entwicklungsumgebung (IDE) Eclipse, aus der Eclipse RCP usrprünglich hervorgegangen ist. Eine Eclipse RCP basierte Anwendung ist um Plug-Ins erweiterbar. Das Framework steuert den Lebenszyklus einer Eclipse-Anwendung und bietet verschiedene Konzepte wie *Views*, *Editoren* und *Perspektiven*, die das Entwickeln einer Anwendung vereinfachen. Eine *View* ist beispielsweise ein kleines Fenster zur Darstellung eines bestimmten Aufgabenbereiches, das vom Benutzer über einen Tab aktiviert werden kann und sich per *Drag and Drop* plazieren lässt. Die Darstellung der GUI-Komponenten basiert auf dem Standard Widget Toolkit (SWT).

2.4 JFreeChart

JFreeChart [jfre] ist eine Open Source Java-Bibliothek zur Darstellung von Diagrammen und Schaubildern. Sie bietet dem Entwickler eine Fülle von unterschiedlichen Diagrammtypen wie etwa Balkendiagramme, Säulendiagramme, Tortendiagramme, Histogramme, etc. Diese Diagramme lassen sich in die grafische Benutzeroberfläche von Java-Programmen einbetten, aber auch in gängige Grafikformate wie PNG und JPEG exportieren.

2.5 Die Statistiksoftware R

R [rpro] ist eine unter freier Lizenz stehende Software für statistische Berechnungen und Grafiken. R wird im Rahmen des GNU-Projektes entwickelt und ist im naturwissenschaftlichen Bereich weit verbreitet. Die Software bietet eine eigene Programmiersprache und ist erweiterbar. Erweiterungen können aus einem Zentralen Archiv bezogen werden, dem *CRAN* (Comprehensive R Archive Network). So auch *rJava*, eine Anbindung von R an Java über das *Java Native Interface* (JNI).

3. Analyse

Im Rahmen der Studienarbeit sollen verschiedene Aspekte von QPME weiterentwickelt werden, wobei der Schwerpunkt auf einer benutzerfreundlichen und umfangreichen grafischen Darstellung der Simulationsergebnisse liegt. Dem Benutzer soll ermöglicht werden, komplexe Anfragen über unterschiedliche Kriterien zu erstellen, nach denen die Ergebnisse einer Simulation gefiltert, aggregiert und visuell aufbereitet werden.

Die grundlegenden Anforderungen, die erfüllt werden sollen, sind auf Basis der Vorbesprechungen mit dem Betreuer entstanden. Im Folgenden werden die einzelnen Arbeitspakete analysiert und deren Anforderungen und Ziele beschrieben.

3.1 Portierung auf neueste Eclipse-Version

Das QPME-Projekt wurde auf Basis von Eclipse RCP 3.1 entwickelt, welches im Jahr 2005 erschienen ist. Seitdem sind neuere Eclipse-Versionen erschienen, welche API-Inkompatibilitäten eingeführt haben. Die Studienarbeit soll als Gelegenheit genutzt werden um QPME auf den aktuellen Stand zu bringen. Daher soll zunächst sichergestellt werden, dass QPME mit der aktuellen Version von Eclipse RCP läuft und dies als Basis für die weitere Entwicklung verwendet werden. Die aktuelle Version zum Zeitpunkt der Studienarbeit sind Eclipse 3.4.1 Ganymede und GEF 3.4.1.

3.2 Zentrale Verwaltung von Warteschlangen

Die Eigenschaften für Warteschlangen müssen derzeit in QPME für jede Warteschlangen-Stelle einzeln definiert werden. QPME soll nun um einen Editor erweitert werden, mit dem die Warteschlangen eines Netzes und deren Eigenschaften wie Name, Anzahl der Server und Scheduling-Strategie zentral definiert werden können. Eine Warteschlangen-Stelle referenziert dann nur noch eine Warteschlange aus dieser zentralen Liste, und übernimmt deren Eigenschaften. Auf diese Weise kann eine Warteschlange für mehrere Stellen verwendet werden. Diese Funktionalität soll analog zur Verwaltung der Färbungen implementiert werden und erfordert eine Anpassung des XML-Schemas des Modells.

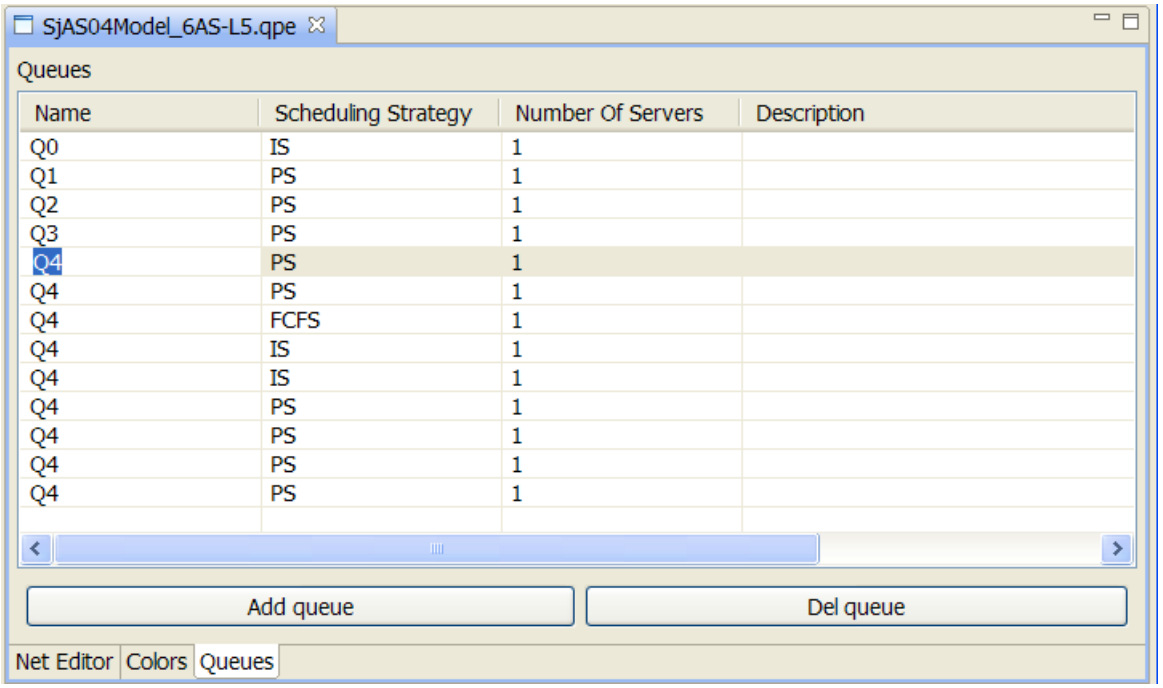

Abbildung 3.1: Zentrale Verwaltung von Warteschlangen

3.3 Speicherung der Simulationsergebnisse

Die Analyse-Ergebnisse eines Simulationslaufes werden bisher als reiner Text direkt auf der Konsole ausgegeben. Hier ein Beispiel für den generierten Report des Simulators zu einer Warteschlangen-Stelle:

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

Overall Queue Util=0.7571974116566562

------------------ Color=0 --------------------
arrivCnt=161471 deptCnt=161468
arrivThrPut=0.014308253925074151 deptThrPut=0.014307988089340334
meanTkPop=3.124693954462401 colUtil=0.7571974116566562
-----
meanST=218.3834439454382 stDevST=322.57718936648314

Steady State Statistics:
numBatchesST=201 batchSizeST=800 stDevStdStateMeanST=45.6472946669
90% c.i. = 218.61956343536986 +/- 5.320603897320468
```

Dieses Format ist nicht sehr gut geeignet für eine weitere maschinelle Verarbeitung. Daher sollen die Simulationsergebnisse nun nach einem erfolgten Simulationslauf in einem XML-Dokument gespeichert werden.

Darin sollen folgende Daten festgehalten werden: Dateiname des Modells, welches der Simulation zugrunde lag, Name der gewählten Simulations-Konfiguration, Datum und Uhrzeit zu denen der Simulationslauf stattfand, sowie die in der Simulation ermittelten Messwerte der beobachteten Netzelemente.

Es gibt vier verschiedene Netzelement-Typen, die während eines Simulationslaufs beobachtet werden können:

1. Eine einfache Stelle
2. Die Warteschlange einer Warteschlangenstelle
3. Das Depository einer Warteschlangenstelle
4. Eine Wartschlange (über alle Warteschlangenstellen betrachtet, von denen sie verwendet wird)

Jedes beobachtete Netzelement besitzt eine eindeutige ID, welche eine Zuordnung des Elements zu seiner Repräsentation im zugrundeliegenden Netzmodell erlaubt, und einen Wert namens *statsLevel*, der angibt, in welchem Umfang Daten zu diesem Element während der Simulation gesammelt wurden.

In dem XML-Dokument sollen für jedes beobachtete Netzelement dessen Typ, ID und *statsLevel* festgehalten werden. Abhängig vom jeweiligen *statsLevel* fallen pro beobachtetem Element unterschiedliche Messwerte an, die gespeichert werden sollen.

Folgende Messwerte fallen derzeit abhängig vom Typ des beobachteten Elements (Nummerierung wie oben) und dessen *statsLevel* an. In Klammern ist jeweils der programminterne Name der jeweiligen Metrik angegeben, welcher auch zur Repräsentation in der XML-Datei verwendet werden soll.

3.4 Anforderungen an die Visualisierung der Simulationsergebnisse

Wie im voherigen Abschnitt gezeigt ist das derzeitige Ausgabeformat des QPN-Simulators nicht sehr übersichtlich. Die Ausgabe erfolgt als reiner Text, die Bezeichnungen der Werte sind kryptisch und bei vielen beobachteten Stellen kann die Ausgabe unüberschaubar lang werden.

Daher soll die Hauptaufgabe der Studienarbeit darin bestehen, eine visuelle Aufbereitung der Simulationsergebnisse zu implementieren. Grundlage dafür wird das XML-Modell der Simulationsergebnisse sein, welches im vorangegangenen Arbeitsschritt entwickelt wurde.

Die während eines Simulationslaufes gesammelten Daten sollen statt wie bisher auf der Konsole nun zunächst innerhalb von QPME in einer eigenen Tab-Komponente tabellarisch dargestellt werden. Hier soll der Anwender wählen können, ob er nun die graphische Aufbereitung aller Daten einer bestimmten Stelle wünscht, oder die Darstellung aller Werte eines bestimmten Messungstyps über alle Stellen.

Die Mittlere Population der Markierungen einer bestimmten Stelle kann beispielsweise nach den unterschiedlichen Farben der Markierungen aufgeschlüsselt werden. Der Anwender soll wählen können, etwa über ein Kontextmenü, welches bei einem Klick mit der rechten Maustaste über der gewünschten Stelle erscheint, ob er dies in Form eines Tortendiagramms oder eines Balkendiagramms dargestellt bekommen möchte.

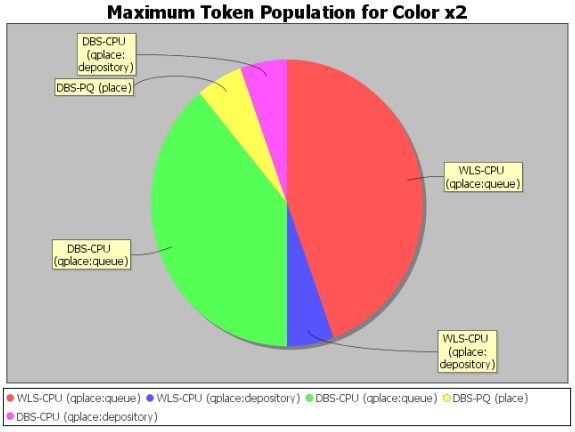

Abbildung 3.2: Ein Kreisdiagramm für die Metrik *Maximum Token Population*

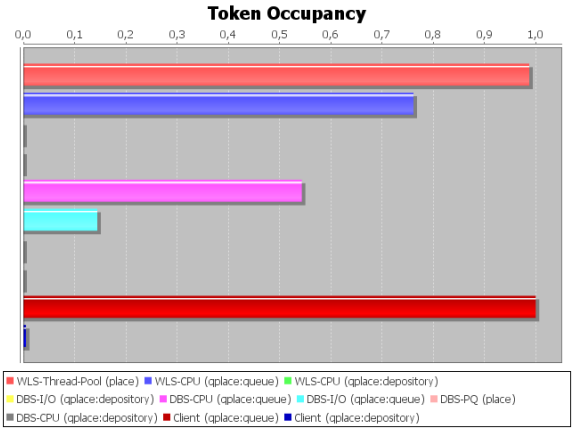

Abbildung 3.3: Ein Balkendiagramm für die Metrik *Token Occupancy*

Weitere für den Anwender interessante Darstellungen sind beispielsweise die Konfidenz-Intervalle der Verweildauer von Markierungen, aufgeschlüsselt nach deren Färbung, oder eine Histogramm-Darstellung der Verweildauer.

Zur Umsetzung dieser Diagramme bietet sich die freie Java-Bibliothek *JFreeChart* [jfre] an, welche die Visualisierung komplexer Diagramme unterschiedlichen Typs, wie etwa Säulendiagramme, Balkendiagramme, Tortendiagramme und Histogramme unterstützt. Die Abbildungen 3.2 und 3.3 demonstrieren die Darstellung zwei dieser Diagrammtypen mit JFreeChart.

Der Benutzer soll eine Datei mit gespeicherten Simulationsergebnissen wählen können, auf deren Grundlage er Visualisierungsanfragen durchführen möchte. Ihm soll eine Übersicht geboten werden über alle in den Simulationsergebnissen vorhandenen Netzelemente (Stellen, Warteschlangen, Depositories) und die gemessenen Metriken hinsichtlich der verschiedenen Farben der Markierungen.

Darauf basierend soll der Benutzer zunächst eine Auswahl treffen können, welche Netzelemente und welche Farben er für eine Visualisierung heranziehen möchte. Zusätzlich soll er eine oder mehrere Metriken wählen können, welche für diese Auswahl visualisiert werden sollen.

Desweiteren soll dem Benutzer die Möglichkeit geboten werden, die Werte der Metriken für die Visualisierung optional zu aggregieren, sowohl hinsichtlich der gewählten

Netzelemente als auch hinsichtlich der gewählten Farben. Mögliche Aggregationen können hierbei beispielsweise die Summe sein oder der Mittelwert.

Für jede zu visualisierende Metrik soll die Art der Visualisierung gewählt werden können: Balkendiagramm, Kreisdiagramm, Tabelle, oder Histogramm, falls dies für die gewählte Metrik sinnvoll ist. Histogrammdaten werden derzeit nur für die Metrik *Token Residence Time* bei einem *statsLevel* >= 4 in den Simulationsergebnissen gespeichert.

Nachdem der Benutzer seine Auswahl für eine Visualisierungsanfrage (*Query*) getroffen hat sollen die Daten dieser Anfrage entsprechend gefiltert, aggregiert und visualisiert werden. Das Resultat kann aus einem oder mehreren Diagrammen bestehen, abhängig davon, wieviele Metriken die Visualisierungsanfrage umfasst und ob Aggregationen gewählt wurden oder nicht.

Der Benutzer soll zudem die Möglichkeit haben, die Größe der dargestellten Diagramme zu beeinflussen und diese als Grafikdatei zu speichern. Auch die Visualisierungsanfrage soll gespeichert werden können, so dass sie jederzeit wiederholt werden kann ohne noch einmal alles neu selektieren zu müssen.

Da ein hoher Grad an Auswahlmöglichkeiten auch eine komplexere Benutzeroberfläche zur Folge hat sollen dem Benutzer zwei unterschiedliche Editoren für Visualisierungsanfragen geboten werden: der *tabellarische Query-Editor* sowie der *Advanced-Query-Editor*. Der tabellarische Editor soll eine einfachere Benutzeroberfläche bieten, jedoch zu Lasten einer geringeren Flexibilität. So ist im tabellarischen Editor keine Aggregation möglich. Der *Advanced-Query-Editor* dagegen soll maximale Flexibilität bieten, ist dafür jedoch etwas weniger intuitiv zu bedienen.

Die Konzepte der beiden Editoren sollen im Folgenden beschrieben werden.

3.5 Der Tabellarische Query-Editor

Der *tabellarische Query-Editor* ist die Editor-Komponente, in die der Benutzer gelangt, sobald er eine Datei mit Simulationsergebnissen öffnet (**.simqpn*). Dieser Editor bietet zwei wichtige Funktionalitäten: zum einen stellt er alle Werte der Metriken für die unterschiedlichen Stellen und Warteschlangenm, die während einer Simulation gemessenen wurden, in tabellarischer Form dar. Zum anderen kann der Benutzer hier sehr schnell eine einfache Visualisierungsanfrage erstellen und ausführen. In diesem Editor ist jedoch nur eine Filterung der darzustellenden Daten möglich, keine Aggregation.

Der tabellarische Editor besteht aus zwei Tabellen: eine Tabelle mit den Daten der Stellen und darunter eine Tabelle mit den Daten der Warteschlangen. Die verschiedenen Metriken bilden die Spalten der Tabellen. Die einzelnen Stellen bilden Knoten einer Baumansicht, die vom Benutzer aufgeklappt werden können und für jede Stelle die Farben ihrer Markierungen als Kindelemente enthalten. Die Felder der Tabelle enthalten die Werte hinsichtlich der jeweiligen Metrik. Bei einem leeren Feld ist kein Messwert vorhanden.

Der Benutzer kann nun mit der Maus und gedrückter [Strg]-Taste entweder in der oberen Tabelle mehrere Stellen und Farben oder in der unteren Tabelle mehrere Warteschlangen selektieren oder deselektieren.

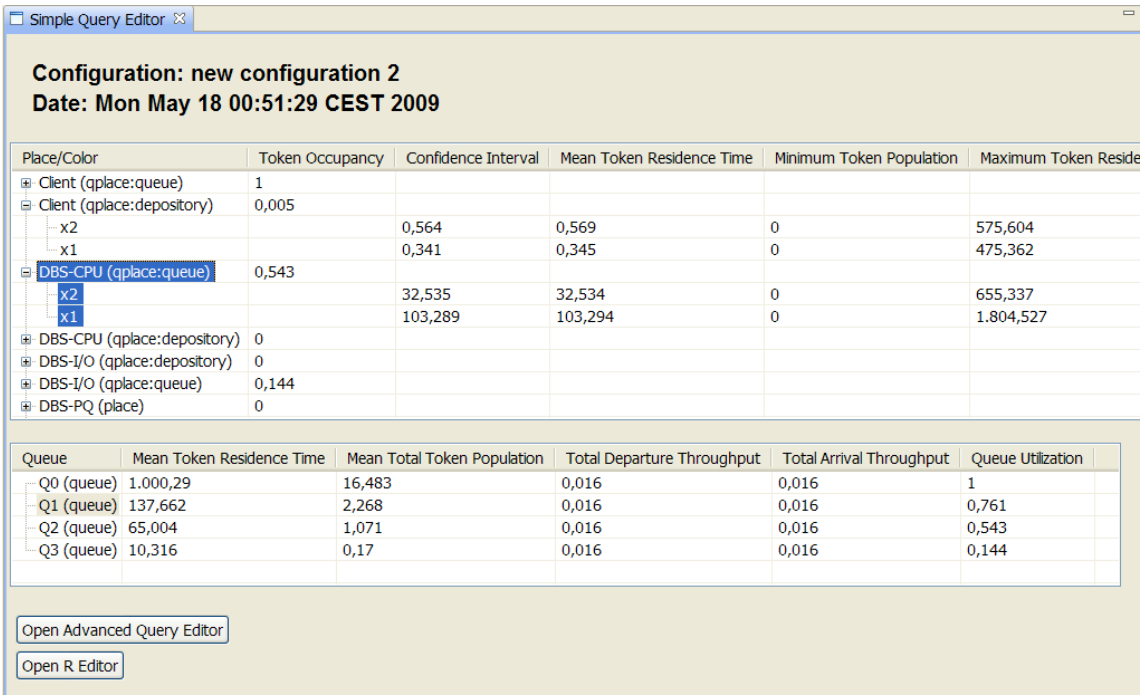

Abbildung 3.4: Der tabellarische Query-Editor

Über einen Rechtsklick auf einer der beiden Tabellen gelangt der Benutzer in ein Kontextmenü. In diesem Menü kann er wählen, welche Metrik er für die Selektion visualisieren möchte. Die Menüpunkte für die verschiedenen Metriken enthalten jeweils ein Untermenü in dem schließlich die gewünschte Art der Visualisierung gewählt werden kann. Nachdem der Benutzer sich für eine Visualisierungsart entschieden hat wird die Visualisierungsanfrage ausgeführt und die Visualisierung dargestellt.

Das Kontextmenü und damit auch die Visualisierungsanfrage bezieht sich jeweils nur auf eine der beiden Tabellen. Da die Warteschlangen andere Metriken haben als einfache Stellen bietet das Kontextmenü dort auch entsprechend andere Menüpunkte.

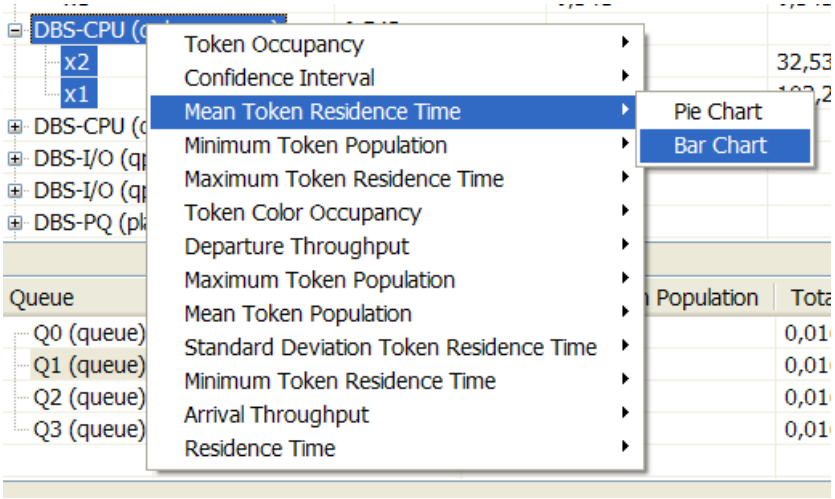

Abbildung 3.5: Das Kontextmenü im tabellarischen Query-Editor

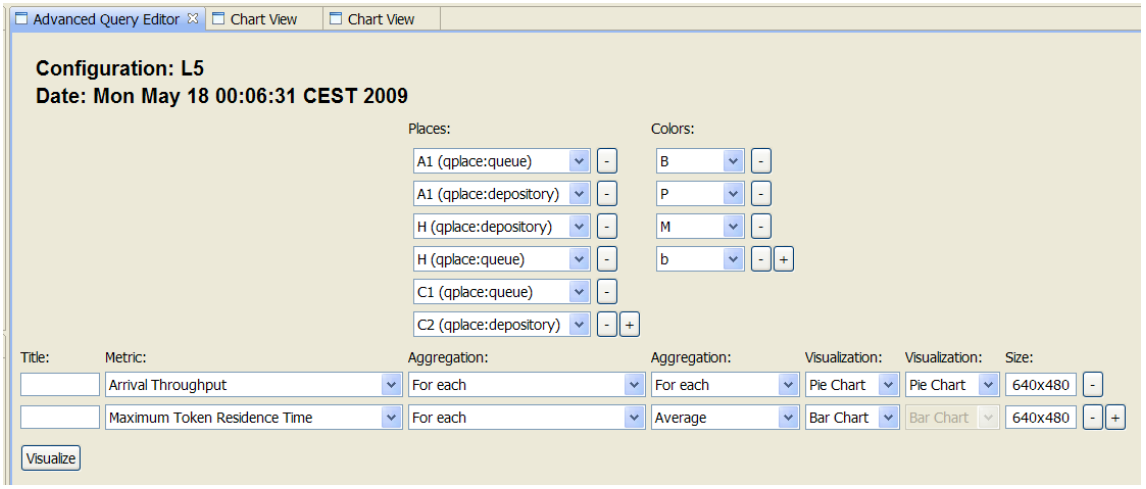

Abbildung 3.6: Der Advanced-Query-Editor

Desweiteren gibt es unterhalb der beiden Tabellen zwei Schaltflächen: über die eine gelangt der Anwender in den *Advanced-Query-Editor*. Dabei wird die in der Tabelle gewählte Selektion übernommen. Die andere Schaltfläche öffnet den *R Editor*, der eine Brücke zur Statistiksoftware R bietet.

3.6 Der Advanced-Query-Editor

Neben der tabellerischen Übersicht soll dem Benutzer eine weitere, flexiblere Möglichkeit geboten werden, die Art der gewünschten graphischen Darstellung auszuwählen. Diese soll sich an den erfahrenen Anwender richten und ihm erlauben auf flexible Weise die Komposition der Darstellung zu konfigurieren, ähnlich einer Datenbankanfrage.

Dieser *Advanced-Query-Editor* soll dem Benutzer möglichst große Flexibilität beim Formulieren einer Visualisierungsanfrage bieten. In diesem Editor können neben der Filterung auch verschiedene Aggregationen gewählt werden, die auf die zu visualisierenden Daten zuvor angewendet werden sollen.

Hier kann der Anwender eine Menge an Stellen und eine Menge an Färbungen auswählen, die ihn interessieren, jeweils entweder eine Teilmenge oder alle. Sowohl für Stellen als auch Färbungen soll nun jeweils eine Art der Aggregierung über eine bestimmte Metrik (Durchsatz, Verweildauer, etc.) ausgewählt werden können. Mögliche Aggregierungen sind:

- Anzeige für jedes Element (keine Aggregierung)
- Summe über Auswahl
- Mittelwert über Auswahl

Zuletzt wählt der Anwender die Art der Ausgabe für die Metrik, beispielsweise *Tortendiagramm*, *Balkendiagramm* oder *Textausgabe*.

Bedienung des Advanced-Query-Editors

Der Benutzer gelangt aus dem tabellarischen Query-Editor über eine Schaltfläche in den *Advanced-Query-Editor*. Hier gibt es in der Mitte zwei Hauptspalten: eine Spalte zur Auswahl der Stellen und eine Spalte zur Auswahl der Farben. Die Auswahl wird über Comboboxen getroffen. Über + und - Schaltflächen können Elemente hinzufgefügt oder wieder entfernt werden. Auf diese Weise lassen sich zwei Mengen definieren: die Menge der Stellen und die Menge der Farben, die für die Visualisierung betrachtet werden sollen.

Unterhalb dieser beiden Spalten kann der Benutzer beliebig viele Zeilen anlegen, ebenfalls über + und - Schaltflächen. Jede dieser Zeilen repräsentiert eine Metrik, die visualisiert werden soll. Der Benutzer kann hier für jede Metrik wählen, welche Aggregation über die Stellen und welche Aggregation über die Farben angewendet werden soll, oder ob keine Aggregation angewendet werden soll (Auswahlmöglichkeit *For each*).

Die Auswahlfelder für die beiden Aggregationen sind so ageordnet, dass sie sich unterhalb der beiden Spalten für die Auswahl der Stellen und der Farben befinden, auf die sie sich jeweils beziehen.

Hinter den beiden Auswahlfeldern für die Aggregationen befinden sich zwei Auswahlfelder für Visualisierungsarten (Balkendiagramm, Kreisdiagramm, ...). Diese beziehen sich auf die beiden Aggregationen.

Wählt der Benutzer in dieser Zeile für eine der beiden oder beide Aggregationen den Wert *For each*, so entstehen für die entsprechende Metrik bei der Visualisierung mehrere Diagramme. Wählt der Benutzer für beide Aggregationen einen anderen Wert als *For each*, so entsteht nur ein einziges Diagramm für die Metrik, das den einzelnen aggregierten Wert darstellt, da sowohl über die Stellen als auch über die Farben aggregiert wurde.

In jeder Metrik-Zeile kann der Benutzer desweiteren die Größe in Pixeln angeben, mit der die Diagramme dargestellt werden sollen, in der Form *BREITExHÖHE* (Beispiel: *600x400*).

Mit einem Klick auf die Schaltfläche *Visualize* wird schließlich die Visualisierung entsprechend der gewählten Anfrage angefordert.

3.7 Integration mit R

Die Integration mit R ist einfach gehalten. Über eine Schaltfläche gelangt der Benutzer aus dem tabellarischen Editor heraus in den *R Editor*. Hier kann in einem Textfeld ein R-Befehl eingegeben werden. Zusätzlich muss eine Datei gewählt werden, welche die Daten enthält, auf die sich der Befehl beziehen soll.

3.8 Zusammenfassung

Hier werden die in den vorangegangenen Abschnitten beschriebenen Anforderungen noch einmal zusammengefasst:

Portierung auf neueste Eclipse-Version

Die aktuellen Versionen sind Eclipse 3.4.1 *Ganymede* und GEF 3.4.1.

Zentrale Verwaltung von Warteschlangen

Die Warteschlangen sollen nicht mehr pro Warteschlangenstelle definiert werden, sondern analog zu den Farben der Markierungen zentral in einem eigenen Editor. Die einzelnen Warteschlangenstellen referenzieren dann jeweils eine der zentral definierten Warteschlangen.

Speicherung der Simulationsergebnisse

Die Simulationsergebnisse sollen im XML-Format gespeichert werden und folgende Daten enthalten:

- Dateiname des Modells, Simulations-Konfiguration, Datum und Uhrzeit
- Beobachtete Netzelemente mit Typ (Stelle, Warteschlange, Depository), ID und *statsLevel*
- Pro Netzelement die Werte der Metriken, aufgeschlüsselt nach Farben der Markierungen.
- Bei eine Elementen mit *statsLevel* $>= 4$ die Histogramm-Daten der Metrik *Token Residence Time*

Visualisierung

Der Benutzer soll in Visualisierungsanfragen folgende Auswahl treffen können:

- Menge der in Betracht zu ziehenden Stellen
- Menge der in Betracht zu ziehenden Farben
- Die zu visualisierenden Metriken
- Die Art der Aggregation jeweils für die Stellen und die Farben
- Die Art der Visualisierung (Balkendiagramm, Kreisdiagramm, ...)
- Darstellungsgröße der Diagramme

Dafür sollen dem Benutzer zwei verschiedene Editoren für Visualisierungsanfragen geboten werden: Den einfacheren *tabellarischen Query-Editor*, jedoch mit geringerer Flexibilität, sowie den flexibleren *Advanced-Query-Editor* für komplexere Visualisierungsanfragen.

4. Entwurf

4.1 Zentrale Verwaltung von Warteschlangen

Für eine zentrale Verwaltung von Warteschlangen ist eine Änderung an dem XML-Format für QPN-Modelle notwendig. Die Informationen für Warteschlangen zur Anzahl der Server und der *Scheduling Strategy* werden nicht mehr als Attribute in den XML-Elementen der Warteschlangen-Stellen gespeichert, sondern zentral unterhalb eines Netzweiten *queues*-Elements. Dabei besitzt jede Warteschlange nun eine ID über die sie nun von Warteschlangenstellen durch das Attribut *queue-ref* referenziert wird:

```
<net>
  <queues>
    <queue id="xxxxxxxxxxx" name="queue_name" number-of-servers="integer"
           strategy="PRIO/PS/FCFS/IS/RANDOM" />
    ...
  </queues>
  <places>
    <place id="yyyyyyyyyyy" departure-discipline="NORMAL"
           type="queueing-place" name="Client"
           queue-ref="xxxxxxxxxxx">
      ...
    </place>
    ...
  </places>
</net>
```

4.2 XML-Speicherung von Simulationsergebnissen

Das XML-Format für die Speicherung der Simulationsergebnisse wurde auf Basis der Anforderungen entworfen, wie sie in Abschnitt 3.3 im beschrieben sind. Eine Datei mit Simulationsergebnissen besitzt die Dateinamenserweiterung *.simpqn* und besitzt folgende Struktur:

```
<simqpn-results model-file="xy.qpe" name="modelname-configuration-date-time"
                configuration-name="" date="">
  <observed-element type="place" name="" id="" stats-level="">
    <metric type="" value="">
    <metric type="" value="">
    ...
    <color name="" id="" real-color="#rrggbb">
      <metric type="arrivThrPut" value="">
      <metric type="meanTkOcp" value="">
      ...
      <histogram type="histST" bucket-size="" num-buckets="">
         <mean></mean>
         <percentiles>
           <percentile for="0.25"></percentile>
           <percentile for="0.50"></percentile>
           <percentile for="0.75"></percentile>
           <percentile for="1.0"></percentile>
         </percentiles>
         <buckets>
           <bucket index="0"></bucket>
           <bucket index="1"></bucket>
           <bucket index="2"></bucket>
           ...
         </buckets>
      </histogram>
    </color>
    <color>
      ...
    </color>
    ...
  </observed-element>

  <observed-element type="qplace:queue">
  </observed-element>

  <observed-element type="qplace:depository">
  </observed-element>

  <observed-element type="queue">
  </observed-element>

</simqpn-results>
```

Die während der Simulation beobachteten Elemente werden jeweils durch ein *observed-element* repräsentiert, wobei folgende Typen möglich sind: *place*, *qplace:queue*, *qplace:depository* und *queue*. Ein solches Element kann entweder direkt Metriken enthalten oder für Metriken, die sich nach Farben aufschlüsseln lassen, innerhalb eines oder mehrerer *color*-Elementen.

Einen Spezialfall bilden Histogramme. Sie werden in einem *histogram*-Element gespeichert und enthalten neben den Werten der Buckets auch Informationen zu Perzentilen, standardmäßig für 25%, 50%, 75% und 100%.

4.3 Aggregation

Im Folgenden wird die Vorgehensweise bei der Aggregation beschrieben.

Hat der Benutzer für eine Visualisierungsanfrage eine Menge an Stellen (*Places*) $p_1, p_2, \ldots, p_m$ und eine Menge an Farben (*Colors*) $c_1, c_2, \ldots, c_n$ ausgewählt, so lässt sich daraus bezüglich einer Metrik eine Matrix aufstellen, bei der die Elemente dieser Matrix die Werte der Metrik bilden, die während der Simulation gemessenen wurden.

Dabei kann es vorkommen, dass einzelne Felder der Matrix keine Werte enthalten, d.h. die Matrix kann Lücken aufweisen. Dies kann beispielsweise der Fall sein, wenn entweder eine Metrik während der Simulation für eine Stelle nicht berücksichtigt wurde, da der *statsLevel* der Stelle zu niedrig ist, oder weil eine Stelle bestimmte Farben von Markierungen nicht enthält.

In den Abbildungen 4.1 bis 4.4 wird eine solche Matrix bezüglich einer Metrik exemplarisch für eine Auswahl von vier Stellen und fünf Farben dargestellt. Die grauen Kästchen repräsentiren Kombinationen, für die kein Wert existiert. Die Abbildungen veranschaulichen die vier verschiedenen Aggregationsmöglichkeiten:

- Keine Aggregation (Abb. 4.1)
- Aggregation über die Stellen (Abb. 4.2)
- Aggregation über die Farben (Abb. 4.3)
- Aggregation sowohl über die Stellen als auch über die Farben (Abb. 4.4)

Eine Aggregation kann beispielsweise die Summe oder der Mittelwert sein. Wichtig bei einer Aggregation wie dem Mittelwert ist, dass nur die Anzahl der tatsächlich vorhandenen Werte in die Berechnung einfließt.

Wünscht der Benutzer keine Aggregation, so entstehen insgesamt bei m gewählten Stellen und n gewählten Farben in der Visualisierung $m + n$ Diagramme (*Charts*).

Wird entweder über die Stellen oder die Farben aggregiert, so entsteht daraus ein Diagramm. Wenn sowohl über Stellen als auch über Farben aggregiert wird, dann ist das Resultat nur ein einziger Wert. Dieser kann durch ein Diagramm, das nur einen Wert enthält dargestellt werden.

4.4 Zusammenfassung

In diesem Kapitel wurden die Änderungen an dem XML-Format für Netzmodelle beschrieben, die für zentral Verwaltete Warteschlangen notwendig sind, sowie ein XML-Format zur Speicherung der Simulationsergebnisse.

Desweiteren wurde das Verfahren beschrieben, mit dem die Daten einer Metrik sowohl hinsichtlich der Stellen als auch der Farben aggregiert werden können und wie diese Daten auf Diagramme für die Visualisierung abgebildet werden.

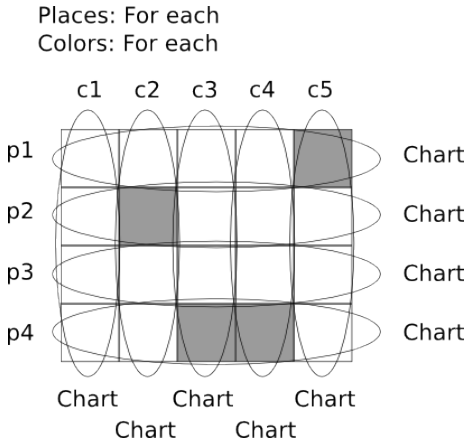

Abbildung 4.1: Keine Aggregation

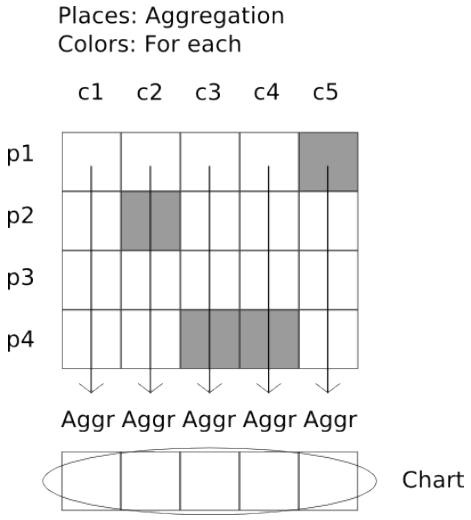

Abbildung 4.2: Aggregation über die Stellen

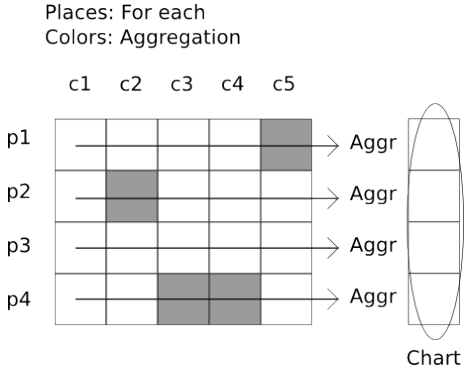

Abbildung 4.3: Aggregation über die Farben

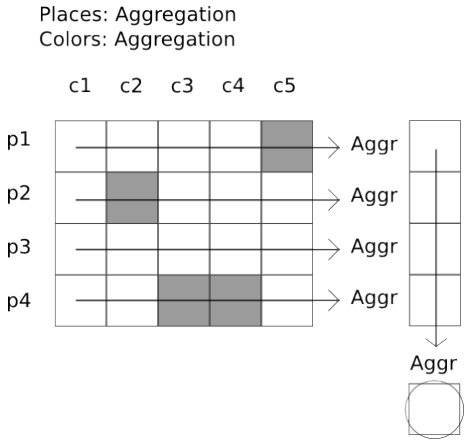

Abbildung 4.4: Aggregation sowohl über die Stellen als auch über die Farben

5. Implementierung

In diesem Kapitel werden einige Details zur Implementierung der neuen Funktionalität vorgestellt. QPME ist in zwei Projekte unterteilt:

- *QPE-Base*: der Netzeditor, basierend auf Eclipse RCP und dem Graphical Editing Framework (GEF) .
- *QPE-SimQPN*: der Simulator, der als Plugin in QPE-Base integriert werden kann, aber auch standalone von der Kommandozeile aus aufgerufen werden kann.

Im Folgenden wird jeweils das Projekt und das Package angegeben, in dem sich die beschriebenen Komponenten befinden.

5.1 Portierung auf neueste Eclipse-Version

Die Portierung auf die neueste Eclipse-Version stellte sich als relativ einfach heraus: Einige Interfaces und Methoden werden in der neueren Eclipse-Version nicht mehr unterstützt oder sind als *deprecated* markiert. In allen Fällen ließen sich jedoch mit Hilfe der Eclipse-API-Dokumentation leicht äquivalente Alternativen finden. QPME läuft nun mit Eclipse 3.4.1 *Ganymede*.

5.2 XML-Speicherung von Simulationsergebnissen

Package: de.tud.cs.simqpn.kernel
Projekt: QPE-SimQPN

StatsDocumentBuilder

Der *StatsDocumentBuilder* baut über die öffentliche Methode *buildDocument()* aus den vom Simulator ermittelten Statistiken ein XML-Dokument zusammen. Dabei wird nach den unterschiedlichen Typen der vom Simulator beobachteten Netzelemente unterschieden (Places, Queues, Depositories, etc.). Desweiteren wird nach

dem *statsLevel* der jeweiligen Elemente entschieden, welche Statistiken vom Simulator ermittelt wurden und in das Dokument aufgenommen werden sollen. Der *RunSimulationWizard* (de.tud.cs.simqpn.plugin.wizard) wurde so erweitert, dass nach abgeschlossenem Simulationslauf der *StatsDocumentBuilder* aufgerufen wird und das resultierende XML-Dokument anschließend in einer Datei nach einem bestimmten Namensschema (enthält unter anderem Timestamp, etc.) gespeichert wird.

5.3 Die Query-Editoren

Package: de.tud.cs.qpe.editors.query
Projekt: QPE-Base

SimpleQueryEditor

Der *SimpleQueryEditor* wird beim Öffnen einer Datei von der *OpenAction* in *de.tud.cs.qpe.rcp.actions.file* aufgerufen, falls die gewählte Datei ein Simulationsergebnis-Dokument ist (Dateierweiterung **.simqpn*). Der *SimpleQueryEditor* implementiert ein *EditorPart* und bekommt ein *QueryEditorInput* übergeben.

Der *SimpleQueryEditor* enthält als GUI-Komponenten neben den beiden Tabellen für Stellen und Warteschlangen einen Button mit der Aufschrift *Open Advanced Query Editor*, über den der **AdvancedQueryEditor** geöffnet wird und einen Button mit der Aufschrift *Open R Editor*, mit dem der Editor zum Absetzen von R-Befehlen geöffnet wird.

AdvancedQueryEditor

Der *AdvancedQueryEditor* implementiert genau wie der *SimpleQueryEditor* ein *EditorPart* und bekommt ein *QueryEditorInput* übergeben.

Der *AdvancedQueryEditor* enthält als GUI-Komponenten ein *QueryComposite* und einen Button mit der Aufschrift *Visualize*, über den die Visualisierung eines Queries gestartet wird.

QueryEditorInput

Die *QueryEditorInput* implementiert eine *IPathEditorInput* und enthält die Eingabe-Daten für Query-Editoren wie den *SimpleQueryEditor* und den *AdvancedQueryEditor*. Weitere Query-Editoren sind denkbar.

Eine *QueryEditorInput* bekommt den Pfad zu einer XML-Datei mit Simulationsergebnissen, läd den XML-Dokumentenbaum aus der Datei und erzeugt aus diesem ein *SimulationResults*-Objekt, auf das über eine öffentliche Methode zugegriffen werden kann. Die Klasse *SimulationResults* wird weiter unten beim Modell beschrieben.

5.4 Das Modell für die Query-Editoren und Queries

Package: de.tud.cs.qpe.editors.query.model
Projekt: QPE-Base

SimulationResults

SimulationResults bekommt XML-Daten mit Simulationsergebnissen übergeben, parst diese und bietet öffentliche Methoden, um auf die geparsten Daten zuzugreifen. Dies sind neben Namen, Datum/Urzeit, Konfiguration der Simulation vor allem die *Place*s sowie die *Color*s, also die Färbungen der Markierungen im Netz, und die dazugehörigen, während der Simulation gemessenen Metriken.

Query

Die *Query* repräsentiert eine komplette Visualisierungs-Anfrage eines Query-Editors. Beispielsweise liefert das *QueryComposite* über die Methode *getQuery()* die derzeit in ihm vom Benutzer zusammengestellte Anfrage. Ein solches Query-Objekt kann dann an den *QueryVisualizer* zur Visualisierung übergeben werden.

Eine *Query* besteht aus zwei Filtermengen A und B (in unserem Fall eine Menge an *Color*s und eine Menge an *Place*s) und mehreren *MetricQueries*, welche der Auswahl von mehreren *MetricRows* entsprechen.

MetricQuery

Eine *MetricQuery* repräsentiert die Auswahl einer *MetricRow*: Titel, Metrik, Aggregationen für die beiden Filter-Gruppen, Visualisierung.

Metric

Repräsentiert eine Metrik. Ist als *enum* umgesetzt, welche die verschiedenen verfügbaren Metriken mit ihren menschenlesbaren Namen sowie ihren Kürzeln für das XML-Format enthält.

Möchte man beispielsweise eine weitere Metrik hinzufügen, so reicht ein Eintrag in dieser *enum*. Sobald eine entsprechende Metrik in einem XML-Dokument mit Simulationsergebnissen vorkommt wird sie auch in den Query-Editoren angezeigt.

MetricValue

Repräsentiert den Wert einer Metrik. Referenziert die Metrik, zu der er gehört.

Aggregation Repräsentiert eine Aggregation. Ist als *enum* umgesetzt, welche die verschiedenen verfügbaren Aggregationen mit ihren Namen und jeweils ihrer spezifischen Aggregationsmethode *aggregate()* enthält. Sie bekommt mehrere *MetricValues* übergeben und liefert einen einzelnen, aggregierten *MetricValue* zurück.

Color, Place

Repräsentieren eine Markierungs-Farbe und eine Stelle eines Queueing-Petrinetzes. Bei beiden sind *equals()* und *hashCode()* überschrieben, um gleiche Farben oder Places in einer *HashMap* oder einem *HashSet* eindeutig identifizieren zu können.

Ein *Place* kann für verschiedene Farben jeweils mehrere *MetricValues* haben.

5.5 Visualisierung

Package: de.tud.cs.qpe.editors.visualization
Projekt: QPE-Base

QueryVisualizer

Der *QueryVisualizer* bekommt im Konstruktor ein *SimulationResults*-Objekt übergeben, das die Simulations-Daten enthält, auf denen er Operieren soll. Die Methode *visualize()* bekommt eine *Query* übergeben, die nun auf die Daten angewendet werden soll. Das Ergebnis ist eine Liste aus *ChartGroups*, welche die Visualisierungen enthalten.

Der *QueryVisualizer* wendet für jede *MetricQuery* der Gesamt-Query die gewählte Aggregation auf die MetricValues an und übergibt die aggregierten Werte an die jeweils gewählte Visualization, und erhält so jeweils ein oder mehrere Charts, die in *ChartGroups* zusammengefasst sind.

Der QueryVisualizer wird einem dem *AdvancedQueryEditor* heraus aufgerufen, sobald der *Visualize*-Button geklickt wird.

ChartGroup

Eine *ChartGroup* ist eine Liste aus *JFreeCharts*, zusammengefasst unter einem gemeinsamen Titel.

Visualization

Dies ist eine abstrakte Klasse, durch welche unterschiedliche Visualisierungen repräsentiert werden können. Die wichtigste Methode ist die abstrakte Methode *createChart()*, die von jeder Visualisierung implementiert werden muss.

Sie erzeugt aus einem Titel und einer Datenreihe aus *MetricValues* ein *JFreeChart*.

Implementierungen dieser abstrakten Klasse sind beispielsweise *PieChartVisualization* und *BarChartVisualization*.

VisualizationEditor

Der *VisualizationEditor* bekommt über seine *VisualizationEditorInput* eine Liste von *ChartGroups*, die er dann layoutet und darstellt.

Er wird von den Query-Editoren geöffnet und mit den Visualisierungsergebnissen des *QueryVisualizers* gespeist, nachdem vom Benutzer eine Visualisierung angefragt wurde.

Er ist nur aus dem Grund als *EditorPart* implementiert, damit er bei den anderen Editor-Komponenten angezeigt wird. Ein tatsächliches Editieren ist nicht möglich.

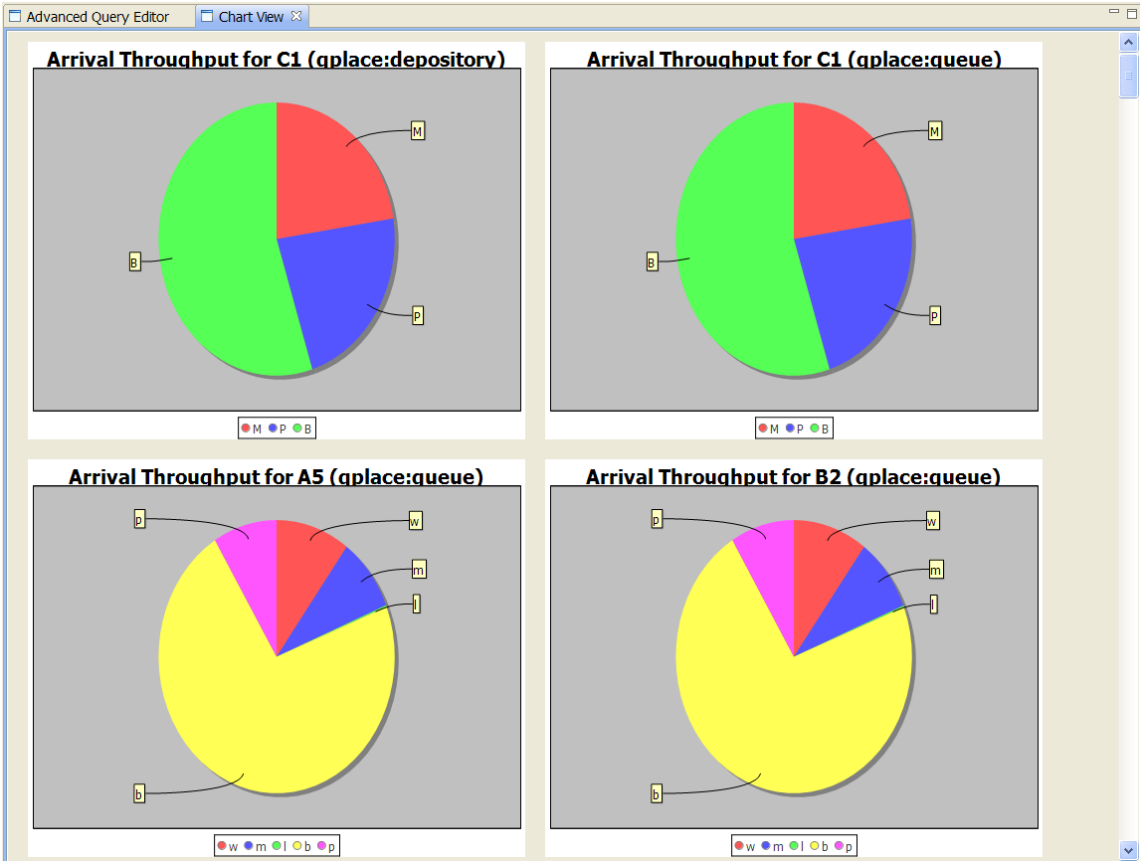

Abbildung 5.1: Darstellung mehrerer Diagramme im *VisualizationEditor*

6. Validierung

Um sicherzustellen, dass bei der Weiterentwicklung von QPME keine Regressionen entstanden sind und dass die neu entwickelten Funktionen korrekt arbeiten wurde das Programm mit mehreren Modellen validiert:

- Modell vom SPECjAppServer2001-Benchmark (Abb. 6.1) [KoBu03]
- Produkt-form Warteschlangennetz von G. Bolch, M. Kirschnick, PEPSY-QNS, Technical Report TR-I4-94-18, University of Erlangen-Nuremberg, Germany, June 1994. (Abb. 6.2) [KoBu06], [peps]
- Modell vom SPECjAppServer2004-Benchmark (Abb. 6.3) [Koun06]

Die Validierung der Arbeit besteht aus zwei Teilen:

- Überprüfen, dass bei der Simulation keine Regressionen im Vergleich zur ursprünglichen Version entstanden sind
- Überprüfen, dass die neu hinzugekommenen Funktionen korrekt arbeiten

6.1 Regressionstest

Um mögliche Regressionen aufzudecken wurden diese Modelle zunächst mit der ursprünglichen Version von QPME simuliert. Danach wurden die Modelle mit der neuen QPME-Version nachmodelliert, da die beiden Modellformate aufgrund der unterschiedlichen Repräsentation der Warteschlangen zueinander inkompatibel sind.

Diese nachgebauten Modelle wurden nun mit der neuen QMPE-Version simuliert und die Simulationsergebnisse mit denen der alten Version verglichen. Die Werte der gemessenen Metriken dürfen dabei nur geringfügig voneinander abweichen. Dabei wurden die Simulationsläufe mehrmals mit verschiedenen *statsLeveln* bis hin zum höchsten Wert ($statsLevel = 5$) durchgeführt um sicherzustellen, dass bei der Validierung alle Metriken abgedeckt wurden.

Bei diesem Vergleich stellte sich heraus, dass die Simulationsergebnisse der alten und der neuen QPME-Version einander entsprechen und somit keine Regressionen entstanden sind.

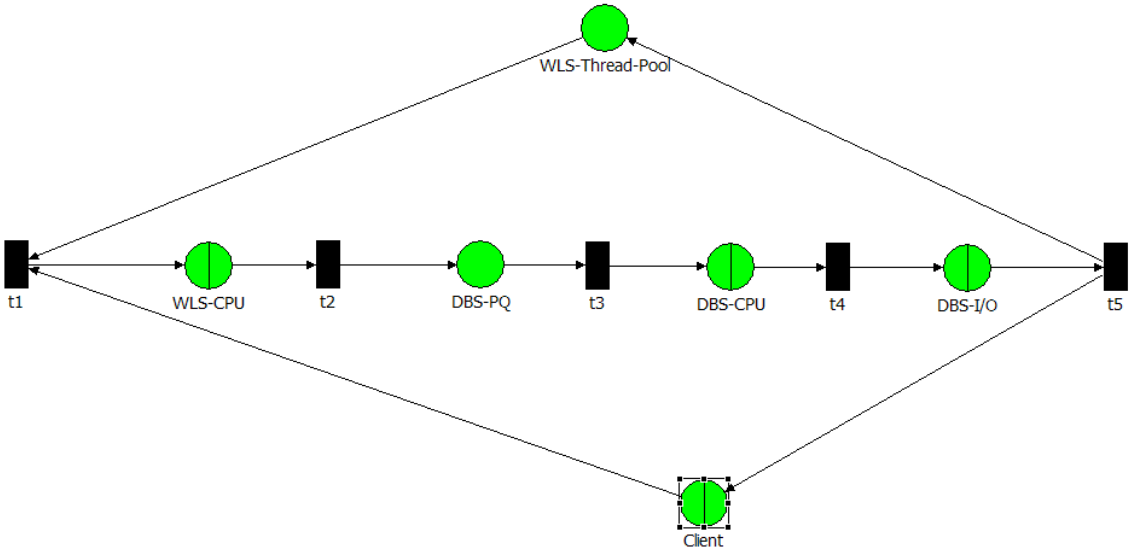

Abbildung 6.1: Netz von ispass03-02.qpe

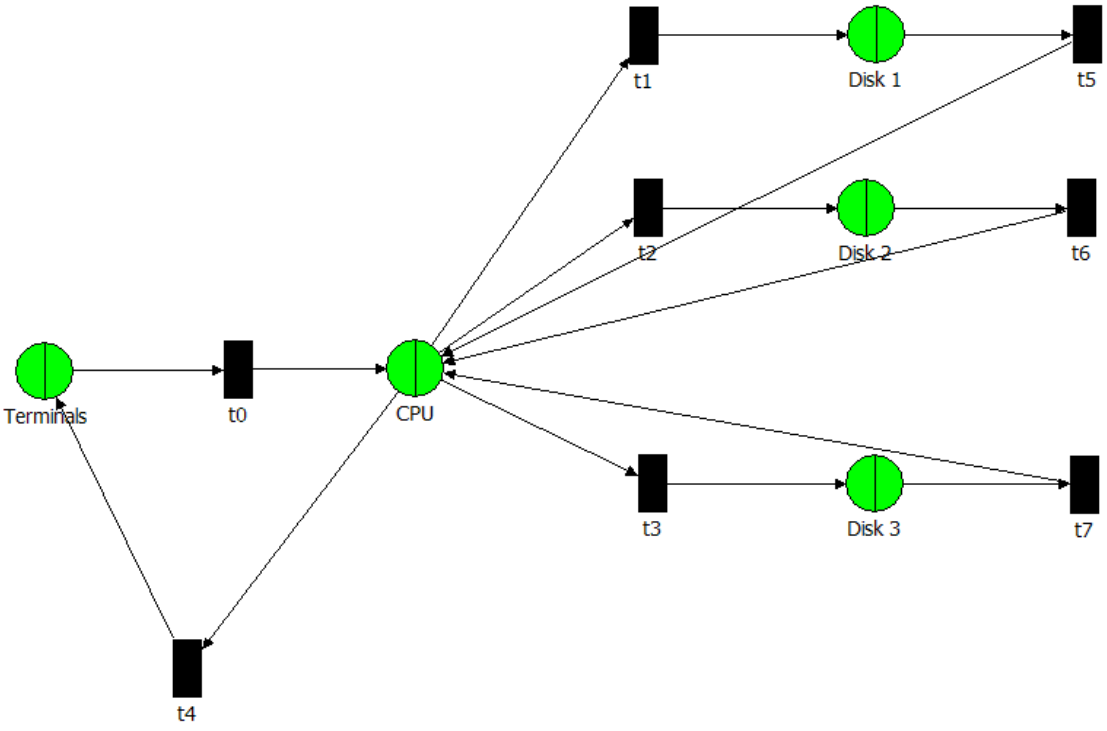

Abbildung 6.2: Netz von pepsy-bcmp2.qpe

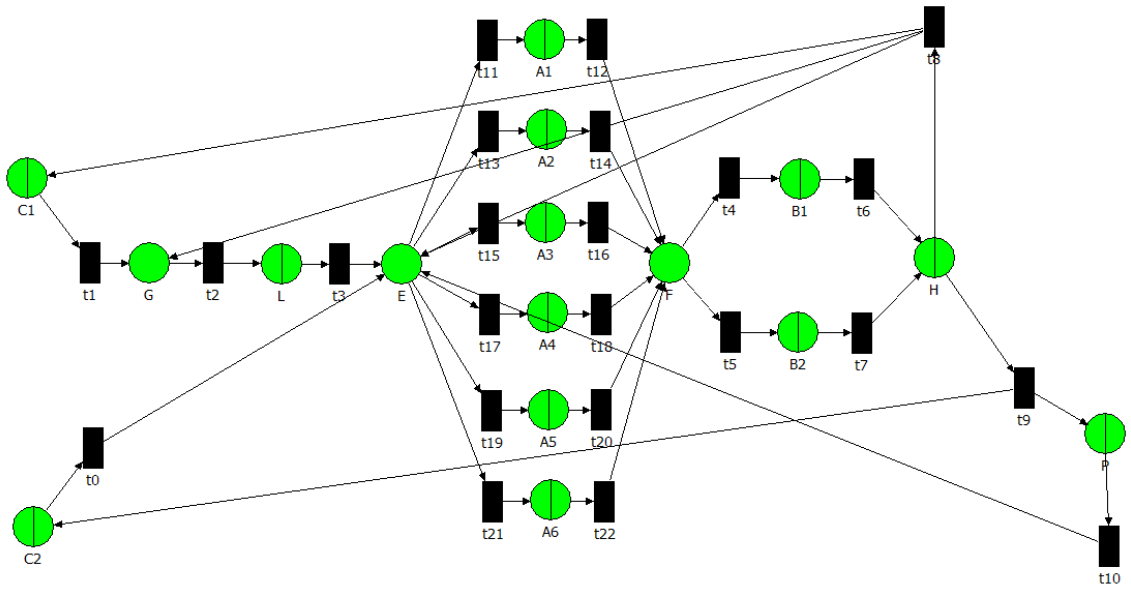

Abbildung 6.3: Netz von SjAS04Model_6AS-L5.qpe

6.2 Validierung der neuen Funktionalität

Um die Korrektheit der Aggregierung und Visualisierung zu überprüfen wurden auf Basis der Simulationsergebnisse oben genannter Modelle mehrere Visualisierungsanfragen durchgeführt. Dabei wurde darauf geachtet, dass alle Features, die sowohl im *tabellarischen Query-Editor* als auch im *Advanced Query Editor* implementiert sind abgedeckt wurden. Die Visualisierungsanfragen wurden mit allen Metriken durchgeführt.

Für jede durchgeführte Anfrage wurden die visualisierten Ergebisse mit manuell aggregierten Daten aus der textuellen Ausgabe verglichen.

Das Histogramm, das bei *statsLevel* >= 4 für die Metrik *Token Residence Time* verfügbar ist wurde mit einem aus den Rohdaten (*statsLevel* = 5) mit der Statistiksoftware R erstellten Histogramm verglichen.

6.3 Zusammenfassung

Bei der Validierung mit mehreren Modellen stellte sich heraus, dass keine Regressionen im Vergleich zur ursprünglichen QPME-Version entstanden sind und dass die neu hinzugekommenen Funktionen zur Filterung, Agreggierung und Visualisierung dem Anwender korrekte Ergebnisse liefern.

7. Zusammenfassung und Ausblick

Zu Beginn dieser Arbeit wurde auf die theoretischen Grundlagen von Queueing-Petri-Netzen und deren Anwendung eingegangen sowie das Modellierungs- und Simulationswerkzeug für Queueing-Petri-Netze QPME vorgestellt, das im Rahmen dieser Studienarbeit weiterentwickelt wurde.

Zunächst wurde die Problemstellung der vorliegenden Arbeit vorgestellt und die Anforderungen untersucht und festgelegt. Es wurden sowohl der Entwurf als auch die Implementierung beschrieben, mit der diese Anforderungen erzielt wurden. Dem Anwender wird nun über die grafische Benutzeroberfläche des Werkzeugs ermöglicht, komplexe Anfragen mittels verschiedener Kriterien an das System zu formulieren, nach denen die Analyse-Daten der Simulation gefiltert, aggregiert und danach visuell aufbereitet werden, um so eine bessere Übersicht über die Daten zu bekommen und die Auswertung zu erleichtern.

Abschließend wurde die Methode beschrieben, mit welcher überprüft wurde, dass durch die Weiterentwicklung von QPME keine Regressionen entstanden sind und auf welche Weise die neuen Funktionalitäten zur Aggregierung und Visualisierung validiert wurden.

Auf Basis dieser Weiterentwicklung soll eine neue Version von QPME veröffentlicht werden.

Literatur

[BaKr02] F. Bause und F. Kritzinger. *Stochastic Petri Nets - An Introduction to the Theory.* Vieweg Verlag. 2002.

[Baus93] 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*, Washington, DC, USA, 1993. IEEE Computer Society, S. 14–23.

[ecli] Eclipse RCP. `http://wiki.eclipse.org/index.php/Rich_Client_Platform`.

[Heim07] Frank Heimburger. Performance Modelling of Java EE Applications using LQNs and QPNs. Diplomarbeit, Technische Universität Darmstadt, Darmstadt, Germany, September 2007.

[jfre] JFreeChart. `http://www.jfree.org/jfreechart/`.

[KoBu03] Samuel Kounev und Alejandro Buchmann. Performance Modeling of Distributed E-Business Applications using Queueing Petri Nets. In *Proceedings of the 2003 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS 2003), Austin, Texas, USA, March 6-8, 2003*, Washington, DC, USA, 2003. IEEE Computer Society, S. 143–155. Best-Paper-Award at ISPASS-2003.

[KoBu06] Samuel Kounev und Alejandro Buchmann. SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation. *Performance Evaluation*, 63(4-5), Mai 2006, S. 364–394.

[KoBu07] Samuel Kounev und Alejandro Buchmann. *Vedran Kordic (ed.) Petri Net, Theory and Application*, Kapitel On the Use of Queueing Petri Nets for Modeling and Performance Analysis of Distributed Systems. Advanced Robotic Systems International, I-Tech Education and Publishing, Vienna, Austria. Februar 2007.

[KoDB06] Samuel Kounev, Christofer Dutz und Alejandro Buchmann. QPME - Queueing Petri Net Modeling Environment. In *In Proceedings of the Third International Conference on the Quantitative Evaluation of Systems - (QEST'06)*, Washington, DC, USA, 2006. IEEE Computer Society, S. 115–116.

[KoDu07] Samuel Kounev und Christofer Dutz. *QPME 1.0 User's Guide.* Technische Universität Darmstadt, Januar 2007.

[KoDu09] Samuel Kounev und Christofer Dutz. QPME - A Performance Modeling Tool Based on Queueing Petri Nets. *ACM SIGMETRICS Performance Evaluation Review (PER), Special Issue on Tools for Computer Performance Modeling and Reliability Analysis*, Band to appear, Januar 2009.

[KoNT07a] Samuel Kounev, Ramon Nou und Jordi Torres. Autonomic QoS-Aware Resource Management in Grid Computing using Online Performance Models. In *Proceedings of the Second International Conference on Performance Evaluation Methodologies and Tools (VALUETOOLS 2007), Nantes, France, October 23-25, 2007*, ICST, Brussels, Belgium, Belgium, 2007. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), S. 1–10.

[KoNT07b] Samuel Kounev, Ramon Nou und Jordi Torres. Using QPN models for QoS Control in Grid Middleware. Technischer Bericht UPC-DAC-RR-CAP-2007-4, Computer Architecture Department, Technical University of Catalonia (UPC), Spain, April 2007.

[Koun05] Samuel Kounev. *Performance Engineering of Distributed Component-Based Systems - Benchmarking, Modeling and Performance Prediction*. Shaker Verlag, Ph.D. Thesis, Technische Universität Darmstadt, Germany. This contribution received Best Dissertation Award from the Vereinigung von Freunden der Technischen Universität zu Darmstadt e.V., Dezember 2005.

[Koun06] Samuel Kounev. Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets. *IEEE Transactions on Software Engineering*, 32(7), Juli 2006, S. 486–502.

[Koun08] Samuel Kounev. QPME (Queueing Petri net Modeling Environment) Homepage. `http://sdq.ipd.uka.de/people/samuel_kounev/projects/QPME`, 2008.

[KSBB08] Samuel Kounev, Kai Sachs, Jean Bacon und Alejandro Buchmann. A Methodology for Performance Modeling of Distributed Event-Based Systems. In *Proceedings of the 11th IEEE International Symposium on Object Oriented Real-Time Distributed Computing (ISORC 2008), Orlando, Florida, USA, May 5-7, 2008*, Washington, DC, USA, 2008. IEEE Computer Society, S. 13–22. Nominated for Best-Paper-Award at ISORC-2008.

[NKJT09] Ramon Nou, Samuel Kounev, Ferran Julia und Jordi Torres. Autonomic QoS control in enterprise Grid environments using online simulation. *Journal of Systems and Software*, 82(3), März 2009, S. 486–502.

[NoKo07] Ramon Nou und Samuel Kounev. Preliminary Analysis of Globus Toolkit 4 to Create Prediction Models. Technischer Bericht UPC-DAC-RR-2007-37, Computer Architecture Department, Technical University of Catalonia (UPC), Spain, Juli 2007.

[NoKT07] Ramon Nou, Samuel Kounev und Jordi Torres. Building Online Performance Models of Grid Middleware with Fine-Grained Load-Balancing:

A Globus Toolkit Case Study. In Katinka Wolter (Hrsg.), *Formal Methods and Stochastic Models for Performance Evaluation, Proceedings of the 4th European Performance Engineering Workshop (EPEW 2007), Berlin, Germany, September 27-28, 2007*, Nr. 4748 der Lecture Notes in Computer Science, Heidelberg, Germany, September 2007. Springer, S. 125–140.

[peps] Pepsy. `http://www4.informatik.uni-erlangen.de/Projects/PEPSY/en/pepsy.html`.

[rpro] R Project. `http://www.r-project.org/`.