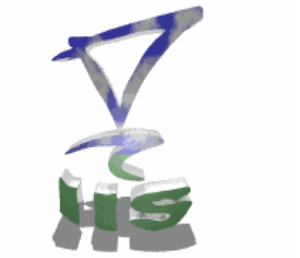
RUHR-UNIVERSITÄT BOCHUM



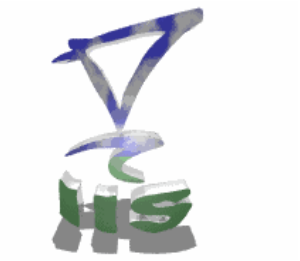
ARBEITSGRUPPE
INTEGRIERTE INFORMATIONSSYSTEME

BACHELORARBEIT

Entwurf und Implementierung
eines automatisierten Mess- und Auswertungssystems
zur Untersuchung des Lastverhaltens von
Multiprocessing Modulen des Apache Webservers

Christian Josef Bell

RUHR-UNIVERSITÄT BOCHUM



ARBEITSGRUPPE
INTEGRIERTE INFORMATIONSSYSTEME

BACHELORARBEIT

Entwurf und Implementierung
eines automatisierten Mess- und Auswertungssystems
zur Untersuchung des Lastverhaltens von
Multiprocessing Modulen des Apache Webservers

vorgelegt von:	Christian Josef Bell
Matrikelnummer:	108 XXX XXXXXX

Bearbeitungszeitraum:	27.04.2007 - 27.08.2007
1. Prüfer:	Prof. Dr.-Ing. Y. Tüchelmann
2. Prüfer:	Prof. Dr. rer. nat. J. Schwenk
Betreuer:	Dipl.-Ing. A. Westhoff

Abstract

Es wird ein automatisiertes Mess- und Auswertungssystem entwickelt, welches es ermöglicht, den Apache Webserver bezüglich seines Lastverhaltens zu untersuchen. Zunächst wird das Konzept, die Modellierung und die Implementierung erklärt. Anschließend folgt eine Erklärung zur Handhabung des Programmes und zur Durchführung von Messungen. Danach werden die beiden Multiprocessing Module Prefork und Worker des Apache Webservers gegenübergestellt. Exemplarische Messergebnisse zeigen, dass das Worker MPM kürzere Antwortzeiten hat. Die Flaschenhälse Prozessor und Arbeitsspeicher werden aufgedeckt. Abschließend erfolgt ein Ausblick in die mögliche Weiterentwicklung des Programmes.

Inhaltsverzeichnis

1 Einleitung **13**

2 Konzept **15**

2.1 Das Testsystem 15

2.1.1 Der Server 15

2.1.2 Der Lastgenerator 16

2.1.3 Der Datenbankserver 17

2.2 Der Messplan 17

2.3 Die Auswertung 17

2.4 Weitere Voraussetzungen an das Programm 18

3 Modellierung und Implementierung **19**

3.1 Die Pakete von JDataMining 19

3.1.1 Das Paket Ant 19

3.1.2 Das Paket Auswertung 24

3.1.3 Das Paket Charts 29

3.1.4 Das Paket CSV 36

3.1.5 Das Paket Export 40

3.1.6 Das Paket GUI 41

3.1.7 Das Paket Hilfsklassen 44

3.1.8 Das Paket JMeter 48

3.1.9 Das Paket Logs 50

3.1.10 Das Paket MFW 51

3.1.11 Das Paket Server 52

3.1.12 Das Paket Threads 55

3.2 Änderungen an Fremdpaketen 58

3.2.1 Änderungen an JFreeChart 58

3.2.2 Änderungen an POI 58

3.2.3 Änderungen an apmfw.c 59

4 Benutzung des Programmes 63
4.1 Installation . . . 63
4.1.1 Installation der Datenbank . . . 63
4.1.2 Einrichtung des Lastgenerators . . . 64
4.1.3 Einrichtung des Servers . . . 64
4.1.4 Erster Start . . . 65
4.2 Funktionen des Programmes . . . 65
4.2.1 Register: Messungen . . . 65
4.2.2 Register: Ergebnisse . . . 68
4.2.3 Register: Querauswertung . . . 71
4.2.4 Register: Einstellungen . . . 72
4.3 Durchführen einer Messung . . . 74

5 Auswertung 77
5.1 Unterschiede zwischen Prefork und Worker . . . 77
5.2 Einschub: Erkenntnisse beim Prefork-MPM . . . 79
5.2.1 Leader-Follower-Implementierung . . . 81
5.2.2 Problemfall Prozesszerstörung . . . 81
5.3 Auflösungstest . . . 86
5.3.1 Zeitauflösung Prefork . . . 86
5.3.2 Zeitauflösung Worker . . . 88
5.4 Auswertung des Prefork-Moduls . . . 88
5.4.1 Standardparameter . . . 88
5.4.2 Stresstest . . . 88
5.4.3 Langzeittest . . . 91
5.4.4 Langzeittest mit Zusatzlast . . . 93
5.5 Auswertung des Worker-Moduls . . . 95
5.5.1 Standardparameter . . . 95
5.5.2 Stresstest . . . 97
5.5.3 Langzeittest . . . 100
5.5.4 Langzeittest mit Zusatzlast . . . 101
5.6 Gemeinsamkeiten von Prefork und Worker . . . 102
5.7 Unterschiede von Prefork und Worker . . . 104
5.8 Zusammenfassung . . . 108

6 Ausblick **111**
6.1 Verbesserungen und Erweiterungen an JDataMining . . . 111
6.2 Verbesserungen und Erweiterungen am Messframework . . . 112
6.3 Weitere Untersuchungen und Messungen . . . 113

Literaturverzeichnis **115**

7 Anhang **117**
7.1 Quelltexte . . . 117
7.1.1 Paket Ant . . . 117
7.1.2 Das Paket Auswertung . . . 130
7.1.3 Das Paket Charts . . . 142
7.1.4 Das Paket CSV . . . 163
7.1.5 Das Paket GUI . . . 172
7.1.6 Das paket Hilfsklassen . . . 205
7.1.7 Das Paket JMeter . . . 216
7.1.8 Das Paket Logs . . . 218
7.1.9 Das Paket MFW . . . 219
7.1.10 Das Paket Server . . . 228
7.1.11 Das Messframework apmfw.c . . . 244
7.1.12 Die Makefile . . . 248
7.2 Inhalte der DVDs . . . 249
7.2.1 DVD 1 . . . 249
7.2.2 DVD 2 . . . 250
7.2.3 DVD 3 . . . 250

Abbildungsverzeichnis

3.1 UML-Klassendiagramm: Die Klasse GUI und ihre Partner 20
3.2 AntStarter 21
3.3 AntWriter 22
3.4 HTMLOverview 24
3.5 PNG 25
3.6 Statistics 26
3.7 ClearableYIntervalSeries 30
3.8 FastXYPlot 31
3.9 InterpolatedDeviationRenderer 32
3.10 JdbcChartListPanel 33
3.11 JdbcChartPanel 34
3.12 JdbcMeanAndStandardDeviationXYDataset 35
3.13 CSV 37
3.14 CSVTableGenerator 37
3.15 MySQLToCSV 40
3.16 MySQLToExcel 41
3.17 ActionListenerClass 42
3.18 ListSelectionListenerClass 43
3.19 DateConverter 44
3.20 DBConnection 45
3.21 ParameterParser 46
3.22 PropertiesSingleton 47
3.23 Querauswertung 48
3.24 JMeterTableGenerator 48
3.25 JMXReader 49
3.26 LogsTableGenerator 50
3.27 MFWPoint, Cut und Results 51
3.28 ValueComputation 52

3.29 Berechnung der Werte in ValueComputation.calculateAndSaveValues() 53
3.30 HttpdMPMConfReader 54
3.31 Parameter 55
3.32 AlleMessungenAuswertenThread 56
3.33 MessungThread 56
3.34 TabelleAuswertenThread 57

4.1 Registerfenster Messungen 66
4.2 Registerfenster Ergebnisse 69
4.3 Registerfenster Querauswertung 71
4.4 Registerfenster Einstellungen 73

5.1 Apache Prefork MPM 78
5.2 Apache Worker MPM 80
5.3 perform_idle_server_maintenance - Teil 1 83
5.4 perform_idle_server_maintenance - Teil 2 85
5.5 90%-Quantile der Requestantwortzeit bei Stresstests 90
5.6 90%-Quantile des freien Arbeitsspeichers bei Stresstests 91
5.7 Aktive Requests beim Prefork MPM mit Standardparametern und 800 Clients (Stresstest) 92
5.8 Requestantwortzeit - Standardparameter - 25 Clients 94
5.9 90%-Quantile der Requestantwortzeit mit Standardparametern 95
5.10 Requestantwortzeit(Prefork) - Standardparameter - 75 Clients - 30 Zusatzclients 96
5.11 Requestantwortzeit des Worker MPM bei 1000 Clients und Standardparametern 98
5.12 Messpunkt 39_Lock_Mutex 99
5.13 Requestantwortzeit - Standardparameter - 25 Clients 101
5.14 90%-Quantile der Requestantwortzeit mit Standardparametern 102
5.15 Requestantwortzeit - Standardparameter - 75 Clients - 30 Zusatzclients 103
5.16 90%-Quantile bei Langzeittests des Prefork MPM mit MaxSpareServers 20 . . 104
5.17 90%-Quantile bei Langzeittest des Worker MPM mit MaxSpareThreads 150 . . 105
5.18 Freier Speicher beim Worker MPM, Stresstest, Standardparameter, 1000Clients 106
5.19 Freier Speicher beim Worker MPM, Stresstest, Standardparameter, 1000Clients 107

Tabellenverzeichnis

5.1 Zeitauflösung beim Prefork MPM 87
5.2 Zeitauflösung beim Worker MPM 89

Abkürzungsverzeichnis

Abkürzung	Bedeutung
APR	Apache Portable Runtime
CGI	Common Gateway Interface
COD	Char Of Death
CSV	Comma Separated Values
GUI	Grapical User Interface
MFW	Messframework
MPM	Multiprocessing Modul
PID	Process IDentifier
POD	Pipe Of Death
TID	Thread IDentifier

1 Einleitung

Im Rahmen dieser Bachelorarbeit wird ein automatisiertes Mess- und Auswertungssystem für den Apache Webserver entwickelt, welches anschließend zur Untersuchung der beiden Multiprocessing Module (MPMs) Prefork und Worker des Apache Webservers dienen wird. Die Ergebnisse werden einander gegenübergestellt und analysiert.

Der Apache Webserver ist ein freier, kostenloser und etablierter Webserver. Er wird von der Apache Software Foundation[1] entwickelt. Auch wenn seine Zahlen seit November 2005 rückläufig sind, so hat er doch mit 50,92 % (Stand: August 2007) den derzeit größten Marktanteil (siehe [Ltd07]).

Für den Betrieb des Apache Webservers stehen verschiedene Module zur Verfügung. Läuft der Apache Webserver beispielsweise unter Linux, so kann zwischen dem prozessbasierten Prefork MPM, dem hybriden (prozess- und threadbasierten) Worker MPM und dem noch experimentellen ereignisbasierten Event MPM gewählt werden. Das Prefork MPM wurde bereits in [Ges05] untersucht. Das Worker MPM wurde in [BNO+07] ausführlich untersucht. Die Analysen zeigten, dass eine manuelle Untersuchung sehr aufwendig ist: Einstellungen an den Servern und den Lastgeneratoren mussten per Hand eingegeben werden, die Messungen mussten manuell gestartet und gestoppt werden, Grafiken mussten in mühsamer Arbeit erstellt werden. Es liegt also nahe, ein System zu entwickeln, welches alle diese Arbeiten und darüber hinaus noch einige mehr dem Anwender abnimmt. Dieses System wird in dieser Bachelorarbeit entwickelt. Es trägt den Namen JDataMining.

Kapitel 2 beginnt mit einer kurzen Beschreibung des Konzeptes von JDataMining. Danach wird in Kapitel 3 die Modellierung und Implementierung erläutert. In Kapitel 4 folgt dann die Installation und Benutzung des Programmes. Es wird auch erklärt, wie eine Messung durchzuführen ist.

Obwohl Prefork MPM und Worker MPM wie soeben erwähnt schon untersucht wurden, gab es noch keinen Vergleich der beiden MPMs untereinander. Daher wurden diese beiden Module mit JDataMining gemessen. Die Auswertung der Ergebnisse befindet sich in Kapitel 5. Darüber hinaus werden dort noch einige Beobachtungen bezüglich der Implementierung der MPMs beschrieben.

[1] Offizielle Seite der Apache Software Foundation: http://www.apache.org/

In Kapitel 6 wird zum Abschluss ein Ausblick geliefert, was an JDataMining noch verbessert werden könnte, was man hinzufügen könnte und wie das Messframework erweitert werden könnte. Es werden auch Vorschläge für weitere Messungen gemacht.

Im Anhang befindet sich eine Auflistung des Inhaltes der beiliegenden DVD sowie alle Quellcodes von JDataMining. Als besonderer Hinweis sei erwähnt, dass diese Arbeit als farbige PDF-Version auf der DVD enthalten ist, was die Betrachtung der Abbildungen (insbesondere Graphen und Screenshots) erleichtert.

Zum Abschluss dieser Einleitung sei noch darauf hingewiesen, dass bei Funktionen die Parameter und Rückgabewerte nur angegeben werden, falls sie zum Verständnis des Textes notwendig sind. Auszüge aus den Quelltexten werden gegebenenfalls um Debugausgaben etc. gekürzt.

2 Konzept

Um ein automatisiertes Mess- und Auswertungssystem für den Apache Webserver zu entwickeln, bedarf es zunächst einer konzeptionellen Phase. Es muss spezifiziert werden, was genau gewünscht ist, was das Programm können muss und was das Programm nicht können muss. Dieses Kapitel beschreibt diese Spezifikationen.

2.1 Das Testsystem

Das Programm soll so entwickelt werden, dass es ein System aus Server, Datenbankserver und Lastgenerator messen kann. Speziell im Falle dieser Arbeit stand ein Testsystem bestehend aus drei Rechnern und einem Switch zur Verfügung. Dies soll hier kurz beschrieben werden:

Das Testsystem besteht aus Server, Lastgenerator, Datenbank und einem Switch, wobei die Datenbank während einer Messung nicht verwendet wurde, so dass dieser Rechner vor dem Messen heruntergefahren wurde, damit er keinen Einfluss auf das Netzwerk hat.

Der Server ist ein AMD Sempron 3100+ mit 1800 MHz, 256 KB Cache und 512 MB RAM. Der Lastgenerator ist ebenfalls ein AMD Sempron 3100+ mit 1000 MHz, 256 KB Cache und 512 MB RAM.

Der Lastgenerator sendet Anfragen an den Server, welcher die Anfragen entgegennimmt und bearbeitet und seine Antwort anschließend an den Lastgenerator zurücksendet. Während dieses Prozesses sollen Daten über die Dauer verschiedener Aktionen des Servers, über die Systemauslastung von Server und Client, über die Requestantwortzeit, über die Speicherauslastung des Servers und des Clients sowie über den Netzwerkverkehr gesammelt werden. Die gewonnenen Informationen müssen in einer Datenbank abgespeichert und verarbeitet werden. Anschließend sollen die Daten in Form von Diagrammen und statistischen Größen ausgewertet werden können.

2.1.1 Der Server

Der Quellcode des Apache Webserver muss mit Messpunkten instrumentalisiert werden, um Informationen aus dem Kern des Apache Webservers gewinnen zu können. Hierzu wird ein

Messframework benötigt, welches Zeitwerte direkt aus dem Apache Quellcode heraus erfassen kann. Des Weiteren soll ein Verfahren entwickelt werden, welches es ermöglicht eine Wartezeit und eine Prozessorverzögerung (siehe Processing-Delay) in den Apache Webserver einzubauen, um die Generierung einer Webseite zu simulieren.

Wartezeit

Die Wartezeit schreibt dem Server vor, wie lange er während einer Anfragebearbeitung warten soll. Damit soll eine Datenbankabfrage simuliert werden. Bei einer Datenbankabfrage wartet nämlich normalerweise der Prozess/Thread auf die Antwort der Datenbank. Anders als bei der Processing-Delay wird dabei nicht der Prozessor belastet (siehe Processing-Delay).

Processing-Delay

Bei der Processing-Delay handelt es sich um einen Zeitwert, welcher die Dauer einer Belastung des Prozessors angibt. Hierzu soll ein Verfahren entwickelt werden, welches es ermöglicht, den Prozessor für eine bestimmt Zeit zu belasten. Der Sinn hinter der Processing-Delay ist, dass so eine Generierung einer Webseite simuliert werden kann. Es wird also ein System für statische Webseiten gemessen, welches jedoch ein Verhalten eines Systems für dynamische Webseiten aufweisen soll.

2.1.2 Der Lastgenerator

Der Lastgenerator soll Anfragen von Clients simulieren.

Auf dem Lastgenerator soll eine grafische Benutzeroberfläche laufen, mit welcher der Anwender alle Einstellungen einer Messung vornehmen kann. Vom Lastgenerator aus werden die gesamten Messungen gesteuert. Auch der Server wird vom Lastgenerator aus gesteuert.

Es müssen die Konfigurationsdateien des Webservers eingelesen werden können, damit die entsprechenden Parameter des Servers variiert werden können. Darüber hinaus sollten die Parameter des Lastgenerators generisch verarbeitet werden, so dass es später möglich ist, das Messsystem für verschiedene Anwendungen zu verwenden.

Das Auswertungssystem (siehe auch 2.3) läuft ebenfalls auf dem Lastgenerator ab. Die gewonnenen Informationen werden von hier aus verarbeitet und zur Datenbank geschickt. Außerdem sollen die gewonnenen Ergebnisse grafisch veranschaulicht werden.

2.1.3 Der Datenbankserver

Auf dem Datenbankserver werden alle gewonnenen Informationen gespeichert. Die Datenbank soll nicht während einer Messung verwendet werden.

Es soll auch möglich sein, die Datenbank nicht extra auf einem eigenen Rechner zu betreiben. Sie darf auch auf dem Lastgenerator oder auf dem Server laufen. Hierbei sollte jedoch beachtet werden, dass dies die Messung erheblich beeinflussen könnte.

2.2 Der Messplan

Bevor eine Messung durchgeführt werden kann, muss ein Messplan erstellt beziehungsweise generiert werden. Eine einzelne Messung sollte wie folgt ablaufen:

1. Parameter des Servers einstellen
2. Lastgenerator-Parameter einstellen
3. Server starten
4. eventuell kurze Zeit warten, damit der Server sich nicht noch in der Initialisierungsphase befindet
5. Skripte zum Messen der Systemauslastung etc. auf Server und Client starten
6. Anfragen simulieren
7. Skripte zum Messen der Systemauslastung etc. auf Server und Client stoppen
8. Server stoppen
9. Gesammelte Daten verarbeiten

2.3 Die Auswertung

Die gesammelten Informationen müssen ausgewertet werden. Hierzu soll es die Möglichkeit geben, dass entsprechende Graphen zu jedem Messpunkt automatisiert erzeugt werden können. Des Weiteren sollen statistische Größen berechnet werden können, zum Beispiel 90%-Quantile, Mittelwerte, Minimum, Maximum. Diese statistischen Werte sollen zusammen mit den generierten Graphen in einer Übersichtsseite dargestellt werden.

Darüber hinaus soll es die Möglichkeit einer Querauswertung geben, das heißt, dass Werte über mehrere Messungen verteilt verglichen werden können sollten.

2.4 Weitere Voraussetzungen an das Programm

Das Programm soll gut erweiterbar sein, damit es leicht möglich ist, das Programm eventuell auf andere Serversysteme und/oder Lastgeneratoren umzustellen. Insbesondere ist es wichtig, dass möglichst alle Parameter generisch eingelesen werden, so dass der Benutzer flexibel Konfigurationsdateien und Lastpläne nach seinen Bedürfnissen einstellen kann.

3 Modellierung und Implementierung

In diesem Kapitel wird zum einen die Modellierung von JDataMining beschrieben, indem die Klassen in Form von UML-Klassendiagrammen dargestellt werden. Zum anderen wird direkt zu jeder Klasse die zugehörige Implementierung beschrieben. Darüber hinaus werden die Zusammenhänge in Form eines UML-Diagrammes erklärt und die Einteilung der Klassen in Pakete.

Alle UML-Diagramme wurden mit dem Eclipse-Plugin EclipseUML 2007 Europa Free Edition[1] generiert.

Bei Start des Programmes JDataMining wird zuallererst die Klasse GUI ausgeführt. Abbildung 3.1 zeigt, welche Klassen in direktem Bezug zur Klasse GUI stehen. Diese Grafik soll als Einstiegspunkt in dieses Kapitel dienen. Die Zusammenhänge dieser Klassen und der wiederum damit in Bezug stehenden Klassen werden im Folgenden beschrieben.

3.1 Die Pakete von JDataMining

Bei der Modellierung wurde auf eine paketorientierte Sichtweise Wert gelegt, da so die einzelnen Klassen direkt verschiedenen Themen zugeordnet werden können. Sollte der Wunsch bestehen, das Programm zu erweitern oder zu verändern, ist das Auffinden von Klassen so um einiges erleichtert. Nachfolgend werden nun in chronologischer Reihenfolge die einzelnen Pakete mit ihren zugehörigen Klassen erklärt.

3.1.1 Das Paket Ant

Das Paket *Ant* beinhaltet alle relevanten Klassen zum Umgang mit Apache Ant.

[1] Herstellerseite: http://www.omondo.de

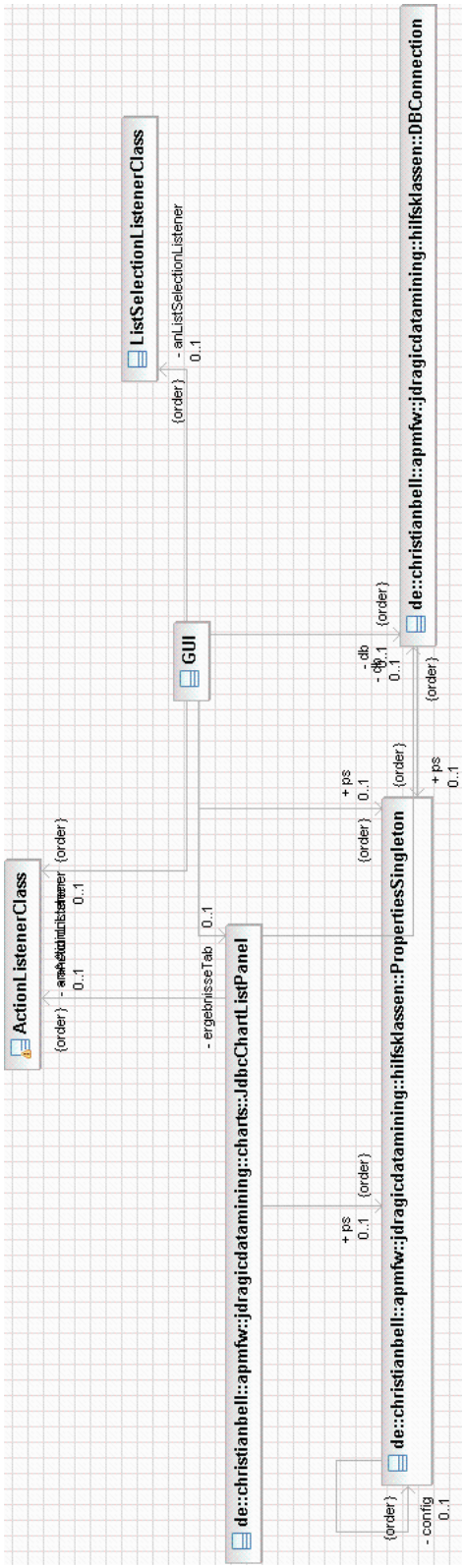


Abbildung 3.1: UML-Klassendiagramm: Die Klasse GUI und ihre Partner

AntStarter

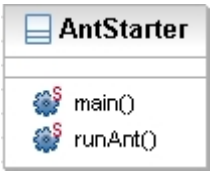


Abbildung 3.2: AntStarter

Diese Klasse beinhaltet die Ausführung einer Ant-Datei und zwar abhängig vom aktuellen Betriebssystem. Zum Stand der Drucklegung dieser Arbeit (August 2007) werden bereits Windows 95, Windows 98, Windows XP und Linuxsysteme unterstützt. Ihre Modellierung ist der Abbildung 3.2 zu entnehmen.

Die Funktion *main()* wird nur zu Testzwecken zum direkten Ausführen der Klasse ohne GUI benötigt.

Die Funktion *runAnt()* führt Ant mit der Standarddatei *build.xml* aus. Dies funktioniert wie folgt: Zunächst wird mit Hilfe der Funktion *java.lang.System.getProperty(String key)* der Name des aktuellen Betriebssystems ausgelesen. Hierzu wird dieser Funktion der Parameter *os.name* übergeben. Die Funktion liefert dann den Namen in Form eines Strings zurück. Nun wird überprüft, ob der String den Teilstring Windows 95, Windows 98, Windows XP oder Linux enthält. Trifft einer dieser Fälle zu, so wird entsprechend dem Betriebssystem ein Präfix gewählt. Für Windows 95/98 ist das *"command.com /c"*, für Windows XP ist es *"cmd /c"* und für Linux lautet das Präfix *"sh -c"*. Das jeweilige Präfix wird vor den Teilstring *"ant -f build.xml"* konkateniert und der neu entstandene String als Parameter der Funktion *java.lang.Runtime.exec(String command)* übergeben. Diese Funktion führt den übergebenen String als Befehl in einem separaten Thread aus. Die Ausgabe des gestarteten Programmes wird als *InputStream* eingelesen, gepuffert und in der Konsole der GUI ausgegeben.

AntWriter

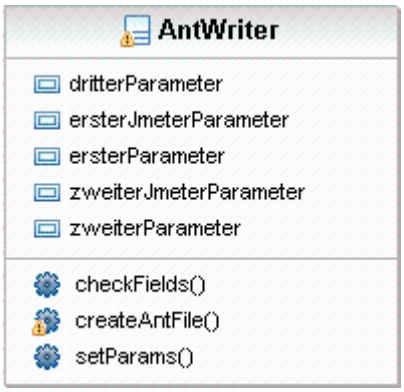


Abbildung 3.3: AntWriter

Die Klasse *AntWriter* ist für die Erzeugung der build.xml-Datei zuständig. Die Modellierung in Abbildung 3.3 zeigt alle ihre Funktionen auf.

Die Funktion *checkFields()* überprüft die Angaben des Benutzers, welche in der GUI gemacht wurden, auf Richtigkeit und Verwendbarkeit für die Erzeugung der *build.xml*. Als erstes ruft sie die Funktion *setParams()* auf, welche weiter unten beschrieben wird. Es wird also versucht, die nötigen Parameter einzulesen und die entsprechenden Variablen damit zu belegen. Sollte dies bereits nicht gelingen, so kann keine Ant-Datei erzeugt werden. Gelingt dies aber, so kann die Funktion *checkFields()* nun alle Parameter auf ihre Richtigkeit überprüfen. Nur wenn alle Parameter gültig sind, dann kann die Ant-Datei erzeugt werden.

Die Funktion *createAntFile()* erzeugt die build.xml-Datei. Nachdem sie überprüft hat, ob die Funktion *checkFields()* erfolgreich war, kann sie mit dem Erstellen der build.xml beginnen. Dazu wird zunächst die XML-Grobstruktur erzeugt. Diese sieht beispielsweise so aus:

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>")
<project name=\"JMeter\" default=\"jmeter\" basedir=\".\">
  <target name=\"jmeter\">
    <taskdef name=\"jmeter\"
     classname=\"org.programmerplanet.ant.taskdefs.
                jmeter.JMeterTask\" />
    <taskdef resource="net/sf/antcontrib/antlib.xml"/>
  </target>
</project>
```

Nun wird dieses Grundgerüst als Dokument vom *SAXBuilder* eingelesen. Entsprechend der gewählten Parameter werden innerhalb von acht geschachtelten For-Schleifen die nötigen Tags

und Attribute in das Dokument eingefügt. Das Verfahren wird an dieser Stelle als Pseudocode dargestellt. Die Implementierung findet sich wie alle Quellcodes im Anhang dieser Arbeit.

```
FOR wartezeit <= wartezeitStop
 FOR processingDelay <= processingDelayStop
  Server- und JMeter-Parameter initialisieren;
  FOR ersterServerParameter <= ersterServerParameterStop
   Ersten Server-Parameter aktualisieren;
   FOR zweiterServerParameter <= zweiterServerParameterStop
    Zweiten Server-Parameter aktualisieren;
    FOR dritterServerParameter <= dritterServerParameterStop
     Dritten Server-Parameter aktualisieren;
     Datei httpd-mpm.conf mit Parameterwerten beschreiben
        und auf Server kopieren;
     FOR ersterJMeterParameter <= ersterJMeterParameterStop
      Ersten JMeter-Parameter aktualisieren;
      FOR zweiterJMeterParameter
            <= zweiterJMeterParameterStop
       Zweiten JMeter-Parameter aktualisieren;
       FOR wiederholungen >= 0
        Id erhöhen;
        Task erzeugen: Server starten;
        Task erzeugen: Log.sh auf Server starten;
        Task erzeugen: Log.sh auf Client starten;
        Task erzeugen: Jmeter-Testplan;
        Task erzeugen: Log.sh auf Server stoppen;
        Task erzeugen: Log.sh auf Client stoppen;
        Task erzeugen: Server stoppen;
        Task erzeugen: CSV-Datei von Server
                       auf Client kopieren;
        Task erzeugen: CSV-Datei eindeutig benennen;
        Task erzeugen: CSV-Datei von Server löschen;
        Task erzeugen: Alle Log-Dateien von
                       Server auf Client kopieren;
        Task erzeugen: Alle Log-Dateien
                       von Server löschen;
```

```
            JMX-Datei erstellen;
          FOR_END
        FOR_END
      FOR_END
    FOR_END
  FOR_END
 FOR_END
FOR_END
```

Die Funktion *setParams()* belegt die Parameter, welche über die GUI eingestellt werden, mit den dort angegebenen Startwerten. Außerdem speichert sie zu den Parametern den Stoppwert, die Schrittweite und den aktuellen Wert.

3.1.2 Das Paket Auswertung

Dieses Paket beinhaltet Klassen, welche zur Auswertung der Messdaten dienen.

HTMLOverview

Abbildung 3.4: HTMLOverview

Die Klasse *HTMLOverview* (siehe Abbildung 3.4) generiert eine HTML-Seite mit statistischen Werten und verlinkten Bildern zu den ihr übergebenen Messungen. Sie hat als einzige Funktion die Funktion *createHTML(Vector<String> tabellenNamen, Vector<String> messpunktNamen, String fileStr)*. Diese Funktion erwartet als Parameter einen Vektor, welcher die Namen der Tabellen (also der Messungen) in der Datenbank, die ausgewertet werden sollen, enthält. Ebenso erwartet die Funktion einen Vektor bestehend aus den auszuwertenden Messpunkten. Als dritter Parameter muss der Ort, an dem die HTML-Seite gespeichert werden soll, als String übergeben werden.

Als erstes wird die HTML-Datei im angegebenen Verzeichnis gespeichert. Als *RandomAccessFile* wird sie danach bearbeitet und zwar, indem sie zeilenweise beschrieben wird. Hierfür sind zwei ineinander geschachtelte While-Schleifen zuständig. Die äußere Schleife durchläuft

die Tabellennamen, während die innere die Messpunkte durchläuft. Es wird zu jedem Messpunkt in jeder Tabelle ein Eintrag in der HTML-Seite erzeugt. Die Werte wie Mittelwert, Minimum, Maximum und 90%-Quantil erhält die Seite durch das jeweilige Aufrufen der Berechnungsfunktion der Klasse *Statistics* (siehe unten). Für den Fall, dass es sich um keine Tabelle mit Messpunkten handelt, sondern beispielsweise um eine aus einer JTL-Datei erzeugten Tabelle, fällt die innere Schleife natürlich weg.

PNG

Abbildung 3.5: PNG

Die Funktion *createPNG(File file, JDBCMeanAndStandardDeviationXYDataset dataset)* der Klasse *PNG* erzeugt aus einem ihr übergebenen Datensatz (zweiter Parameter) eine PNG-Grafik mit dem Dateinamen des ersten Parameters. Zuerst übergibt diese Funktion den Datensatz an die Funktion *createChart()* der Klasse *JdbcChartPanel* (siehe unten), welche einen Chart zurückliefert. Dieser Chart wird dann der Funktion *org.jfree.chart.ChartUtilities.saveChartAsPNG()* übergeben, die den Chart als PNG-Datei speichert.

Statistics

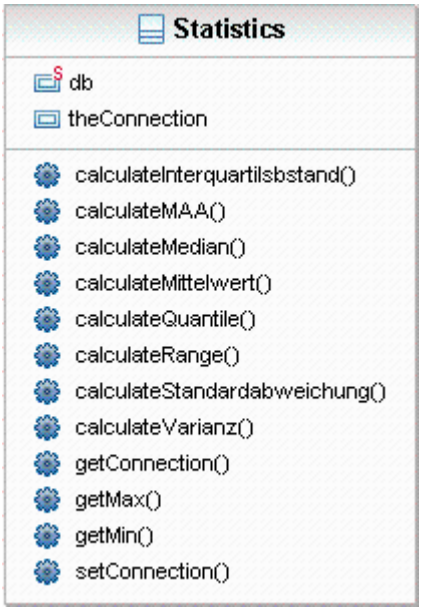


Abbildung 3.6: Statistics

Die Klasse *Statistics* bietet einige Funktionen zur Berechnung von statistischen Werten. Die Funktionen im Einzelnen:

Die Funktion *calculateInterquartilsabstand()* berechnet den Abstand zwischen dem 75%-Quantil und dem 25%-Quantil. Hierzu ruft sie die Funktion *calculateQuantile(double p, String table, String messpunkt)* einmal mit p = 0.75 und einmal mit p = 0.25 auf. Die Ergebnisse werden voneinander subtrahiert.

Die mittlere absolute Abweichung wird von der Funktion *calculateMAA()* berechnet. Es handelt sich dabei um ein Streuungsmaß, welches die mittlere absolute Abweichung zum Median angibt. Zunächst wird daher die Funktion *calculateMedian()* zur Berechnung des Medians aufgerufen. Dieser Wert wird zwischengespeichert. Nun wird in einer While-Schleife der Median von jedem Messwert subtrahiert, die jeweilige Differenz wird vorzeichenlos (also als absolute Zahl) zu den anderen Differenzen addiert. Gleichzeitig wird in der While-Schleife die Anzahl der Messwerte gezählt. Im Anschluss an die While-Schleife wird die Summe durch die Anzahl der Messwerte geteilt. Die folgende Funktion sollte diese Berechnung besser veranschaulichen

(N = Anzahl Messwerte, Xi = i-ter Messwert, Med = Median):

$$MAA = \frac{1}{N} \sum_{i=1}^{N} |x_i - Med|$$

Die Funktion *calculateMedian()* berechnet den Median. Als erstes werden die Messwerte aufsteigend nach Größe sortiert und die Anzahl der Messwerte errechnet. Dann wird überprüft, ob die Anzahl der Messwerte gerade oder ungerade ist. Sollte sie ungerade sein, so wird der Wert an der Stelle (n + 1) / 2 als Median ausgewählt. Sollte die Zahl gerade sein, dann werden die Werte an der Stelle n / 2 und an der Stelle (n + 1) /2 addiert und die Summe halbiert. Das Ergebnis ist der Median. Folgende Formeln zeigen noch einmal die Berechnungen:

Falls die Anzahl Messwerte gerade ist:

$$Med = \frac{1}{2}(x_{\frac{n}{2}} + x_{\frac{n+1}{2}})$$

Falls die Anzahl Messwerte ungerade ist:

$$Med = x_{\frac{n+1}{2}}$$

Von der Funktion *calculateMittelwert()* wird das arithmetische Mittel bestimmt. Alle Messwerte werden aufaddiert und durch die Gesamtanzahl geteilt. Dies ist die zugehörige mathematische Formel:

$$\bar{X} = \frac{1}{N} \sum_{i=1}^{N} x_i$$

Die Funktion *calculateQuantile(double p, String table, String messpunkt*) bestimmt das (p * 100)-Quantil, wobei p als Parameter übergeben wird und zwischen 0 und 1 liegen muss. Mit dieser Funktion kann beispielsweise das 90%-Quantil berechnet werden, welches im Auswertungskapitel (siehe Kapitel 5) eine Rolle spielt. Wie auch beim Median[2] werden zuerst die Messwerte aufsteigend nach Größe sortiert und die Anzahl der Messwerte bestimmt. Dann wird die Anzahl der Messwerte mit p multipliziert. Ist das Produkt keine ganze Zahl, so wird die unmittelbar auf das Produkt folgende ganze Zahl als Index verwendet und der Messwert mit diesem Index als Quantil gewählt. Sollte das Produkt dagegen eine ganze Zahl sein, so wird der Messwert an der Stelle, die das Produkt angibt, sowie der darauffolgende Messwert addiert und die entstandene Summe halbiert. Das Ergebnis ist dann das Quantil. Zum besseren Verständnis hier ein Ausschnitt aus dem Quellcode:

[2]Der Median ist eigentlich auch ein Quantil. Er entspricht dem 50%-Quantil.

```
if(messpunkt == null)
{
  result = statement.executeQuery("SELECT value FROM " + table
           + " ORDER BY value");
}
else
{
  result = statement.executeQuery("SELECT value FROM " + table
       + " WHERE mfwpoint='" + messpunkt + "' ORDER BY value");
}
while (result.next()) //Zaehle die Anzahl Messpunkte
{
  anzahl++;
}
//Erst ab dieser Groesse erhaelt man sinnvolle Ergebnisse
if(anzahl > 10)
{
  double perzentil = (double)anzahl*p;
  //Falls nicht ganzzahlig, dann waehle naechst hoehere Zahl
  //Wenn ganzzahlig, dann veraendert diese Funktion nichts
  int ceil = (int)Math.ceil(perzentil);
  result.beforeFirst();
  int count = 0;
  while(result.next())
  {
    count++;
    if (count == ceil)
    {
       break; //Halte an der richtigen Stelle an
    }
  }
  long value = result.getLong("value");
  result.next();
  long value2 = result.getLong("value");
  if(perzentil%1 == 0) //n*p ganzzahlig
  {
```

```
        result.close();
        statement.close();
        return (value+value2)/2;
      }
      else //n*p nicht ganzzahlig
      {
        result.close();
        statement.close();
        return value;
      }
    }
```

Die Funktion *calculateRange()* bestimmt den Abstand zwischen Minimum und Maximum, indem sie das Ergebnis der Funktion *getMin()* von dem Ergebnis der Funktion *getMax()* subtrahiert.

Die Standardabweichung wird von der Funktion *calculateStandardabweichung()* errechnet. Sie ruft dazu nur die Funktion *calculateVarianz()* auf und bildet davon die Wurzel:

```
return Math.sqrt(calculateVarianz(table, messpunkt));
```

Die Funktion *calculateVarianz()* berechnet die Stichprobenvarianz nach folgender Formel:

$$V = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{X})^2$$

Die Funktion *getConnection()* liefert die Datenbankverbindung zurück, die eventuell mit der Klasse *Statistics* aufgebaut wurde.

Das Minimum und das Maximum werden mit den Funktionen *getMax()* und *getMin()* bestimmt. Sie führen lediglich eine SQL-Abfrage durch. SQL liefert das Minimum beziehungsweise das Maximum.

Der Funktion *setConnection()* kann eine Verbindung übergeben werden. So verwendet die Klasse für alle Funktionen die gleiche Datenbankanbindung.

3.1.3 Das Paket Charts

Das Paket *Charts* enthält verschiedene Klassen zur Erzeugung eines Graphen. Alle Klassen (außer der Klasse *FastXYPlot*) wurden bereits von Alexander Hergenroeder entwickelt. Aufgrund der Vollständigkeit und weil einige Details an diesen Klassen geändert wurden, werden sie im Folgenden aufgelistet. Eine ausführlichere Beschreibung der Klassen gibt es in [Her07].

ClearableYIntervalSeries

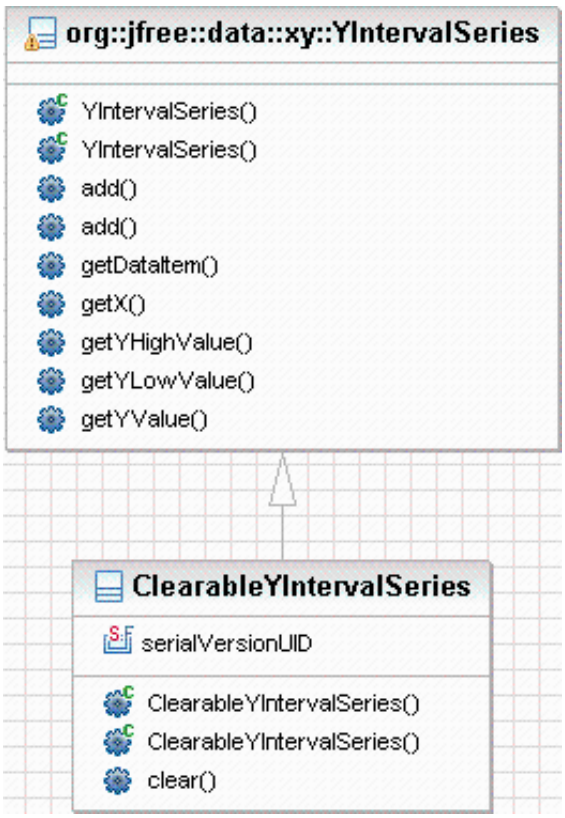


Abbildung 3.7: ClearableYIntervalSeries

An dieser Klasse wurden keine Änderungen vorgenommen. Diese Klasse (siehe Abbildung 3.7) ermöglicht es, die Methode *clear()* der Klasse *YIntervalSeries* von JFreeChart auch von außerhalb aufzurufen. Die Funktion *clear()* ermöglicht das Entfernen von jeglichen Objekten in einer Serie.

FastXYPlot

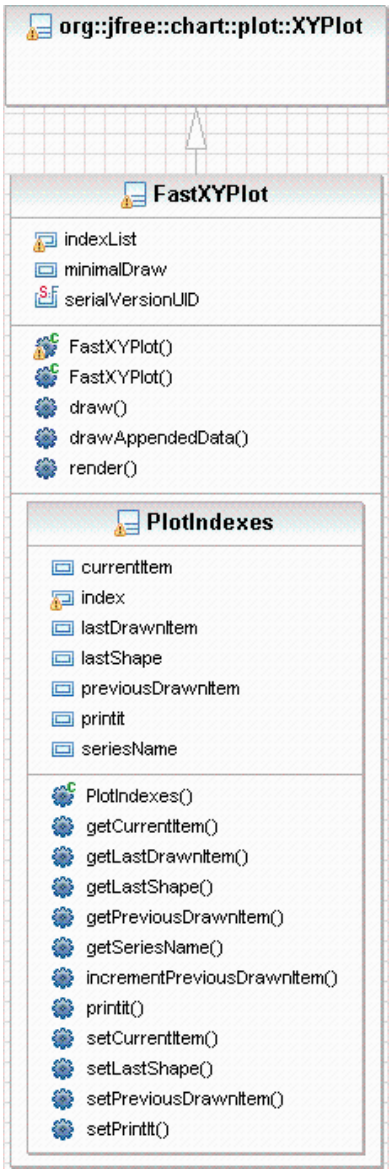


Abbildung 3.8: FastXYPlot

Die Klasse *FastXYPlot* (siehe Abbildung 3.8) ist eine Klasse, die ursprünglich von Lindsay Pender geschrieben wurde (vergleiche [Pen05]) und mittlerweile von der JFreeChart-Community weiter entwickelt wird (siehe [JC06]). Sie wurde bisher jedoch nicht in den Kern von *JFreeChart* eingearbeitet, sondern steht nur als Patch zur Verfügung. Die Klasse beschleunigt das Plotten von Datenpunkten beim Zeichnen eines Charts, indem sie von der Klasse *XYPlot* (aus Platzgründen wurden die Attribute und Methoden dieser Klasse nicht in Abbildung 3.8 dargestellt) erbt und einige Funktionen mit effizienteren Funktionen überschreibt.

InterpolatedDeviationRenderer

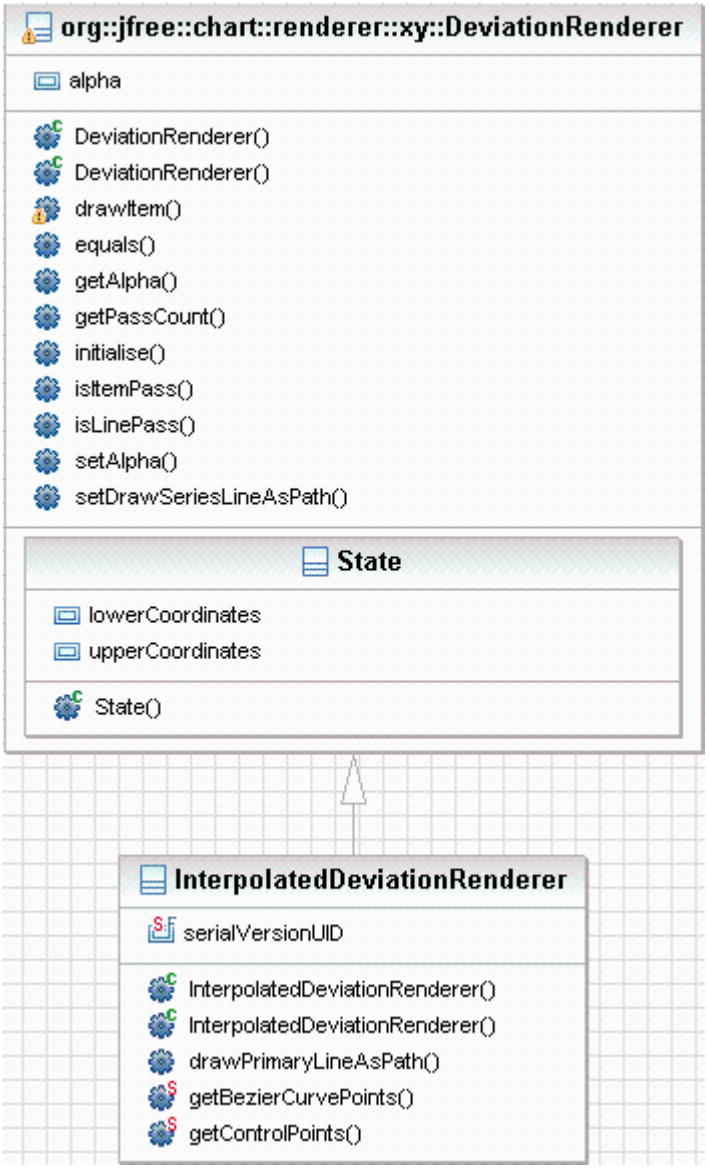


Abbildung 3.9: InterpolatedDeviationRenderer

Diese Klasse (siehe Abbildung 3.9) ist für die Interpolation der Graphen basierend auf dem *DeviationRenderer* zuständig. Sie berechnet die Bézier-Interpolation. An dieser Klasse wurde nichts verändert.

JdbcChartListPanel

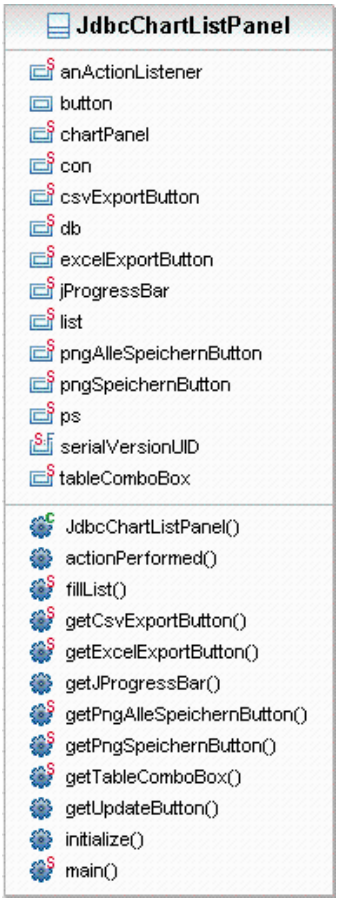


Abbildung 3.10: JdbcChartListPanel

Die Klasse *JdbcChartListPanel* war die Hauptklasse des Projektes [Her07]. Nun ist sie ein Bestandteil des Registers Ergebnisse der GUI. Hinzugefügt wurden vor allem weitere Schaltflächen zur Interaktion mit dem Programm:

- Eine CSV-Export-Schaltfläche zum Exportieren der aktuell gewählten Tabelle in eine CSV-Datei.
- Eine Excel-Export-Schaltfläche zum Exportieren der aktuell gewählten Tabelle in eine Excel-Datei.
- Eine Schaltfläche “Tabelle auswerten” zum Erzeugen einer Auswertungsübersicht in HTML zur aktuell gewählten Tabelle.
- Eine Schaltfläche “Alle Tabellen auswerten” zum Erzeugen einer Auswertungsübersicht in HTML des aktuellen Messnamens.

Darüber hinaus wurde die Funktion der Schaltfläche “Update” so angepasst, dass auch Tabellen ohne Messpunkte (aber mit Messdaten) angezeigt werden können.

JdbcChartPanel

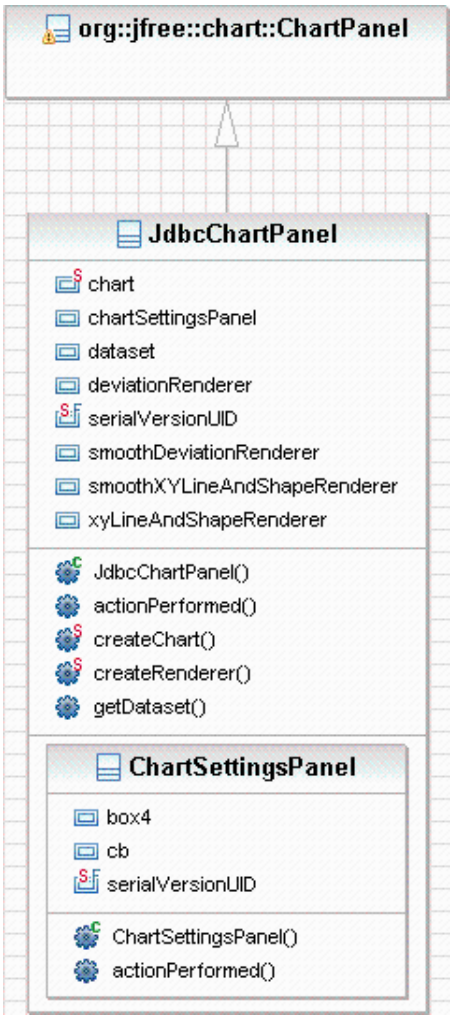


Abbildung 3.11: JdbcChartPanel

Die Klasse *JdbcChartPanel* erbt von der JFreeChart-Klasse *ChartPanel* (wird in Abbildung 3.11 zur Platzersparnis ohne Methoden und Attribute angezeigt). Sie bietet die Möglichkeit zwischen verschiedenen Renderern über ein Menü zu wählen. In dieser Klasse wurde die Funktion *createChart()* um mehrere Fallunterscheidungen erweitert, die je nach gewählter Tabelle eine entsprechende Beschriftung für den Graphen auswählen.

JDBCMeanAndStandardDeviationXYDataset

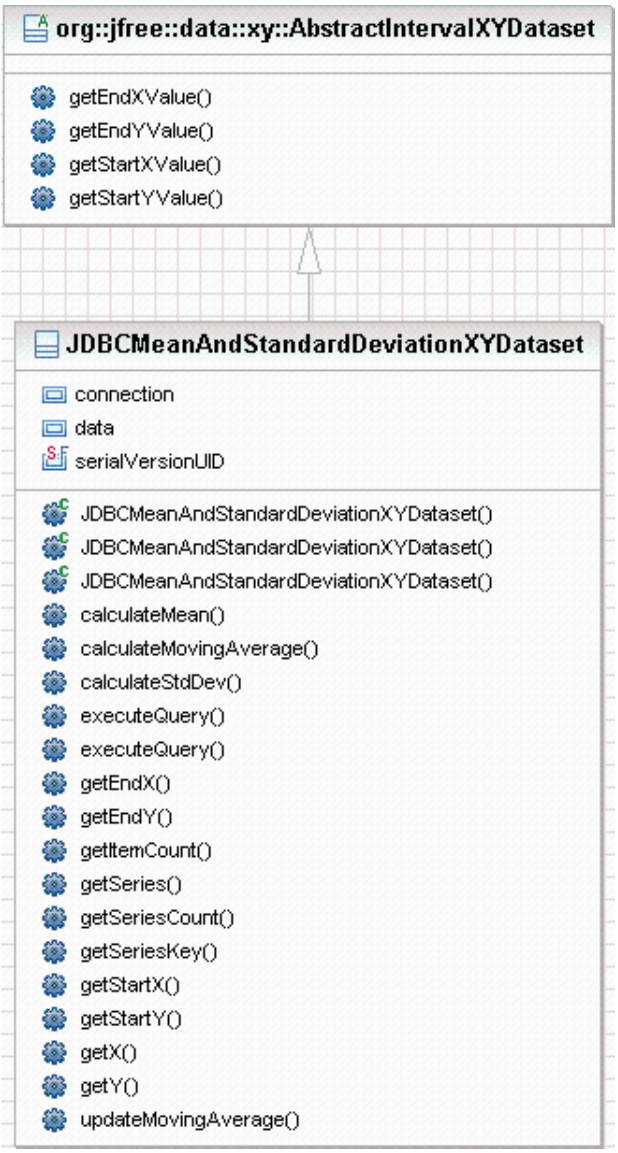


Abbildung 3.12: JdbcMeanAndStandardDeviationXYDataset

Diese Klasse (Abbildung 3.12) erzeugt aus einem Datensatz einer Datenbank Datensätze für einen Chart und zwar für alle drei Serien Mittelwert, gleitender Mittelwert und die normale Verteilung der Werte.

An dieser Klasse wurden zwei wesentliche Dinge geändert, die die Performance erheblich verbessert haben. Zum einen wird in der Funktion *executeQuery()* nun bei der Erzeugung des Objektes *value* vom Typ *ClearableYIntervalSeries* der Parameter *autoSort* auf *false* gesetzt. Dieser Parameter veranlasste *JFreeChart* die Sortierung aller Punkte vorzunehmen. Nun übernimmt dies die MySQL-Datenbank, die für solche Tätigkeiten im Gegensatz zu *JFreeChart*

optimiert ist. Zum anderen wurde in der selben Funktion ein Verfahren implementiert, welches erst beim letzten übergebenen Punkt ein *SeriesChangedEvent* sendet. Zuvor wurden alle Punkte mit der Funktion *YIntervalSeries.add()* nacheinander hinzugefügt und für jeden Punkt wurden entsprechend alle Listener, die an der Serie gelauscht haben, benachrichtigt (siehe hierzu auch die Veränderung an der Klasse *YIntervalSeries* in Abschnitt 3.2.1). Das Verfahren sieht wie folgt aus:

```
while (resultSet.next())
{
  count++;
}
resultSet.first();
int i = 1;
while (i < count)
{
  Double x = resultSet.getDouble(1);
  Double y = resultSet.getDouble(2);
  resultSet.next();
  value.add(x, y, y, y, false );
  i++;
}
if (count > 0)
{
  Double x = resultSet.getDouble(1);
  Double y = resultSet.getDouble(2);
  value.add(x, y, y, y, true );
}
this.data.add(value);
```

Dank dieser beiden Änderungen sowie einer leichten Reduzierung der Auflösung der PNG-Bilder konnte die Performance beispielsweise beim Auswerten von Messungen um geschätzte 75% gesenkt werden.

3.1.4 Das Paket CSV

Das Paket *CSV* beinhaltet zwei Klassen zum Übertragen der Informationen in eine Datenbank.

CSV

Abbildung 3.13: CSV

Die CSV-Klasse (siehe Abbildung 3.13) ist für das Schreiben der Informationen aus der CSV-Datei in die Datenbank zuständig. Im Wesentlichen machen dies die beiden Funktionen *getStartAndStop()* und *loadCSVIntoDatabase()*.

Die Funktion *getStartAndStop()* sucht in der CSV-Datei den Startzeitpunkt und den Stoppzeitpunkt heraus. Dazu wird die CSV-Datei zeilenweise durchlaufen und nach einem Vorkommen von den Strings *"__START__"* und *"__STOP__"* durchsucht. Sobald ein String gefunden wird, wird die zugehörige Zeit zwischengespeichert. Wenn beide Werte gefunden wurden, dann werden beide Werte als String-Array zurückgegeben. Ansonsten wird ein Fehler gemeldet.

Die Funktion *loadCSVIntoDatabase()* liest die CSV ein und speichert jede Zeile, deren Zeitwert zwischen dem Startzeitpunkt und dem Stoppzeitpunkt liegt, in der Datenbank. Hierzu muss sie natürlich zuvor die Funktion *getStartAndStop()* aufgerufen haben. Die Werte werden nicht sofort in die Datenbank geschrieben, sondern zuerst in ein sogenanntes Batch abgelegt. Immer wenn 10.000 Einträge in der Batch liegen, dann wird der Inhalt zur Datenbank gesendet. Dadurch wird ein großer Performancevorteil erzielt, da nicht jeder Wert einzeln zur Datenbank geschickt werden muss. Am Ende der Funktion werden weitere eventuell vorhandene Einträge, die noch in der Batch liegen, zur Datenbank übertragen.

CSVTableGenerator

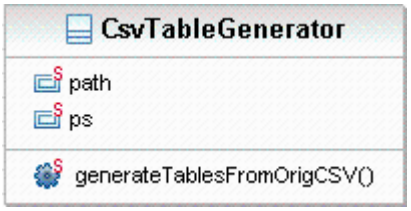


Abbildung 3.14: CSVTableGenerator

Die Klasse *CSVTableGenerator* (siehe Abbildung 3.14) wird aufgerufen, wenn eine oder mehrere CSV-Dateien in die Datenbank geschrieben werden sollen. Dazu an dieser Stelle zur besseren Erklärung ein kurzer Ausschnitt aus dem Quellcode (gekürzt um verschiedene Ausgaben etc.):

Als Erstes werden die benötigten Variablen deklariert:

```
ps = PropertiesSingleton.getInstance();
DBConnection db = new DBConnection();
Connection connection = null;
Statement statement = null;
ResultSet result = null;
ValueComputation calc = new ValueComputation();
CSV csvFile = new CSV();
path = ps.getProperty("csvPath");
File dir = new File(path);
```

Dann wird das Verzeichnis *dir* nach Dateien durchsucht. Alle gefundenen Dateien werden in *entries* gespeichert:

```
File[] entries = dir.listFiles();
if(entries == null)
 System.out.println(dir + " ist kein gueltiges Verzeichnis!");
else if(entries.length <= 0)
  System.out.println(dir + " enthaelt keine Dateien oder
                    Verzeichnisse!");
```

Sollten Dateien in dem Verzeichnis vorhanden gewesen sein, so darf nun eine Verbindung zur Datenbank aufgebaut werden:

```
else
{
  try
    {
      connection = db.getConnection();
      statement =  connection.createStatement();
      calc.setConnection(connection);
    }
    catch ............
```

Nun wird die Schleife alle Einträge durchlaufen. Dabei dürfen jedoch nur Einträge berücksichtigt werden, die einer Datei mit Endung ".csv" entsprechen.

```
for( int i=0; i<entries.length; i++ )
{
  if(!entries[i].isDirectory()
     && entries[i].toString().endsWith(".csv"))
  {
```

Es wird die Funktion *cleanTable()* der Klasse *DBConnection* aufgerufen, welche im Falle einer existierenden Tabelle namens *"orig"*[3] diese Tabelle aus der Datenbank entfernt. Anschließend wird sie neu erstellt. So ist sichergestellt, dass sie erstens leer ist und zweitens die richtige Struktur aufweist:

```
db.cleanTable("orig", statement);
statement.execute("CREATE TABLE IF NOT EXISTS orig (ID
                   BIGINT(20) UNSIGNED AUTO_INCREMENT,
                   PRIMARY KEY (ID), MFWPOINT VARCHAR(100)
                   NOT NULL, TSTAMP BIGINT(20) UNSIGNED
                   NOT NULL, STATE VARCHAR(10) NOT NULL,
                   TIME BIGINT(20) NOT NULL, PID INT(10)
                   UNSIGNED NOT NULL, TID INT(10) UNSIGNED
                   NOT NULL) ENGINE=MyISAM CHARACTER SET
                   utf8 COLLATE utf8_general_ci");
```

Nun wird die aktuelle Datei als CSV-Datei vermerkt und die Funktion *loadCSVIntoDatabase()* (siehe Beschreibung zur Klasse *CSV*) ausgeführt. Somit werden die Werte aus der aktuellen CSV-Datei in die Tabelle namens *"orig"* geladen:

```
csvFile.setCsvFile(entries[i].toString());
csvFile.loadCSVIntoDatabase("orig");
```

Nachdem alle Informationen zur Berechnung in der Tabelle *"orig"* vorliegen, wird mittels einer SQL-Query herausgefunden, welche unterschiedlichen Messpunkte in dieser Tabelle vorhanden sind. Alle Messpunkte (außer Start und Stopp) werden in einen String-Vektor gespeichert, so dass die verschiedenen Messpunktnamen zur Verfügung stehen:

[3] Dies ist die temporäre Tabelle, in der später Werte zur Berechnung zwischengespeichert werden.

```
result = statement.executeQuery("SELECT DISTINCT
                        mfwpoint FROM orig");
Vector<String> v = new Vector<String>();
while(result.next())
{
  if(!result.getString("mfwPoint").contains("__ST"))
     v.add(result.getString("mfwpoint"));
}
```

Mit diesen Informationen können nun mit der Funktion *calculateAndSaveDifferences()* (siehe Beschreibung der Klasse *ValueComputation*) alle Zeitwerte bestimmt werden:

```
calc.calculateAndSaveDifferences(entries[i].toString(), v);
```

3.1.5 Das Paket Export

In diesem Paket befinden sich Klassen zum Export von Daten aus einer Datenbank in ein bestimmtes Format.

MySQLToCSV

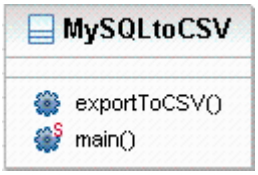


Abbildung 3.15: MySQLToCSV

Die Klasse *MySQLToCSV* exportiert MySQL-Daten in eine CSV-Datei. Die Funktion *exportToCSV()* führt lediglich diese SQL-Query aus:

```
statement.execute("SELECT * FROM " + table + " INTO OUTFILE '"
                  + fileStr + "' FIELDS TERMINATED BY ','");
```

MySQLToXLS

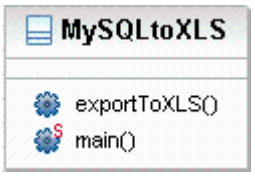


Abbildung 3.16: MySQLToExcel

Die Klasse *MySQLToXLS* exportiert MySQL-Daten in eine XLS-Datei. Die Klasse greift auf Funktionalitäten des Fremdpaketes *org.apache.poi*[4] zu. Dies ist ein Paket, welches es ermöglicht, die verschiedenen Office-Formate von Microsoft aus Javaanwendungen heraus zu erzeugen.

Die Funktion *exportToCSV()* ist für das Exportieren der Daten zuständig. Dazu wird ein sogenanntes *Workbook* mit zunächst einem *Sheet* (Datenblatt) erstellt. Dann wird die Datenbank abgefragt, welche Spalten die Tabelle vorzuweisen hat. Die Spaltennamen werden dann in die erste Zeile des ersten Sheets eingetragen. Anschließend werden die Daten aus der Datenbank gelesen und in einem *ResultSet* gespeichert. In einer While-Schleife wird nun zu jedem Eintrag des ResultSets eine neue Zeile in dem *Sheet* erzeugt und die jeweilige Zeile mit dem Eintrag aus dem *ResultSet* gefüllt. Gleichzeitig wird ein Zähler hochgezählt, der sich die Anzahl der Zeilen merkt. Sollte dieser Zähler größer als 65535 werden, so wird ein weiteres *Sheet* erzeugt und der Zähler wieder auf Null gesetzt. Dies ist notwendig, da Microsoft Excel in den meisten Versionen eine Zeilenbegrenzung von 65536 Zeilen hat. Hinweis: Es können insgesamt nur drei Sheets in einem Workbook erstellt werden. Die Entwickler des Paketes *org.apache.poi* haben diese Begrenzung aus Performancegründen eingebaut.

Nachdem alle Daten in das Workbook geschrieben wurden, wird es an einen Ausgabestrom übergeben. Es wird also in diesem Fall in eine XLS-Datei geschrieben.

3.1.6 Das Paket GUI

In diesem Paket befindet sich die Hauptklasse namens *GUI* sowie Listener-Klassen.

[4] Herstellerseite: http://poi.apache.org/

ActionListenerClass

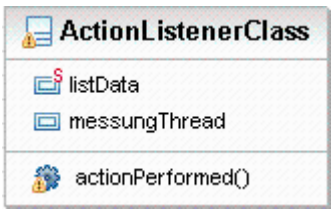


Abbildung 3.17: ActionListenerClass

Diese Klasse (siehe Abbildung 3.17) hört die verschiedenen Schaltflächen der GUI ab.

- Wird die Schaltfläche “Messung starten” geklickt, dann führt die Funktion *actionPerformed()* die Funktion *start()* der Klasse *MessungThread* aus.
- Bei Klick auf “Messung abbrechen” wird die Funktion *interrupt()* der Klasse *MessungThread* aufgerufen.
- Wenn ein Modul ausgewählt wird, so muss die Ansicht entsprechend dem gewählten Modul aktualisiert werden. Hierzu werden die Funktionen *getServerParamsList()*, *getStartWertField()*, *getStopWertField()*, und *getStepField()* der Klasse *GUI* gestartet, welche die jeweiligen Felder mit den richten Daten und Parametern füllt.
- Wird die Schaltfläche “Speichern” gedrückt, dann werden alle Felder der GUI abgespeichert. Dazu werden sie alle eingelesen und jeweils der Funktion *setProperty()* der Klasse *PropertiesSingleton* übergeben.
- Sollte die Schaltfläche “Alle Tabellen auswerten” angeklickt werden, so wird die Funktion *start()* der Klasse *AlleMessungenAuswertenThread* ausgeführt.
- Bei der Schaltfläche “Tabelle auswerten” wird die Funktion *start()* der Klasse *TabelleAuswertenThread* gestartet.
- Wenn im Register “Ergebnisse” eine andere Tabelle gewählt wird, so muss die Messpunktliste aktualisiert werden. Daher wird die Funktion *fillList()* der Klasse *JdbcChartListPanel* aufgerufen.
- Für den Fall, dass im Register “Querauswertung” eine Tabelle ausgewählt wird, so muss auch hier die Messpunktliste erneuert werden. Dazu wird die Funktion *fillList()* der Klasse *GUI* aufgerufen.

- Die Schaltfläche “Hinzufügen” fügt dem gewählten Messpunkt den Vektor *listData* hinzu und übergibt diesen Vektor der Funktion *GUI.quantileList.setListData()*. Anschließend wird mit *GUI.quantileList.repaint()* die Anzeige aktualisiert, so dass nun der hinzugefügte Wert in der Liste sichtbar ist.
- Wird die Excel-Export-Schaltfläche gewählt, so wird die Funktion *exportToXLS()* der Klasse *MySQLToXLS* ausgeführt.
- Wird die CSV-Export-Schaltfläche gewählt, so wird die Funktion *exportToCSV()* der Klasse *MySQLToCSV* ausgeführt.
- Bei Wahl der Schaltfläche “Graph zeichnen” wird zu jedem Eintrag der Liste im Querauswertungsregister das 90%-Quantil mit *ValueComputation.calculateQuantile()* berechnet. Jeder errechnete Wert wird einer Serie hinzugefügt und dann wird die endgültige Serie der Funktion *ChartFactory.createXYLineChart()* übergeben, die einen Chart zurückliefert. Der Chart wird dann mit *ChartUtilities.saveChartAsPNG()* gespeichert.

GUI

Die Klasse *GUI* ist die Benutzeroberfläche für den Anwender. Aufgrund ihrer Größe und da der Code zum Großteil automatisch generiert wurde, ist das Klassendiagramm hier nicht dargestellt[5]. Startet man das Programm *JDataMining*, so wird als erstes die Funktion *main()* der Klasse *GUI* ausgeführt. Innerhalb der Funktion *main()* wird zuallererst die Datei *config.properties* der Funktion *PropertiesSingleton.getInstance()* übergeben. Dies stellt sicher, dass sofort zu Anfang die Einstellungen des Programmes verfügbar sind und zwar für das ganze Programm nur als eine einzige Instanz (siehe dazu auch die Beschreibung der Klasse *PropertiesSingleton*). Erst danach wird die GUI initialisiert.

ListSelectionListenerClass

Abbildung 3.18: ListSelectionListenerClass

Die Klasse *ListSelectionListenerClass* (siehe Abbildung 3.18) hört die Server-Parameterliste und die JMeter-Parameterliste ab. Wird ein Eintrag in einer Liste ausgewählt, so werden in der

[5] Das Klassendiagramm befindet sich jedoch für Interessierte auf der beiliegenden DVD.

Funktion *valueChanged()* entweder die Funktionen *GUI.getStartWertField()*, *GUI.getStopWertField()* und *GUI.getStepField()* aufgerufen, falls die Änderung in der Server-Parameterliste stattgefunden hat oder aber es werden die Funktionen *GUI.getJmeterStartWertField()*, *GUI.getJmeterStopWertField()* und *GUI.getJmeterStepField()* aufgerufen, falls die Änderung in der JMeter-Parameterliste stattgefunden hat. In beiden Fällen werden die jeweils zugehörigen Felder für den Startwert, den Stoppwert und die Schrittweite aktualisiert.

3.1.7 Das Paket Hilfsklassen

In diesem Paket befinden sich Klassen, die nicht direkt einem Paket zuzuordnen sind und/oder nicht direkt im Programm verwendet werden.

DateConverter

Abbildung 3.19: DateConverter

Diese Klasse (siehe Abbildung 3.19) ermöglicht das Umwandeln eines Datums in einen Zeitstempel. Es wird bisher das englische und das deutsche Datumsformat unterstützt. Die Klasse wird insbesondere beim Parsen der Log-Dateien benötigt, da die Log-Dateien ein Datum mit Uhrzeit mitloggen, dieses aber für die Verwendung im Programm in einen Zeitstempel umgewandelt werden muss.

DBConnection

Abbildung 3.20: DBConnection

Der Konstruktor *DBConnection()* der Klasse *DBConnection* (siehe Abbildung 3.20) holt sich zunächst die Instanz der Klasse *PropertiesSingleton*, also die Einstellungen des Programmes. Nun kann er aus diesen Einstellungen den Namen der Datenbank, den Benutzernamen, das Passwort, den Host und den Namen der temporären Tabelle einlesen und in seinen Attributen speichern.

Die Funktion *cleanTable()* löscht eine Tabelle mittels der SQL-Query:

```
statement.executeUpdate("DROP TABLE IF EXISTS " +table);
```

Mit *closeConnection()* wird eine bestehende Datenbankverbindung geschlossen.

Mit der Funktion *connect()* kann eine Datenbankverbindung hergestellt werden.

Die Funktion *getAllSections()* liefert alle Messwerte zu einem Messpunkt sortiert zurück. Dabei wird in folgender Reihenfolge nach TID, PID und TSTAMP (also Thread-ID, Prozess-

ID, Zeitstempel) sortiert. Hierzu wird beim Worker MPM die folgende SQL-Query ausgeführt (beim Prefork MPM wird die TID weggelassen):

```
statement.executeQuery("Select * FROM " + origTable
          + " WHERE MFWPOINT='" +messpunkt
          + "' AND STATE='" + state
          + "' ORDER BY 'TID', 'PID', 'TSTAMP'");
```

Die Funktion *getMesspunktnamen()* erzeugt einen String-Vektor mit allen Messpunktnamen, die in der Tabelle auftreten. Dabei führt sie zuerst die SQL-Query *statement.executeQuery-("SELECT DISTINCT MFWPOINT FROM " + table);* aus. Diese Query liefert alle Messpunktnamen der Tabelle. Jeder Messpunktname wird dank des Schlüsselwortes *DISTINCT* nur ein einziges Mal vermerkt, alle weiteren Vorkommen werden ignoriert. Das Ergebnis der Query wird in einem *ResultSet* gespeichet. Dieses *ResultSet* wird nun in einer While-Schleife durchlaufen. Jeder Eintrag wird dabei zum String-Vektor hinzugefügt, welcher anschließend an die aufrufende Funktion zurückgegeben wird.

ParameterParser

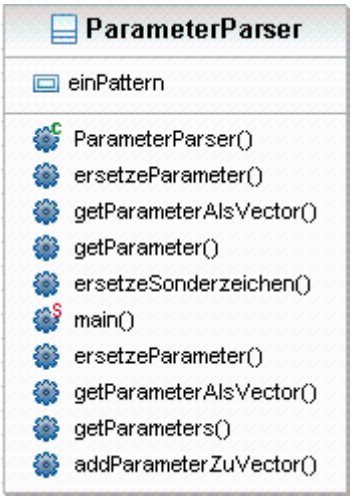


Abbildung 3.21: ParameterParser

Die in Abbildung 3.21 abgebildete Klasse *ParameterParser* wurde von André Westhoff in [Wes02] entwickelt. Sie dient dem Ersetzen von Parameterplatzhaltern mit richtigen Werten in einer Datei. Der Quellcode wurde nachträglich noch an die Javaversion 5 angepasst, das heißt, dass Vektoren etc. um Typsicherheit erweitert wurden.

PropertiesSingleton

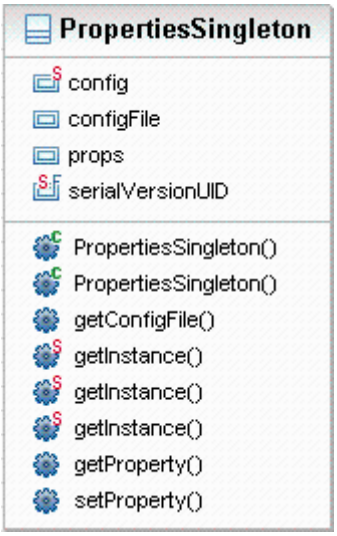


Abbildung 3.22: PropertiesSingleton

Die Klasse *PropertiesSingleton* (siehe Abbildung 3.22) wurde nach dem Singletonmuster implementiert. Man kann von ihr nur ein einziges Objekt in der gesamten Anwendung erzeugen. Dies hat den Vorteil, dass die Konfigurationsdatei mit dieser Klasse nur ein einziges Mal bei Programmstart eingelesen werden muss und von da an die Einstellungen jederzeit von den meisten Klassen erreichbar sind. Außerdem ist so sichergestellt, dass die Anwendung an jeder Stelle mit den gleichen Einstellungen arbeitet.

Die Klasse wurde von Stephan Wiesner entwickelt (siehe [Wie04]). Da die Klasse nur die Einstellungen am Anfang aus einer Datei lesen konnte und ab dann die Einstellungswerte zur Verfügung hatte, konnten die Einstellungen immer wieder aus dem anfangs erzeugten Objekt der Klasse *PropertiesSingleton* mit der Funktion *getProperty(String key)* ausgelesen werden. Leider gab es jedoch nicht die Möglichkeit, diese Einstellungen während der Laufzeit zu verändern. Daher wurde nachträglich die Funktion *setProperty(String key, String value)* hinzugefügt. Mit dieser Funktion ist es nun möglich, während der Laufzeit des Programmes die Einstellungen aus dem Programm heraus zu verändern. Die Änderungen werden aber nicht nur im Speicher gehalten, sondern auch direkt in die Konfigurationsdatei geschrieben, damit nach einem Programmneustart die Änderungen noch vorhanden sind.

Querauswertung

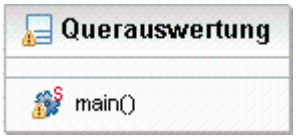


Abbildung 3.23: Querauswertung

Die Klasse *Querauswertung* wird nicht im Programm *JDataMining* verwendet. Aufgrund der einzigen Funktion *main()* (siehe Abbildung 3.23) ist sie ohne das eigentliche Programm ausführbar. Sie war für das Auswertungskapitel eine kleine Hilfestellung, da sie zu einer ihr angegebenen Messung und einem angegebenen Messpunkt alle 90%-Quantile berechnet und davon den Durchschnitt bildet. Außerdem erzeugt sie einen Graphen mit 90%-Quantilen.

3.1.8 Das Paket JMeter

JMeterTableGenerator

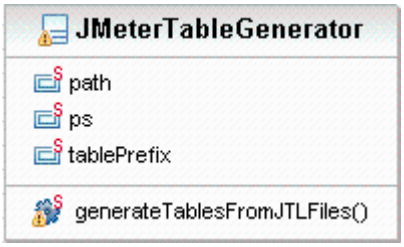


Abbildung 3.24: JMeterTableGenerator

Für das Funktionieren dieser Klasse wird das Paket *Core* von Martin Gasse verwendet (siehe [Gas06]). Die Klasse *JMeterTableGenerator* (siehe Abbildung 3.24) ist für das Generieren der Tabellen aus JTL-Dateien zuständig. In der Funktion *generateTablesFromJTLFiles()* wird als Erstes der Pfad zu dem Ordner mit den gespeicherten JTL-Dateien aus der Einstellungsdatei gesucht. Der Pfad wird dem Konstruktor des neu erzeugten Objektes *FileGetter* übergeben. Anschließend sollen die Dateien geparst werden. Daher wird folgende Zeile ausgeführt, welche die Dateien an ein Objekt der Klasse JTLParser weiterleitet:

```
lJTLParser.setFilesToParse(lFileGetter.getAllLogs());
```

Mit dem Befehl *lJTLParser.generateSQLQueryList("");* werden dann die Inhalte der Dateien in die Datenbank geschrieben.

JMXReader

Abbildung 3.25: JMXReader

Die Klasse *JMXReader* (siehe Abbildung 3.25) kann eine JMeter-Vorlage einlesen und dort die gesetzten Parameter herauslesen. Die Funktion *getParams()* holt sich den Pfad zu der JMeter-Vorlage aus dem PropertiesSingelton-Objekt. Die dort gefundene Datei wird zeilenweise eingelesen. Falls sie auf eine Zeile trifft, die den Teilstring *"!–PARAM::"* enthält, so extrahiert sie aus der Zeile den Parameternamen und fügt diesen Namen einem Vektor hinzu, welcher zum Schluss der aufrufenden Funktion zurückgegeben wird.

Eine Zeile könnte zum Beispiel so aussehen:

```
<stringProp name="ThreadGroup.delay">!--PARAM::mainDelay--!
     </stringProp>
```

Da der Teilstring *"!–PARAM::"* dort auftaucht, weiß die Funktion, dass sie diese Zeile aufspalten muss. Dies geschieht mit den folgenden zwei Codezeilen:

```
String[] strArr = currentLine.split("::");
String[] strArr2 = strArr[1].split("--!");
```

Mit *v.add(strArr2[0]);* wird der Name dann dem Vektor hinzugefügt.

3.1.9 Das Paket Logs

LogsTableGenerator

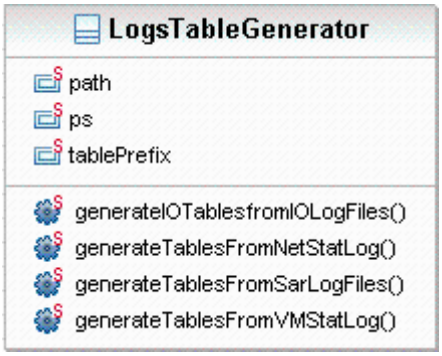


Abbildung 3.26: LogsTableGenerator

Diese Klasse sorgt für das Generieren von Tabellen aus Log-Dateien. Für diese Klasse wird ebenfalls das Paket *Core* von Martin Gasse ([Gas06]) gebraucht. Alle vier Funktionen, die in Abbildung 3.26 dargestellt sind, funktionieren nach dem gleichen Prinzip. Als Beispiel wird hier die Funktionsweise der Funktion *generateIOTablesFromIOLogFiles()* erläutert. Begonnen wird mit dem Lesen des Log-Pfades aus dem PropertiesSingleton-Objekt:

```
ps = PropertiesSingleton.getInstance();
path = ps.getProperty("logspath");
```

Dann wird ein neues Objekt der Klasse *IOStatLogParser* (bzw. *NetStatLogParser*, *VMStatLogParser* oder *SarLogParser*) und ein neues Objekt der Klasse *FileGetter* (mit übergebenen Pfad) erzeugt:

```
IOStatLogParser lIOStatLogParser = new IOStatLogParser();
FileGetter lFileGetter = new FileGetter(path);
```

Anschließend werden die IOStatLog-Dateien dem Objekt *lIOStatLogParser* zum Parsen übergeben und dann eine Query aus den geparsten Informationen erzeugt, welche an die Datenbank geschickt wird:

```
lIOStatLogParser.setFilesToParse(lFileGetter.getAllLogs());
lIOStatLogParser.commit(
   lIOStatLogParser.generateSQLQueryList(""));
```

3.1.10 Das Paket MFW

Dieses Paket enthält Klassen zur Verarbeitung der Informationen des Messframeworks.

Cut, MFWPoint und Results

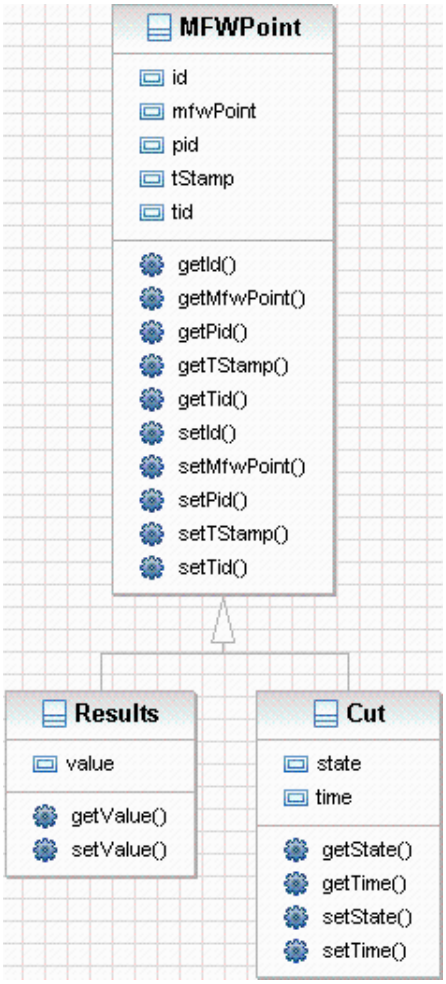


Abbildung 3.27: MFWPoint, Cut und Results

Die Klasse *MFWPoint* (siehe Abbildung 3.27) repräsentiert einen Messpunkt. Sie besitzt die Attribute *id*, *pid*, *tid*, *mfwPoint* und *tStamp*. Alle diese Attribute entsprechen Feldern in einer Tabelle. Das Attribut *id* ist der Primärschlüssel, der von der Datenbank automatisch vergeben wird. Das Attribut *pid* ist die Prozess-ID und entsprechend ist das Attribut *tid* die Thread-ID. *MfwPoint* trägt den Namen eines Messpunktes und *tStamp* speichert den Zeitstempel.

Die Klasse *Cut* erweitert die Klasse *MFWPoint* um die Attribute *state* und *time*. *State* wird für den Status eines Messpunktes verwendet. Er kann entweder mit on, off oder error belegt sein. Das Attribut *time* nimmt den gemessenen Prozessortick auf.

Die Klasse *Results* erweitert die Klasse *MFWPoint* um das Attribut *value*. Dies ist der berechnete Wert, der aus zwei Objekten der Klasse *Cut* gebildet wird (zur Berechnung siehe Abschnitt *ValueComputation*).

ValueComputation

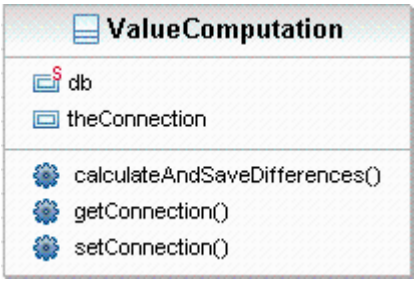


Abbildung 3.28: ValueComputation

Die Klasse *ValueComputation* (siehe Abbildung 3.28) berechnet die Zeitwerte aus den Daten die in der temporären Tabelle vorliegen. Die Funktion *calculateAndSaveDifferences()* bewerkstelligt diese Berechnungen. Zu Beginn wird die neu zu erzeugende Tabelle aus der Datenbank entfernt, falls sie schon in der Datenbank vorhanden ist:

```
statement2.execute("DROP TABLE IF EXISTS " + name);
```

Nun muss sie neu erstellt werden:

```
statement2.execute("CREATE TABLE IF NOT EXISTS " + name
                   + " (ID BIGINT(20) UNSIGNED AUTO_INCREMENT,
                   PRIMARY KEY (ID), MFWPOINT VARCHAR(100)
                   NOT NULL, TSTAMP BIGINT(20) UNSIGNED
                   NOT NULL, VALUE BIGINT(20) NOT NULL,
                   PID INT(10) UNSIGNED NOT NULL,
                   TID INT(10) UNSIGNED NOT NULL)
                   ENGINE=MyISAM CHARACTER SET utf8
                   COLLATE utf8_general_ci");
```

Abbildung 3.29 zeigt den weiteren Verlauf des Programmes.

3.1.11 Das Paket Server

Das Paket Server beinhaltet die serverrelevanten Klassen.

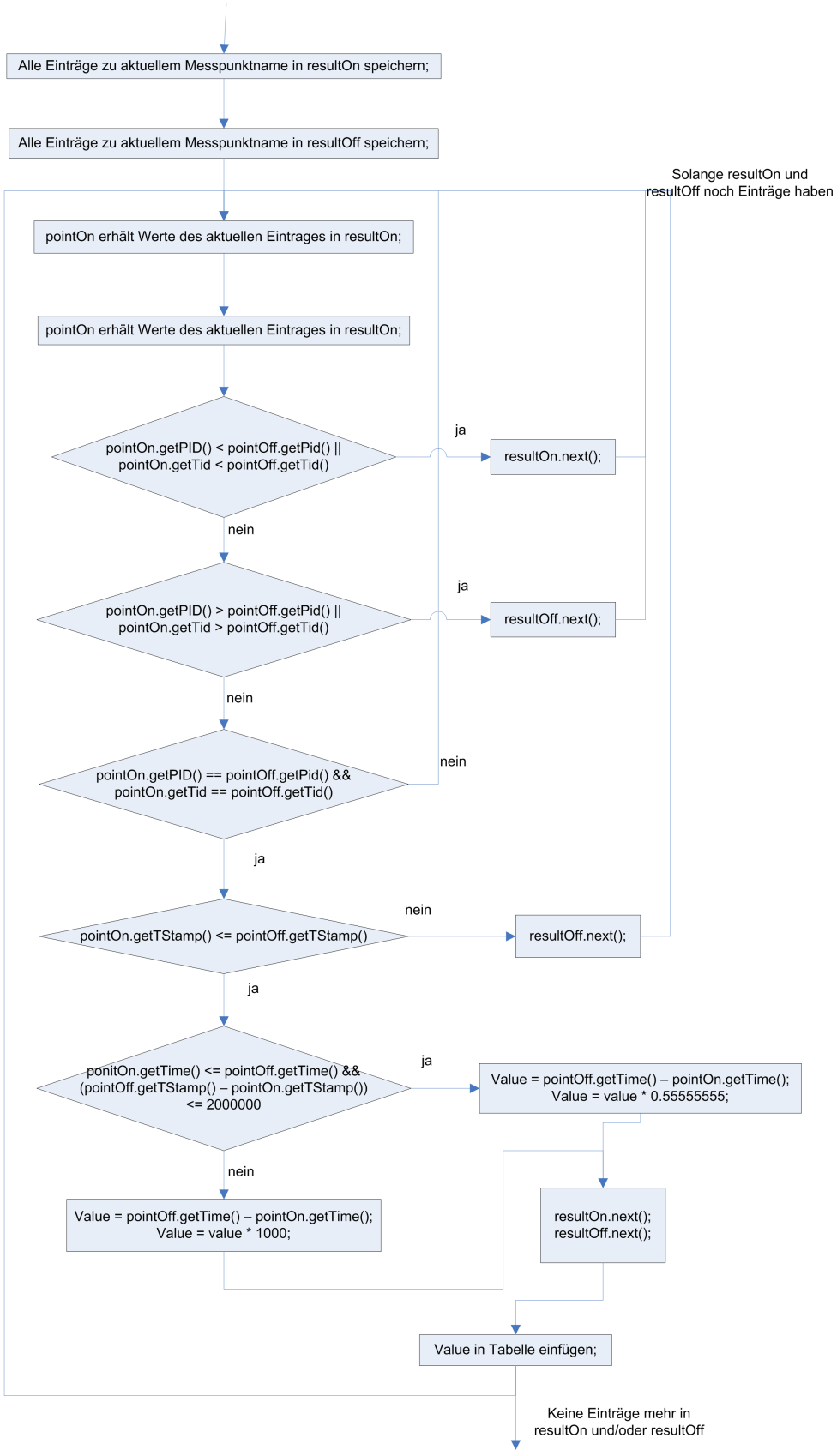


Abbildung 3.29: Berechnung der Werte in ValueComputation.calculateAndSaveValues()

HttpdMPMConfReader

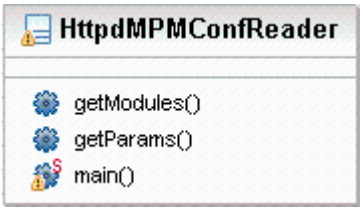


Abbildung 3.30: HttpdMPMConfReader

Die Klasse *HttpdMPMConfReader* (siehe Abbildung 3.30) liest die Parameter der Konfigurationsdatei des Servers ein. Hierzu verwendet sie die beiden Funktionen *getModules()* und *getParams()*.

Die Funktion *getModules()* stellt fest, welche Module in der Konfigurationsdatei zur Verfügung stehen. Sie holt sich zunächst aus dem PropertiesSingleton-Objekt den Speicherort der Konfigurationsdateivorlage. Diese Vorlage durchläuft sie nun zeilenweise. Wenn sie auf den Teilstring *"<IfModule mpm_"* trifft, dann hat sie ein Modul entdeckt. Daher muss nun aus dieser Zeile der Modulname extrahiert werden, indem der String mit der split()-Methode aufgeteilt wird. Dabei ist der Unterstrich das Trennzeichen. Nun steht ein Array zur Verfügung mit verschiedenen Teilstrings. Wenn das Array drei Felder lang ist, so verbirgt sich der Modulname auf jeden Fall an der Indexstelle eins. Wenn das Array aber länger als drei Felder ist, so muss der Modulname zusammengesetzt werden, da er selbst einen Unterstrich in seinem Modulnamen enthält. Dazu wird ab der Indexstelle eins wieder zwischen jedem Feld ein Unterstrich hinzugefügt. Der Modulname besteht dann aus den Feldern beginnend mit dem Indexfeld eins und endend mit dem vorletzten Indexfeld, jeweils getrennt durch Unterstriche. In jedem Fall wird der Modulname zu einem String-Vektor hinzugefügt und dann wird in der Datei weiter gesucht, ob es noch weitere Module gibt. Zum Schluss wird der Vektor an die aufrufende Funktion zurückgeliefert. Diese Funktion wird unter anderem von der GUI verwendet um die Modulauswahlbox zu füllen.

Die Funktion *getParams()* liefert entsprechend dem gewünschten Modul alle Parameter zurück, die in der Konfigurationsdatei aufgelistet sind. Dabei verfährt sie ähnlich wie die Funktion *getModuls()*. Sie sucht zunächst nach dem Teilstring *"<IfModul mpm_*MODULNAME*_module>"*, wobei MODULNAME durch das gesuchte Modul ersetzt wird. Ab jetzt liest sie jede Zeile ein und speichert die gefundenen Parameter in einem Vektor, bis sie auf den Teilstring *"</IfModule>"* stößt. Der Vektor wird an die aufrufende Funktion übergeben.

Parameter

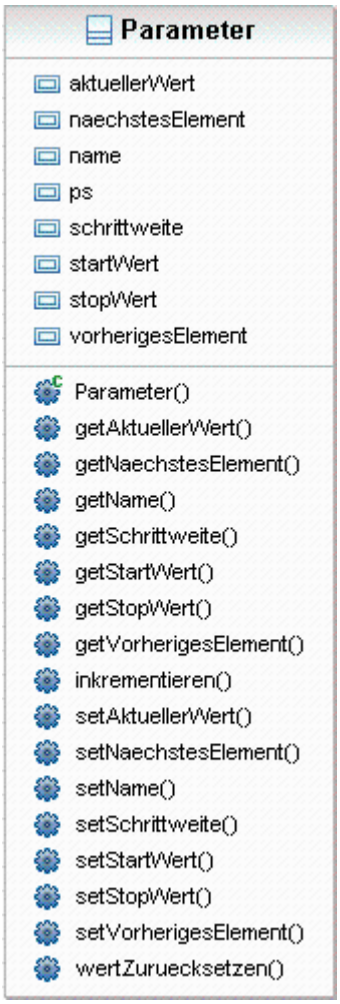


Abbildung 3.31: Parameter

Die Klasse *Parameter* (siehe Abbildung 3.31) repräsentiert einen Parameter mit den Werten Startwert, Stoppwert, aktueller Wert und Schrittweite. Des Weiteren trägt ein Parameter einen Namen, zum Beispiel StartServers oder MinSpareServers.

Die Funktionen *getNaechstesElement()*, *setNaechstesElement()*, *getVorherigesElement()*, *setVorherigesElement()*, *inkrementieren()* und *wertZuruecksetzen()* werden nicht im Programm verwendet.

3.1.12 Das Paket Threads

Das Paket Threads enthält Klassen, die als Threads ablaufen sollen, damit während ihres Betriebes das Programm weiterhin verwendet werden kann.

AlleMessungenAuswertenThread

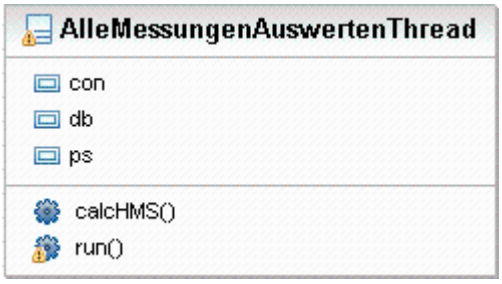


Abbildung 3.32: AlleMessungenAuswertenThread

Die Klasse *AlleMessungenAuswertenThread* erbt von der Klasse Thread. Sie besitzt die beiden Funktionen *calcHMS()*[6] und *run()*.

Die Funktion *calcHMS()* ist nur ein kleines Feature, was ermöglichen soll, dass nach einer Auswertung die Auswertungsdauer in einem ansehnlicheren Format dargestellt werden kann. Sie wandelt eine Sekundenzeitangabe in Stunden, Minuten und Sekunden um.

Die Funktion *run()* ist für die Auswertung der Messung zuständig. Als erstes holt sie sich den Namen der auszuwertenden Messungen aus dem PropertiesSingleton-Objekt. Nun durchsucht sie die Tabellenliste der GUI nach Tabellen mit diesem Namen als Bestandteil und merkt sich alle zutreffenden Tabellen in einem Vektor. Außerdem merkt sie sich die Messpunkte in einem weiteren Vektor, die zu einer Tabelle gehören. Nachdem diese Informationen feststehen, fragt sie alle Tabellen, welche in dem Vektor mit den Tabellennamen sind, in der Datenbank der Reihe nach ab. Zu jedem Messpunkt jeder Tabelle erstellt sie mit der Funktion *PNG.createPNG()* eine PNG-Datei. Wenn alle Bilder erstellt wurden, dann werden die Messpunktnamen und Tabellennamen an die Funktion *HTMLOverview.createHTML()* gesendet, um eine HTML-Übersicht der Messungen zu erstellen.

MessungThread

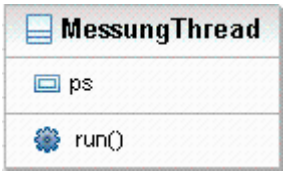


Abbildung 3.33: MessungThread

[6]Diese Funktion wurde aus [Nie01] übernommen.

MessungThread (siehe Abbildung) ist eine Klasse, die von der Klasse *Thread* erbt. Innerhalb dieses Threads laufen die Messungen ab. Begonnen wird in der Funktion *run()* mit dem Aufruf der Funktion *AntWriter.createAntFile()*. Diese erzeugt die Ant-Datei, die den Messplan darstellt. Um den Messplan auszuführen, wird die Funktion *AntStarter.runAnt()* aufgerufen. Die Messungen laufen nun der Reihe nach durch. Sobald sie beendet sind, müssen alle Informationen in die Datenbank geschrieben werden. Daher werden anschließend diese Funktionen nacheinander ausgewählt:

```
JMeterTableGenerator.generateTablesFromJTLFiles();
LogsTableGenerator.generateTablesFromNetStatLog();
LogsTableGenerator.generateIOTablesfromIOLogFiles();
LogsTableGenerator.generateTablesFromSarLogFiles();
LogsTableGenerator.generateTablesFromVMStatLog();
CsvTableGenerator.generateTablesFromOrigCSV();
```

Für den Fall, dass vom Benutzer das automatische Auswerten eingestellt wurde, wird zum Abschluss der Thread *AlleMessungenAuswertenThread* gestartet.

Zwischen allen Befehlen in der Klasse *MessungThread* wird überprüft, ob der Benutzer auf “Messung abbrechen” geklickt hat. Falls ja, dann wird der Thread nicht mehr weiter ausgeführt und beendet.

TabelleAuswertenThread

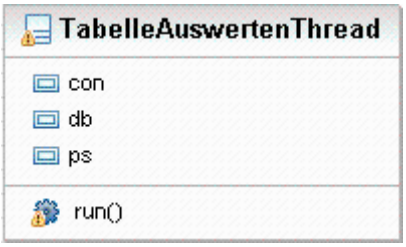


Abbildung 3.34: TabelleAuswertenThread

Diese Klasse (siehe Abbildung 3.34) funktioniert im Prinzip ähnlich wie die Klasse *AlleMessungenAuswertenThread* mit dem Unterschied, dass sie nur die aktuell ausgewählte Tabelle berücksichtigt.

3.2 Änderungen an Fremdpaketen

An manchen Klassen von Fremdpaketen wurden leichte Änderungen vorgenommen, um die Klassen an das Programm effizienter anzupassen. Die Änderungen werden im Folgenden geschildert.

3.2.1 Änderungen an JFreeChart

Der Klasse *YIntervalSeries* im Paket *org.jfree.data.xy* wurde nachfolgende Funktion hinzugefügt, da die bestehende Funktion nicht die Variable *notify* besaß und stattdessen immer die Superklasse an dieser Stelle mit *true* aufgerufen hat:

```
public void add(double x, double y, double yLow,
                double yHigh, boolean notify)
{
 super.add(new YIntervalDataItem(x, y, yLow,
                                 yHigh), notify);
}
```

Nun kann von der Klasse *JdbcMeanAndStandardDeviationXYDataset* aus diese Funktion für alle Punkte außer dem letzten mit *notify* gleich *false* aufgerufen werden, während für den letzten Punkt die Funktion mit *notify* gleich *true* aufgerufen wird (siehe hierzu auch Abschnitt 3.1.3).

3.2.2 Änderungen an POI

Microsoft hat die maximale Zeilenanzahl bei Excel auf 65536 Zeilen begrenzt. In *POI* gibt es auch speziell eine Sicherheitsabfrage, ob die Zeilenzahl größer oder gleich Null ist und kleiner als 65536 ist. Das Problem ist jedoch, dass diese Begrenzung nicht annähernd ausgereizt wird. Ein Zeilenindex größer als 32767 ist nicht möglich, da in *POI* beim Erzeugen von Zellen nur der Datentyp *short* erlaubt ist. Dieser Datentyp umfasst zwar insgesamt 65536 Zahlen, allerdings liegen diese Zahlen im Zahlenbereich von -32768 bis 32767. Folglich wurde im Konstruktor der Klasse *HSSFRow* des Paketes *org.apache.poi.hssf.usermodel* der Datentyp der Variablen *rowNum* von *short* nach *int* geändert. Nun zog sich jedoch diese Änderung als Fehler durch das ganze Programm. Entsprechend musste noch in einigen anderen Klassen der Datentyp von *short* nach *int* geändert werden.

Diese Änderungen sind nötig gewesen, weil es gewünscht ist, dass möglichst alle Zeilen in Excel ausgefüllt werden können. Beim Messen des Apache Webservers fallen viele Daten an, daher muss der Spielraum an freien Zellen möglichst ausgereizt werden.

3.2.3 Änderungen an apmfw.c

Die Datei *apmfw.c* wurde von Simon Olofsson in [BNO+07] entwickelt, wo auch eine ausführliche Erklärung zu finden ist. Dank dieser Datei ist es möglich, in den Apache Quellcode Messpunkte zu setzen, deren Messwerte von *apmfw.c* in die CSV-Datei geschrieben werden.

Es musste noch ein Verfahren implementiert werden, welches eine Wartezeit und eine Processing-Delay pro Anfrage ermöglicht.

Bei der Wartezeit handelt es sich um eine Zeitangabe in ms, während dieser der Prozess/Thread schlafen soll. Dadurch könnte also theoretisch eine Datenbankabfrage simuliert werden, da bei einer solchen Abfrage der entsprechende Prozess/Thread ebenfalls auf die Antwort der Datenbank warten muss.

Bei der Processing-Delay handelt es sich um eine Zeit in ms, während dieser der Prozess/Thread den Prozessor belasten soll. Der Prozess/Thread soll also etwas machen, das dafür sorgt, dass Rechenleistung verbraucht wird. Dies ist in der Theorie eine Simulation einer Generierung einer dynamischen Webseite.

Um diese beiden Zeiten zu implementieren, war es zunächst nötig die Funktion *apmfw_check_url()* anzupassen. Sie wurde um folgenden Else-Zweig erweitert:

```
else if (strstr(url, "Testseite.html") && query != NULL)
{
  apmfw_wait_and_process(query);
}
```

Dieser Zweig wird aufgerufen, wenn als URL die Seite *Testseite.html* aufgerufen wird und diese Seite eine Query hat. In der Query werden die Parameter für die Wartezeit, die Processing-Delay und die Methode der Processing-Delay (siehe unten) mitgeliefert. Die Funktion *apmfw_wait_and_process()*, welche in dem Else-Zweig aufgerufen wird, ist ebenfalls neu. Sie sieht wie folgt aus:

```
int apmfw_wait_and_process(char *query)
{
  char* s = query;
  char* res;
  char* res1;
  res = strsep(&s,"=");
  res = strsep(&s,"&");
  apmfw_write("waitTime", "on");
  apr_sleep(atoi(res) * 1000);
```

```
    apmfw_write("waitTime", "off");
    res = strsep(&s,"=");
    res = strsep(&s, "&");
    res1 = strsep(&s, "=");
    apmfw_processing_delay(apr_atoi64(res), atoi(s));
    return APR_SUCCESS;
}
```

Die Query wird in dieser Funktion als Erstes auseinander genommen, das heißt, dass zuerst der Parameterwert für *waitTime* (also die Wartezeit) extrahiert wird. Dieser Wert wird mit der Funktion *atoi()* in eine Zahl umgewandelt und mit 1000 multipliziert, um sie in den Mikrosekunden-Bereich zu verschieben. Dieser neue Wert wird dann der Funktion *apr_sleep()* übergeben, welche den Prozess/Thread für die angegebene Zeit in μs schlafen lässt. Vor und nach dieser Funktion *apr_sleep()* befinden sich die Messpunkte *waitTime*, damit gemessen werden kann, wieviel der Prozess/Thread tatsächlich geschlafen hat.

Nachdem der Prozess wieder aktiv ist, wird die Query weiter auseinander genommen, um die Parameterwerte für *processingDelay* und die zugehörige Methode zu extrahieren. Beide Werte werden in Zahlen umgewandelt und der Funktion *apmfw_processing_delay()* übergeben.

Die Funktion *apmfw_processing_delay()* ist auch neu implementiert worden. Sie wird nun schrittweise erklärt. Als Erstes wird die aktuelle Zeit bestimmt und zwar zweimal, damit die Differenz davon in der Variable *time* abgespeichert werden kann:

```
int apmfw_processing_delay(uint64_t processingDelay,
                           int method)
{
  apmfw_write("processingDelay", "on");
  uint64_t startTime = apmfw_rdtsc();
  uint64_t currentTime = apmfw_rdtsc();
  uint64_t time = currentTime - startTime;
```

Falls die Variable *method* mit eins belegt ist, wird die Methode eins für die Processing-Delay verwendet. Bei der Methode eins gibt es eine äußere Schleife, welche solange läuft, wie die Differenz der aktuellen Zeit zur Startzeit kleiner als die gewünschte Processing-Delay ist. Die innere Schleife ist eine Schleife, deren Anzahl Durchläufe zuvor genau so bestimmt wurde, dass sie möglichst genau eine ms benötigt. In ihr werden zwei Rechenoperationen ausgeführt, die Rechenleistung produzieren. Außerdem wird die Anzahl Schleifendurchläufe der äußeren Schleife mitgezählt, um sie später mittels des Messframeworks in die CSV-Datei zu schreiben:

```
if(method == 1)
{
  int count = 0;
  processingDelay *= 1000000;
  while(time < processingDelay)
  {
    count++;
    int x = 0;
    int j = 0;
    for(j = 0; j < 893310; j++)
    {
      x = x + 1;
      x = x - 1;
    }
    currentTime = apmfw_rdtsc();
    time = currentTime - startTime;
  }
  apmfw_write("processingDelay", "off");
  int i;
  char str[15];
  i = sprintf(str, "%d", count);
  apmfw_write("Methode1Durchlaeufe", str);
}
```

Bei Methode 2 läuft die äußere Schleife nicht bis sie die gewünschte Processing-Delay überschritten hat, sondern sie läuft eine festgelegte Anzahl an Durchläufen. Wenn beispielsweise die Processing-Delay 20 ms betragen soll, dann wird die äußere Schleife 20 mal durchlaufen. Die innere Schleife braucht nämlich, wie gerade schon erwähnt, eine ms für alle Durchgänge.

```
else if (method == 2)
{
  int count;
  for(count = 0; count < processingDelay; count++)
  {
    int x = 0;
    int j = 0;
    for(j = 0; j < 893310; j++)
```

```
        {
          x = x + 1;
          x = x - 1;
        }
      }
      apmfw_write("processingDelay", "off");
    }
```

Der letzte Else-Zweig ist zum Bestimmen der nötigen Anzahl Durchläufe für 1 ms Rechenzeit der inneren Funktion gedacht. Da auf jedem System diese Schleife eine andere Zeit verbrauchen wird, muss vor dem Messen durch Verändern der Abbruchbedingung eine Zahl gesucht werden, die die Schleife auf 1 ms Rechenzeit bringt:

```
    else
    {
      apmfw_write("Schleifendauer", "on");
      int x = 0;
      int j = 0;
      for(j = 0; j < 893310; j++)
      {
        x = x + 1;
        x = x - 1;
      }
      apmfw_write("Schleifendauer", "off");
      apmfw_write("processingDelay", "off");
    }
    return APR_SUCCESS;
  }
```

4 Benutzung des Programmes

Dieses Kapitel beschreibt die Benutzung des Programmes JDataMining beginnend bei der Einrichtung des Testsystems, über die Handhabung der GUI bis hin zur Auswertung. Dabei wird davon ausgegangen, dass Grundkenntnisse in Linux oder Windows bereits vorhanden sind.

4.1 Installation

Bevor JDataMining verwendet werden kann, müssen zunächst einige Installationen und Konfigurationen vorgenommen werden. Dies wird in den folgenden Abschnitten erläutert.

4.1.1 Installation der Datenbank

Die Installation der Datenbank wird hier für das Betriebssystem Windows XP erklärt, da sich die Datenbank des Testsystems in dieser Arbeit auch unter Windows XP befindet.

Am einfachsten ist es, wenn das aktuelle Paket der Distribution XAMPP von der Seite www.apachefriends.org heruntergeladen wird. Dieses Paket bietet zwar mehr Funktionen als eigentlich notwendig sind, dafür ist die Installation aber sehr einfach und neben MySQL wird auch direkt PHPMyAdmin mitgeliefert, so dass die Datenbank direkt administriert werden kann. Es wird hier speziell die Installer-Version empfohlen. Ein Doppelklick auf das heruntergeladene Paket startet die Installation.

Nach erfolgreicher Installation wird das neu installierte Programm “XAMPP Control Panel” gestartet. Dort wird auf die beiden Startknöpfe neben “Apache” und neben “MySQL” geklickt. Als nächstes wird ein Internetbrowser geöffnet (zum Beispiel der Internet Explorer oder der Mozilla Firefox) und in die Adresszeile wird “localhost/phpmyadmin” eingetippt. Nach Drücken auf die Taste “Enter” erscheint das Datenbankadministrationstool PHPMyAdmin.

Mit Hilfe dieses Tools wird eine Datenbank erstellt. Dazu wird der Name der neuen Datenbank (zum Beispiel “apmfw”) in das Feld “Neue Datenbank anlegen” eingetragen und mit einem Klick auf “Anlegen” angelegt. Auf Wunsch kann auch ein Benutzer mit Passwort für die Datenbank angelegt werden oder es wird einfach die Starteinstellung verwendet (Benutzername root ohne Passwort; Achtung: Diese Variante nur in Testumgebungen verwenden, da sie ein

hohes Sicherheitsrisiko darstellt).

4.1.2 Einrichtung des Lastgenerators

Da der Lastgenerator des Testsystems in dieser Arbeit auf Suse Linux läuft, wird die Installation und Einrichtung für Suse Linux beschrieben.

Sollte noch kein Java oder eine Javaversion unter Java 5 auf dem Rechner installiert sein, so muss dies zunächst installiert werden. Deshalb wird die Seite http://java.sun.com besucht und von dort die aktuelle Javaversion heruntergeladen und installiert. Eventuell muss noch der *CLASSPATH* und *JAVA_HOME* eingetragen werden. Hierzu wird in der Konsole “sudo gedit /etc/enviroment” oder einer der vielen ähnlichen Befehle zum Editieren von Dateien eingegeben. In der nun geöffneten Datei ergänzt man die Zeilen JAVA_HOME="<Pfad zum Javaverzeichnis>” und CLASSPATH=”<Pfad zum lib-Verzeichnis von Java>”. Sollte bei *PATH* Java ebenfalls noch nicht eingetragen sein, so fügt man in dieser Zeile noch den Pfad zum bin-Verzeichnis von Java ein.

Nun müssen noch einige Plugins für Java installiert werden, damit das Programm reibungslos laufen kann. Die benötigten Plugins sind auf der dieser Arbeit beiliegenden DVD im Ordner Plugins enthalten. Der Ordner apache-ant-1.7.0[1] wird von der DVD an einen beliebigen Ort des Rechners kopiert. Die Plugins ant-jmeter.jar[2], ant-contrib-0.3.jar[3] und jsch-0.1.32.jar[4] sind bereits an der richtigen Stelle diesem Ordner beigelegt. Der Ordner jakarta-jmeter-2.2[5] wird ebenfalls von der DVD an einen beliebigen Ort auf dem Rechner kopiert. Es muss nun noch die Umgebungsvariable für Ant gesetzt werden. Hierzu wird wie oben beschrieben */etc/enviroment* editiert. Die Variable *PATH* wird mit dem bin-Verzeichnis von Ant erweitert. Hinzu kommt die Variable *ANT_HOME*. Sie muss auf das Ant-Verzeichnis verweisen.

Die Datei Log.sh wird von der DVD in das Verzeichnis $HOME/bin kopiert.

Auch SSH sollte auf dem Lastgenerator installiert sein. Ist dies nicht der Fall, so kann es beispielsweise über Yast nachinstalliert werden.

Zu guter Letzt wird der Ordner JDataMining auf den Rechner kopiert.

4.1.3 Einrichtung des Servers

Damit später der Lastgenerator Befehle auf dem Server ausführen kann, sollte SSH installiert sein. Dies kann zum Beispiel über Yast installiert werden.

[1] Herstellerseite: http://ant.apache.org/
[2] Herstellerseite: http://www.programmerplanet.org/pages/projects/jmeter-ant-task.php
[3] Herstellerseite: http://ant-contrib.sourceforge.net
[4] Herstellerseite: http://www.jcraft.com/jsch/
[5] Herstellerseite: http://jakarta.apache.org/jmeter/

Auch auf dem Server muss die Datei Log.sh von der beiliegenden DVD in das Verzeichnis $HOME/bin kopiert werden.

Jetzt wird die Datei “Makefile” von der DVD in ein beliebiges Verzeichnis auf dem Server kopiert. Mit root-Rechten wird in einer Konsole diese Datei mit dem Befehl “make worker-inst” oder “make preforkinst” ausgeführt. Es wird nun automatisch der instrumentalisierte Programmcode aus dem SVN-Repository heruntergeladen, der Apache Webserver kompiliert und letztendlich installiert.

Mit dem Befehl “/usr/local/apache2/bin/apachectl start” können Sie testen, ob der Server funktioniert. Mit “/usr/local/apache2/bin/apachectl stop” beenden Sie den Server wieder.

4.1.4 Erster Start

Wenn das Programm jdatamining.jar das erste Mal ausgeführt wird, dann kann es sein, dass es mit der Meldung “Fehler beim Lesen der Datenbank” wieder abbricht. Dies liegt daran, dass das Programm von einem Tool Gebrauch macht, welches nur starten kann, wenn es eine Datenbankverbindung hat. Deshalb muss dem Programm mitgeteilt werden, wie die Datenbank heißt und wie die Zugangsdaten lauten. Hierzu wird in den Ordner jdatamining/config gewechselt und die dort vorhandene Datei config.properties mit folgenden Werten editiert:

- dbname=<Der Name der Datenbank>
- host=jdbc\:mysql\://<Der Host der Datenbank, zum Beispiel localhost>
- password=<Das Passwort des Benutzers der Datenbank>
- user=<Der Benutzername des Benutzers der Datenbank>

Wenn nun das Programm gestartet wird, sollte es richtig ausgeführt werden.

4.2 Funktionen des Programmes

In dem nachfolgenden Abschnitt werden alle Elemente von JDataMining erklärt. Dabei wird in der Reihenfolge der vorhandenen Register vorgegangen.

4.2.1 Register: Messungen

Das Registerfenster *Messungen* ist das Startfenster der Anwendung. Die Abbildung 4.1 zeigt einen Screenshot dieses Registerfensters. Im Folgenden werden die einzelnen Felder erklärt.

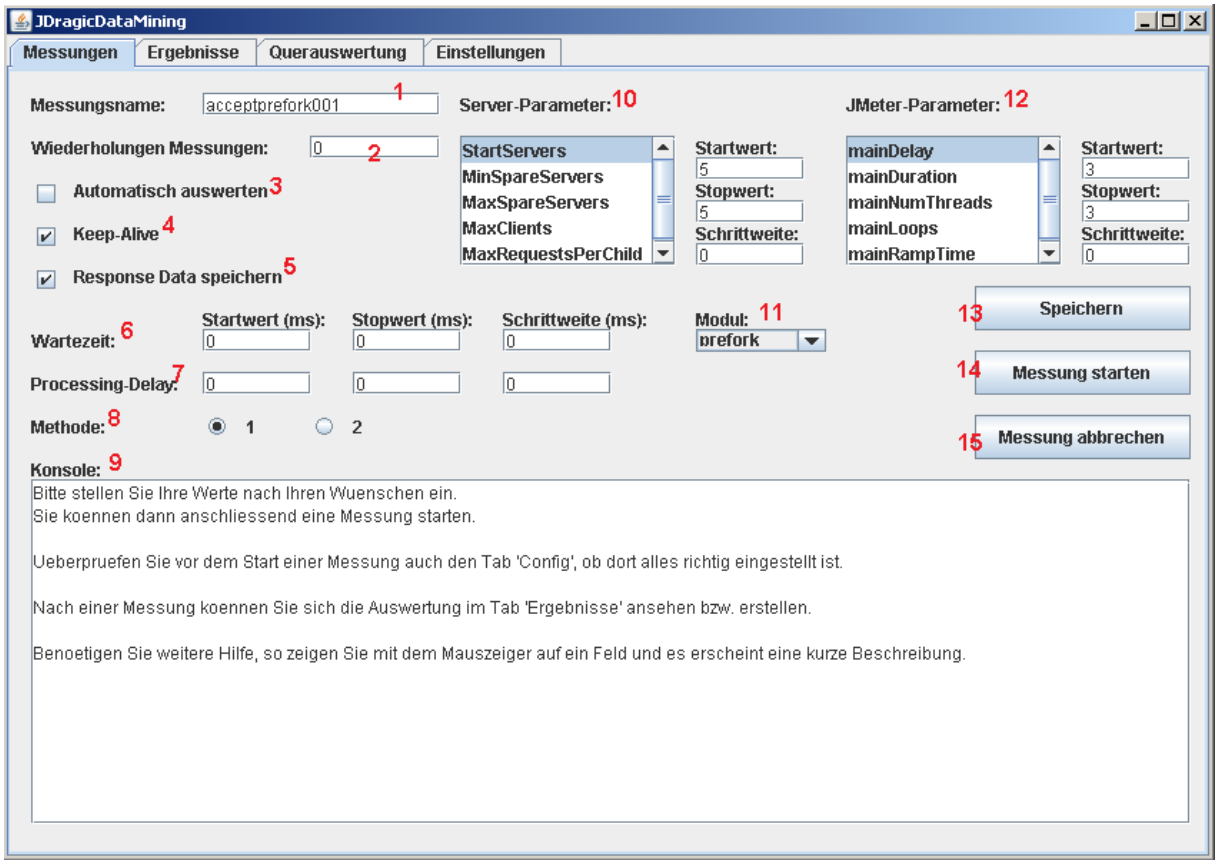


Abbildung 4.1: Registerfenster Messungen

1. Messungsname: Mit diesem Feld kann man einen Messnamen vergeben. Alle Messungen, die dann gestartet werden, tragen diesen Namen. Anmerkung: Zusätzlich bekommt jede einzelne Messung vom Programm automatisch eine eindeutige Identifikationsnummer zugewiesen.

2. Wiederholungen Messungen: In dieses Feld kann eine Zahl eingetragen werden, welche die Anzahl bestimmt, wie oft die einzelnen Messungen wiederholt werden sollen. Das Wiederholen von Messungen ist vor allem dann sinnvoll, wenn sichergestellt werden soll, dass die Ergebnisse einer Messung nicht zufälliger Natur sind.

3. Automatisch auswerten: Wenn dieses Feld aktiviert ist, dann wird im Anschluss an alle Messungen automatisch eine HTML-Seite mit statistischen Größen und verlinkten Graphen erzeugt.

4. Keep-Alive: Dieses Feld bestimmt, ob die Anfragen von JMeter mit oder ohne Keep-Alive gestellt werden sollen.

5. Response Data speichern: Sollte dieses Feld aktiviert sein, so speichert JMeter zu jeder Anfrage die kompletten Daten, die es vom Server erhält. Achtung: Bei vielen Anfragen und/oder großen Dateien auf dem Server können die von JMeter erzeugten Dateien sehr groß werden.

6. Wartezeit: Die Wartezeit bestimmt eine Zeit in ms wie lange beim Beantworten der Anfrage zunächst gewartet werden soll. Mit diesem Wert wird eine Anfrage an eine Datenbank simuliert. Der Wert beginnt mit dem angegebenen Startwert und endet mit dem angegebenen Stoppwert. Die Schrittweite bestimmt dabei die Erhöhung des Parameters bis zum Stoppwert. Hinweis: Der Stoppwert muss größer als der Startwert sein. Darüber hinaus müssen sowohl Startwert als auch Stoppwert restlos durch die Schrittweite teilbar sein.

7. Processing-Delay: Dieser Wert simuliert die Generierung einer Website. Der Wert wird in ms angegeben und verursacht Prozessorrechenleistung. Die Felder für Startwert, Stoppwert und Schrittweite verhalten sich wie bei der Wartezeit.

8. Processing-Delay Methode: Hier wird die Methode der Processing-Delay bestimmt. Eine Beschreibung der einzelnen Methoden ist in Kapitel 3.2.3 zu finden.

9. Konsole: In der Konsole werden alle möglichen Ausgaben des Programmes dargestellt.

10. Server-Parameter: Dieser Bereich listet alle Parameter auf, die für das aktuell aktivierte Modul (siehe Punkt 11) des Servers zur Verfügung stehen. Der Wert für den jeweiligen Parameter beginnt mit dem angegebenen Startwert und endet mit dem angegebenen

Stoppwert. Die Schrittweite bestimmt dabei die jeweilige Erhöhung des Parameters bis zum Stoppwert. Hinweis: Der Stoppwert muss größer als der Startwert sein. Darüber hinaus müssen sowohl Startwert als auch Stoppwert restlos durch die Schrittweite teilbar sein.

11. Modul: Hier wird das zu verwendende Servermodul eingestellt. Dabei ist jedoch zu beachten, dass das hier eingestellte Modul auch auf dem Server installiert sein sollte, da sonst das Programm die Messung nicht durchführen wird.

12. JMeter-Parameter: Dies sind die Parameter, die in JMeter variiert werden können. Insbesondere die Anzahl der Anfragen und der Clients sind zu erwähnen. Auch hier beginnt der Wert für den jeweiligen Parameter mit dem angegebenen Startwert und endet mit dem angegebenen Stoppwert. Die Schrittweite bestimmt dabei die jeweilige Erhöhung des Parameters bis zum Stoppwert. Hinweis: Der Stoppwert muss größer als der Startwert sein. Darüber hinaus müssen sowohl Startwert als auch Stoppwert restlos durch die Schrittweite teilbar sein.

13. Speichern: Mit Klick auf diese Schaltfläche werden die aktuell in der Registerkarte Messungen angezeigten Angaben abgespeichert.

14. Messung starten: Diese Schaltfläche startet die Messungen.

15. Messung abbrechen: An vereinzelten Stellen im Programmablauf kann die Messung mit dieser Schaltfläche abgebrochen werden. Dies ist nur jeweils zwischen Generierung des Messplanes, Messung, Schreiben der JTL-Tabellen, Schreiben der NetStatLog-Tabellen, Schreiben der IOStatLog-Tabellen, Schreiben der SarLog-Tabellen, Schreiben der VMStatLog-Tabellen und Schreiben der CSV-Tabellen möglich.

4.2.2 Register: Ergebnisse

Das Registerfenster *Ergebnisse*[6] (siehe Abbildung 4.2) ist in zwei Bereiche aufgeteilt. Der linke Teil beherbergt die Fläche für die Anzeige von Graphen und verschiedene Interaktionselemente. Der rechte Teil zeigt die Messpunkte zu der aktuell aktivierten Liste an. Die einzelnen Funktionen im Detail:

1. Die Fläche für die Graphen: Sobald der Button Update geklickt wird, wird in dieser Fläche ein Graph angezeigt. angezeigt. Mit einem Rechtsklick in dieses Feld öffnet sich ein

[6]Dieses Fenster entspricht der eingebetteten Klasse JdbcChartListPanel (siehe Kapitel 3.1.3)

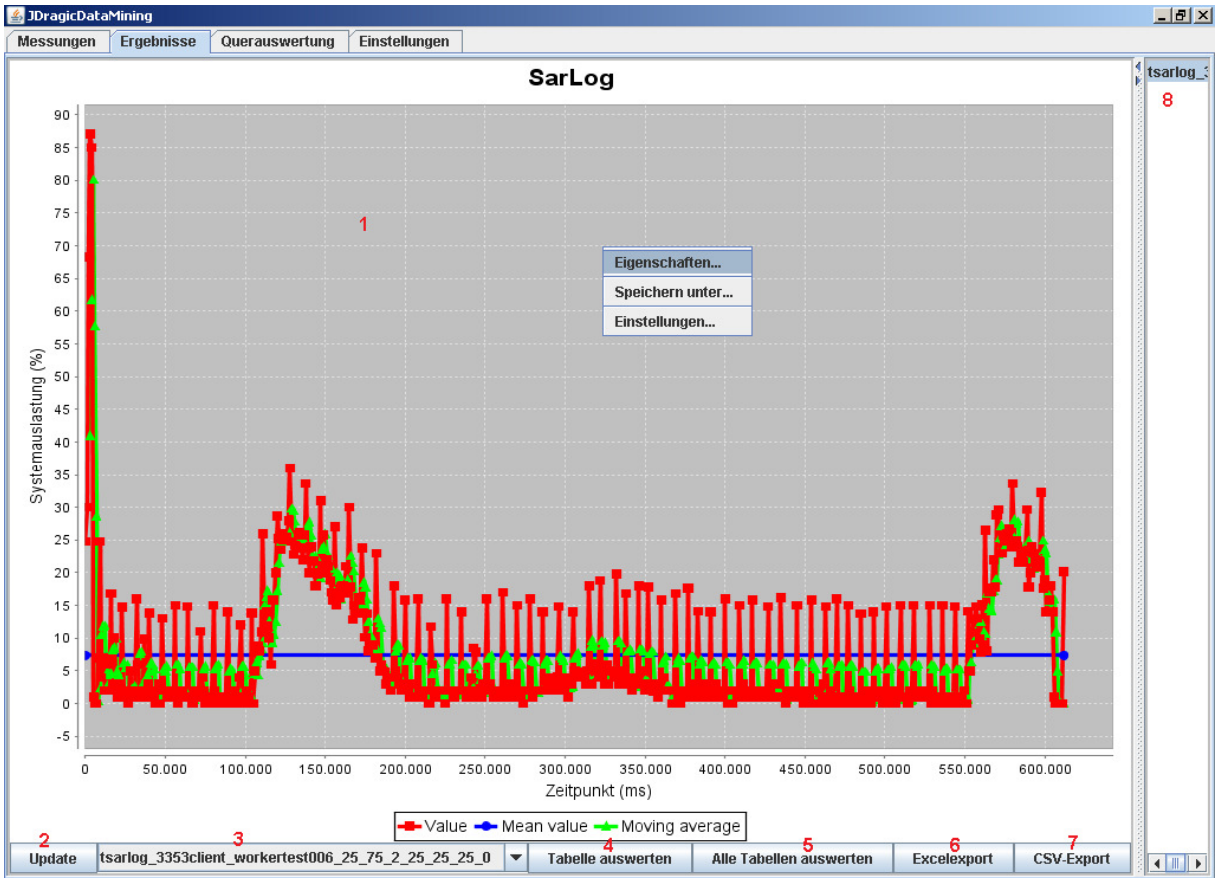


Abbildung 4.2: Registerfenster Ergebnisse

Dialogfenster, in dem man die Optionen Eigenschaften, Speichern unter und Einstellungen wählen kann.
Eigenschaften öffnet ein neues Fenster mit drei Registerkarten. Die erste Registerkarte heißt *Überschrift*: Hier kann festgelegt werden, ob über dem Diagramm eine Überschrift angezeigt werden soll oder nicht. Im Feld Text wird die Überschrift eingegeben. Des Weiteren kann man noch die Schriftart und die Farbe bestimmen.
Die Registerkarte *Diagramm-Eigenschaften* ermöglicht die Beschriftungen der Rubrikenachse und der Werteachse. Auch hier kann die Farbe und die Schriftart bestimmt werden. Außerdem kann angegeben werden, ob Markierungen gezeichnet werden und ob der Wertebereich automatisch justiert werden soll. Ansonsten kann man den Minimalwert und den Maximalwert angeben. Weiterhin kann die Rahmenart bestimmt werden, sowie die Rahmenfarbe, die Hintergrundfarbe und die Orientierung des Bildes.
Die Registerkarte *Sonstiges* bietet die Möglichkeit zu wählen, ob die Zeichnung geglättet werden soll oder nicht. Außerdem können noch weitere Farb- und Darstellungseinstellungen getroffen werden, falls entsprechende Editoren implementiert sind.
Die Option *Speichern unter* des Dialogfensters speichert eine Abbildung des aktuellen Graphen.
Die Option *Einstellungen* öffnet ein neues Fenster. In diesem Fenster kann gewählt werden, ob die Graphen interpoliert werden sollen und welche Graphen genau gezeichnet werden sollen. Dabei ist Value ein Graph aus den gemessenen Werten: Mittelwert zeichnet die Linie ein, wo sich der Mittelwert befindet, gleitender Mittelwert zeichnet einen gleitenden Mittelwert wahlweise für drei, fünf, sieben oder neun benachbarte Punkte ein und Standardabweichung zeichnet einen hellen Bereich ein, welcher der Standardabweichung entspricht. Als letzte Auswahlmöglichkeit kann bestimmt werden, ob Punkte und/ oder Linien gezeichnet werden sollen.

2. Die Schaltfläche *Update* aktualisiert die Abbildung je nach ausgewählter Tabelle in der Tabellenauswahlliste und nach dem ausgewählten Messpunkt in der Messpunktauswahlliste.

3. Die *Tabellenauswahlliste* listet alle Tabellen auf, die in der Datenbank gefunden wurden.

4. Die Schaltfläche *Tabelle auswerten* erstellt eine HTML-Seite mit statistischen Größen und Bildern zu der aktuell ausgewählten Tabelle.

5. Die Schaltfläche *Alle Tabellen auswerten* erstellt eine HTML-Seite mit statistischen Größen und Bildern zu allen Messungen, die im Feld Messungsname (siehe 4.2.1 Punkt 1) der eingegebenen Messung zuzuordnen sind.

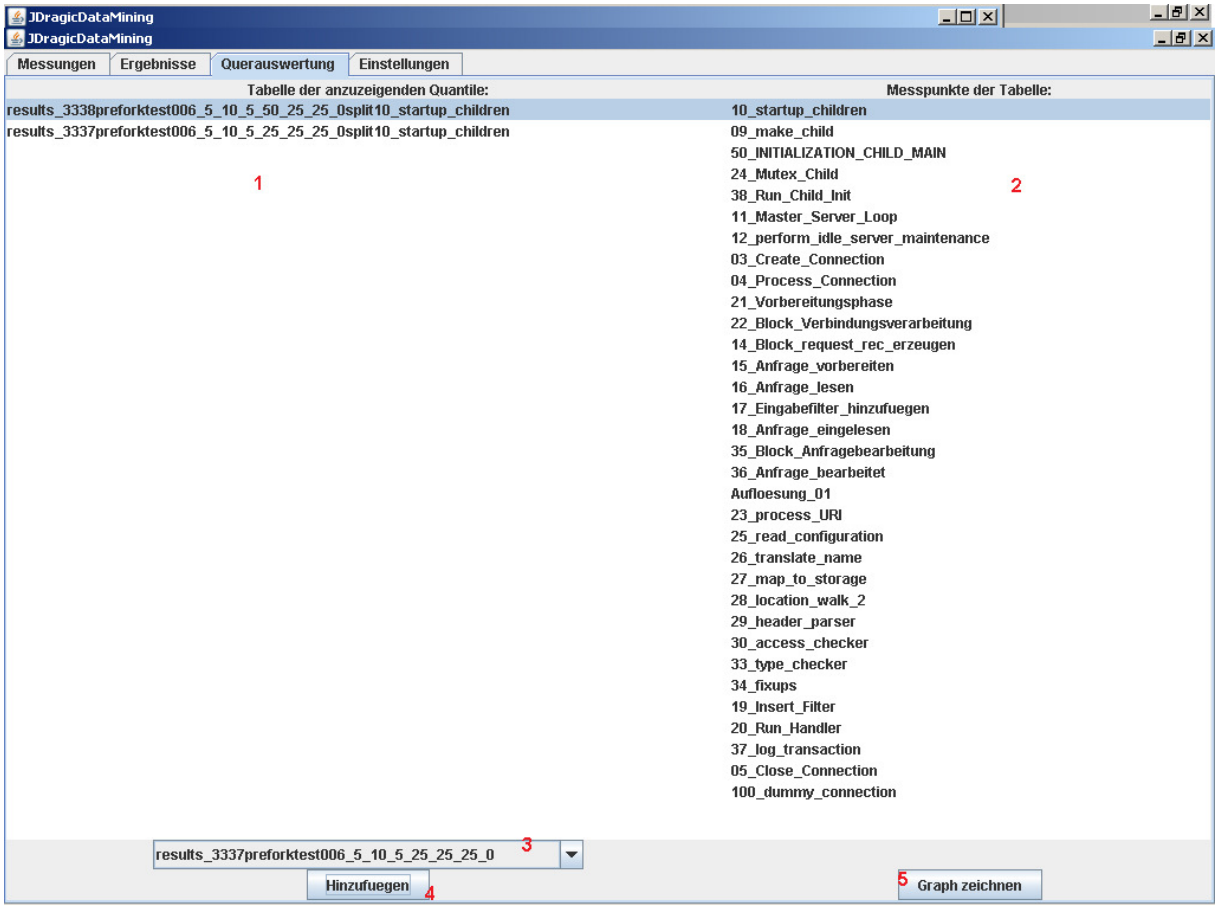


Abbildung 4.3: Registerfenster Querauswertung

6. Die Schaltfläche *Excelexport* exportiert die aktuell gewählte Tabelle in eine Excel-Datei.

7. Die Schaltfläche *CSV-Export* exportiert die aktuell gewählte Tabelle in eine CSV-Datei.

8. Die *Fortschrittsanzeige* zeigt an, wie weit die Auswertung der Tabellen fortgeschritten ist.

9. Die *Messpunktauswahlliste* listet alle Messpunkte auf, welche innerhalb einer Tabelle zu finden sind.

4.2.3 Register: Querauswertung

Das Registerfenster Querauswertung (siehe Abbildung 4.3) dient einer Querauswertung über verschiedene Messungen. In diesem Register werden 90%-Quantile der gewählten Tabellen als Graph dargestellt.

- Tabelle der anzuzeigenden Quantile: In dieser Tabelle werden die zu zeichnenden Tabellen eingetragen.
- Messpunkte der Tabelle: In dieser Tabelle sind alle Messpunkte der aktuell gewählten Datenbanktabelle aufgelistet.
- Tabellenauswahlbox: Hier kann eine Tabelle aus der Datenbank ausgewählt werden.
- Hinzufügen: Mit dieser Schaltfläche wird ein Messpunkt aus der Messpunktetabelle zur Auswahl der zu zeichnenden Quantile hinzugefügt.
- Graph zeichnen: Diese Schaltfläche zeichnet den Graphen.

4.2.4 Register: Einstellungen

In dem Register Einstellungen werden die wichtigen Einstellungen zum Betreiben des Programmes getroffen. Abbildung 4.4 zeigt einen Screenshot dieses Registers. Die Felder im Detail:

1. Ergebnispfad: Hier wird der Pfad angegeben, unter dem die generierten HTML-Seiten abgespeichert werden sollen.
2. Temp-Tabelle: In dieses Feld wird der Name der Tabelle angegeben, die auf dem Server während einer Messung angelegt wird.
3. Host: Dies ist die Adresse des Hosts.
4. Datenbankname: Gibt den Namen der Datenbank an.
5. Benutzername: Dies ist der Benutzername der Datenbank.
6. Passwort: Das zugehörige Passwort für die Datenbank.
7. max. Heap-Speicher: Gibt die maximale Größe des zu verwendenden Heap-Speichers an. Wird in der aktuellen Version vorerst nicht benötigt.
8. Konfigurationsdatei: Gibt den Pfad an, unter dem die Vorlage der Server-Konfigurationsdatei liegt.
9. Dateiname JMeter-XML: Gibt den Namen des generierten Messplanes an.
10. JMeter-Home: Der Pfad zur JMeter-Installation.

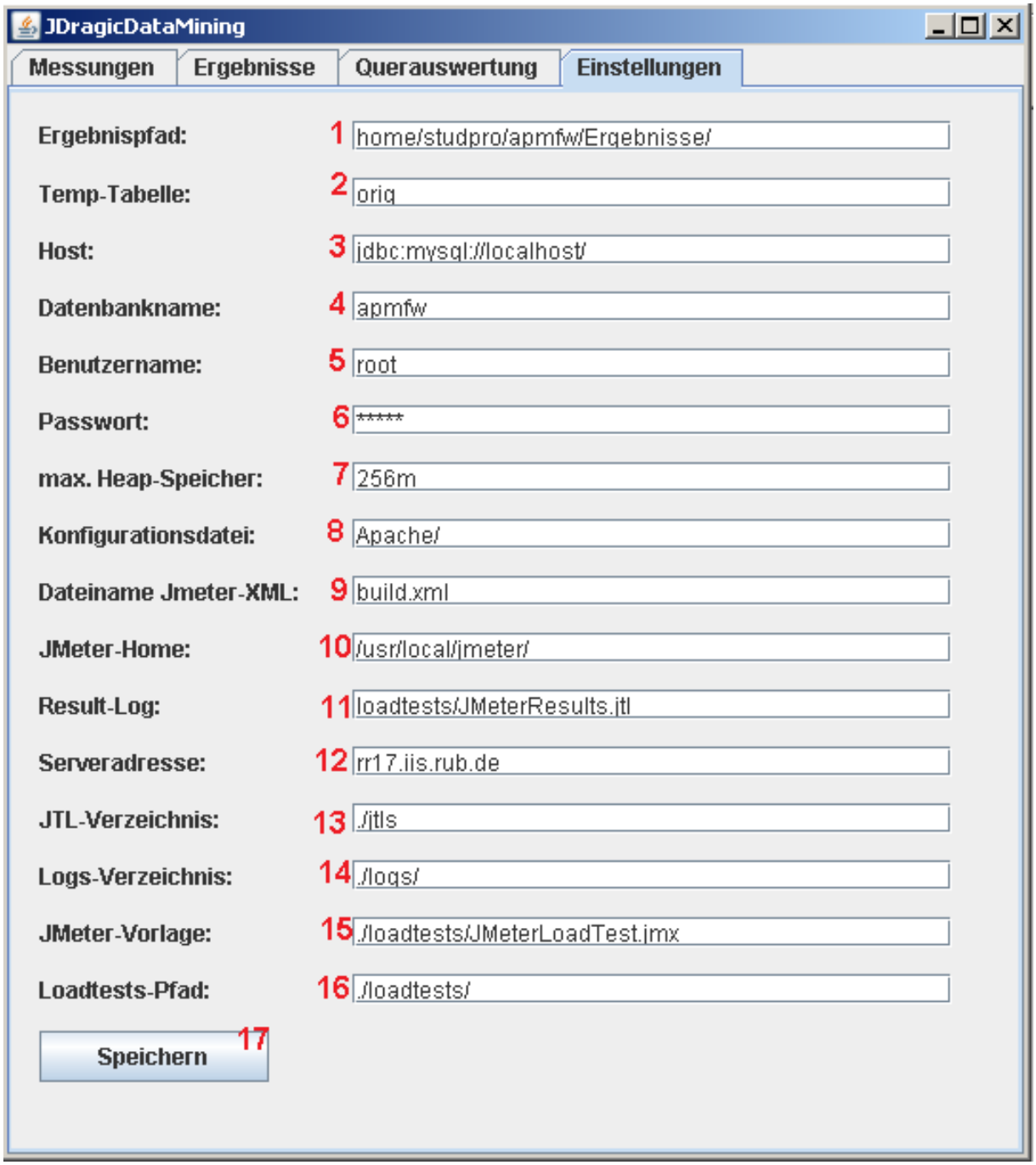


Abbildung 4.4: Registerfenster Einstellungen

11. Result-Log: Gibt den Namen der Result-Log an. Das ist eine HTML-Seite, die von JMeter generiert wird.

12. Serveradresse: Die Adresse des zu messenden Servers.

13. JTL-Verzeichnis: Pfad zum Verzeichnis, in dem die erzeugten JTL-Dateien abgelegt werden sollen.

14. Logs-Verzeichnis: Hier werden alle Log-Dateien abgelegt.

15. JMeter-Vorlage: Pfad und Name der JMeter-Vorlagedatei.

16. Loadtests-Pfad: Hier sind alle JMeter-Messpläne abgespeichert.

17. Die Schaltfläche Speichern speichert die getroffenen Einstellungen.

4.3 Durchführen einer Messung

Bevor mit dem Messen begonnen werden kann, muss sichergestellt sein, dass der instrumentalisierte Quellcode auf dem Server installiert ist (siehe 4.1.3). Außerdem darf der Webserver nicht gestartet sein. Falls er es ist, sollte er beendet werden.

Nun wird JDataMining mit root-Rechten gestartet und das Register Einstellungen angeklickt. Hier sollten alle Einstellungen den Wünschen entsprechend konfiguriert werden. Neben den Dateipfaden zum Speichern der verschiedenen Messdateien, der Serveradresse und der Datenbankanbindung ist insbesondere wichtig, wo sich die JMeter-Vorlage und die Server-Konfigurationsdateivorlage befinden. Hierzu werden entweder die auf der beiliegenden DVD vorhandenen Vorlagen ausgewählt oder aber eigene Vorlagen erstellt. Zum Erstellen einer Vorlage wird zunächst mit dem Programm JMeter ein Testplan erzeugt. Anschließend wird der Testplan in einem Editor aufgerufen und nach sinnvollen Stellen für Parameterplatzhalter durchsucht. An allen Stellen, an denen ein Parameter variiert werden soll, wird ein Platzhalter nach dieser Form eingefügt “–!PARAM::NAME–!”, wobei NAME durch einen Parameternamen ersetzt wird.

Sind nun alle Einstellungen getroffen, wird das Register Messungen angeklickt. Als Erstes wird das gewünschte Modul ausgewählt. Entsprechend verändern sich die zur Auswahl stehenden Server-Parameter. Nun wird ein Name für die Gesamtmessung vergeben. Alle Parameter können theoretisch variiert werden, wobei jedoch eine Beschränkung einprogrammiert wurde. Von den Server-Parametern können maximal drei Parameter und von den JMeter-Parametern können maximal zwei Parameter variiert werden. Grund: Angenommen, es werden drei Server-Parameter und zwei JMeter-Parameter jeweils mit zwei möglichen Werten belegt, so gibt es be-

reits 2^5 = 32 mögliche Kombinationen. Hinzu kommt noch die Wartezeit und die Processing-Delay (angenommen ebenfalls jeweils zwei mögliche Werte), dann gibt es 2^7 = 128 mögliche Kombinationen. Da es empfehlenswert ist, Messungen zu wiederholen, um Zufallstreffer auszuschließen, werden zum Beispiel die Messungen noch zweimal wiederholt. Nun sind es bereits 512 Messungen. Da in der Realität jedoch meistens einzelne Parameter nicht nur um zwei, sondern gleich um einige Werte mehr verändert werden, ist es offensichtlich wie schnell die Komplexität ansteigen kann und eine ausführliche Betrachtung sämtlicher Messergebnisse behindert wird. Geht man davon aus, dass jede einzelne Messung zum Beispiel nur eine Minute dauert, ist man bei diesem Beispiel schon bei 512 Minuten Laufzeit. Hinzu käme noch die Zeit zur Berechnung der Werte und gegebenenfalls die Zeit zur Generierung der HTML-Seite mit Bildern.

Die Parameter werden wie folgt eingestellt: Die gewünschten Parameter erhalten einen Startwert und einen Stoppwert. Die Schrittweite gibt an, in welchen Schritten die Parameter variiert werden. Es ist wichtig, dass sowohl Startwert als auch Stoppwert durch die Schrittweite restlos teilbar sind. Ist dies nicht der Fall, wird der Benutzer vom Programm darauf hingewiesen. Wenn bei Schrittweite eine Null gesetzt wird, so heißt dies, dass dieser Parameter nicht variiert werden soll. Es wird hier nur der Startwert verwendet. Mit Klick auf die Schaltfläche *Speichern*, werden die Angaben gespeichert.

Nun kann noch bestimmt werden, ob nach der Messung automatisch ausgewertet werden soll, ob die Anfragen mit oder ohne Keep-Alive gestellt werden sollen und ob die Response Data gespeichert werden soll.

Ein Klick auf *Messung starten* startet die Messung. In der Konsole erscheint immer der aktuelle Stand jeder Messung und was das Programm gerade macht. Tipp: Es ist ratsam, die Messungen über Nacht, am Wochenende oder zu einem Zeitpunkt, an dem nicht am System gearbeitet wird, laufen zu lassen, denn das Messen kann mehrere Stunden dauern und währenddessen sollte das System nicht von außen beeinflusst werden.

5 Auswertung

Mit dem in den vorherigen Kapiteln entwickelten und vorgestellten Programm JDataMining wurde der Apache Webserver mit dem Modul Worker und dem Modul Prefork untersucht. Die Ergebnisse werden in diesem Kapitel aufgezeigt. Das Prefork- und das Worker-Modul werden gegenübergestellt und verglichen. Eventuelle Gemeinsamkeiten und Unterschiede werden herausgearbeitet.

Insgesamt wurden 2736 Messungen durchgeführt. Jedoch wird aus forschungspragmatischen Gründen hier nur eine sehr kleine Auswahl an Messergebnissen präsentiert. Alle Ergebnisse finden sich aber auf der beiliegenden DVD. Darüber hinaus befindet sich auf der DVD eine Tabellenübersicht über alle Parameterveränderungen jeder einzelnen Messung.

Bevor mit der eigentlichen Auswertung begonnen wird, werden zunächst die wesentlichen Unterschiede beider Module erklärt. Anschließend erfolgt eine kurze Erläuterung zu der zeitlichen Auflösung des Messframeworks an den Orten der Messpunkte.

5.1 Unterschiede zwischen Prefork und Worker

Sowohl das Prefork-Modul als auch das Worker-Modul sind Multiprocessing-Module (MPMs). Multiprocessing-Module wurden entwickelt, um die Zuordnung von Verbindungen zu Prozessen bzw. Threads vom Apache Kern zu lösen. Dadurch wird die Optimierung des Webservers für verschiedene Plattformen vereinfacht. Es wird genau ein MPM beim Kompilieren des Servers eingebunden. Unter Linux stehen beispielsweise die MPMs Worker, Prefork sowie das experimentelle MPM Event zur Auswahl. Eine komplette Liste aller MPMs für alle Plattformen gibt es unter [Fou06].

Das Prefork-Modul ist ein rein prozessbasiertes Modul. Bereits während der Initialisierung des Servers werden ein paar Kindprozesse erzeugt. Diese Prozesse lauschen die Sockets des Servers ab. Zu einem Zeitpunkt ist jedoch nur ein Prozess der sogenannte Listener. Deshalb wird ein Mechanismus benötigt, der für die korrekte Synchronisierung sorgt. Beim Apache Webserver wird hierzu ein Mutex verwendet. Derjenige Prozess, welcher Listener ist, hat diesen Mutex gesperrt. Wenn der Listener nun auf einem Socket eine Anfrage bemerkt, nimmt er diese Anfrage entgegen, gibt den Mutex frei, damit ein anderer Prozess Listener werden kann, und er

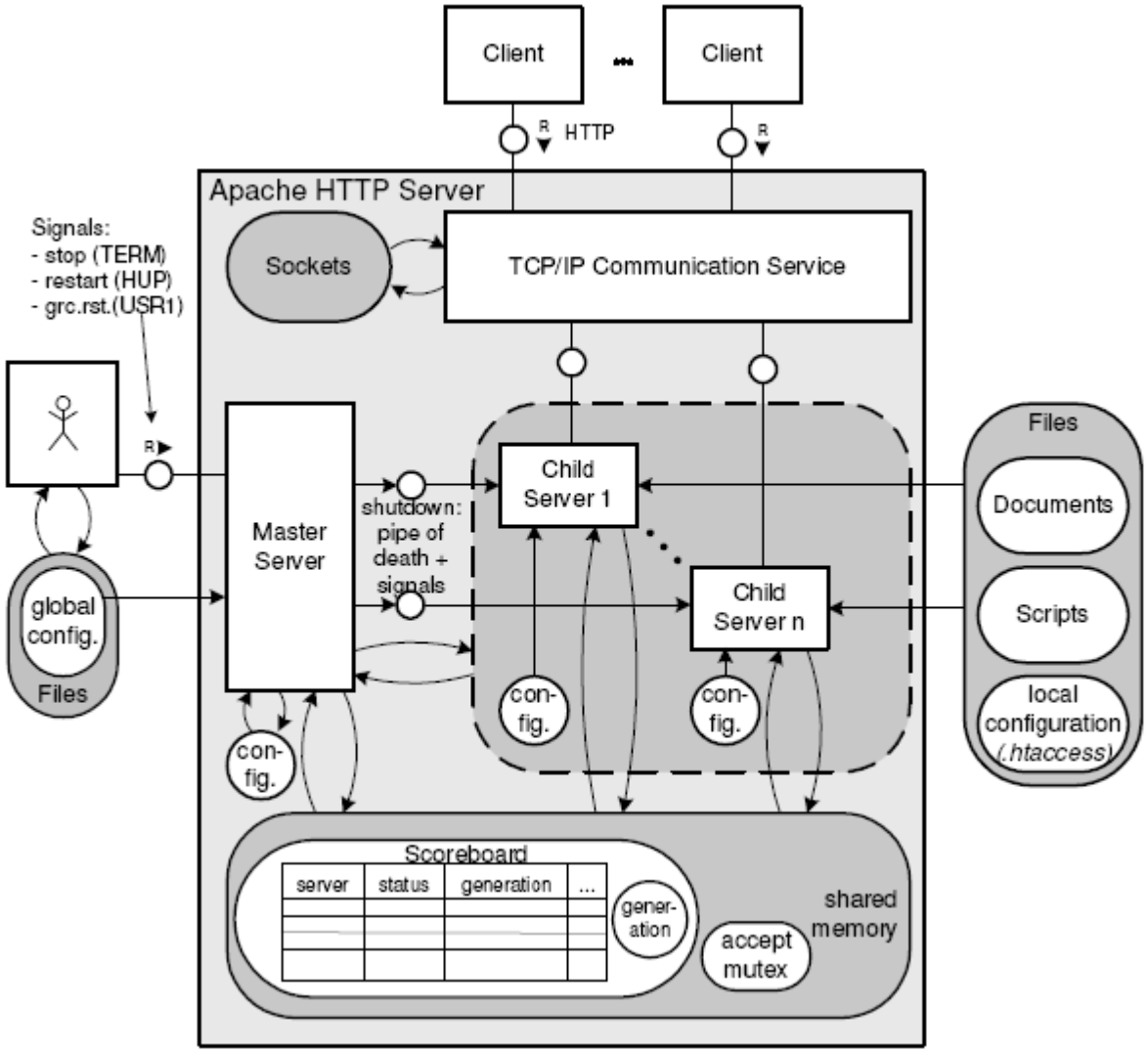


Abbildung 5.1: Apache Prefork MPM

bearbeitet die Anfrage. Da immer auf einen Listener ein Nachfolger folgt, wird dieses Modell auch Leader-Follower-Modell genannt. Eine ausführliche Beschreibung des Prefork-Moduls findet sich in [GKKS04]. Des Weiteren ist [Ges05] zu empfehlen, da hier der Apache Webserver mit Prefork-Modul untersucht und auf sein Lastverhalten gemessen wurde. Die Abbildung 5.1 zeigt den groben Aufbau des Apache Webservers mit dem Prefork-Modul. Es ist aus [GKKS04] entnommen.

Das Worker-Modul ähnelt auf den ersten Blick dem Prefork-Modul. Es ist eine Mischung aus prozessbasiertem und threadbasiertem Modul. Es werden ebenfalls bereits während der Initialisierungsphase des Servers ein paar Kindprozesse erzeugt. Diese Prozesse dienen jedoch nicht direkt der Anfragenbearbeitung wie beim Prefork-Modul. Stattdessen hat jeder Kindprozess

eine bestimmte Anzahl an Threads. Diese Threads übernehmen die Aufgabe der Anfragenbearbeitung. In jedem Prozess gibt es einen Listener-Thread (also gibt es folglich mehrere Listener im Server, was wiederum eine Synchronisierung durch einen Mutex erfordert) und mehrere Worker-Threads. Sobald ein Listener-Thread eine Anfrage auf einem Socket entgegennimmt, übergibt er diese Anfrage an einen unbeschäftigten Worker-Thread, welcher sich dann um die Bearbeitung der Anfrage kümmert. Der Listener-Thread selbst bleibt weiterhin ein Listener, jedoch sind zunächst die anderen Listener-Threads an der Reihe, da mittels einer Warteschlange dafür gesorgt wird, dass sich die Listener-Threads abwechseln. Die Übergabe einer Anfrage vom Listener-Thread zum Worker-Thread geschieht nur innerhalb eines Prozesses und ist somit nicht prozessübergreifend. Es ist also nicht möglich, die Anfrage an einen Worker-Thread aus einem anderen Prozess zu übergeben. Das Worker-Modul ist ausführlich in [GKKS04] beschrieben. In [BNO+07] wurde der Quellcode des Apache Webservers ausführlich in Hinsicht auf das Worker-Modul untersucht sowie Messungen zur Untersuchung des Lastverhaltens des Apache Webservers mit dem Worker-Modul durchgeführt. Die Abbildung 5.2 zeigt den groben Aufbau des Apache Webservers mit eingebundenem Worker-Modul (Quelle: [GKKS04]).

Der wesentliche Unterschied zwischen Prefork- und Worker-Modul besteht also darin, dass die Anfragen beim Prefork-Modul von Prozessen und beim Worker-Modul von Threads (und nur noch indirekt von Prozessen) bearbeitet werden. Vor- und Nachteile beider Module liegen auf der Hand. Wo das Prefork-Modul mit seinen schwergewichtigen Prozessen viel Speicher verbraucht, verwendet das Worker-Modul leichtgewichtige Threads. Aus Sicht der Performance müsste also das Worker-Modul besser abschneiden. Andererseits müsste das Prefork-Modul stabiler sein, denn wenn ein Prozess abstürzt, so sind die anderen Prozesse davon nicht betroffen, da jeder Prozess seinen eigenen Speicherplatz hat. Threads teilen sich dagegen einen bestimmten Speicherplatz und sollte ein Thread abstürzen, so würden eventuell alle Threads mit dem gleichen Speicherbereich (also innerhalb des gleichen Prozesses) auch abstürzen.

Je nach späterem Einsatzgebiet des Apache-Webservers ist also entsprechend das richtige MPM auszuwählen. Die nachfolgenden Abschnitte werden Messergebnisse aufzeigen, welche mit dem in dieser Arbeit zuvor programmierten Messprogramm gewonnen wurden. Es wird gezeigt, in welcher Situation welches MPM die besseren Ergebnisse erzielt, wo sie eventuell ein gleiches Verhalten haben und wo sich eventuell Flaschenhälse innerhalb der MPMs befinden.

5.2 Einschub: Erkenntnisse beim Prefork-MPM

Während der Untersuchung des Lastverhaltens beider MPMs sind ein paar Erkenntnisse zum Vorschein gekommen, die aufzeigen, dass die MPMs anders als erwartet implementiert wurden. An dieser Stelle werden nun zwei interessante Implementierungen des Prefork MPMs vorge-

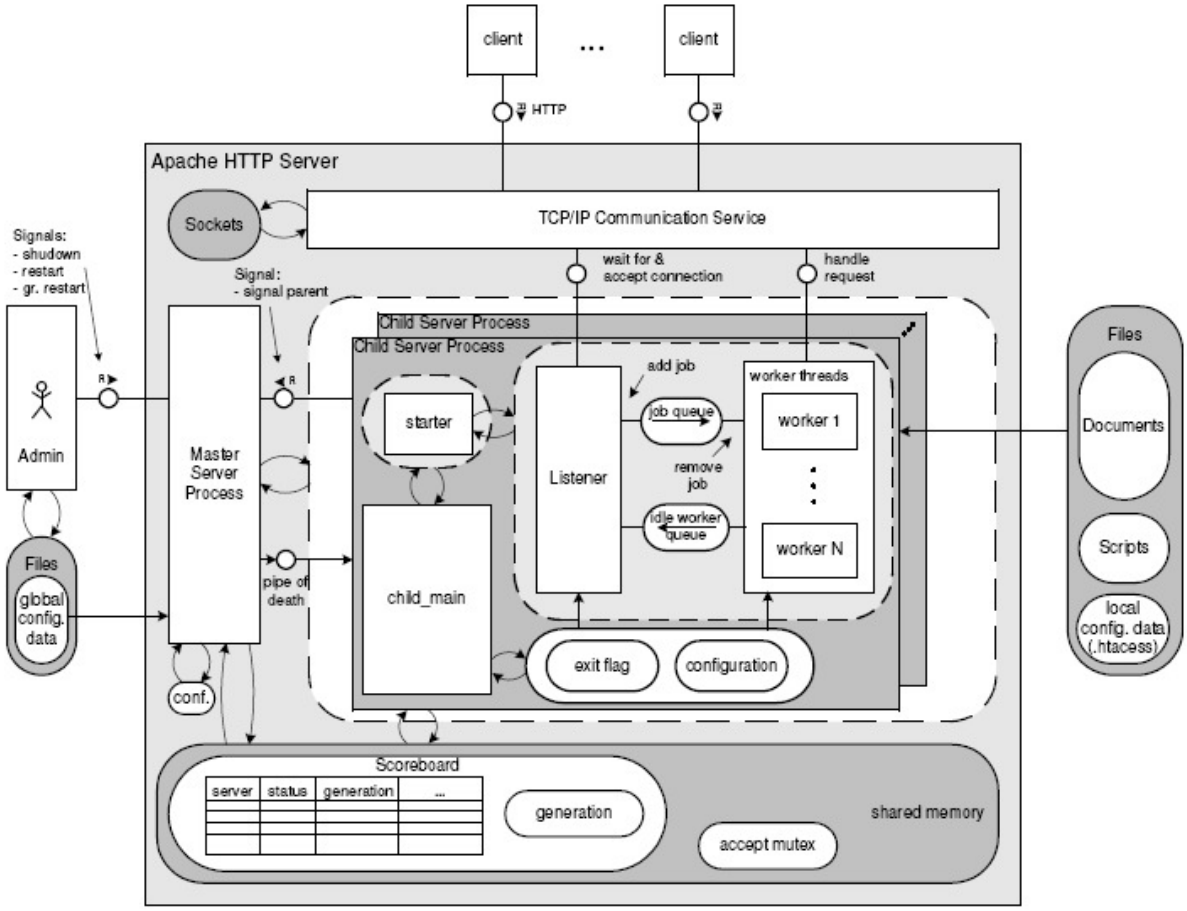


Abbildung 5.2: Apache Worker MPM

stellt.

5.2.1 Leader-Follower-Implementierung

In der Literatur wird oft davon gesprochen, dass das Prefork-MPM ein Beispiel für das Leader-Follower-Modell ist. Das Leader-Follower-Prinzip ist ein Gestaltungsmuster (Design Pattern) zur Regelung von beispielsweise Zuordnungen von Anfragen an Prozesse. Für das Prefork-Modul würde es bedeuten, dass der aktuelle Listener der aktuelle Leader ist. Alle idle Listener sind potentielle Follower. Ein bestimmter idle Listener ist der tatsächliche Follower. Dieser wird durch das Leader-Follower-Prinzip bestimmt. Auf einen Leader muss immer ein Follower folgen, welcher dann als nächstes zum Leader wird. Leader, welche ihre Arbeit erledigt haben, werden ans Ende der Leader-Follower-Warteschlange gesetzt.

Angeblich ist dieses Muster im Prefork MPM implementiert. Bei genauerer Betrachtung, insbesondere des Quellcodes, stellt sich jedoch heraus, dass dieses Muster nur indirekt im Prefork MPM verwendet wird. Tatsächlich wird es dem Betriebssystem überlassen, inwiefern dieses Muster gehandhabt wird. Im Prefork MPM selbst blockieren alle potentiellen Listener im Zustand *accept_mutex_on* (siehe auch 5.2.2). Nur ein potentieller Listener kann allerdings zu einem Zeitpunkt ein tatsächlicher Listener sein. Wer dies wird, das entscheidet das Betriebssystem. Es ist also möglich, dass verschiedene Betriebssysteme auf verschiedene Arten den aktuellen Listener bestimmen.

5.2.2 Problemfall Prozesszerstörung

Da ein potentieller Listener beim Warten auf eine Anfrage im Zustand *accept_mutex_on* blockiert, könnte es sein, dass er nicht bemerkt, dass er eventuell zerstört werden soll. Daher muss es die Möglichkeit geben, es dem Prozess mitzuteilen. Dies könnte beispielsweise mit Signalen geschehen. Jedoch ist dies im Prefork-Modul nicht der Fall. Die Entwickler des Prefork-MPM haben sich für eine Lösung entschieden, welche direkt in allen Betriebssystemen funktioniert und so mussten sie keine Fallunterscheidungen zwischen Betriebssystemen verwenden.

In den BSD-Betriebssystemen, wie zum Beispiel FreeBSD, gibt es einen Kernel-Filter, welcher ein *accept()* auf die Sockets legt. Es wird so lange für dahinter liegende Anwendungen wie dem Apache der Socket blockiert, bis eine vollständige HTTP-Anfrage eingetroffen ist. Solange dies nicht der Fall ist, wird die ankommende Verbindung gepuffert.

Damit ein Prozess beendet werden kann, wird nun eine sogenannte Dummy-Verbindung aufgebaut, das heißt, der Server verbindet sich selbst mit dem Socket und schickt dann eine kurze Dummy-Anfrage über diese Verbindung. Die Verbindung wird danach sofort geschlossen. Der

Prozess hat nun eine Anfrage erhalten. Er kann also den Mutex freigeben. Nun ist er nicht mehr in dem blockierenden Zustand *accept_mutex_on* und kann nun prüfen, ob er zerstört werden soll.

Hinweis: Mit diesem Verfahren kann immer nur der aktuelle Listener zerstört werden. Soll also ein bestimmter Prozess beendet werden, welcher sich in einem blockierendem *accept_mutex_on* Zustand befindet, so muss dieser Prozess der Listener werden. Wann und wie ein Prozess Listener wird, ist in Kapitel 5.2.1 beschrieben.

Wie wird überhaupt bestimmt, ob ein Prozess sterben muss? Hierzu ist die Funktion *perform_idle_server_maintenance* zuständig. Sie wurde für das Worker MPM bereits in [BNO+07] beschrieben, soll hier aber nochmals detaillierter für das Prefork MPM erläutert werden.

Die Funktion *perform_idle_server_maintenance* beginnt zunächst mit einer For-Schleife. Sollte der Indexzähler der Schleife mindestens so groß sein wie die Variable *ap_max_daemons_limit* und sollte des Weiteren die Anzahl freier Slots im Scoreboard gleich der *idle_spawn_rate*[1] sein, so wird die Schleife sofort wieder verlassen. Andernfalls iteriert die Schleife über alle Einträge der Prozesse im Scoreboard, um den aktuellen Status zu erfahren. In jedem Durchgang wird bei dem aktuellen Eintrag überprüft, ob der Status gleich *SERVER_DEAD* ist. Falls ja, wird noch überprüft, ob die Anzahl freier Slots kleiner als die *idle_spawn_rate* ist. Trifft auch dies zu, dann wird dieser Slot als frei markiert und die Anzahl freier Slots um eins inkrementiert.

Sollte der Status nicht gleich *SERVER_DEAD* sein, so wird die Anzahl insgesamt nicht toter Prozesse um eins erhöht und der zuletzt bemerkte Sloteintrag eines nicht toten Prozesses vermerkt. Falls darüber hinaus der Status sogar gleich *SERVER_STARTING* oder *SERVER_READY* ist, so wird der Zähler der idle-Prozesse um eins inkrementiert.

Nachdem die Schleife beendet wurde, wird die Variable *ap_max_daemons_limit* mit dem Index des zuletzt gefundenen toten Prozesses belegt und um eins erhöht. Die Schleife ist also dafür zuständig, die Statusinformationen aller Prozesse zu sammeln und zu merken. Anschließend kennt sie die Anzahl nicht toter Prozesse, die Anzahl freier Slots im Scoreboard, die Anzahl unbeschäftigter Prozesse und die Position des zuletzt gefundenen nicht toten Prozesses. Abbildung 5.3 zeigt den Aufbau dieser Schleife.

Mit den nun vorliegenden Informationen wird der nächste Teilabschnitt der Funktion *perform_idle_server_maintenance* betreten. Dieser Abschnitt ist in Abbildung 5.4 visualisiert. Hier wird zunächst überprüft, ob die Anzahl unbeschäftigter Prozesse größer ist als die maximale Grenze für freie Daemons (*ap_daemons_max_free*). Falls ja, dann bedeutet dies, dass zu viele

[1] Die idle_spawn_rate gibt die Anzahl der im nächsten Wartungszyklus zu erzeugenden Kinder an. Sie wird in jedem Zyklus verdoppelt bis zu der maximalen Grenze von *MAX_SPAWN_RATE* (standardmäßig bei 32). Sie wird zurück auf eins gesetzt, falls es in einem Wartungszyklus nicht nötig war neue Kinder zu produzieren.

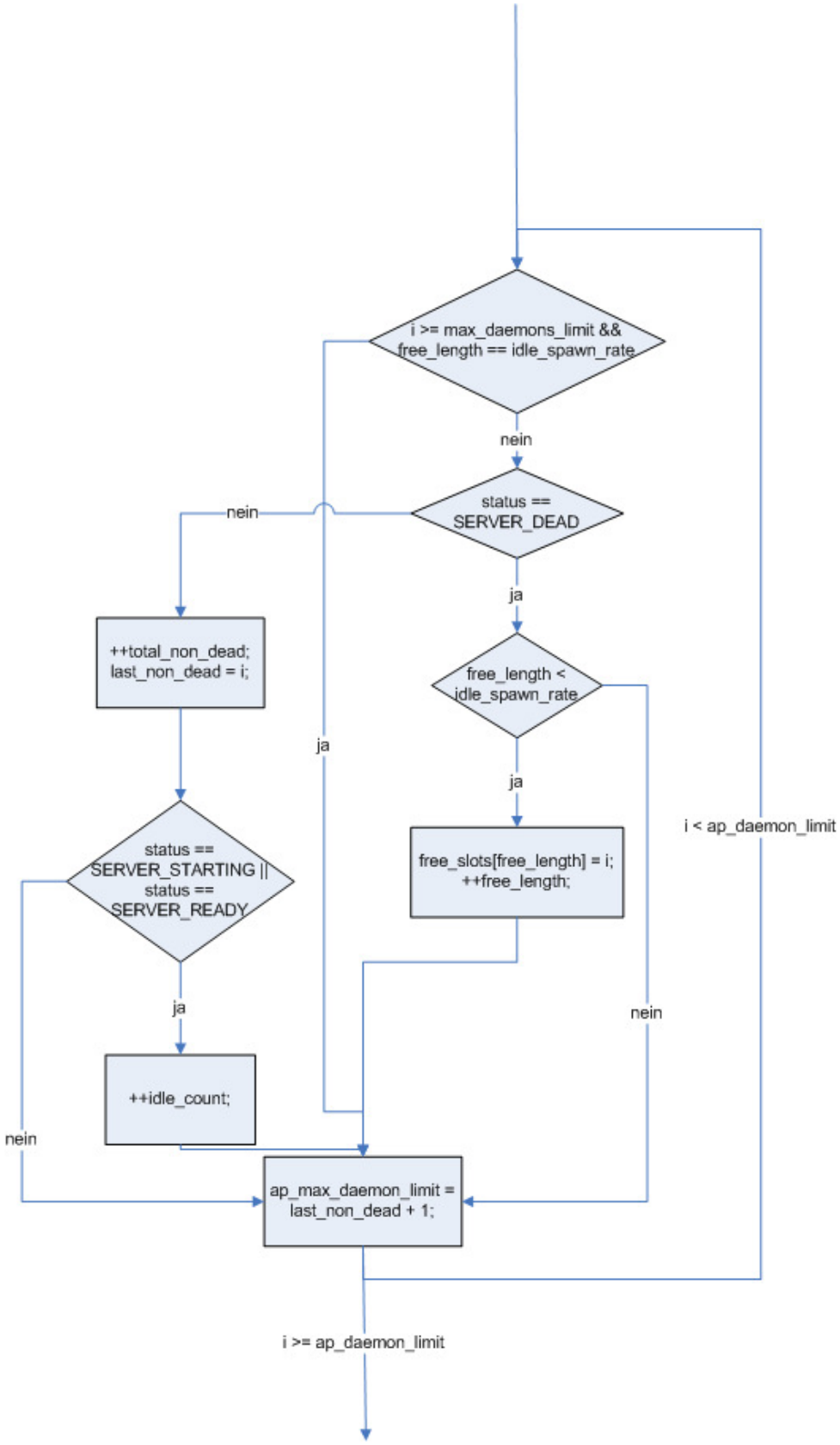


Abbildung 5.3: perform_idle_server_maintenance - Teil 1

unbeschäftigte Prozesse vorhanden sind. Deshalb wird ein Signal in die Pipe-of-Death geschrieben, welches für die Zerstörung eines Prozesses sorgt. Darüber hinaus wird die *idle_spawn_rate* auf 1 zurückgesetzt, damit im nächsten Zyklus nicht unnötigerweise zu viele Prozesse nachproduziert werden. Die Funktion *perform_idle_server_maintenance* ist dann in diesem Fall bereits fertig.

Falls nein, so wird überprüft, ob eventuell die Anzahl unbeschäftigter Prozesse kleiner ist als die minimale Grenze für freie Daemons (*ap_daemons_min_free*). Trifft dies nicht zu, so wird die *idle_spawn_rate* auf 1 gesetzt (denn es sind genügend unbeschäftigte Prozesse vorhanden und es müssen im nächsten Zyklus nicht mehr so viele Prozesse nacherzeugt werden) und auch hier ist die Funktion *perform_idle_server_maintenance* dann beendet. Sollte es jedoch zutreffen, dann gibt es zu wenig unbeschäftigte Prozesse. Es wird mit der Überprüfung fortgefahren, ob freie Slots im Scoreboard vorhanden sind. Gibt es keine freien Slots, dann wird die *idle_spawn_rate* auf 1 gesetzt, da nicht mehr genug Platz für weitere Prozesse vorhanden ist. Hiermit wäre die Funktion *perform_idle_server_maintenance* ebenfalls beendet.

Sollten jedoch freie Slots zur Verfügung stehen, so wird die Funktion *make_child* (in [BNO+07] ist diese Funktion ausführlich beschrieben) innerhalb einer Schleife so oft aufgerufen, wie es freie Slots gibt. Dies sorgt dafür, dass neue Prozesse erzeugt werden und diese Prozesse einen freien Slot im Scoreboard zugeordnet bekommen.

Nachdem nun Prozesse erzeugt wurden, wird kontrolliert, ob die Variable *hold_off_on_exponentiell_spawning*[2] > 0 ist. Falls ja, dann wird sie um eins dekrementiert und die Funktion *perform_idle_server_maintenance* erreicht ihr Ende. Falls nein, dann wird überprüft, ob die *idle_spawn_rate* kleiner als *MAX_SPAWN_RATE* (Standard ist 32) ist. Ist dies der Fall, wird die *idle_spawn_rate* verdoppelt und *perform_idle_server_maintenance* ist beendet. Ansonsten ist die Funktion direkt beendet.

Anmerkung: Beim Worker MPM existiert die Funktion *perform_idle_server_maintenance* ebenfalls. Sie ist im Wesentlichen gleich implementiert. Der hauptsächliche Unterschied ist selbstverständlich, dass nicht der Status der Prozesse, sondern der Status der einzelnen Threads kontrolliert wird.

[2]Diese Variable ist bei Start des Servers auf 10 gesetzt. Mit jedem Durchgang der Funktion perform_idle_server_maintenance wird sie um eins dekrementiert, falls sie größer null ist. Sobald sie gleich 0 ist, bleibt sie auf diesem Wert und ab dann darf die idle_spawn_rate, falls nötig, verdoppelt werden. Dieses Verfahren stellt sicher, dass bei einem Graceful-Restart nicht sofort zu viele Prozesse erzeugt werden, denn im Speicher könnten noch alte Prozesse der vorherigen Servergeneration arbeiten.

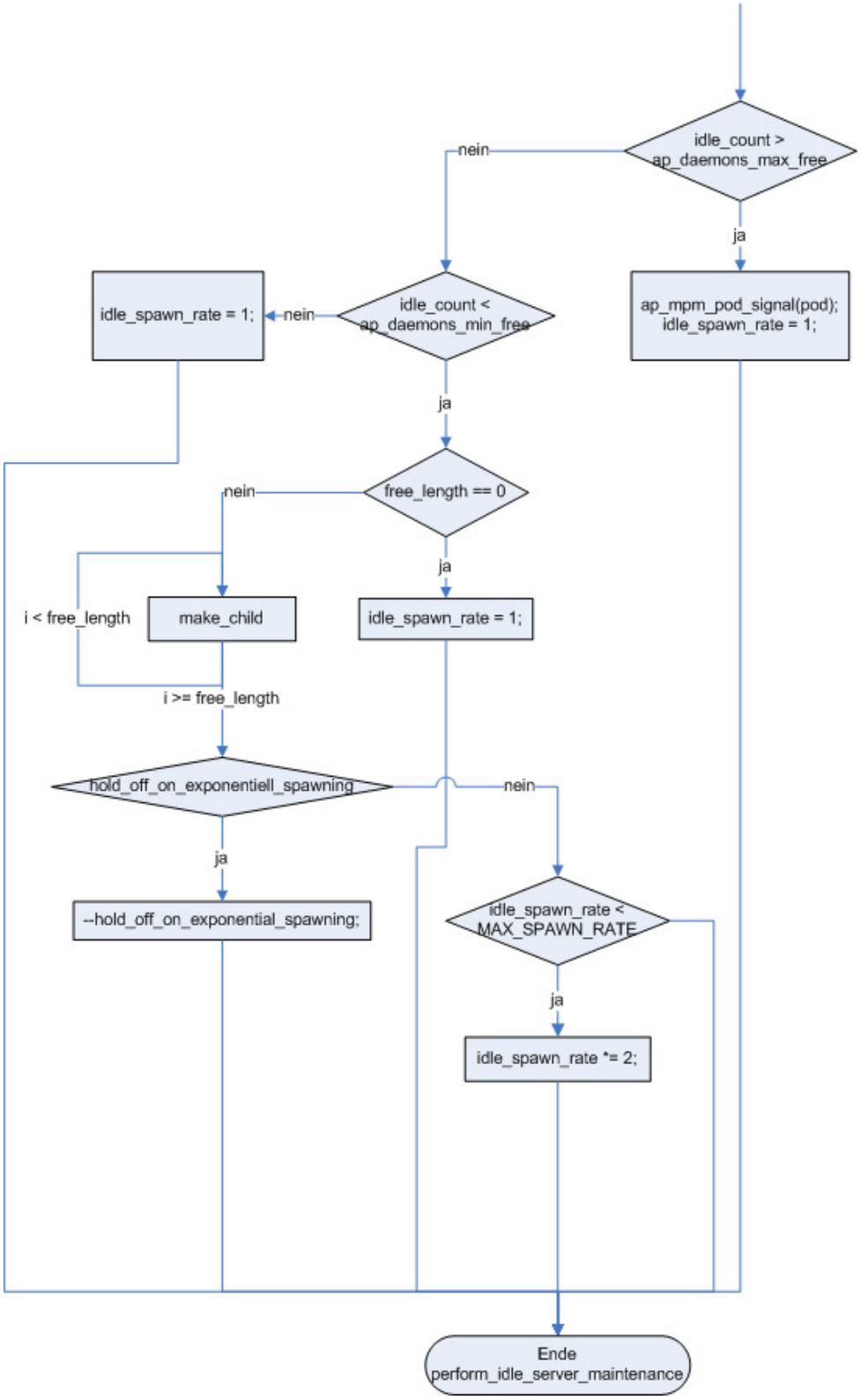


Abbildung 5.4: perform_idle_server_maintenance - Teil 2

5.3 Auflösungstest

Bevor Aussagen über das Verhalten eines MPMs gemacht werden können, muss zunächst die zeitliche Auflösung an allen Stellen im Quellcode, an denen später ein Messpunkt stehen wird, überprüft werden. Hierzu wurde der Quellcode des Apache Webservers mit Auflösungsmesspunkten instrumentalisiert, das heißt, dass an jeder Stelle eines späteren Messpunktes die Zeit gemessen wurde, wie lange das Messframework braucht, um einen Messwert an exakt dieser Stelle im Quellcode abzuspeichern. Die gewonnenen Ergebnisse müssen bei der eigentlichen Auswertung berücksichtigt werden, damit keine falschen Rückschlüsse zum Verhalten des Apache Webservers getroffen werden.

Für das Prefork MPM und für das Worker MPM wurden insgesamt jeweils 200 Messungen durchgeführt. Es wurde die Anzahl der Clients (100 - 2000) und die Wartezeit pro Anfrage (100 ms - 1000 ms) variiert.

Die Zeitauflösungen werden mittels nachfolgender Formel bestimmt, wobei M für den Messpunkt steht, T steht für die aktuelle Tabelle (also der aktuellen Messung), W steht für den zu bestimmenden Wert, N ist die Gesamtzahl der Messungen und quantile() ist eine Funktion, welche zu einer Tabelle und einem zugehörigen Messpunkt das 90%-Quantil bestimmt. Diese Formel sollte natürlich nur verwendet werden, wenn die Streuung der Quantilswerte über die einzelnen Messungen nicht zu hoch ist. Bei zu hoher Streuung sollte eher eine Spannweite der Zeitauflösung angegeben werden.

$$W_j = \frac{\sum_{i=1}^{N} quantile(T_i(M_j))}{N}$$

5.3.1 Zeitauflösung Prefork

Tabelle 5.1 zeigt die Zeitauflösung des Messframeworks zu jedem einzelnen Messpunkt für das Prefork MPM. Anders als beim Worker MPM (siehe 5.3.2) kann beim Prefork MPM leider kein exakter Wert für die Zeitauflösung angegeben werden. Die meisten Werte liegen zwar bei einer ähnlichen Zeitauflösung wie beim Worker MPM, jedoch gibt es zu oft zu viele Abweichungen davon. Deshalb wird hier in der Tabelle ein Wert ohne die hohen Ausschläge, dann der Wert unter Berücksichtigung hoher Ausschläge und der maximale Ausschlag angegeben.

Das MFW für das Prefork MPM hat vermutlich wegen der hohen Speicherauslastung beim Prefork MPM Probleme, alle Messinformationen richtig abzuspeichern (siehe hierzu 5.4). Es wird demnach empfohlen, vor jeder Messung des Prefork MPM zunächst einen Auflösungstest durchzuführen und zwar mit den gleichen Parametereinstellungen wie in der eigentlichen Messung. Im Kapitel 6.2 gibt es Ideen zur Beseitigung dieses Problems.

Messpunkt	ohne Ausschläge	mit Ausschlägen	Maximum
36_Anfrage_bearbeitet	ca. 5 μs	234,547 μs	4,5 ms
Aufloesung_01	ca. 5 μs	258,717 μs	2,65 ms
Aufloesung_02	ca. 5 μs	190,734 μs	4 ms
23_process_URI	ca. 5 μs	117,81 μs	3,2 ms
25_read_configuration	ca. 5 μs	242,95 μs	3,8 ms
26_translate_name	ca. 5 μs	150,261 μs	2,1 ms
27_map_to_storage	ca. 5 μs	174,741 μs	1,8 ms
28_location_walk_2	ca. 8 μs	220,552 μs	3,2 ms
29_header_parser	ca. 5 μs	190,283 μs	2,7 ms
30_access_checker	ca. 5 μs	213,827 μs	4,3 ms
33_type_checker	ca. 5 μs	118,013 μs	3,2 ms
34_fixups	ca. 5 μs	165,635 μs	2,2 ms
19_Insert_Filter	ca. 5 μs	93,332 μs	2 ms
20_Run_Handler	ca. 5 μs	228,145 μs	3,5 ms
37_log_transaction	ca. 70 μs	376,877 μs	4,6 ms
05_Close_Connection	ca. 6 μs	39,493 μs	1,7 ms
12_perform_idle_server_maintenance	46 - 60 μs	—	—
11_Master_Server_Loop	45 - 50 μs	—	—
03_Create_Connection	Extreme Schwankungen 43 μs - 95 ms	—	—
04_Process_Connection	ca. 5 μs	249,184 μs	4 ms
21_Vorbereitungsphase	ca. 5 μs	244,602 μs	4 ms
22_Block_Verbindungsverarbeitung	ca. 5 μs	171,054 μs	3,1 ms
14_Block_request_rec_erzeugen	ca. 5 μs	202,1 μs	3,7 ms
15_Anfrage_vorbereiten	ca. 5 μs	184,179 μs	3,8 ms
16_Anfrage_lesen	ca. 5 μs	168,237 μs	2,2 ms
17_Eingabefilter_hinzufuegen	ca. 5 μs	154,505 μs	3,25 ms
18_Anfrage_eingelesen	ca. 5 μs	164,49 μs	3,8 ms
35_Block_Anfragebearbeitung	ca. 5 μs	159,454 μs	2,4 ms
09_make_child	45,981 μs	—	—
50_INITIALIZATION_CHILD_MAIN	60,72 μs	—	—
24_Mutex_Child	12,269 μs	—	—
38_Run_Child_Init	7,956 μs	—	—

Tabelle 5.1: Zeitauflösung beim Prefork MPM

5.3.2 Zeitauflösung Worker

Die Tabelle 5.2 zeigt die Zeitauflösung des Messframeworks zu jedem einzelnen Messpunkt für das Worker MPM. Die fehlenden Werte konnten aufgrund einer zu geringen Datenmenge nicht ermittelt werden.

5.4 Auswertung des Prefork-Moduls

Vorbemerkung: Aufgrund der relativ groben Zeitauflösung beim Prefork MPM ist Vorsicht geboten in Bezug auf Aussagen zu den Messpunkten im Apache Quellcode. Ein Blick auf die Zeitauflösungstabelle (siehe Tabelle 5.1) hilft dabei, den Grad der Genauigkeit einzuschätzen. Alle anderen gesammelten Daten können aber ohne weiteres verwendet werden. Daher werden die Messpunkte in diesem Unterkapitel nicht erwähnt, da hier für klare Aussagen zunächst eine Veränderung des Messframeworks aus [BNO+07] notwendig ist (siehe auch Kapitel 6).

5.4.1 Standardparameter

Wenn im Folgenden von Standardparametern die Rede ist, dann sind folgende Einstellungen damit gemeint:

- StartServers 5
- MaxClients 256
- MinSpareServers 5
- MaxSpareServers 10
- MaxRequestsPerChild 0

5.4.2 Stresstest

Für die Stresstests wurden folgende Parametervariationen verwendet:

- Wartezeit: 1,4 ms - 5,6 ms; Schrittweite: 1,4 ms
- Processing-Delay: 1, 4ms - 5,6 ms; Schrittweite: 1,4 ms
- Clients: 200 - 1000; Schrittweite 200
- MinSpareServers: 5 - 10; Schrittweite: 5

Messpunkt	Zeitauflösung
36_Anfrage_bearbeitet	5,297 μs
Aufloesung_01	5,328 μs
Aufloesung_02	5,231 μs
23_process_URI	5,242 μs
25_read_configuration	5,381 μs
26_translate_name	5,257 μs
27_map_to_storage	5,372 μs
28_location_walk_2	6,386 μs
29_header_parser	5,250 μs
30_access_checker	5,348 μs
33_type_checker	5,317 μs
34_fixups	5,222 μs
19_Insert_Filter	5,490 μs
20_Run_Handler	5,273 μs
37_log_transaction	64,118 μs
05_Close_Connection	6 μs - 40 μs
07_worker_loop	44,528 μs
41_Unlock_Mutex	7,5 μs - 52 μs
45_ADD_REQUEST_TO_QUEUE	5,367 μs
44_WAIT_FOR_IDLE_WORKER	5,5 μs - 45 μs
40_Child_Process_Main_Loop	8 μs - 14 μs
03_Create_Connection	5,361 μs
04_Process_Connection	5,485 μs
21_Vorbereitungsphase	5,429 μs
22_Block_Verbindungsverarbeitung	5,647 μs
14_Block_request_rec_erzeugen	5,574 μs
15_Anfrage_vorbereiten	7,116 μs
16_Anfrage_lesen	5,334 μs
17_Eingabefilter_hinzufuegen	8,830 μs
18_Anfrage_eingelesen	5,411 μs
35_Block_Anfragebearbeitung	5,433 μs
12_perform_idle_server_maintenance	57,389 μs
11_Master_Server_Loop	10 μs - 55 μs
09_make_child	—
50_INITIALIZATION_CHILD_MAIN	—
24_Mutex_Child	—
38_Run_Child_Init	—
51_CREATE_START_THREADS	—
48_CREATE_WORKER_AND_LISTENER_THREADS	—
06_worker_init	15 μs - 57 μs
49_EXIT_START_THREADS	—
43_INITIALIZATION_LISTENER_THREAD	—
52_KILL_CHILD	—
46_CLOSE_LISTENERS	—
47_ap_queue_term	—

Tabelle 5.2: Zeitauflösung beim Worker MPM

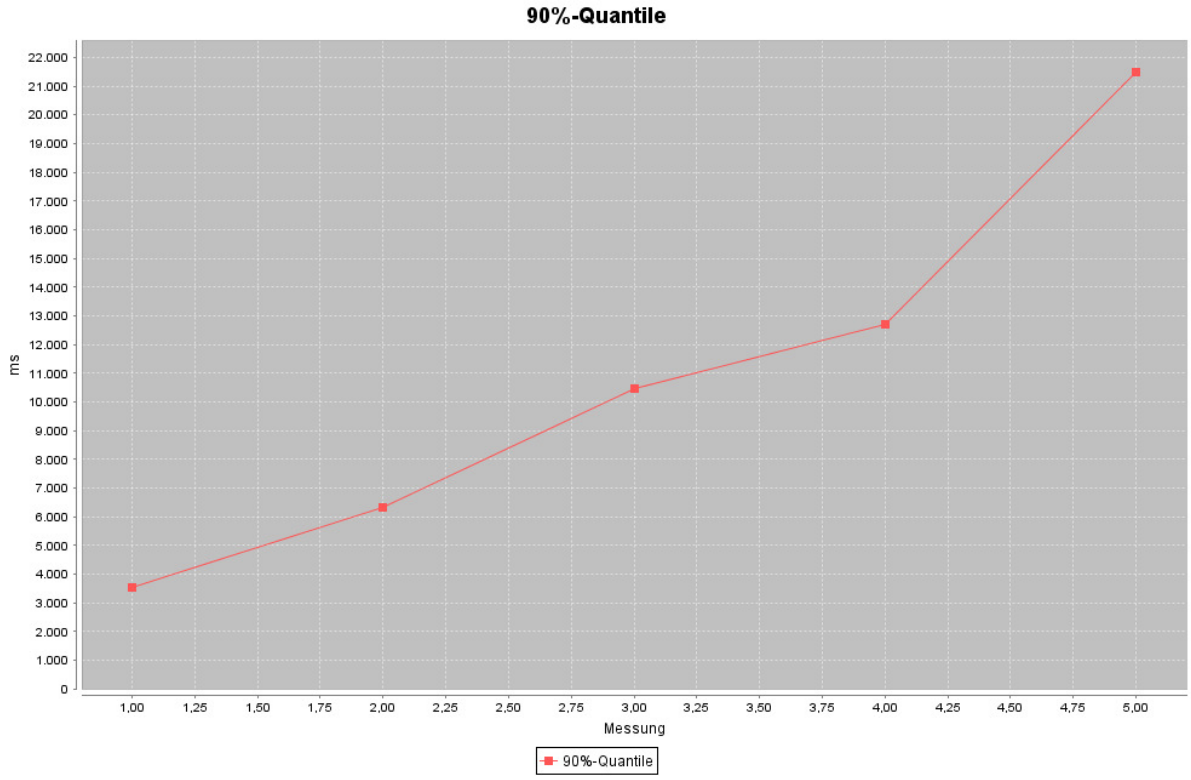


Abbildung 5.5: 90%-Quantile der Requestantwortzeit bei Stresstests

- MaxSpareServers: 10 - 20; Schrittweite: 10
- StartServers: 5
- Methode: 1 und 2

Beispielergebnis: Requestantwortzeit

Abbildung 5.5 zeigt den Verlauf des 90%-Quantils einer Messung mit Standardparametern und je 200, 400, 600, 800 oder 1000 Clients. Alle anderen Messungen hatten ein ähnliches Verhalten. Dieses Ergebnis ist selbstverständlich, denn je mehr Anfragen eintreffen um so mehr muss der Server leisten. Dadurch leidet die Gesamtperformance und entsprechend dauert das Bearbeiten jeder einzelnen Anfrage etwas länger.

Beispielergebnis: Freier Arbeitsspeicher

Mit wachsender Anzahl Clients nimmt der freie Arbeitsspeicher leicht ab, denn mehr Prozesse belegen auch mehr Speicher (siehe Abbildung 5.6 mit den gleichen Bedingungen wie im vorherigen Abschnitt).

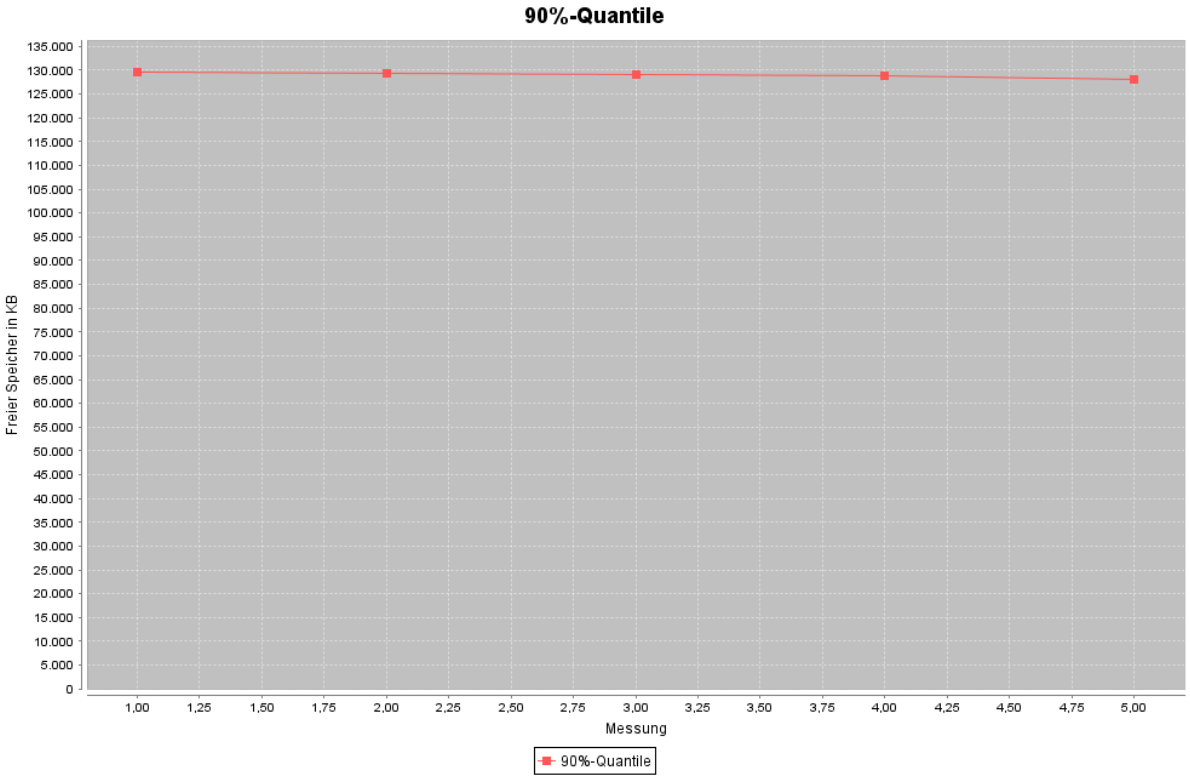


Abbildung 5.6: 90%-Quantile des freien Arbeitsspeichers bei Stresstests

Beispielergebnis: Prozessorauslastung

Die Prozessorauslastung lag in allen Fällen immer über 90% bishin zu 98%. Selbst bei nur 200 Clients wurde das System enorm belastet. Das Testsystem ist alleine schon aus diesem Grund nicht für einen Serverbetrieb geeignet, wenn erwartet werden kann, dass viele Benutzer gleichzeitig auf das System zugreifen.

Beispielergebnis: Aktive Requests

Mit steigender Anzahl der Clients wächst auch das Maximum der gleichzeitig vorhandenen aktiven Requests. Abbildung 5.7 zeigt einen exemplarischen Graphen mit Standardparametern und 800 Clients. Speziell bei diesem Graphen beträgt das Maximum 762 aktive Requests.

5.4.3 Langzeittest

Für den Langzeittest wurden 16 unterschiedliche Messungen durchgeführt. Jede einzelne Messung lief über einen Zeitraum von zehn Minuten. Folgende Parameterwerte wurden verwendet:

- Wartezeit 1,4 ms

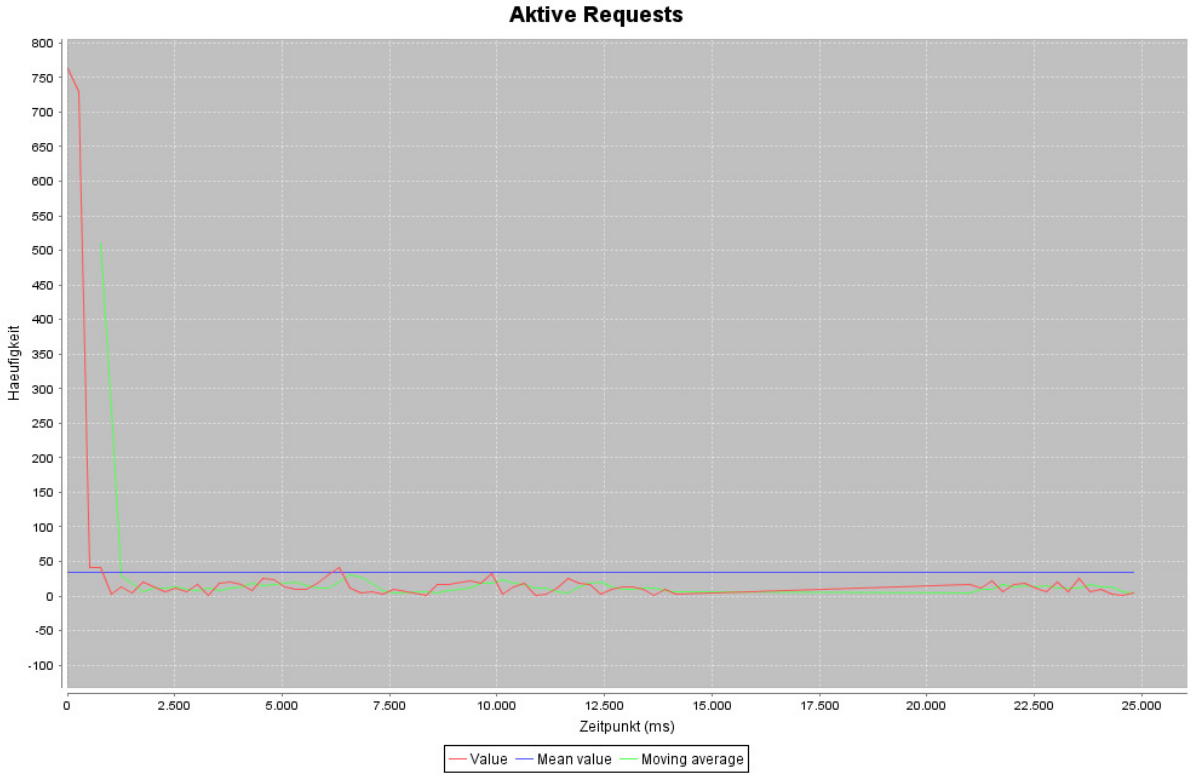


Abbildung 5.7: Aktive Requests beim Prefork MPM mit Standardparametern und 800 Clients (Stresstest)

- Processing-Delay: 1,4 ms
- Clients: 25 - 100; Schrittweite: 25
- Requests pro Client: 3
- MinSpareServers: 5 - 10; Schrittweite: 5
- MaxSpareServers: 10 - 20; Schrittweite: 10
- StartServers: 5
- MaxClients: 256
- Methode: 2
- Dauer: 10 Minuten

Beispielergebnis: Requestantwortzeit

Trotz vereinzelter Ausreißer schwinkt sich bei allen Langzeittests die Requestantwortzeit auf einen bestimmten Wert ein. Abbildung 5.8 zeigt einen beispielhaften Verlauf bei Standardparametern und 25 Clients. Alle anderen Messungen verhalten sich ähnlich. Abbildung 5.9 zeigt die Entwicklung des 90%-Quantils bei Standardparametern und jeweils 25, 50, 75 und 100 Clients. Mit wachsender Zahl der Clients steigt auch das 90%-Quantil der Requestantwortzeit an. Dies liegt daran, dass der Server durch eine höhere Anzahl Clients mehr leisten muss. Bei 25 Clients sind es 31 ms, bei 50 Clients 34 ms, bei 75 Clients 45 ms und bei 100 Clients sind es 48 ms. Weitere Graphen zur Requestantwortzeit werden in Kapitel 5.6 gezeigt.

Beispielergebnis: Systemauslastung

Die Systemauslastung bleibt anders als bei den Stresstests immer unter 30%. Grund: Der Server muss nicht so viele Anfragen gleichzeitig bewältigen.

5.4.4 Langzeittest mit Zusatzlast

Folgende Parameterwerte wurden verwendet:

- Wartezeit ca. 0,56 ms
- Processing-Delay: ca. 0,56 ms

Abbildung 5.8: Requestantwortzeit - Standardparameter - 25 Clients

- Clients: 25 - 75; Schrittweite: 25
- Zusatzclients: 10 -30; Schrittweite: 10
- Requests pro Client: 3
- MinSpareServers: 5 - 10; Schrittweite: 5
- MaxSpareServers: 10 - 20; Schrittweite: 10
- StartServers: 5
- MaxClients: 256
- Methode: 2
- Dauer: 15 Minuten

Beispielergebnis: Requestantwortzeit

Es ist keinerlei Effekt durch die Zusatzlast zu identifizieren. Abbildung 5.10 zeigt, dass sich die Antwortzeit auf einen bestimmten Wert einpendelt (Standardparameter, 75 Clients, 30 Zusatz-

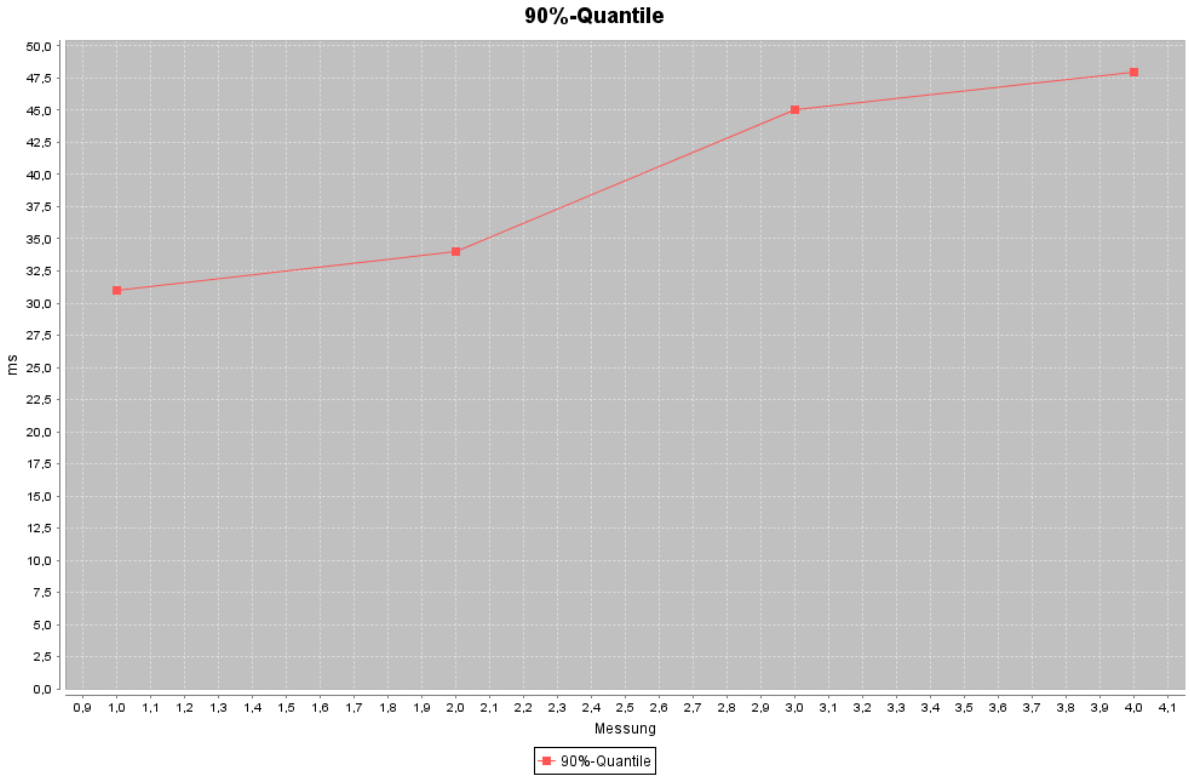


Abbildung 5.9: 90%-Quantile der Requestantwortzeit mit Standardparametern

clients). Bei allen Messungen beträgt das 90%-Quantil 13 ms.

5.5 Auswertung des Worker-Moduls

5.5.1 Standardparameter

Wenn im Folgenden von Standardparametern die Rede ist, dann sind folgende Einstellungen damit gemeint:

- StartServers 2
- MaxClients 150
- MinSpareThreads 25
- MaxSpareThreads 75
- ThreadsPerChild 25
- MaxRequestsPerChild 0

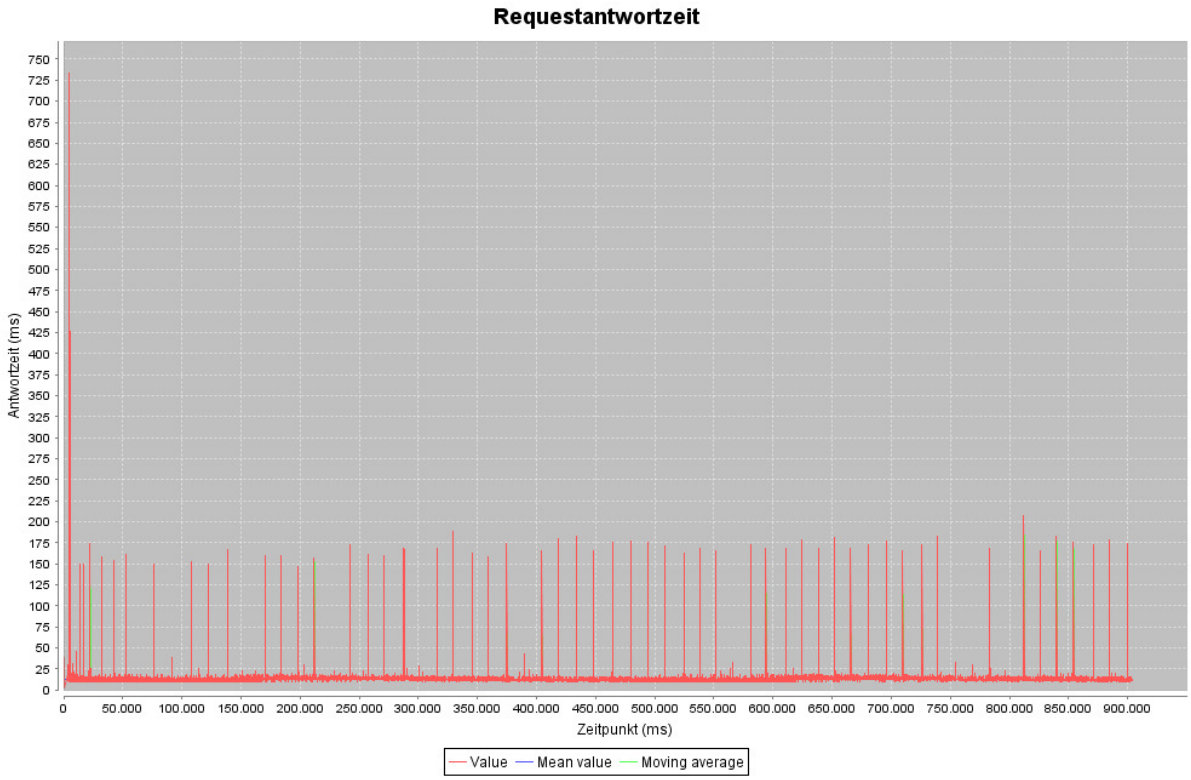


Abbildung 5.10: Requestantwortzeit(Prefork) - Standardparameter - 75 Clients - 30 Zusatzclients

5.5.2 Stresstest

Für die Stresstests wurden folgende Parametervariationen verwendet:

- Wartezeit: 1,4 ms - 5,6 ms; Schrittweite: 1,4 ms
- Processing-Delay: 1, 4ms - 5,6 ms; Schrittweite: 1,4 ms
- Clients: 200 - 1000; Schrittweite 200
- Requests pro Client: 3
- MinSpareThreads: 25 - 50; Schrittweite: 25
- MaxSpareThreads: 75 - 150; Schrittweite: 75
- StartServers: 2
- Methode: 1 und 2

Beispielergebnis: Requestantwortzeit

Die Abbildung 5.11 zeigt einen beispielhaften Verlauf zu 1000 Clients bei Standardparametern, einer Wartezeit von 1,4 ms und einer Processing-Delay von 1,4 ms. Zu beobachten ist, dass anfangs die Antwortzeiten wesentlich höher als gegen Ende der Messung sind. Dies hat mehrere Ursachen: Einerseits wird das Worker MPM beim Eintreffen der Anfragen feststellen, dass es nicht genug unbeschäftigte Worker-Threads hat. Daher werden Prozesse mit Threads nachproduziert. Dies erhöht natürlich die Bearbeitungszeit für jede einzelne Anfrage. Andererseits wird festgestellt, dass der Lastgenerator bei Beginn einer Messung die meisten aktiven Requests zählt. Es sind also insgesamt mehr Requests aktiv[3], das heißt, dass der Server mehr Arbeit verrichten muss. Folglich steigt die Antwortzeit jeder einzelnen Anfrage.

Weiterhin wird festgestellt, dass das 90%-Quantil mit steigender Anzahl Clients zunimmt.

Beispielergebnis: Freier Arbeitsspeicher

Mit steigender Anzahl Clients nimmt der freie Arbeitsspeicher leicht ab. Bei Standardparametern ist etwas mehr Arbeitsspeicher frei als nach den Modifikationen.

[3] Bedauerlicherweise passt der Lastgenerator seine erzeugte Last in beschränktem Maße an den Server an. Stellt er fest, dass der Server nicht mehr schnell genug auf die Last reagiert, drosselt der Lastgenerator seine Last etwas. Dies ist auch ein Grund dafür, dass die meisten aktiven Requests zu Beginn einer Messung gezählt werden.

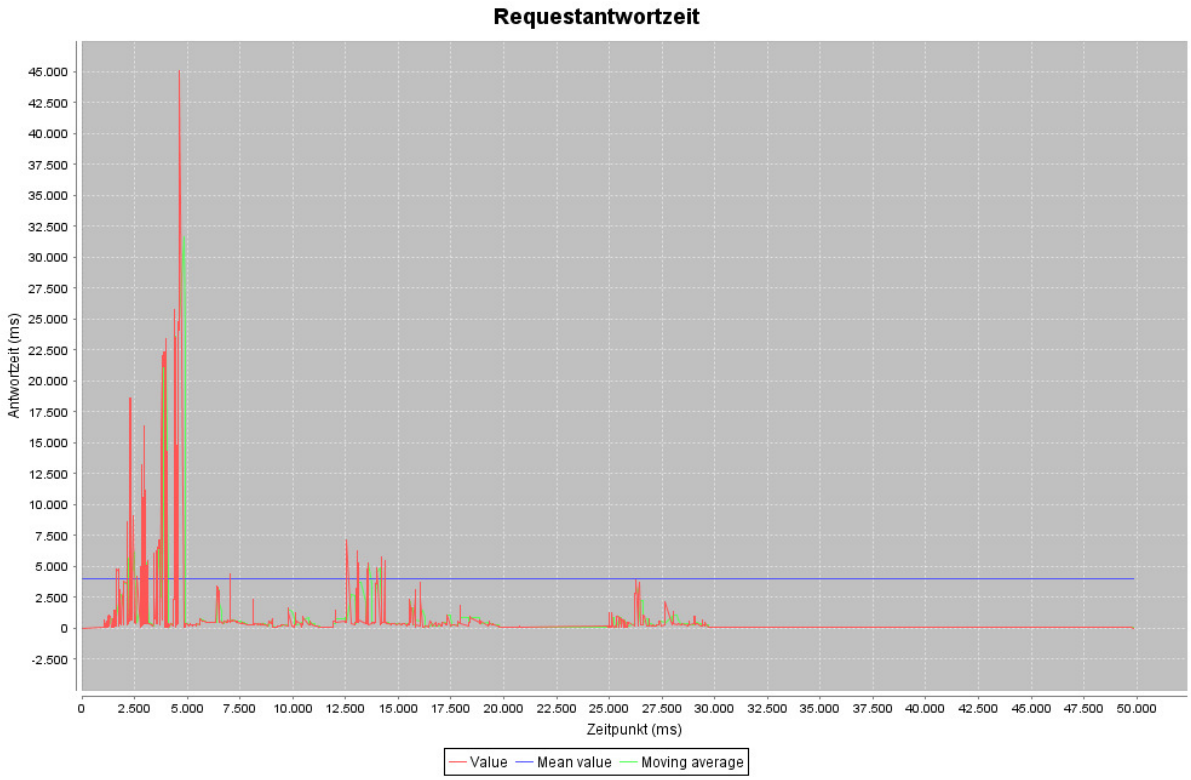


Abbildung 5.11: Requestantwortzeit des Worker MPM bei 1000 Clients und Standardparametern

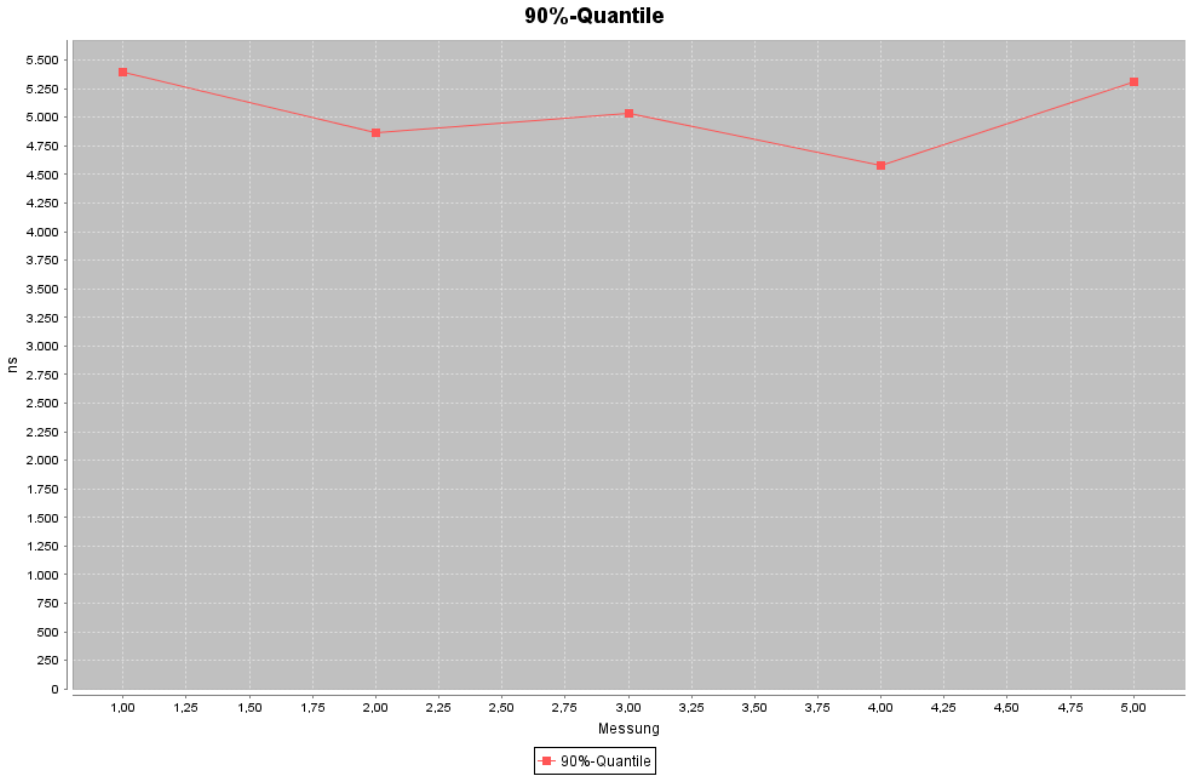


Abbildung 5.12: Messpunkt 39_Lock_Mutex

Beispielergebnis: Prozessorauslastung

Die Prozessorauslastung liegt in allen Fällen immer über 90% bishin zu 98%. Selbst bei nur 200 Clients wird das System enorm belastet. Das Testsystem ist alleine schon aus diesem Grund nicht für einen Serverbetrieb geeignet, wenn erwartet werden kann, dass viele Benutzer gleichzeitig auf das System zugreifen.

Beispielergebnis: 39_Lock_Mutex

Abbildung 5.12 zeigt die Entwicklung des Messpunktes 39_Lock_Mutex mit Standardparametern und jeweils 200, 400, 600, 800 und 1000 Clients. Der Wert liegt im Schnitt bei etwa 5 μs. Dies bedeutet also, dass beim Sperren des Mutex kaum Zeit gebraucht wird. Da dies der einzige Synchronisationsmechanismus bei der Anfragebearbeitung ist, ist die Synchronisierung kein Flaschenhals.

5.5.3 Langzeittest

Für den Langzeittest wurden 16 unterschiedliche Messungen durchgeführt. Jede einzelne Messung lief über einen Zeitraum von zehn Minuten. Folgende Parameterwerte wurden verwendet:

- Wartezeit 1,4 ms
- Processing-Delay: 1,4 ms
- Clients: 25 - 100; Schrittweite: 25
- MinSpareThreads: 25 - 50; Schrittweite: 25
- MaxSpareThreads: 75 - 150; Schrittweite: 75
- StartServers: 2
- MaxClients: 150
- ServerLimit: 16
- ThreadsPerChild: 25
- Methode: 2
- Dauer: 10 Minuten

Beispielergebnis: Requestantwortzeit

Trotz vereinzelter Ausreißer schwinkt sich bei allen Langzeittests die Requestantwortzeit auf einen bestimmten Wert ein. Abbildung 5.13 zeigt einen beispielhaften Verlauf bei Standardparametern und 25 Clients. Alle anderen Messungen verhalten sich ähnlich. Abbildung 5.14 zeigt die Entwicklung des 90%-Quantils bei Standardparametern und jeweils 25, 50, 75 und 100 Clients. Mit wachsender Zahl der Clients steigt auch das 90%-Quantil der Requestantwortzeit an. Dies liegt daran, dass der Server durch eine höhere Anzahl Clients mehr Arbeit verrichten muss. Bei 25 Clients sind es 30 ms, bei 50 Clients 34 ms, bei 75 Clients 38 ms und bei 100 Clients sind es 44 ms. Weitere Graphen zur Requestantwortzeit werden in Kapitel 5.6 gezeigt.

Beispielergebnis: Systemauslastung

Die Systemauslastung bleibt anders als bei den Stresstests immer unter 30%. Grund: Der Server muss nicht so viele Anfragen gleichzeitig bewältigen.

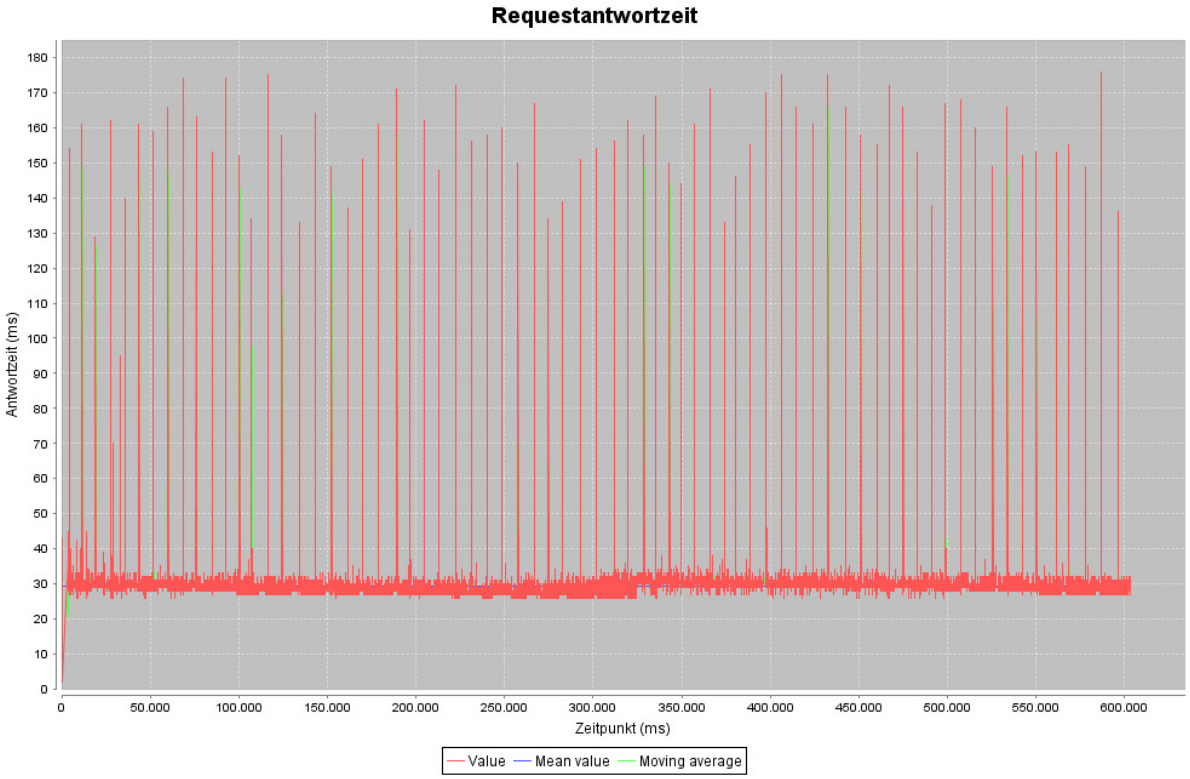


Abbildung 5.13: Requestantwortzeit - Standardparameter - 25 Clients

5.5.4 Langzeittest mit Zusatzlast

- Wartezeit ca. 0,56 ms
- Processing-Delay: ca. 0,56 ms
- Clients: 25 - 75; Schrittweite: 25
- Zusatzclients: 10 - 30; Schrittweite: 10
- MinSpareThreads: 25 - 50; Schrittweite: 25
- MaxSpareThreads: 75 - 150; Schrittweite: 75
- StartServers: 2
- MaxClients: 150
- ServerLimit: 16
- ThreadsPerChild: 25

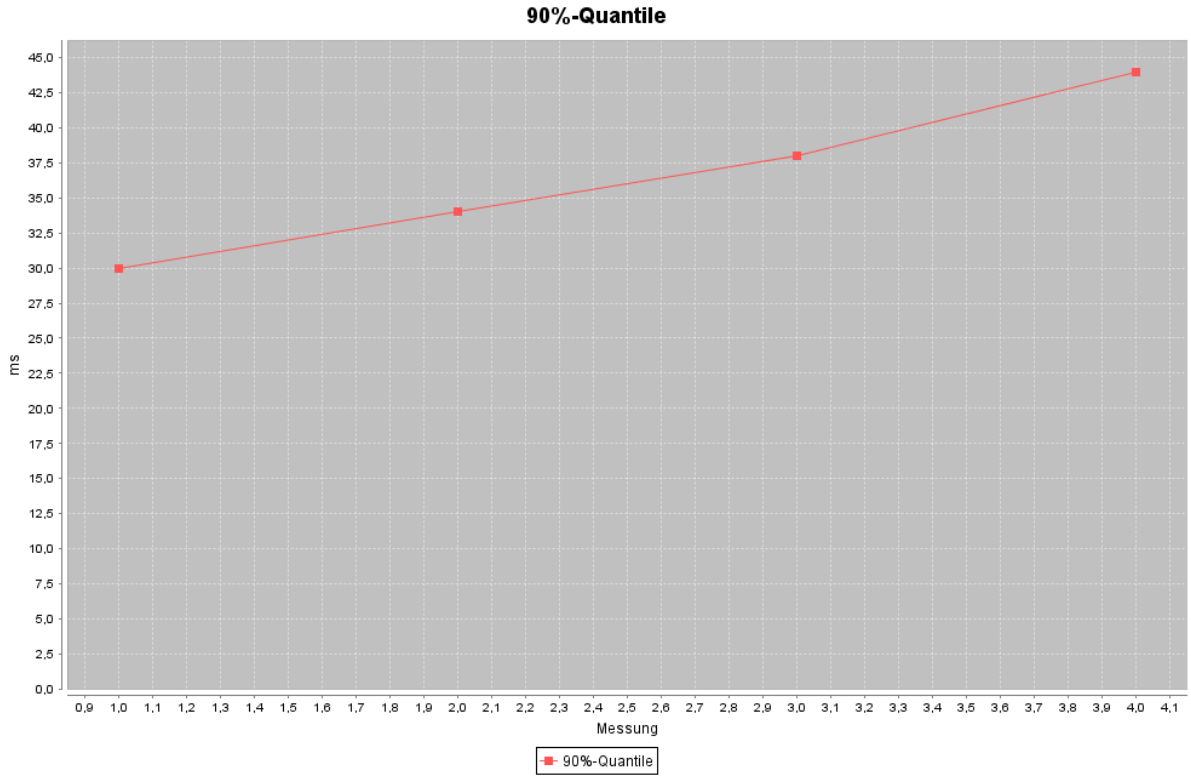


Abbildung 5.14: 90%-Quantile der Requestantwortzeit mit Standardparametern

- Methode: 2
- Dauer: 15 Minuten

Beispielergebnis: Requestantwortzeit

Es ist keinerlei Effekt durch die Zusatzlast zu identifizieren. Abbildung 5.15 zeigt, dass sich die Antwortzeit auf einen bestimmten Wert einpendelt (Standardparameter, 75 Clients, 30 Zusatzclients). Bei allen Messungen beträgt das 90%-Quantil 13 ms.

5.6 Gemeinsamkeiten von Prefork und Worker

Systemauslastung bei Stresstests

Die Graphen beider Module sind sich über alle Messungen extrem ähnlich. Das 90%-Quantil liegt für beide Module und alle Messungen immer über 90% (meistens zwischen 95% und 98%). Selbst bei nur 200 Clients und einer Processing-Delay von 1,4 ms wurden keine besseren Werte ermittelt. Dies könnte beim vorhandenen Testsystem einen Flaschenhals darstellen.

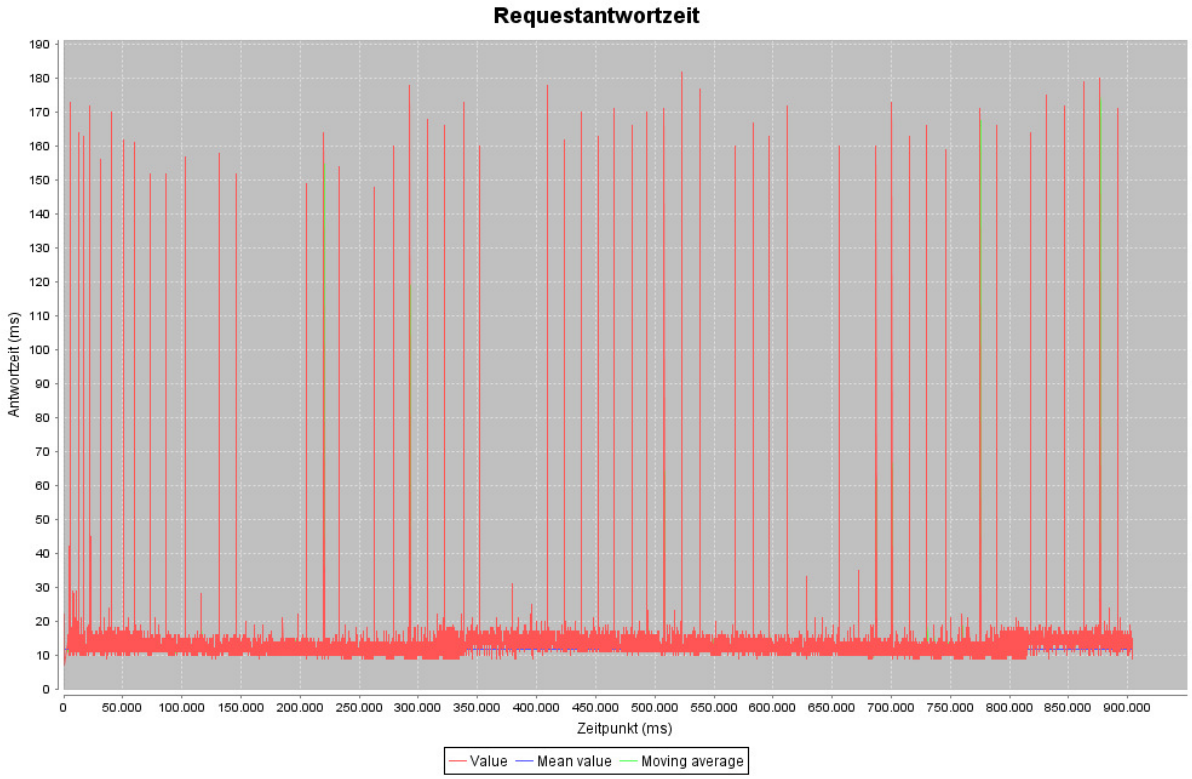


Abbildung 5.15: Requestantwortzeit - Standardparameter - 75 Clients - 30 Zusatzclients

Requestantwortzeit bei Stresstests

Beide Module haben gemeinsam, dass die Requestantwortzeit mit zunehmender Anzahl an Clients steigt.

Requestantwortzeit bei Langzeittests

Sehr interessant ist das Verhalten beider Module in Bezug auf die Requestantwortzeit, denn beide Module weisen ein ähnliches Verhalten auf. Vergleicht man die beiden Graphen zur Requestantwortzeit in Kapitel 5.4.3 und Kapitel 5.5.3, so stellt man fest, dass deren Verlauf sehr ähnlich ist. Dies mag vielleicht noch nicht verwunderlich sein. Aber betrachtet man die nachfolgenden Graphen, so zeigt sich, dass auch das Ändern von Parametern einen ähnlichen Effekt bei beiden Modulen erzielt. Zunächst einmal zeigt Abbildung 5.16 den Graphen der 90%-Quantile zum Prefork MPM mit MaxSpareServers verdoppelt auf 20 (Rest Standard). Abbildung 5.17 zeigt den entsprechenden Graphen zum Worker MPM mit MaxSpareThreads verdoppelt auf 150 (Rest Standard). Es zeigt sich wieder ein ähnlicher Verlauf. Sogar der Einbruch bei 100 Clients tritt bei beiden Modulen auf.

Werden stattdessen die Parameter MinSpareServers auf 10 bzw. MinSpareThreads auf 50

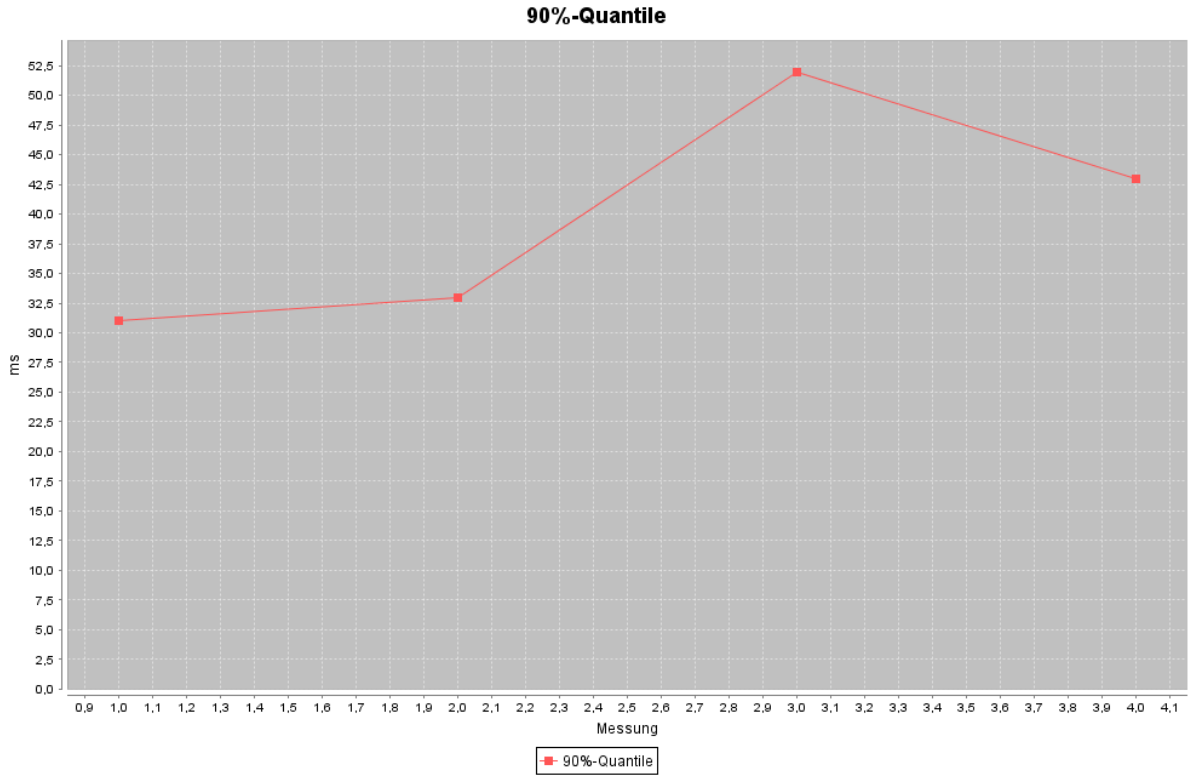


Abbildung 5.16: 90%-Quantile bei Langzeittests des Prefork MPM mit MaxSpareServers 20

verdoppelt, zeigt sich wieder, dass dies einen ähnlichen Effekt bei beiden Modulen hat. Sowohl beim Prefork MPM als auch beim Worker MPM steigt die Requestantwortzeit ähnlich an wie bei den Standardeinstellungen.

Werden sowohl MinSpareThreads, MaxSpareThreads, MinSpareServers und MaxSpareServers verdoppelt erhält man wiederum ähnliche Graphen.

Requestantwortzeit bei Langzeittests mit Zusastzlast

Sowohl beim Prefork MPM als auch beim Worker MPM betragen die 90%-Quantile immer 13 ms. Veränderungen der Parameter und der Clientanzahl hatten auf diesen Wert keinerlei Einfluss.

5.7 Unterschiede von Prefork und Worker

Requestanwortzeit bei Stresstests

Während die Kurvenverläufe im Vergleich zwischen Prefork MPM und Worker MPM ähnlich aussehen, unterscheiden sie sich doch in der tatsächlichen Antwortzeit. Das Worker MPM liegt

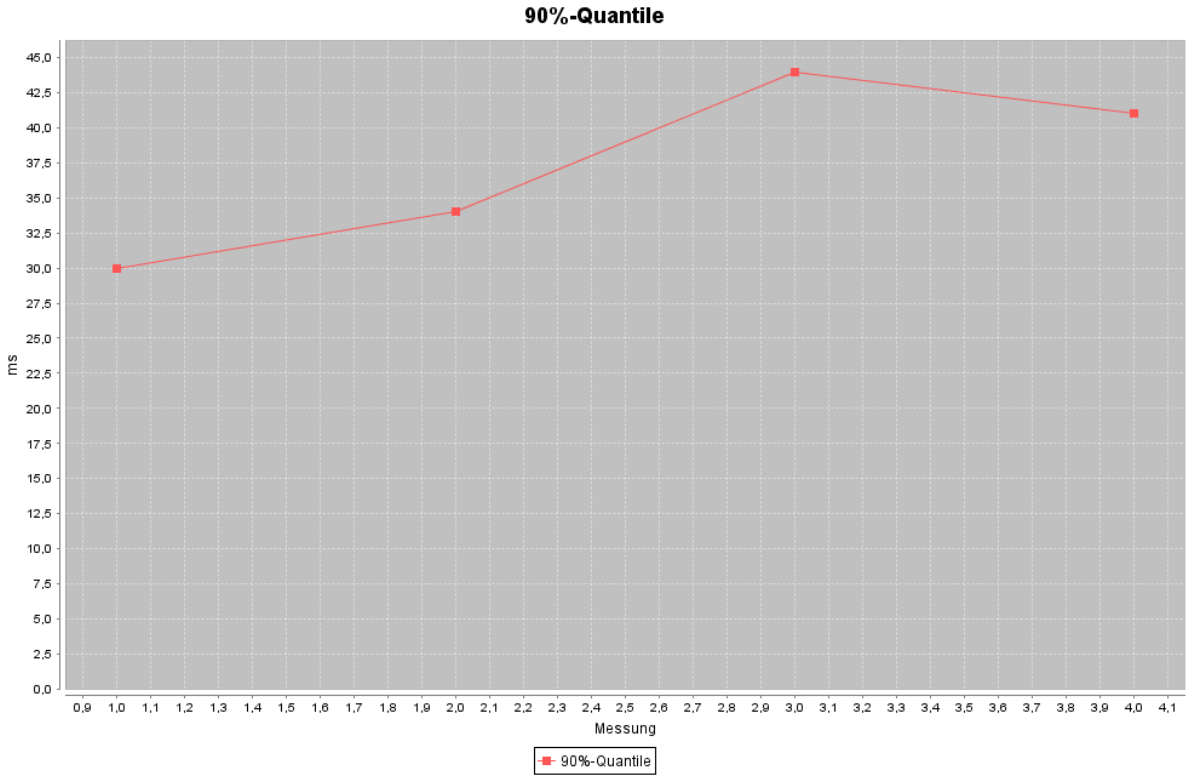


Abbildung 5.17: 90%-Quantile bei Langzeittest des Worker MPM mit MaxSpareThreads 150

bei fast allen Stresstests klar vor dem Prefork MPM. Es scheint so, als ob das Worker MPM auf sehr viele plötzlich eintretende Anfragen besser reagieren kann. Dies wird klar, wenn bedacht wird, dass das Prefork MPM nach dem Serverstart nur 5 Prozesse aktiv hat (Standardeinstellung) und somit sehr viele Prozesse nachproduzieren muss, um genug Prozesse zum Bearbeiten der gleichzeitig eintreffenden Anfragen zur Verfügung zu haben. Das Worker MPM hat dagegen direkt zu anfangs 2 Prozesse mit je 64 Threads aktiv (Standardeinstellung). Das Worker MPM muss zwar wahrscheinlich auch weitere Prozesse erzeugen, hat aber mit jedem weiteren Prozess direkt 64 weitere Threads für die Anfragenbearbeitung zur Verfügung.

Freier Arbeitsspeicher bei Stresstests

Das Prefork MPM hat ohne Ausnahme den Arbeitsspeicher stärker belegt als das Worker MPM. Als Beispiel sollen hier die Bilder 5.18 und 5.19 dienen. Abbildung 5.18 zeigt die Entwicklung des freien Speichers in KB für das Prefork MPM gemessen mit Standardparametern, 1000 Clients, einer Wartezeit von 1,4 ms und einer Processing-Delay von 1,4 ms. Das 90%-Quantil beträgt etwa 128 MB freier Speicher. Abbildung 5.19 zeigt die Entwicklung des freien Speichers in KB für das Worker MPM gemessen mit Standardparametern, 1000 Clients, einer Wartezeit von 1,4 ms und einer Processing-Delay von 1,4 ms. Das 90%-Quantil beträgt hier etwa 401 MB

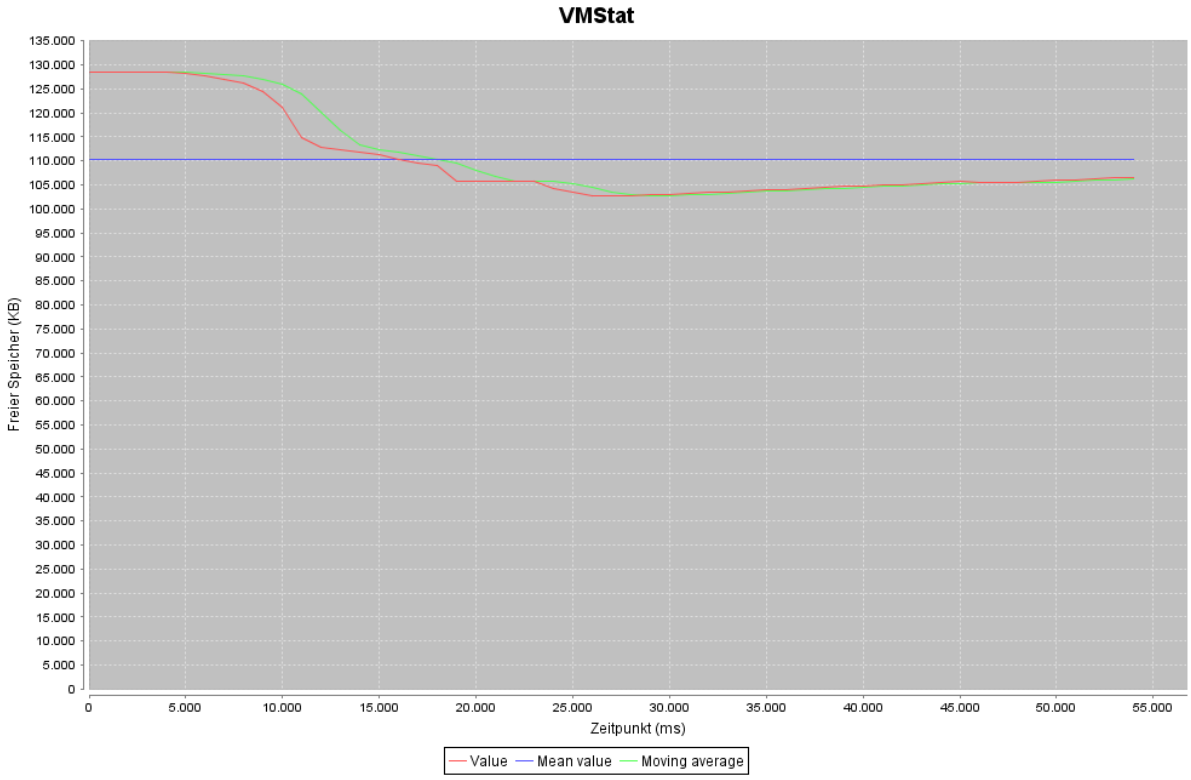


Abbildung 5.18: Freier Speicher beim Worker MPM, Stresstest, Standardparameter, 1000Clients

freier Speicher. Diese Ergebnisse sind nicht überraschend, da bekannt ist, dass Prozesse den Speicher stärker belasten als Threads. Dennoch ist dies ein klares Zeichen, dass beim Prefork MPM je nach Ausstattung des Rechners der Arbeitsspeicher zum Flaschenhals werden kann. Interessanter ist jedoch die Entwicklung des freien Arbeitsspeichers auf dem Lastgenerator, da das verhalten hier genau umgekehrt zu sein scheint. Der Lastgenerator hat beim Worker MPM mehr Speicher belegt als beim Prefork MPM. Ein Grund könnte sein, dass aufgrund der insgesamt schnelleren Bearbeitung der Anfragen im Worker MPM (siehe vorherigen Abschnitt), der Lastgenerator dadurch zur gleichen Zeit mehr Anfragen verwalten muss, so dass der Arbeitsspeicher mehr belastet wird, während beim Prefork MPM die Verwaltung der Anfragen im Lastgenerator kontinuierlicher geschehen kann.

Requestantwortzeit bei Langzeittests

Obwohl, wie in Kapitel 5.7 beschrieben, beide Module ein ähnliches Verhalten aufzeigen, unterscheiden sie sich doch insgesamt in der tatsächlichen Antwortzeit. Hier hat das Worker MPM in fast allen Messungen eine gleiche oder meistens sogar eine geringere Antwortzeit als das Prefork MPM. Es gibt nur zwei Ausnahmen: Bei Verdopplung des Parameters MinSpareServe-

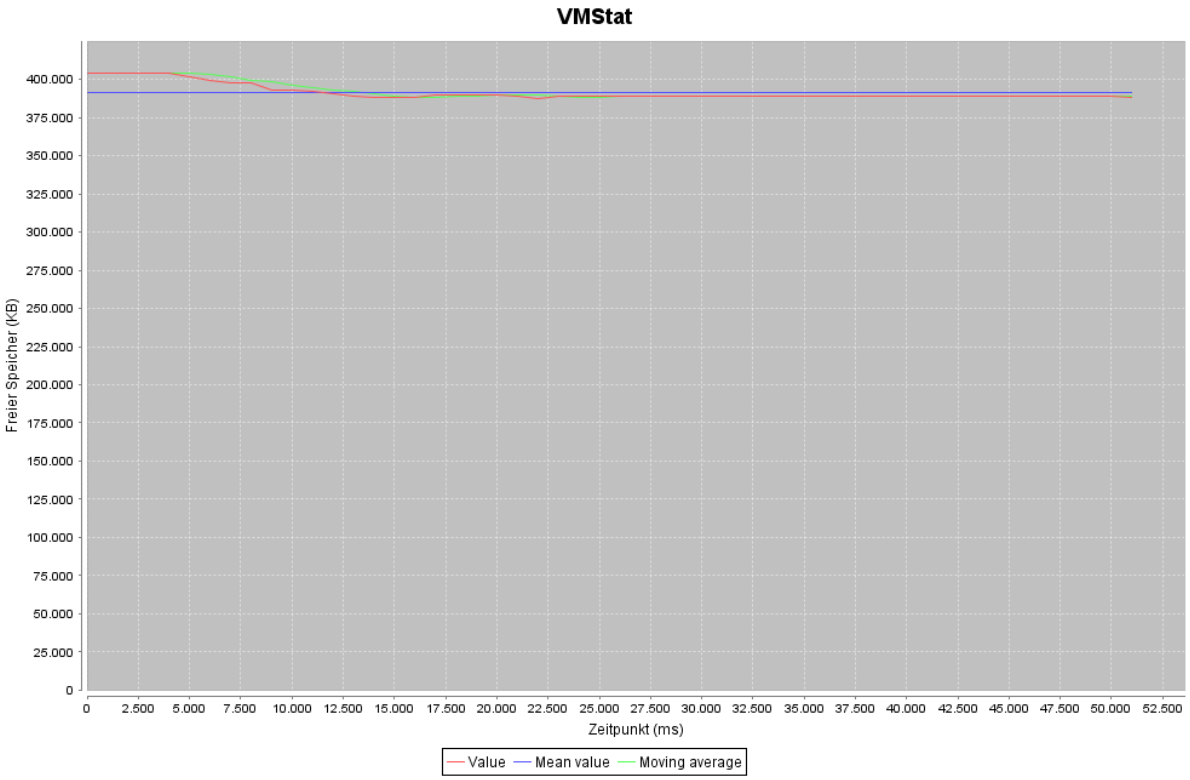


Abbildung 5.19: Freier Speicher beim Worker MPM, Stresstest, Standardparameter, 1000Clients

rs/MinSpareThreads und 100 Clients ist das Prefork MPM um ca. 5 ms besser (65 ms zu 70 ms) basierend auf dem 90%-Quantil. Bei Verdopplung von MaxSpareServers/MaxSpareThreads und 100 Clients ist das Prefork MPM um ca. 43 ms besser (56 ms zu 99 ms) basierend auf dem 90%-Quantil.

Messpunkt 09_make_child

Ein Vergleich zwischen Prefork MPM und Worker MPM in Bezug auf den Messpunkt *09_make_child* zeigt, dass während der Laufzeit von 10 Minuten (Langzeittest) bei jeder Messung das Worker MPM nie mehr als zehn (erst ab zehn Werten wird das Quantil berechnet) Kindprozesse erzeugt hat. Das Prefork MPM erzeugt dagegen viel mehr Prozesse. Der Messpunkt *09_make_child* ist sogar beim Prefork MPM ein aussagekräftiger Wert, da hier die Zeitauflösung etwa 46 μs beträgt und es keine Schwankungen gibt. Das Minimum dieses Messpunktes beträgt bei allen Messungen abzüglich der Zeitauflösung ca. 450 μs. Da viel mehr Prozesse erzeugt werden, als beim Worker MPM kostet es zum Einen mehr Zeit und zum Anderen ist der Speicher viel höher belastet.

5.8 Zusammenfassung

Es wurden die beiden MPMs Prefork und Worker unter den Bedingungen Stresstest, Langzeittest und Langzeittest mit Zusatzlast gemessen. Dabei wurden verschiedene Parameter und die Anzahl der Clients variiert. Von der großen Anzahl an Ergebnissen wurden in diesem Kapitel einige wenige herausgegriffen und näher erläutert. Als Flaschenhals stellte sich bei dem vorhandenen Testsystem primär die CPU heraus, da sie bei beiden MPMs voll ausgelastet war. Beim Prefork MPM kam zusätzlich noch hinzu, dass die erzeugten Prozesse sehr viel Arbeitsspeicher belegt haben. Neben diesen beiden hardwareseitigen Flaschenhälsen, ist des Weiteren die Prozesserzeugung ein zwar kleiner, aber dennoch beeinflussender Zeitfaktor. Allein die Tatsache, dass mehr Prozesse beim Prefork MPM erzeugt werden müssen als beim Worker MPM, beeinflusst die Messergebnisse der Prefork MPM.

Das Sperren eines Mutex stellt dagegen keinen Flaschenhals dar. Die Zeiten, die zum Sperren des Mutex benötigt wurden, waren beim Worker MPM minimal. Für das Prefork MPM muss diese Aussage noch mittels weiterer Messungen untersucht werden. Hierfür muss jedoch zunächst das MFW verbessert werden (siehe Kapitel 6).

Insbesondere aus Sicht des Endanwenders konnte das Worker MPM besser abschneiden, da es die Anfragen in kürzerer Zeit beantworten konnte. Dies zeigte sich vorallem bei den Stresstests.

Während aller Messungen liefen beide Module stabil. Die Fehlerquote lag bei 0%. Das heißt, dass auch das Worker MPM innerhalb der verwendeten Messungsbedingungen ausreichend stabil zu sein scheint.

6 Ausblick

Dieses Kapitel soll erklären, inwiefern das Programm JDataMining verbessert werden kann, welche Probleme noch gelöst werden sollten und welche weiteren Messungen sinnvoll wären. Das Kapitel stellt keinen Anspruch auf Vollständigkeit, sondern listet lediglich einige Vorschläge auf. Die Liste kann also nach Belieben erweitert werden.

6.1 Verbesserungen und Erweiterungen an JDataMining

- Zusätzlich zu der generierten HTML-Übersicht der Messergebnisse könnte ein PDF-Bericht erzeugt werden.
- Eine Tabelle, die alle Messungen mit ID und veränderten Parametern auflistet, sollte automatisch erzeugt werden.
- Vielleicht könnte über ein Tool nachgedacht werden, dass die Ergebnisse bewertet und anschließend eine Rangfolge der besten Ergebnisse erstellt. So könnten die Parametereinstellungen optimal für ein MPM an einen bestimmten Rechner angepasst werden. Natürlich müsste ein vernünftiges Gewichtungssystem entwickelt werden und dieses System müsste validiert werden.
- Im Register Ergebnisse sollte es die Wahl zwischen verschiedenen Darstellungen geben, denn nicht immer ist ein Liniengraph sinnvoll. Bei Häufigkeitsverteilungen wären beispielsweise Balkendiagramme besser geeignet.
- Das Herunterladen des instrumentalisierten Quellcodes aus dem SVN-Repository sowie dessen Kompilierung und Installation sollte über JDataMining steuerbar sein, anstatt die Makefile auf dem Server auszuführen.
- Es könnte über einen E-Mail-Versand nachgedacht werden, der bei einem Fehler dem Anwender eine E-Mail mit Fehlerbericht schreibt, so dass der Anwender schneller über

ein mögliches Problem benachrichtigt werden kann und nicht erst, wenn er zu einem späteren Zeitpunkt die Testergebnisse am Testsystem überprüfen will. Je nachdem könnte das Problem dann auch direkt per Fernzugriff behoben werden.

- Dateien wie die CSV-Dateien, Log-Dateien, etc. sollten nach erfolgreicher Bearbeitung in einen anderen Ordner verschoben werden.

- Im Register Einstellungen sollten die Dateien und Pfade zusätzlich durch eine Schaltfläche “Durchsuchen...” ausgewählt werden können. Bisher können sie nur eingetippt werden.

- JDataMining läuft zur Zeit unter Linux. Für ein einwandfreies Funktionieren unter Windows müsste noch ein Skript entwickelt werden, welches Daten über den Netzwerkverkehr, die Systemauslastung etc. liefert, so wie es Log.sh unter Linux bereits macht.

- Es sollte noch eine Auswahlbox im Registerfenster Querauswertung eingebaut werden, welche es ermöglicht, zwischen verschiedenen statistischen Größen zu wählen. Bisher wird immer nur das 90%-Quantil berechnet. Es stehen im Quellcode aber bereits viele weitere verschiedene statistische Berechnungen zur Verfügung. Das Gleiche gilt auch für das Erzeugen der HTML-Seite. Hier sollte man individuell bestimmen können, welche Werte in der HTML-Seite angezeigt werden.

- Eventuell wäre ein Registerfenster “Hilfe” angebracht. Hier könnte der Benutzer bei Problemen zunächst nachsehen. Falls dort keine Lösung für das Problem zu finden ist, kann der Benutzer dem Autor des Programmes eine E-Mail senden.

- Für den Fall, dass Fehler beim Programmablauf geschehen, könnten die Fehler in eine Log-Datei geschrieben werden, anstatt sie auf der Systemkonsole auszugeben.

6.2 Verbesserungen und Erweiterungen am Messframework

Zur Verbesserung der zeitlichen Auflösung und zur eventuellen Lösung des Zeitauflösungsproblems beim Prefork MPM wäre es angebracht, das bestehende Messframework komplett auf ein Shared-Memory-System umzustellen. Bisher werden die Messdaten noch zu oft während des Betriebes in eine Datei geschrieben.

Eventuell sollte geprüft werden, ob eine direkte Berechnung der Zeitwerte im Messframework einen postiven oder negativen Einfluss auf die Zeitauflösung hätte. Eigentlich müsste

davon auszugehen sein, dass die Zeitauflösung dadurch verschlechtert wird, da während des Serverbetriebs zusätzliche Berechnungen anfallen. Andererseits müsste so nur noch die Hälfte an Daten in die CSV-Datei geschrieben werden. Es könnte also sein, dass sich dies wiederum positiv auf die Zeitauflösung auswirken könnte. Angenommen, der Effekt wäre positiv, so könnte zu einem Großteil auf die Zeitberechnungen in JDataMining verzichtet werden. Das Programm würde seine Laufzeit dadurch erheblich verkürzen.

Eine Verbesserung des Messframeworks in Bezug auf die Erfassung der Prozessorticks wäre je nachdem sinnvoll. Zur Zeit werden nur die unteren 32 Bit der Prozessortickangabe berücksichtigt. Eine Erweiterung auf die gesamten 64 Bit würde das Problem mit den Überläufen, welche ca. alle 2 Sekunden stattfinden, aufheben. So müsste dann nicht mehr der Zeitstempel für den Prozessortickwert einspringen, falls ein Überlauf stattfindet. Dies könnte weiterhin die Exaktheit der Messwerte verbessern.

6.3 Weitere Untersuchungen und Messungen

Eine Idee für weitere Messungen wäre, bei der Parametervariation ausgefallenere Varianten zu verwenden. Es könnte beispielsweise überprüft werden, was passiert, wenn man die Parameter alle auf den minimalst oder maximalst möglichen Wert setzt. Oder was passieren würde, wenn beim Worker MPM der Parameter ThreadsPerChild auf 1 gesetzt würde. Oder der Parameter MaxRequestsPerChild wird auf 1 gesetzt, so dass ein Prozess immer nur eine Anfrage bearbeiten darf und sich dann zerstören muss. Lauter solcher Variationen könnten eventuell noch einige interessante Dinge aufzeigen.

Da bei den Messungen in dieser Arbeit der Server mit den verwendeten Modulen nie abgestürzt ist und die Fehlerquote bei 0% lag, wäre es interessant, ob es Parameterkonstellationen oder anderweitige Bedingungen gibt, bei denen ein Fehler mit großer Wahrscheinlichkeit auftreten würde. Hier könnte man die Fehlerrate der gemessenen Module vergleichen.

Des Weiteren würde sich anbieten, JDataMining mit weiteren MPMs zu testen. Beispielsweise könnte das Verhalten des experimentellen Event MPM untersucht werden und mit den bisher untersuchten MPMs Worker und Prefork verglichen werden.

Literaturverzeichnis

[BNO+07] Christian Bell, Pedram Nazari, Simon Olofsson, Jan Schrupp und Stephanie Wille. *Erweiterung und Validierung eines Modells zur Simulation des Apache Webservers*. Lehrstuhl Integrierte Informationssysteme, Ruhr-Universität Bochum, 2007.

[Fou06] The Apache Software Foundation. *Apache HTTP Server Version 2.2 - Modul-Index*. http://httpd.apache.org/docs/2.2/mod/, 2006.

[Gas06] Martin Gasse. *Implementierung eines Modells zur Simulation des Webservers Tomcat*. Lehrstuhl Integrierte Informationssysteme, Ruhr-Universität Bochum, Ruhr-Universität Bochum, 2006.

[Ges05] Ulrich Gesing. *Simulationsmodell des Apache Webservers*. Lehrstuhl Integrierte Informationssysteme, Ruhr-Universität Bochum, Ruhr-Universität Bochum, 2005.

[GKKS04] Bernhard Gröne, Andreas Knöpfel, Rudolf Kugel und Oliver Schmidt. *The Apache Modeling Project*. Hasso-Plattner-Institute for Software Systems Engineering, Potsdam, 2004.

[Her07] Alexander Hergenroeder. *Erstellung einer grafischen Oberfläche für Performancemessung eines Apache-Servers*. Lehrstuhl Integrierte Informationssysteme, Ruhr-Universität Bochum, 2007.

[JC06] JFreeChart-Community. To Those Who Use JFreeChart for Dynamic Plotting/Large Sets. http://www.jfree.org/phpBB2/viewtopic.php?t=18592-&start=0&postdays=0&postorder=asc&highlight=, August 2006.

[Ltd07] Netcraft Ltd. August 2007 Web Server Survey. http://news.netcraft.com/archives-/2007/08/06/august_2007_web_server_survey.html, August 2007.

[Nie01] Robert Nielsen. Working in Java time - Learn the basics of calculating elapsed time in Java. http://www.javaworld.com/javaworld/jw-03-2001/jw-0330-time.html?page=1, März 2001.

[Pen05] Lindsay Pender. Another dynamic data solution. http://www.jfree.org/phpBB2-/viewtopic.php?t=11499&start=0&postdays=0&postorder=asc&highlight=, Januar 2005.

[Wes02] André Westhoff. *Entwicklung und algorithmische Umsetzung von Simulationsszenarien zur Qualitätsbewertung von lokalen Netzwerken, Diplomarbeit*. Arbeitsgruppe Integrierte Informationssysteme, Ruhr-Universität Bochum, 2002.

[Wie04] Stephan Wiesner. *Java. Der Code*. Mitp-Verlag, 2004.

7 Anhang

7.1 Quelltexte

Es werden nur Klassen aufgeführt, die im Hauptprogramm enthalten sind und keine Fremdklassen sind (Ausnahme: Modifizierte Fremdklassen).

7.1.1 Paket Ant

AntStarter

```
package de.christianbell.apmfw.jdragicdatamining.ant;
import java.io.BufferedReader; import java.io.IOException;
import java.io.InputStreamReader;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
/*
* Diese Klasse beinhaltet die Ausfuehrung einer Ant-Datei
* abhaengig vom Betriebssystem
* @author Christian Bell
*/
public class AntStarter
{
   public static void runAnt() throws IOException, InterruptedException
   {
     Runtime runtime = Runtime.getRuntime();
     Process proc = null;
     String prefix = null;
     String operatingSystem = System.getProperty("os.name");
     operatingSystem = operatingSystem.toLowerCase();
     System.out.println("Erkanntes Betriebssystem: " + operatingSystem);
     if (operatingSystem.indexOf("windows") != -1)
     {
       if (operatingSystem.indexOf("95") != -1)
       {
         prefix = "command.com /c ";
       }
       else if (operatingSystem.indexOf("98") != -1)
       {
         prefix = "command.com /c ";
       }
       else if (operatingSystem.indexOf("xp") != -1)
       {
```

```
        prefix = "cmd /c ";
      }
    }
    else if(operatingSystem.indexOf("linux") != -1)
    {
      prefix = "sh -c ";
    }
    if (prefix == null)
    {
      System.out.println("Unbekanntes Betriebssystem!!!");
    }
    else
    {
      GUI.konsoleArea.append("Starte Ant\n");
      GUI.autoScroll();
      try
      {
        proc = runtime.exec(prefix + "ant -f build.xml");
        BufferedReader procout = new BufferedReader(new
                       InputStreamReader(proc.getInputStream()));
        String line;
        while ((line = procout.readLine()) != null)
        {
          GUI.konsoleArea.append("  :: " + line +"\n");
          GUI.autoScroll();
        }
        GUI.konsoleArea.append("Ant ist fertig\n");
        GUI.autoScroll();
      }
      catch(Exception e)
      {
        e.printStackTrace();
      }
    }
  }

  public static void main(String[] args)
  {
    try
    {
      runAnt();
    }
    catch (IOException e)
    {
      e.printStackTrace();
    }
    catch (InterruptedException e)
    {
      e.printStackTrace();
    }
  }
}
```

AntWriter

```
package de.christianbell.apmfw.jdragicdatamining.ant;
import java.io.BufferedOutputStream; import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.DataOutputStream;
import java.io.File;
import java.io.FileOutputStream;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Hashtable;
import org.jdom.*;
import org.jdom.input.SAXBuilder;
import org.jdom.output.Format;
import org.jdom.output.XMLOutputter;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.ParameterParser;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
import de.christianbell.apmfw.jdragicdatamining.server.*;
/*
 * Diese Klasse kuemmert sich um die Erzeugung der Ant-Datei zum Messen
 * @author Christian Bell
 * @version 0.1.1
 */
public class AntWriter
{
   private static PropertiesSingleton ps = PropertiesSingleton.getInstance();
   private Parameter ersterParameter = null;
   private Parameter zweiterParameter = null;
   private Parameter dritterParameter = null;
   private Parameter ersterJmeterParameter = null;
   private Parameter zweiterJmeterParameter = null;

   public boolean setParams()
   {
     int i = 0;
     int max = 3;
     int maxJmeter = 2;
     int start, stop, step;
     int size = GUI.getServerParamsList().getModel().getSize();
     String elem;
     String modul = GUI.getModulComboBox().getSelectedItem().toString();
     //Setzt die ServerParameter und ueberprueft ob drei Parameter variiert werden
     while(i < size)
     {
       elem = GUI.getServerParamsList().getModel().getElementAt(i).toString();
       start = Integer.valueOf(ps.getProperty(modul+elem+"Start"));
       stop = Integer.valueOf(ps.getProperty(modul+elem+"Stop"));
       step = Integer.valueOf(ps.getProperty(modul+elem+"Step"));
       if (step > 0)
       {
         if (max == 3)
           ersterParameter = new Parameter(start, stop, step, elem);
```

```
    else if(max == 2)
      zweiterParameter = new Parameter(start, stop, step, elem);
    else if(max == 1)
      dritterParameter = new Parameter(start, stop, step, elem);
    max--;
  }
  if (max < 0)
  {
    GUI.konsoleArea.append("Bitte hoechstens drei Server-Parameter variieren
                           und alle anderen auf Schrittweite 0 setzen.");
    return false;
  }
  i++;
}
i=0;
if (max == 3)
{
  System.out.println("max ist gleich 3!");
  while(i < size)
  {
    elem = GUI.getServerParamsList().getModel().getElementAt(i).toString();
    start = Integer.valueOf(ps.getProperty(modul+elem+"Start"));
    stop = Integer.valueOf(ps.getProperty(modul+elem+"Stop"));
    step = Integer.valueOf(ps.getProperty(modul+elem+"Step"));
    if (max == 3)
      ersterParameter = new Parameter(start, stop, step, elem);
    else if(max == 2)
      zweiterParameter = new Parameter(start, stop, step, elem);
    else if(max == 1)
      dritterParameter = new Parameter(start, stop, step, elem);
    max--;
    i++;
  }
}
else if (max == 2)
{
  boolean zweiterSchonBesucht = false;
  boolean dritterSchonBesucht = false;
  while(i < size)
  {
    elem = GUI.getServerParamsList().getModel().getElementAt(i).toString();
    start = Integer.valueOf(ps.getProperty(modul+elem+"Start"));
    stop = Integer.valueOf(ps.getProperty(modul+elem+"Stop"));
    step = Integer.valueOf(ps.getProperty(modul+elem+"Step"));
    if (step == 0)
    {
      if(!zweiterSchonBesucht)
        zweiterParameter = new Parameter(start, stop, step, elem);
      else if(zweiterSchonBesucht && !dritterSchonBesucht)
        dritterParameter = new Parameter(start, stop, step, elem);
    }
    i++;
  }
}
```

```
else if (max == 1)
{
  boolean schonBesucht = false;
  while(i < size)
  {
    elem = GUI.getServerParamsList().getModel().getElementAt(i).toString();
    start = Integer.valueOf(ps.getProperty(modul+elem+"Start"));
    stop = Integer.valueOf(ps.getProperty(modul+elem+"Stop"));
    step = Integer.valueOf(ps.getProperty(modul+elem+"Step"));
    if (step == 0 && !schonBesucht)
    {
      dritterParameter = new Parameter(start, stop, step, elem);
      schonBesucht = true;
    }
    i++;
  }
}
i = 0;
GUI gui = new GUI();
int sizeJmeter = gui.getJmeterParamsList().getModel().getSize();
//Setzt die JMeterParamter und ueberprueft ob zwei Parameter gesetzt wurden
while(i < sizeJmeter)
{
  elem = gui.getJmeterParamsList().getModel().getElementAt(i).toString();
  start = Integer.valueOf(ps.getProperty(elem+"Start"));
  stop = Integer.valueOf(ps.getProperty(elem+"Stop"));
  step = Integer.valueOf(ps.getProperty(elem+"Step"));
  if (step > 0)
  {
    if (maxJmeter == 2)
      ersterJmeterParameter = new Parameter(start, stop, step, elem);
    else if(maxJmeter == 1)
      zweiterJmeterParameter = new Parameter(start, stop, step, elem);
    maxJmeter--;
  }
  if (maxJmeter < 0)
  {
    GUI.konsoleArea.append("Bitte hoechstens drei Server-Parameter variieren
                           und alle anderen auf Schrittweite 0 setzen.");
    return false;
  }
  i++;
}
i=0;
if (maxJmeter == 2)
{
while(i < sizeJmeter)
{
  elem = gui.getJmeterParamsList().getModel().getElementAt(i).toString();
  start = Integer.valueOf(ps.getProperty(elem+"Start"));
  stop = Integer.valueOf(ps.getProperty(elem+"Stop"));
  step = Integer.valueOf(ps.getProperty(elem+"Step"));
  if (maxJmeter == 2)
    ersterJmeterParameter = new Parameter(start, stop, step, elem);
```

```
        else if(maxJmeter == 1)
          zweiterJmeterParameter = new Parameter(start, stop, step, elem);
        maxJmeter--;
        i++;
      }
    }
    else if (maxJmeter == 1)
    {
      boolean schonBesucht = false;
      while(i < sizeJmeter)
      {
        elem = gui.getJmeterParamsList().getModel().getElementAt(i).toString();
        start = Integer.valueOf(ps.getProperty(elem+"Start"));
        stop = Integer.valueOf(ps.getProperty(elem+"Stop"));
        step = Integer.valueOf(ps.getProperty(elem+"Step"));
        if (step == 0 && !schonBesucht)
        {
          zweiterJmeterParameter = new Parameter(start, stop, step, elem);
          schonBesucht = true;
        }
        i++;
      }
    }
    return true;
}
/*
  * Diese Funktion ueberprueft die Angaben des Benutzers
  * @return Boolean false, wenn was falsch ist
  */
public boolean checkFields()
{ //TODO Alles ueperpruefen
    //Ueberpruefe, ob hoechstens drei Parameter gesetzt (speichere dann auch Params)
    if(!setParams())
      return false;
    //Ueberpruefe CGI-Einstellungen
    int cgiStart = Integer.valueOf(ps.getProperty("cgiStart"));
    int cgiStop = Integer.valueOf(ps.getProperty("cgiStop"));
    int cgiStep = Integer.valueOf(ps.getProperty("cgiStep"));
    if(cgiStep!=0 && (cgiStop - cgiStart)%cgiStep != 0)
    {
      GUI.konsoleArea.append("Das Verhaeltnis der CGI-Parameter stimmt nicht!\n"
                             + "Format: Stopwert > Startwert
                             und Stopwert - Startwert teilbar durch Schrittweite!");
      GUI.autoScroll();
      return false;
    }
    return true;
}
/*
  * Diese Funktion erstellt die build.xml
*/
public void createAntFile() throws IOException, JDOMException
{
    //Die Ant-Datei wird nur geschrieben,
```

```
//wenn die Felder der GUI alle Bedingungen erfuellen
if(checkFields())
{
  //Die zu erzeugende Ant-Datei
  File originalFile = new File(ps.getProperty("jmeterxml"));
  if(originalFile.exists())
    originalFile.delete();
  originalFile.createNewFile();
  RandomAccessFile rafile = new RandomAccessFile(originalFile, "rw");
  //grobes Geruest erstellen
  rafile.writeBytes("<?xml version=\"1.0\" encoding=\"UTF-8\"?>");
  rafile.writeBytes("<project name=\"JMeter\" default=\"jmeter\" basedir=\".\">");
  rafile.writeBytes("<target name=\"jmeter\">");
  rafile.writeBytes("<taskdef name=\"jmeter\"
          classname=\"org.programmerplanet.ant.taskdefs.jmeter.JMeterTask\" />");
  rafile.writeBytes("<taskdef
            resource=\"net/sf/antcontrib/antcontrib.properties\">");
  rafile.writeBytes("<classpath>");
  rafile.writeBytes("<pathelement
        location=\"/usr/java/jdk1.5.0_07/lib/ant-contrib-0.3.jar\"/>");
  rafile.writeBytes("</classpath></taskdef>");
  rafile.writeBytes("</target>");
  rafile.writeBytes("</project>");
  rafile.close();
  //XML-Datei mit SAXBuilder als doc einlesen
  Document doc = new SAXBuilder().build(originalFile);
  //Hole Wurzelelement
  Element project = doc.getRootElement();
  //Hole das Element 'target'
  Element target = project.getChild( "target" );
  //Erzeuge ein neues Element 'jmeter'
  String name;
  String outHTML;
  String jmeterhome = ps.getProperty("jmeterhome");
  String resultlog;
  String style = ps.getProperty("xmlstilvorlage");
  int cgiDurchgang = 0;
  ParameterParser paramPars = new ParameterParser("!--PARAM::", "--!");
  //TODO In Config aufnehmen
  String patchPfad = "/home/prakti/apmfw/mfw/apmfw-patch/";
  File jmeterFile = new File(ps.getProperty("jmetervorlage"));
  File loadtestsPath = new File(ps.getProperty("loadtests"));
  int id = Integer.valueOf(ps.getProperty("id"));
  for(int cgi = Integer.valueOf(ps.getProperty("cgiStart"));
          cgi <= Integer.valueOf(ps.getProperty("cgiStop"));
          cgi += Integer.valueOf(ps.getProperty("cgiStep")))
  {
    for(int processingDelay = Integer.valueOf(ps.getProperty("processingDelayStart"));
            processingDelay <= Integer.valueOf(ps.getProperty("processingDelayStop"));
            processingDelay +=
              Integer.valueOf(ps.getProperty("processingDelayStep")))
    {
      Element sleep17 = new Element("sleep");
      sleep17.setAttribute("seconds", "2");
```

```
target.addContent(sleep17);
int i = 0;
int size = GUI.getServerParamsList().getModel().getSize();
String elem = null;
String modul = GUI.getModulComboBox().getSelectedItem().toString();
Hashtable<String,String> ServerErsetzliste = new Hashtable<String, String>();
File file = new File("Apache/httpd-mpm.conf");
while(i < size)
{
  elem = GUI.getServerParamsList().getModel().getElementAt(i).toString();
  ServerErsetzliste.put(modul + elem, ps.getProperty(modul+elem+"Start"));
  i++;
}
int j = 0;
GUI gui = new GUI();
int sizeJmeterList = gui.getJmeterParamsList().getModel().getSize();
Hashtable<String,String> JmeterErsetzliste = new Hashtable<String, String>();
while(j < sizeJmeterList)
{
  elem = gui.getJmeterParamsList().getModel().getElementAt(j).toString();
  JmeterErsetzliste.put(elem, ps.getProperty(elem+"Start"));
  j++;
}
JmeterErsetzliste.put("cgizeit", Integer.valueOf(cgi).toString());
JmeterErsetzliste.put("processingDelay",
           Integer.valueOf(processingDelay).toString());
JmeterErsetzliste.put("keepAlive", ps.getProperty("keepAlive"));
int count = 0;
if(ersterParameter.getSchrittweite() <= 0)
  ersterParameter.setSchrittweite(1);
for(int first = ersterParameter.getStartWert();
        first <= ersterParameter.getStopWert();
        first += ersterParameter.getSchrittweite())
{
  ServerErsetzliste.put(modul + ersterParameter.getName(),
            Integer.valueOf(first).toString());
  if(zweiterParameter.getSchrittweite() <= 0)
    zweiterParameter.setSchrittweite(1);
  for(int second = zweiterParameter.getStartWert();
          second <= zweiterParameter.getStopWert();
          second += zweiterParameter.getSchrittweite())
  {
    ServerErsetzliste.put(modul + zweiterParameter.getName(),
              Integer.valueOf(second).toString());
    if(dritterParameter.getSchrittweite() <= 0)
      dritterParameter.setSchrittweite(1);
    for(int third = dritterParameter.getStartWert();
            third <= dritterParameter.getStopWert();
            third += dritterParameter.getSchrittweite())
    {
      ServerErsetzliste.put(modul + dritterParameter.getName(),
                Integer.valueOf(third).toString());
      try
      {
```

```
   FileReader fileReader = new FileReader(file);
   BufferedReader buffReader = new BufferedReader(fileReader);
   StringBuffer sb = new StringBuffer();
   String line;
   line = buffReader.readLine();
   sb.append(line + "\n");
   while(line != null)
   {
     sb.append(line +"\n");
     line = buffReader.readLine();
   }
   buffReader.close();
  //Ersetze Parameter
  String newString = paramPars.ersetzeParameter(ServerErsetzliste,
                     sb.toString());
  File ordner = new File("Apache/" + count +"/");
  if(!ordner.exists())
    ordner.mkdirs();
  File newFile = new File("Apache/" + count + "/httpd-mpm.conf");
  FileWriter fileWriter = new FileWriter(newFile);
  BufferedWriter buffWriter = new BufferedWriter(fileWriter);
  buffWriter.append(newString);
  buffWriter.close();
  fileReader.close();
  fileWriter.close();
  count++;
  Element scp = new Element("scp");
  scp.setAttribute("file", newFile.getAbsolutePath());
  scp.setAttribute("todir", ps.getProperty("serveruser") +":"
                  + ps.getProperty("password") + "@"
                  + ps.getProperty("server")
                  + ":/usr/local/apache2/conf/extra/");
  scp.setAttribute("trust", "yes");
  target.addContent(scp);
  Element sleep18 = new Element("sleep");
  sleep18.setAttribute("seconds", "1");
  target.addContent(sleep18);
}
catch(IOException e)
{
  e.printStackTrace();
}
for(int jmeterFirst = ersterJmeterParameter.getStartWert();
        jmeterFirst <= ersterJmeterParameter.getStopWert();
        jmeterFirst += ersterJmeterParameter.getSchrittweite())
{
  JmeterErsetzliste.put(ersterJmeterParameter.getName(),
        Integer.valueOf(jmeterFirst).toString());
  if(zweiterJmeterParameter.getSchrittweite() <= 0)
    zweiterJmeterParameter.setSchrittweite(1);
  for(int jmeterSecond = zweiterJmeterParameter.getStartWert();
          jmeterSecond <= zweiterJmeterParameter.getStopWert();
          jmeterSecond += zweiterJmeterParameter.getSchrittweite())
  {
```

```
JmeterErsetzliste.put(zweiterJmeterParameter.getName(),
       Integer.valueOf(jmeterSecond).toString());
name = ps.getProperty("messungsname") + "_" + first + "_" + second
        + "_" + third + "_" + jmeterFirst + "_" + jmeterSecond
        + "_" + cgi + "_" + processingDelay;
for(int wiederholungen =
        Integer.valueOf(ps.getProperty("anzahlWiederholungen"));
    wiederholungen >= 0; wiederholungen--)
{
  id++;
  //Server starten
  Element sshExecServerStart = new Element("sshexec");
  sshExecServerStart.setAttribute("host", ps.getProperty("server"));
  sshExecServerStart.setAttribute("username",
                    ps.getProperty("serveruser"));
  sshExecServerStart.setAttribute("password",
                    ps.getProperty("password"));
  sshExecServerStart.setAttribute("command",
                    "/usr/local/apache2/bin/apachectl start");
  sshExecServerStart.setAttribute("trust", "yes");
  target.addContent(sshExecServerStart);
  Element sleep1 = new Element("sleep");
  sleep1.setAttribute("seconds", "10");
  target.addContent(sleep1);
  //log.sh Server start
  Element forgetServerStart = new Element("forget");
  Element sshServerLogStart = new Element("sshexec");
  sshServerLogStart.setAttribute("host", ps.getProperty("server"));
  sshServerLogStart.setAttribute("username",
                   ps.getProperty("serveruser"));
  sshServerLogStart.setAttribute("password",
                   ps.getProperty("password"));
  sshServerLogStart.setAttribute("command",
                   "log.sh /home/prakti/apmfw/logs/cj/" + id
                   + "Server_" + name + "_" + wiederholungen
                   + " start");
  forgetServerStart.addContent(sshServerLogStart);
  target.addContent(forgetServerStart);
  //log.sh Client start
  Element forget = new Element("forget");
  Element execClientLogStart = new Element("exec");
  execClientLogStart.setAttribute("executable", "sh");
  Element arg = new Element("arg");
  arg.setAttribute("line", "log.sh " + ps.getProperty("logspath")
          + id +"Client_" + name + "_" + wiederholungen + " start");
  execClientLogStart.addContent(arg);
  forget.addContent(execClientLogStart);
  target.addContent(forget);
  //Kurz schlafen, damit SSH fertig wird, bevor gemessen wird
  Element sleep3 = new Element("sleep");
  sleep3.setAttribute("seconds", "1");
  target.addContent(sleep3);
  //Erzeuge Testplaene-Tasks
  resultlog = ps.getProperty("jtlpath") + "/" + id + name +"_"
```

```
            + wiederholungen + ".jtl";
outHTML = loadtestsPath + "/" + id + name +"_" + wiederholungen
          + ".html";
Element jmeter = new Element( "jmeter" );
//Setze die Attribute des Elementes 'jmeter'
jmeter.setAttribute("jmeterhome",jmeterhome);
jmeter.setAttribute("resultlog", resultlog);
jmeter.setAttribute("testplan", loadtestsPath + "/" + id + name
                    + ".jmx");
//Fuege Element 'jmeter' dem Element 'target' hinzu
target.addContent(jmeter);
//Erzeuge neues Element 'property'
Element property = new Element("property");
//Setze die Attribute des Elements 'property'
property.setAttribute("name",
     "jmeter.save.saveservice.response_data");
property.setAttribute("value","true");
//Fuege 'property' dem Element 'jmeter' hinzu
jmeter.addContent(property);
property = new Element("property");
property.setAttribute("name",
     "jmeter.save.saveservice.output_format");
property.setAttribute("value","xml");
//Fuege 'property' dem Element 'jmeter' hinzu
jmeter.addContent(property);
Element sleep4 = new Element("sleep");
sleep4.setAttribute("seconds", "2");
target.addContent(sleep4);
//log.sh Server stop
Element forgetServerStop = new Element("forget");
Element sshServerLogStop = new Element("sshexec");
sshServerLogStop.setAttribute("host",
      ps.getProperty("server"));
sshServerLogStop.setAttribute("username",
      ps.getProperty("serveruser"));
sshServerLogStop.setAttribute("password",
      ps.getProperty("password"));
sshServerLogStop.setAttribute("command",
      "log.sh /home/prakti/apmfw/logs/cj/"
      +id +"Server_" + name + "_" + wiederholungen + " stop");
forgetServerStop.addContent(sshServerLogStop);
target.addContent(forgetServerStop);
Element execClientLogStop = new Element("exec");
execClientLogStop.setAttribute("executable", "sh");
Element arg2 = new Element("arg");
arg2.setAttribute("line", "log.sh " + ps.getProperty("logspath")
  + id + "Client_" + name + "_" + wiederholungen + " stop");
execClientLogStop.addContent(arg2);
target.addContent(execClientLogStop);
//Server stoppen
Element sshExecServerStop = new Element("sshexec");
sshExecServerStop.setAttribute("host", ps.getProperty("server"));
sshExecServerStop.setAttribute("username",
      ps.getProperty("serveruser"));
```

```
sshExecServerStop.setAttribute("password",
      ps.getProperty("password"));
sshExecServerStop.setAttribute("command",
      "/usr/local/apache2/bin/apachectl stop");
sshExecServerStop.setAttribute("trust", "yes");
target.addContent(sshExecServerStop);
Element sleep7 = new Element("sleep");
sleep7.setAttribute("seconds", "4");
target.addContent(sleep7);
//Orig-Datei holen
Element scpOrigCSV = new Element("scp");
scpOrigCSV.setAttribute("file", ps.getProperty("serveruser")
  +":" + ps.getProperty("password") + "@"
  + ps.getProperty("server")
  + ":/usr/local/apache2/htdocs/apmfw/orig.csv");
scpOrigCSV.setAttribute("todir", "./Apache/origs/");
scpOrigCSV.setAttribute("trust", "yes");
target.addContent(scpOrigCSV);
Element sleep8 = new Element("sleep");
sleep8.setAttribute("seconds", "1");
target.addContent(sleep8);
//Benennt die Orig-Datei in spezifischen Messnamen um
Element move = new Element("move");
move.setAttribute("file", "./Apache/origs/orig.csv");
move.setAttribute("tofile", "./Apache/origs/results_" + id
     + name + "_" + wiederholungen + ".csv");
target.addContent(move);
//Orig von Server loeschen
Element sshExecOrigDelete = new Element("sshexec");
sshExecOrigDelete.setAttribute("host", ps.getProperty("server"));
sshExecOrigDelete.setAttribute("username",
     ps.getProperty("serveruser"));
sshExecOrigDelete.setAttribute("password",
     ps.getProperty("password"));
sshExecOrigDelete.setAttribute("command",
     "rm -r /usr/local/apache2/htdocs/apmfw/orig.csv");
sshExecOrigDelete.setAttribute("trust", "yes");
target.addContent(sshExecOrigDelete);
//Logs vom Server holen
//IOStat
Element scpIOStat = new Element("scp");
scpIOStat.setAttribute("file", ps.getProperty("serveruser")
  +":" + ps.getProperty("password") + "@"
  + ps.getProperty("server") + ":/home/prakti/apmfw/logs/cj/"
  + id + "Server_" + name + "_" + wiederholungen
  + "_iostatlog.txt");
scpIOStat.setAttribute("todir", ps.getProperty("logspath"));
scpIOStat.setAttribute("trust", "yes");
target.addContent(scpIOStat);
//TODO ALLE LOGS VOM SERVER LOESCHEN
//NetStat
Element scpNetStat = new Element("scp");
scpNetStat.setAttribute("file", ps.getProperty("serveruser")
    +":" + ps.getProperty("password") + "@"
```

```
        + ps.getProperty("server") + ":/home/prakti/apmfw/logs/cj/"
        + id + "Server_" + name + "_" + wiederholungen
        + "_netstatlog.txt");
     scpNetStat.setAttribute("todir", ps.getProperty("logspath"));
     scpNetStat.setAttribute("trust", "yes");
     target.addContent(scpNetStat);
     //SarLog
    Element scpSarLog = new Element("scp");
    scpSarLog.setAttribute("file", ps.getProperty("serveruser")
        +":" + ps.getProperty("password") + "@"
        + ps.getProperty("server") + ":/home/prakti/apmfw/logs/cj/"
        + id + "Server_" + name + "_" + wiederholungen + "_sarlog.txt");
    scpSarLog.setAttribute("todir", ps.getProperty("logspath"));
    scpSarLog.setAttribute("trust", "yes");
    target.addContent(scpSarLog);
    //VmStat
    Element scpVMStat = new Element("scp");
    scpVMStat.setAttribute("file", ps.getProperty("serveruser")
        +":" + ps.getProperty("password") + "@"
        + ps.getProperty("server") + ":/home/prakti/apmfw/logs/cj/"
        + id + "Server_" + name + "_" + wiederholungen
        + "_vmstatlog.txt");
    scpVMStat.setAttribute("todir", ps.getProperty("logspath"));
    scpVMStat.setAttribute("trust", "yes");
    target.addContent(scpVMStat);
    Element sleep15 = new Element("sleep");
    sleep15.setAttribute("seconds", "3");
    target.addContent(sleep15);
  }
  //Erzeuge JMX-Dateien
  try
  {
    FileReader fileReader = new FileReader(jmeterFile);
    BufferedReader buffReader = new BufferedReader(fileReader);
    StringBuffer sb = new StringBuffer();
    String line;
    line = buffReader.readLine();
    while(line != null)
    {
      sb.append(line +"\n");
      line = buffReader.readLine();
    }
    buffReader.close();
    //Ersetze Parameter
    String newString = paramPars.ersetzeParameter(JmeterErsetzliste,
                       sb.toString());
    File newFile = new File(loadtestsPath + "/" + Integer.toString(id)
                  + name + ".jmx");
    if(newFile.exists())
      newFile.delete();
    newFile.createNewFile();
    FileWriter fileWriter = new FileWriter(newFile);
    BufferedWriter buffWriter = new BufferedWriter(fileWriter);
    buffWriter.append(newString);
```

```
                    buffWriter.close();
                    fileReader.close();
                    fileWriter.close();
                  }
                  catch(IOException e)
                  {
                    e.printStackTrace();
                  }
                }
              }
            }
          }
          //Erzeuge eine XML-Ausgabe mit einem 'schoenen' Format
          XMLOutputter out = new XMLOutputter( Format.getPrettyFormat() );
          //Erzeuge eine Ausgabedatei
          DataOutputStream outFile = new DataOutputStream(
                                     new BufferedOutputStream(
                                     new FileOutputStream(originalFile)));
          //Schreibe die XML-Ausgabe in die Ausgabedatei
          GUI.konsoleArea.append(out.outputString(doc));
          out.output( doc, outFile );
          GUI.autoScroll();
          if(Integer.valueOf(ps.getProperty("processingDelayStep")) == 0)
            processingDelay++;
        }
        if(id > 99999)
          id = 0;
        ps.setProperty("id", Integer.toString(id));
        if(Integer.valueOf(ps.getProperty("cgiStep")) == 0)
          cgi++;
      }
    }
}}}
```

7.1.2 Das Paket Auswertung

HTMLOverview

```
package de.christianbell.apmfw.jdragicdatamining.auswertung;
import java.io.File;
import java.io.RandomAccessFile;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Iterator;
import java.util.Vector;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
/*
 * Diese Klasse generiert eine Uebersicht ueber die gemessenen
 * Informationen als HTML-Seite
 * @author Christian Bell
 */
public class HTMLOverview
```

```
{
  /**Erzeugt die HTML-Seite
  *
  * @param tabellenName Namen der Tabellen
  * @param messpunktNamen Namen der Messpunkte
  * @param fileStr Dateiname
  * @exception SQLException
  * @exception Exception
  */
  public void createHTML(Vector<String> tabellenNamen,
                         Vector<String> messpunktNamen, String fileStr)
  {
    try
    {
      File file = new File(fileStr + "/index.html");
      if(file.exists())
      {
        file.delete();
        file.createNewFile();
      }
      RandomAccessFile rafile = new RandomAccessFile(fileStr + "/index.html", "rw");
      rafile.writeBytes("<html>\n");
      rafile.writeBytes("<head>\n");
      rafile.writeBytes("<title>\n");
      rafile.writeBytes("Messergebnisse\n");
      rafile.writeBytes("</title>\n");
      rafile.writeBytes("</head>\n");
      rafile.writeBytes("<body>\n");
      rafile.writeBytes("<h1>Messergebnisse</h1>\n");
      rafile.writeBytes("<div>\n");
      rafile.writeBytes("<table border=\"1\">\n");
      rafile.writeBytes("<tr>\n");
      rafile.writeBytes("<td><b>Messpunkt</b></td>\n");
      rafile.writeBytes("<td><b>Mittelwert</b></td>\n");
      rafile.writeBytes("<td><b>MIN</b></td>\n");
      rafile.writeBytes("<td><b>MAX</b></td>\n");
      //rafile.writeBytes("<td><b>Standardabweichung</b></td>\n");
      //rafile.writeBytes("<td><b>Varianz</b></td>\n");
      //rafile.writeBytes("<td><b>Mittlere absolute Abweichung</b></td>\n");
      //rafile.writeBytes("<td><b>Median</b></td>\n");
      rafile.writeBytes("<td><b>90%-Quantil</b></td>\n");
      //rafile.writeBytes("<td><b>Interquartilsabstand</b></td>\n");
      //rafile.writeBytes("<td><b>Range</b></td>\n");
      rafile.writeBytes("</tr>\n");
      Iterator tableItr = tabellenNamen.iterator();
      Iterator messpunktItr = messpunktNamen.iterator();
      String currentTableName;
      String currentMesspunktName;
      double quantil;
      DBConnection db = new DBConnection();
      Connection theConnection = db.getConnection();
      Statistics valComp = new Statistics();
      valComp.setConnection(theConnection);
      while(tableItr.hasNext())
```

```
{
  currentTableName = tableItr.next().toString();
  rafile.writeBytes("<tr><td><b>" + currentTableName + "</b></td></tr>\n");
  if(currentTableName.contains("results"))
  {
    while(messpunktItr.hasNext())
    {
    currentMesspunktName = messpunktItr.next().toString();
    GUI.konsoleArea.append(currentTableName + "_" + currentMesspunktName + "\n");
    GUI.autoScroll();
    rafile.writeBytes("<tr>\n");
    try
    {
      rafile.writeBytes("<td><a href=\"" + currentTableName + "_"
                        + currentMesspunktName + ".png\">"
                        + currentMesspunktName + "</a>" + "</td>\n");
      rafile.writeBytes("<td>" + valComp.calculateMittelwert(currentTableName,
                         currentMesspunktName) + "</td>\n");
      rafile.writeBytes("<td>" + valComp.getMin(currentTableName,
                         currentMesspunktName) + "</td>");
      rafile.writeBytes("<td>" + valComp.getMax(currentTableName,
                         currentMesspunktName) + "</td>");
      quantil = valComp.calculateQuantile(0.9, currentTableName,
                         currentMesspunktName);
      rafile.writeBytes("<td>" + (quantil >= 0.0 ? quantil : "Zu wenig Daten!")
                         + "</td>");
      //rafile.writeBytes("<td>"
                         + valComp.calculateStandardabweichung(currentTableName,
                         currentMesspunktName) + "</td>");
      //rafile.writeBytes("<td>" + valComp.calculateVarianz(currentTableName,
                         currentMesspunktName) + "</td>");
      //rafile.writeBytes("<td>" + valComp.calculateMAA(currentTableName,
                         currentMesspunktName) + "</td>");
      //rafile.writeBytes("<td>" + valComp.calculateMedian(currentTableName,
                         currentMesspunktName) + "</td>");
      //interquartilsabstand =
                         valComp.calculateInterquartilsbstand(currentTableName,
                             currentMesspunktName);
      //rafile.writeBytes("<td>" + (interquartilsabstand >= 0.0 ?
                         interquartilsabstand : "Zu wenig Daten!") + "</td>");
      //rafile.writeBytes("<td>" + valComp.calculateRange(currentTableName,
                         currentMesspunktName) + "</td>");
      rafile.writeBytes("</tr>\n");
    }
    catch(Exception e)
    {
      System.out.println("Fehler in Charts.createHTML(): " + e.getMessage());
      e.printStackTrace();
    }
  }
  messpunktItr = messpunktNamen.iterator();
}
else
{
```

```
        rafile.writeBytes("<tr>\n");
        GUI.konsoleArea.append(currentTableName + "\n");
        GUI.autoScroll();
        try
        {
          rafile.writeBytes("<td><a href=\"" + currentTableName + ".png\">"#
                            + currentTableName + "</a>" + "</td>\n");
          rafile.writeBytes("<td>" + valComp.calculateMittelwert(currentTableName,
                            null) + "</td>\n");
          rafile.writeBytes("<td>" + valComp.getMin(currentTableName, null) + "</td>");
          rafile.writeBytes("<td>" + valComp.getMax(currentTableName, null) + "</td>");
          quantil = valComp.calculateQuantile(0.9, currentTableName, null);
          rafile.writeBytes("<td>" + (quantil >= 0.0 ? quantil : "Zu wenig Daten!")
                            + "</td>");
          //rafile.writeBytes("<td>"
                             + valComp.calculateStandardabweichung(currentTableName,
                              null) + "</td>");
          //rafile.writeBytes("<td>" + valComp.calculateVarianz(currentTableName,
                             null) + "</td>");
          //rafile.writeBytes("<td>" + valComp.calculateMAA(currentTableName, null)
                             + "</td>");
          //rafile.writeBytes("<td>" + valComp.calculateMedian(currentTableName,
                             null) + "</td>");
          //interquartilsabstand =
                             valComp.calculateInterquartilsbstand(currentTableName,
                                null);
          //rafile.writeBytes("<td>" + (interquartilsabstand >= 0.0 ?
                             interquartilsabstand : "Zu wenig Daten!") + "</td>");
          //rafile.writeBytes("<td>" + valComp.calculateRange(currentTableName, null)
                              + "</td>");
          rafile.writeBytes("</tr>\n");
        }
        catch(Exception e)
        {
          System.out.println("Fehler in Charts.createHTML(): " + e.getMessage());
          e.printStackTrace();
        }
      }
    }
    rafile.writeBytes("</table>\n");
    rafile.writeBytes("</div>\n");
    rafile.writeBytes("</body>\n");
    rafile.writeBytes("</html>");
    rafile.close();
    db.closeConnection(theConnection);
  }
  catch(Exception e)
  {
    System.out.println("Fehler in Charts.createHTML(): " + e.getMessage());
  }
 }
}
```

PNG

```
package de.christianbell.apmfw.jdragicdatamining.auswertung;
import java.io.File;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import de.christianbell.apmfw.jdragicdatamining.charts.JDBCMeanAndStandardDeviationXYDataset;
import de.christianbell.apmfw.jdragicdatamining.charts.JdbcChartPanel;
/*
 * Diese Klasse erzeugt eine PNG-Datei aus einem Chart
 * @author Christian Bell
 */
public class PNG
{
   /*Erzeugt ein PNG zu einem Messpunkt
   *
   * @param file Der Dateiname fuer die PNG
   * @param dataset Der Datensatz
   * @param tableName Tabellenname
   * @exception Exception
   */
   public void createPNG(File file, JDBCMeanAndStandardDeviationXYDataset dataset,
                         String tablename)
   {
     JFreeChart chart = JdbcChartPanel.createChart(dataset, tablename);
     try
     {
       ChartUtilities.saveChartAsPNG(file, chart, 1000, 640);
     }
     catch(Exception e)
     {
       System.out.println("Exception " + e.getMessage());
     }
   }
}
```

Statistics

```
package de.christianbell.apmfw.jdragicdatamining.auswertung;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
import de.christianbell.apmfw.jdragicdatamining.mfw.Cut;
import de.christianbell.apmfw.jdragicdatamining.mfw.Results;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Iterator;
import java.util.Vector;
/*
 * Diese Klasse bietet viele statistische Berechnungen
 * @author Christian Bell
```

```
 */
public class Statistics
{
   static DBConnection db;
   public Connection theConnection;
   public Connection getConnection() {
     return theConnection;
   }
   public void setConnection(Connection theConnection) {
     this.theConnection = theConnection;
   }
   /*
   * Diese Funktion berechnet die mittlere absolute Abweichung
   */
   public  double calculateMAA(String table, String messpunkt)
   {
     try
     {
       if(db == null)
       {
         db = new DBConnection();
       }
       if(theConnection == null)
       {
         theConnection = db.getConnection();
       }
       Statement statement = theConnection.createStatement();
       double sum = 0;
       long anzahl = 0;
       ResultSet result = null;
       if(messpunkt == null)
       {
         result = statement.executeQuery("SELECT value FROM " + table);
       }
       else
       {
         result = statement.executeQuery("SELECT value FROM " + table
                          + " WHERE mfwpoint='" + messpunkt +"'");
       }
       double median = calculateMedian(table, messpunkt);
       while(result.next())
       {
         sum += Math.abs((double)result.getLong("value") - median);
         anzahl++;
       }
       result.close();
       statement.close();
       return sum/(double)anzahl;
     }
     catch(Exception e)
     {
       System.out.println(e.getMessage());
     }
     return -1.0;
```

```
}
public  double calculateMittelwert(String table, String messpunkt)
{
  try
  {
    if(db == null)
    {
      db = new DBConnection();
    }
    if(theConnection == null)
    {
      theConnection = db.getConnection();
    }
    Statement statement = theConnection.createStatement();
    long sum = 0;
    long anzahl = 0;
    ResultSet result = null;
    if(messpunkt == null)
    {
      result = statement.executeQuery("SELECT value FROM " + table);
    }
    else
    {
      result = statement.executeQuery("SELECT value FROM " + table
                        + " WHERE mfwpoint='" + messpunkt + "'");
    }
    while(result.next())
    {
      sum += result.getLong("value");
      anzahl++;
    }
    result.close();
    statement.close();
    return (double)sum/(double)anzahl;
  }
  catch(Exception e)
  {
    System.out.println(e.getMessage());
  }
  return -1.0;
}
public  double calculateRange(String table, String messpunkt)
{
  if(messpunkt == null)
    return getMax(table, messpunkt) - getMin(table, messpunkt);
  else return getMax(table, messpunkt) - getMin(table, messpunkt);
}
public  double getMax(String table, String messpunkt)
{
  try
  {
    if(db == null)
    {
      db = new DBConnection();
```

```
    }
    if(theConnection == null)
    {
      theConnection = db.getConnection();
    }
    Statement statement = theConnection.createStatement();
    ResultSet result = null;
    if(messpunkt == null)
    {
      result = statement.executeQuery("SELECT MAX(VALUE) FROM " + table);
    }
    else
    {
      result = statement.executeQuery("SELECT MAX(VALUE) FROM " + table
                        + " WHERE mfwpoint='" + messpunkt + "'");
    }
    result.next();
    double max = result.getDouble("MAX(VALUE)");
    result.close();
    statement.close();
    return max;
  }
  catch (SQLException e)
  {
    e.printStackTrace();
  }
  catch (InstantiationException e)
  {
    e.printStackTrace();
  }
    catch (ClassNotFoundException e)
  {
    e.printStackTrace();
  }
  catch(IllegalAccessException e)
  {
    e.printStackTrace();
  }
  return -1;
}
public  double getMin(String table, String messpunkt)
{
  try
  {
    if(db == null)
    {
      db = new DBConnection();
    }
    if(theConnection == null)
    {
      theConnection = db.getConnection();
    }
    Statement statement = theConnection.createStatement();
    ResultSet result = null;
```

```
      if(messpunkt == null)
      {
        result = statement.executeQuery("SELECT MIN(VALUE) FROM " + table);
      }
      else
      {
        result = statement.executeQuery("SELECT MIN(VALUE) FROM " + table
                        + " WHERE mfwpoint='" + messpunkt + "'");
      }
      result.next();
      double min = result.getDouble("MIN(VALUE)");
      result.close();
      statement.close();
      return min;
    }
    catch (SQLException e)
    {
      e.printStackTrace();
    }
    catch (InstantiationException e)
    {
      e.printStackTrace();
    }
    catch (ClassNotFoundException e)
    {
      e.printStackTrace();
    }
    catch(IllegalAccessException e)
    {
      e.printStackTrace();
    }
    return -1;
  }
  public  double calculateStandardabweichung(String table, String messpunkt)
  {
    return Math.sqrt(calculateVarianz(table, messpunkt));
  }
  public  double calculateVarianz(String table, String messpunkt)
  {
    try
    {
      if(db == null)
      {
        db = new DBConnection();
      }
      if(theConnection == null)
      {
        theConnection = db.getConnection();
      }
      Statement statement = theConnection.createStatement();
      long anzahl = 0;
      double mittelwert = calculateMittelwert(table, messpunkt);
      ResultSet result = null;
      if(messpunkt == null)
```

```
    {
      result = statement.executeQuery("SELECT value FROM " + table);
    }
    else
    {
      result = statement.executeQuery("SELECT value FROM " + table
                        + " WHERE mfwpoint='" + messpunkt + "'");
    }
    double quadrat = 0.0;
    while(result.next())
    {
      anzahl++;
      quadrat += Math.pow((double)result.getLong("value") - mittelwert, 2);
    }
    double varianz = quadrat/(double)(anzahl-1);
    result.close();
    statement.close();
    return varianz;
  }
  catch(Exception e)
  {
    System.out.println(e.getMessage());
  }
  return -1;
}
public  double calculateInterquartilsbstand(String table, String messpunkt)
{
  double quantil75 = calculateQuantile(0.75, table, messpunkt);
  double quantil25 = calculateQuantile(0.25, table, messpunkt);
  if(quantil75 <= 0 || quantil25 <= 0)
    return  -1.0; //Zu wenig Daten
  else return quantil75 - quantil25;
}
public  double calculateMedian(String table, String messpunkt)
{
  try
  {
    if(db == null)
    {
      db = new DBConnection();
    }
    if(theConnection == null)
    {
      theConnection = db.getConnection();
    }
    Statement statement = theConnection.createStatement();
    int anzahl = 0;
    ResultSet result = null;
    if(messpunkt == null)
    {
      result = statement.executeQuery("SELECT value FROM " + table
                        + " ORDER BY value");
    }
    else
```

```
  {
    result = statement.executeQuery("SELECT value FROM " + table
                      + " WHERE mfwpoint='" + messpunkt + "' ORDER BY value");
  }
  while (result.next())
  {
    anzahl++;
  }
  if(anzahl > 1)
  {
    result.beforeFirst();
    int count = 0;
    if(anzahl % 2 != 0)
    {
      while(result.next())
      {
        count++;
        if (count == (anzahl+1)/2)
        {
          break;
        }
      }
      double value = (double)result.getLong("value");
      result.close();
      statement.close();
      return value;
    }
    else
    {
      while(result.next())
      {
        count++;
        if (count == (anzahl)/2)
        {
          break;
        }
      }
      long value = result.getLong("value");
      result.next();
      long value2 = result.getLong("value");
      result.close();
      statement.close();
      return ((double)value + (double)value2) / 2.0;
    }
  }
  else
  {
    //Zu wenig Daten
    return -1.0;
  }
}
catch (SQLException e)
{
  e.printStackTrace();
```

```
    return -1.0;
  }
  catch (IllegalAccessException e)
  {
    e.printStackTrace();
    return -1.0;
  }
  catch (ClassNotFoundException e)
  {
    e.printStackTrace();
    return -1.0;
  }
  catch(InstantiationException e)
  {
    e.printStackTrace();
    return -1.0;
  }
}
/*
* Diese Funktion berechnet aus uebergebenen Daten ein Perzentil
* @param p Die gewuenschte Prozentzahl zw. 0 und 1
* @param table Name der Tabelle mit den Daten
* @param Eine Verbindung zur Datenbank, falls null wird sie erstellt
* @return double Das Pezentil
*/
public  double calculateQuantile(double p, String table, String messpunkt)
{
  try
  {
    if(db == null)
    {
      db = new DBConnection();
    }
    if(theConnection == null)
    {
      theConnection = db.getConnection();
    }
    Statement statement = theConnection.createStatement();
    int anzahl = 0;
    ResultSet result = null;
    if(messpunkt == null)
    {
      result = statement.executeQuery("SELECT value FROM " + table
                        + " ORDER BY value");
    }
    else
    {
      result = statement.executeQuery("SELECT value FROM " + table
                        + " WHERE mfwpoint='" + messpunkt + "' ORDER BY value");
    }
    while (result.next())
    {
      anzahl++;
    }
```

```
        if(anzahl > 10)
        {
          double perzentil = (double)anzahl*p;
          int ceil = (int)Math.ceil(perzentil);
          result.beforeFirst();
          int count = 0;
          while(result.next())
          {
            count++;
            if (count == ceil)
            {
              break;
            }
          }
          long value = result.getLong("value");
          result.next();
          long value2 = result.getLong("value");
          if(perzentil%1 == 0)//n*p gerade
          {
            result.close();
            statement.close();
            return (value+value2)/2;
          }
          else
          {
            result.close();
            statement.close();
            return value;
          }
        }
        else
        {
          return -1.0;
        }
      }
      catch (Exception e)
      {
        e.printStackTrace();
        return -1.0;
      }
    }
}
```

7.1.3 Das Paket Charts

JdbcChartListPanel

```
package de.christianbell.apmfw.jdragicdatamining.charts;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.GridBagLayout;
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Iterator;
import java.util.Vector;
import javax.swing.BorderFactory;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.JSplitPane;
import de.christianbell.apmfw.jdragicdatamining.gui.ActionListenerClass;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
import de.christianbell.apmfw.jdragicdatamining.threads.AlleMessungenAuswertenThread;
import de.christianbell.apmfw.jdragicdatamining.threads.MessungThread;
import java.awt.Dimension;
import java.io.File;
import javax.swing.JComboBox;
/**
 * Die Klasse JdbcChartListPanel erstellt eine Flaeche,
 *die im linken Bereich ein Diagramm und im rechten
 * Bereich eine Auswahlliste anzeigt. Durch betaetigen
 * des Update-Knopfs wird der Verlauf der in der Auswahlliste
 * selektierten Variable angezeigt.
 * @author Alexander Hergenroeder, modified by Christian Bell
 */
public class JdbcChartListPanel extends JPanel implements ActionListener
{
  private static final long serialVersionUID = 6247355791357901960L;
  private static DBConnection db = null;
  public static PropertiesSingleton ps = null;
  /** Diagramm */
  public static JdbcChartPanel chartPanel;
  /** Auswahlliste */
  public static JList list;
  /** Update-Button */
  private JButton button;
  /** Kommunikation mit der Datenbank */
  private static Connection con;
  private static JButton pngSpeichernButton = null;
  public static JComboBox tableComboBox = null;
  private static JButton pngAlleSpeichernButton = null;
  private static ActionListenerClass anActionListener = new ActionListenerClass();
  private static JButton excelExportButton = null;
  private static JButton csvExportButton = null;
  /**
  *      Erzeugt ein neues JdbcChartListPanel-Objekt
  */
  public JdbcChartListPanel()
```

```
{
  super(new BorderLayout());
  initialize();
  try
  {
    ps = PropertiesSingleton.getInstance();
  }
  catch(Exception e)
  {
    System.out.println(e.getMessage());
  }
  //Verbindung mit der Datenbank herstellen
  try
  {
    if(db == null) {
       db = new DBConnection();
    }
    con = db.getConnection();
  }
  catch (Exception e)
  {
    JOptionPane.showMessageDialog(null, "Fehler beim Verbinden mit der Datenbank",
                                  "", JOptionPane.ERROR_MESSAGE);
    e.printStackTrace();
  }
  //Neuer Datensatz zur Aufnahme der in der Datenbank gespeicherten Daten erzeugen
  JDBCMeanAndStandardDeviationXYDataset dataset =
                    new JDBCMeanAndStandardDeviationXYDataset(con);
  //Auswahlliste erzeugen und mit Werten aus der Datenbank fuellen
  list = new JList();
  try
  {
    getTableComboBox();
    fillList(tableComboBox.getSelectedItem().toString());
  }
  catch (SQLException e)
  {
    JOptionPane.showMessageDialog( null, "Fehler beim Lesen der Messpunknamen
                        aus der Datenbank", "", JOptionPane.ERROR_MESSAGE);
    e.printStackTrace();
    return;
  }
  JScrollPane listPane = new JScrollPane(list,
                 JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
  JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED );
  listPane.setBorder(BorderFactory.createLineBorder(Color.lightGray, 2));
  //Chart-Panel erzeugen
  chartPanel = new JdbcChartPanel(dataset,
               tableComboBox.getSelectedItem().toString());
  JPanel buttonPanel = new JPanel(new GridBagLayout());
  buttonPanel.setBackground(Color.white);
  buttonPanel.add(getUpdateButton());
  buttonPanel.add(getTableComboBox());
  buttonPanel.add(getPngSpeichernButton());
```

```
  buttonPanel.add(getPngAlleSpeichernButton());
  buttonPanel.add(getExcelExportButton());
  buttonPanel.add(getCsvExportButton());
  JPanel panel = new JPanel(new BorderLayout());
  panel.add(buttonPanel, BorderLayout.SOUTH);
  panel.add(chartPanel, BorderLayout.CENTER);
  panel.setBorder(BorderFactory.createLineBorder(Color.lightGray, 2));
  JSplitPane horizSplitter = new JSplitPane(JSplitPane.HORIZONTAL_SPLIT,
                             true, panel, listPane);
  horizSplitter.setDividerLocation(2.0/3.0);
  horizSplitter.setOneTouchExpandable(true);
  add(horizSplitter);
}
/**
* This method initializes this
*/
private void initialize() {
  this.setSize(new Dimension(212, 117));
}
/**
*      Erzeugt den Update-Button
*      @return Update-Button
*/
public JButton getUpdateButton()
{
  if(button == null)
  {
    button = new JButton("Update");
    button.setActionCommand("update");
    button.addActionListener(this);
  }
  return button;
}
public void actionPerformed(ActionEvent e)
{
//Pruefen, ob die Verbindung zur Datenbank steht,
//falls nicht -> Verbindung aufbauen
if(e.getActionCommand().equals(getUpdateButton().getActionCommand()))
{
  if(con == null)
  {
    try
    {
      if(db == null)
      {
        db = new DBConnection();
      }
      con = db.getConnection();
    }
    catch (Exception e1)
    {
    JOptionPane.showMessageDialog( this, "Fehler beim Verbinden mit der Datenbank",
                                   "", JOptionPane.ERROR_MESSAGE);
    e1.printStackTrace();
```

```
    return;
  }
}
//Pruefen, ob die Auswahlliste mindestens ein Element enthaelt,
//falls nicht -> Auswahlliste fuellen
if(list.getModel().getSize() == 0)
{
  try
  {
    fillList(tableComboBox.getSelectedItem().toString());
  }
  catch (SQLException e2)
  {
    JOptionPane.showMessageDialog( this, "Fehler beim Lesen der Messpunknamen
                        aus der Datenbank", "", JOptionPane.ERROR_MESSAGE);
    e2.printStackTrace();
    return;
  }
}
try
{
  JDBCMeanAndStandardDeviationXYDataset dataset =
       (JDBCMeanAndStandardDeviationXYDataset)chartPanel.getDataset();
  if(tableComboBox.getSelectedItem().toString().contains("results"))
  {
    Statement statement = con.createStatement();
    ResultSet result = statement.executeQuery("SELECT MIN(tstamp) FROM "
                      + tableComboBox.getSelectedItem().toString()
                      + " where mfwPoint='" + (String) list.getSelectedValue()
                      + "'");
    result.next();
    long min = result.getLong("MIN(tstamp)");
    if(min != 0)
    {
      statement.executeUpdate("UPDATE " + tableComboBox.getSelectedItem().toString()
                            + " SET tstamp=tstamp-" + min + " where mfwPoint='"
                            + (String) list.getSelectedValue() +"'");
    }
    dataset.executeQuery(con, "SELECT tStamp, value FROM "
                           + tableComboBox.getSelectedItem().toString()
                           + " where mfwPoint='" + (String) list.getSelectedValue()
                           + "' ORDER BY tstamp");
  }
  else if(tableComboBox.getSelectedItem().toString().contains("iostat")
          || tableComboBox.getSelectedItem().toString().contains("sarlog"))
  {
    Statement statement = con.createStatement();
    statement.executeUpdate("UPDATE " + tableComboBox.getSelectedItem().toString()
                           + " SET value = REPLACE(value,',','.')");
    ResultSet result = statement.executeQuery("SELECT MIN(tstamp) FROM "
                           + tableComboBox.getSelectedItem().toString());
    result.next();
    long min = result.getLong("MIN(tstamp)");
    if(min != 0)
```

```
      {
        statement.executeUpdate("UPDATE " + tableComboBox.getSelectedItem().toString()
                                + " SET tstamp=tstamp-" + min);
      }
      dataset.executeQuery(con, "SELECT tStamp, CAST(value AS DECIMAL(5,2))
                                 AS value FROM "
                                 + tableComboBox.getSelectedItem().toString()
                                 + " ORDER BY tstamp");
    }
    else
    {
      Statement statement = con.createStatement();
      ResultSet result = statement.executeQuery("SELECT MIN(tstamp) FROM "
                              + tableComboBox.getSelectedItem().toString());
      result.next();
      long min = result.getLong("MIN(tstamp)");
      if(min != 0)
      {
        statement.executeUpdate("UPDATE " + tableComboBox.getSelectedItem().toString()
                                + " SET tstamp=tstamp-" + min);
      }
      dataset.executeQuery(con, "SELECT tStamp, value FROM "
                                + tableComboBox.getSelectedItem().toString()
                                + " ORDER BY tstamp");
    }
    chartPanel = new JdbcChartPanel(dataset,
                    tableComboBox.getSelectedItem().toString());
    chartPanel.repaint();
  }
  catch (Exception e1)
  {
    e1.printStackTrace();
    JOptionPane.showMessageDialog(this, "Fehler beim Datenbankzugriff.",
                                  "", JOptionPane.ERROR_MESSAGE);
  }
}}
/**
* fuellt die Auswahlliste
* @throws SQLException, falls das Abrufen der Daten aus der Datenbank fehlschlaegt
*/
public static void fillList(String table) throws SQLException
{
  if(list != null && con != null)
  {
    if(db == null)
    {
      db = new DBConnection();
    }
    try{
      list.setListData(db.getMesspunktNamen(con, table));
      list.setSelectedIndex(0);
    }
    catch (Exception e) {
    }
```

```
    }
   }

/**
* This method initializes pngSpeichernButton
* @return javax.swing.JButton
*/
public static JButton getPngSpeichernButton()
{
   if (pngSpeichernButton == null)
   {
     pngSpeichernButton = new JButton("Tabelle auswerten");
     pngSpeichernButton.setToolTipText("Wertet die aktuelle Tabelle aus!");
     pngSpeichernButton.setActionCommand("pngSpeichern");
     pngSpeichernButton.addActionListener(anActionListener);
   }
   return pngSpeichernButton;
}
/**
* This method initializes tableComboBox
* @return javax.swing.JComboBox
*/
public static JComboBox getTableComboBox()
{
   if (tableComboBox == null)
   {
     tableComboBox = new JComboBox();
     Vector <String> v = new Vector <String>();
     if(con == null)
     {
       try
       {
         if(db == null)
         {
           db = new DBConnection();
         }
         con = db.getConnection();
       }
       catch (Exception e1)
       {
         JOptionPane.showMessageDialog( null, "Fehler beim Verbinden mit der Datenbank",
                              "", JOptionPane.ERROR_MESSAGE);
         e1.printStackTrace();
       }
     }
     try
     {
       v = db.getTables(con);
     }
     catch (SQLException e)
     {
       e.printStackTrace();
     }
     Iterator itr = v.iterator();
```

```
      tableComboBox.removeAllItems();
      while(itr.hasNext())
      {
        tableComboBox.addItem(itr.next());
      }
      tableComboBox.addActionListener(anActionListener);
      tableComboBox.setActionCommand("changeTable");
    }
    return tableComboBox;
}
/**
* This method initializes pngAlleSpeichernButton
* @return javax.swing.JButton
*/
public static JButton getPngAlleSpeichernButton() {
    if (pngAlleSpeichernButton == null) {
      pngAlleSpeichernButton = new JButton("Alle Tabellen auswerten");
      pngAlleSpeichernButton.setToolTipText("Wertet alle Tabellen aus,
                  die zum eingegebenen Messungsnamen gehoeren!");
      pngAlleSpeichernButton.setActionCommand("pngAlleSpeichern");
      pngAlleSpeichernButton.addActionListener(anActionListener);
    }
    return pngAlleSpeichernButton;
}

/**
* This method initializes excelExportButton
* @return javax.swing.JButton
*/
public static JButton getExcelExportButton() {
    if (excelExportButton == null) {
      excelExportButton = new JButton("Excelexport");
      excelExportButton.setToolTipText("Schreibt die aktuelle Tabelle in
                   eine Excel-Datei!");
      excelExportButton.setActionCommand("excelExport");
      excelExportButton.addActionListener(anActionListener);
    }
    return excelExportButton;
}
/**
* This method initializes csvExportButton
* @return javax.swing.JButton
*/
public static JButton getCsvExportButton() {
    if (csvExportButton == null) {
      csvExportButton = new JButton("CSV-Export");
      csvExportButton.setToolTipText("Schreibt die aktuelle Tabelle in eine CSV-Datei!");
      csvExportButton.setActionCommand("csvExport");
      csvExportButton.addActionListener(anActionListener);
    }
    return csvExportButton;
}
public static void main(String[] args)
{
```

```
    JFrame demo;
    demo = new JFrame();
    demo.setLayout(new BorderLayout());
    demo.add(new JdbcChartListPanel());
    demo.pack();
    demo.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    demo.setVisible(true);
  }
  }
```

JdbcChartPanel

```
package de.christianbell.apmfw.jdragicdatamining.charts;
import java.awt.BasicStroke;
import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.GridLayout;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.Vector;
import javax.swing.BorderFactory;
import javax.swing.JCheckBox;
import javax.swing.JComboBox;
import javax.swing.JLabel;
import javax.swing.JMenuItem;
import javax.swing.JOptionPane;
import javax.swing.JPanel;
import javax.swing.JPopupMenu;
import layout.TableLayout;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartPanel;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.DeviationRenderer;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.XYDataset;
import org.jfree.ui.RectangleInsets;
/** Erweitert die Klasse ChartPanel um folgende Funktionalitaeten:
 *  1. Moeglichkeit unterschiedliche Renderer fuer die Anzeige
 * zu benutzen (Interpolation, Deviation)
 *  2. Ein- und Ausschaltbarkeit von Serienplots sowie Plotelementen
 */
public class JdbcChartPanel extends ChartPanel
{
  private static final long serialVersionUID = -456724516646688126L;
  /** Daten */
  private XYDataset dataset;
  /** Renderer */
  private DeviationRenderer deviationRenderer;
  private InterpolatedDeviationRenderer smoothDeviationRenderer;
  private XYLineAndShapeRenderer xyLineAndShapeRenderer;
  private InterpolatedXYLineAndShapeRenderer smoothXYLineAndShapeRenderer;
  private ChartSettingsPanel chartSettingsPanel;
```

```
public JdbcChartPanel(XYDataset dataset, String tablename)
{
  super(createChart(dataset, tablename), true, true, false, false, true);
  this.dataset = dataset;
  //wird benutzt, um die Standardabweichung als Schattierung
  //um den Mittelwert herum anzuzeigen
  deviationRenderer = new DeviationRenderer(true, true);
  createRenderer(deviationRenderer);
  //interpolierte Version vom deviationRenderer
  smoothDeviationRenderer = new InterpolatedDeviationRenderer(true, true);
  createRenderer(smoothDeviationRenderer);
  //Datenpunkte (x,y) werden durch Linien verbunden
  xyLineAndShapeRenderer = new XYLineAndShapeRenderer();
  createRenderer(xyLineAndShapeRenderer);
  //interpolierte Version vom xyLineAndShapeRenderer
  smoothXYLineAndShapeRenderer = new InterpolatedXYLineAndShapeRenderer();
  createRenderer(smoothXYLineAndShapeRenderer);
  getChart().getXYPlot().setRenderer(0,xyLineAndShapeRenderer);
  setLayout(new BorderLayout());
  //das default PopUp-Menue um den Eintrag "Einstellungen..." erweitert
  JPopupMenu menu = this.getPopupMenu();
  menu.addSeparator();
  JMenuItem settings = new JMenuItem("Einstellungen...");
  settings.addActionListener(this);
  settings.setActionCommand("settings");
  menu.add(settings);
  this.setPopupMenu(menu);
  chartSettingsPanel = new ChartSettingsPanel(getChart());
  this.setBounds(0, 0, 500, 400);
  this.validate();
}

/**
* Gibt den Dataset zurueck
* @return dataset
*/
public XYDataset getDataset()
{
  return dataset;
}
/**
* Erzeugt das Diagramm
* @param dataset
* @return chart
*/
public static JFreeChart chart;
public static JFreeChart createChart(XYDataset dataset, String tablename)
{
  if(tablename.contains("iostat"))
    chart = ChartFactory.createXYLineChart("IOStat", "Zeitpunkt (ms)",
            "rkB/s", dataset, PlotOrientation.VERTICAL, true, true, false);
  else if(tablename.contains("sarlog"))
    chart = ChartFactory.createXYLineChart("SarLog", "Zeitpunkt (ms)",
            "Systemauslastung (%)", dataset, PlotOrientation.VERTICAL,
```

```
            true, true, false);
  else if(tablename.contains("aktreq"))
    chart = ChartFactory.createXYLineChart("Aktive Requests", "Zeitpunkt (ms)",
            "Aktive Requests", dataset, PlotOrientation.VERTICAL, true, true, false);
  else if(tablename.contains("haeufigkeitresp"))
    chart = ChartFactory.createXYLineChart("Aktive Requests", "Zeitpunkt (ms)",
            "Haeufigkeit", dataset, PlotOrientation.VERTICAL, true, true, false);
  else if(tablename.contains("jtl"))
    chart = ChartFactory.createXYLineChart("Requestantwortzeit", "Zeitpunkt (ms)",
            "Antwortzeit (ms)", dataset, PlotOrientation.VERTICAL, true, true, false);
  else if(tablename.contains("netstat"))
    chart = ChartFactory.createXYLineChart("NetStat", "Zeitpunkt (ms)", "RX Packets",
            dataset, PlotOrientation.VERTICAL, true, true, false);
  else if(tablename.contains("vmstat"))
    chart = ChartFactory.createXYLineChart("VMStat", "Zeitpunkt (ms)",
            "Freier Speicher (KB)", dataset, PlotOrientation.VERTICAL,
            true, true, false);
  else //Results
    chart = ChartFactory.createXYLineChart("Messpunkt", "Zeitpunkt (MikroSek)",
            "Dauer in ns", dataset, PlotOrientation.VERTICAL, true, true, false);
    chart.setBackgroundPaint(Color.white);
    XYPlot plot = (XYPlot) chart.getPlot();
    plot.setBackgroundPaint(Color.lightGray);
    plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 5.0));
    plot.setDomainGridlinePaint(Color.white);
    plot.setRangeGridlinePaint(Color.white);
    return chart;
  }

/**
* Setzt Farben und Liniendicke vom Renderer
* @param renderer
*/
private static void  createRenderer(XYLineAndShapeRenderer renderer)
{
      renderer.setSeriesStroke(0, new BasicStroke(3.0f));
      renderer.setSeriesStroke(1, new BasicStroke(3.0f));
      renderer.setSeriesStroke(2, new BasicStroke(3.0f));
      renderer.setSeriesPaint(0, Color.red);
      renderer.setSeriesPaint(1, Color.blue);
      renderer.setSeriesPaint(2, Color.green);
}

public void actionPerformed(ActionEvent e)
{
  super.actionPerformed(e);
  if(e.getActionCommand().equals("settings"))
  {
    Object[] options = {"OK"};
    JOptionPane.showOptionDialog(this,chartSettingsPanel,"",
                                 JOptionPane.DEFAULT_OPTION,
                            JOptionPane.PLAIN_MESSAGE,null,
                                 options, options[0]);
  }
```

```
  }
/** Die Klasse stellt das Dialogfenster dar,
* das sich unter dem Menueeintrag "Einstellungen" vebirgt.
*/
private class ChartSettingsPanel extends JPanel implements ActionListener
{
   private static final long serialVersionUID = 6925257471449886091L;
   JFreeChart chart;
   JCheckBox box4;
   JComboBox cb;
   public ChartSettingsPanel(JFreeChart chart)
   {
     super();
     this.chart = chart;
     double size[][] = {{TableLayout.FILL}, {0.2, 0.5, 0.3}};
     setLayout(new TableLayout(size));
     //Interpolation
     JCheckBox box1 = new JCheckBox("Interpolieren");
     box1.setActionCommand("interpolation");
     box1.addActionListener(this);
     box1.setSelected(false);
     add(box1, "0, 0");
     //Seriensichtbarkeit
     JCheckBox box2 = new JCheckBox("Value");
     box2.setActionCommand("value");
     box2.addActionListener(this);
     box2.setSelected(true);
     JCheckBox box3 = new JCheckBox("Mittelwert");
     box3.setActionCommand("mean value");
     box3.addActionListener(this);
     box3.setSelected(true);
     box4 = new JCheckBox("mit Standartabweichung");
     box4.setActionCommand("standard deviation");
     box4.addActionListener(this);
     box4.setSelected(false);
     JPanel p1 = new JPanel(new GridLayout(1,2));
     p1.add(box3);
     p1.add(box4);
     JCheckBox box5 = new JCheckBox("Gleitender Mittelwert");
     box5.setActionCommand("moving average");
     box5.addActionListener(this);
     box5.setSelected(true);
     Vector <Integer>  vector = new Vector <Integer>();
     vector.add(new Integer(3));
     vector.add(new Integer(5));
     vector.add(new Integer(7));
     vector.add(new Integer(9));
     cb = new JComboBox(vector);
     cb.setActionCommand("step");
     cb.addActionListener(this);
     double size1[][] = {{0.6, 0.4}, {TableLayout.FILL}};
     JPanel p2 = new JPanel(new TableLayout(size1));
     p2.add(new JLabel("Schritt"),"0,0");
     p2.add(cb,"1,0");
```

```
  JPanel p3 = new JPanel(new GridLayout(1,2));
  p3.add(box5);
  p3.add(p2);
  JPanel panel1 = new JPanel(new GridLayout(3,1));
  panel1.setBorder(BorderFactory.createTitledBorder(
                  BorderFactory.createLineBorder(Color.lightGray),
                  "Seriensichtbarkeit"));
  panel1.add(box2);
  panel1.add(p1);
  panel1.add(p3);
  add(panel1, "0,1");
  //Punkten- und Liniensichtbarkeit
  JCheckBox box6 = new JCheckBox("Punkte");
  box6.setActionCommand("shapes");
  box6.addActionListener(this);
  box6.setSelected(true);
  JCheckBox box7 = new JCheckBox("Linien");
  box7.setActionCommand("lines");
  box7.addActionListener(this);
  box7.setSelected(true);
  JPanel panel2 = new JPanel(new GridLayout(1,2));
  panel2.setBorder(BorderFactory.createTitledBorder(
                  BorderFactory.createLineBorder(Color.lightGray),
                  "Sichtbarkeit von Kurvenkomponenten"));
  panel2.add(box6);
  panel2.add(box7);
  add(panel2, "0,2");
}

public void actionPerformed(ActionEvent e)
{
  int series = -1;
  if (e.getActionCommand().equals("value"))
     series = 0;
  else if (e.getActionCommand().equals("mean value"))
     series = 1;
  else if (e.getActionCommand().equals("moving average"))
     series = 2;
  //Punkten sichtbar/unsichtbar machen
  if (e.getActionCommand().equals("shapes"))
  {
    boolean visible = ((XYLineAndShapeRenderer)
          chart.getXYPlot().getRenderer()).getBaseShapesVisible();
    deviationRenderer.setBaseShapesVisible(!visible);
    smoothDeviationRenderer.setBaseShapesVisible(!visible);
    xyLineAndShapeRenderer.setBaseShapesVisible(!visible);
    smoothXYLineAndShapeRenderer.setBaseShapesVisible(!visible);
  }
  //Linien sichtbar/unsichtbar machen
  else if (e.getActionCommand().equals("lines"))
 {
   boolean visible = ((XYLineAndShapeRenderer)
           chart.getXYPlot().getRenderer()).getBaseLinesVisible();
   deviationRenderer.setBaseLinesVisible(!visible);
```

```
      smoothDeviationRenderer.setBaseLinesVisible(!visible);
      xyLineAndShapeRenderer.setBaseLinesVisible(!visible);
      smoothXYLineAndShapeRenderer.setBaseLinesVisible(!visible);
    }
    //Interpolation ein-/ausschalten
    else if (e.getActionCommand().equals("interpolation"))
    {
      if(chart.getXYPlot().getRenderer() instanceof InterpolatedDeviationRenderer)
        chart.getXYPlot().setRenderer(deviationRenderer);
      else if(chart.getXYPlot().getRenderer() instanceof DeviationRenderer)
        chart.getXYPlot().setRenderer(smoothDeviationRenderer);
      else if(chart.getXYPlot().getRenderer() instanceof
        chart.getXYPlot().setRenderer(xyLineAndShapeRenderer);
      else if(chart.getXYPlot().getRenderer() instanceof XYLineAndShapeRenderer)
        chart.getXYPlot().setRenderer(smoothXYLineAndShapeRenderer);
    }
    //Standartabweichung zeigen/verbergen
    else if (e.getActionCommand().equals("standard deviation"))
    {
      if( box4.isSelected())
      {
        if(chart.getXYPlot().getRenderer() instanceof
                        InterpolatedXYLineAndShapeRenderer)
          chart.getXYPlot().setRenderer(smoothDeviationRenderer);
        else if(chart.getXYPlot().getRenderer() instanceof XYLineAndShapeRenderer)
          chart.getXYPlot().setRenderer(deviationRenderer);
      }
      else
      {
       if(chart.getXYPlot().getRenderer() instanceof InterpolatedDeviationRenderer)
         chart.getXYPlot().setRenderer(smoothXYLineAndShapeRenderer);
       else if(chart.getXYPlot().getRenderer() instanceof DeviationRenderer)
         chart.getXYPlot().setRenderer(xyLineAndShapeRenderer);
      }
    }
    //Anzahl der Punkte ueber die gleitender Mittelwert gebildet wird aendern
    else if (e.getActionCommand().equals("step"))
    {
      if(dataset instanceof JDBCMeanAndStandardDeviationXYDataset)
        ((JDBCMeanAndStandardDeviationXYDataset)dataset).updateMovingAverage(
                  (Integer)cb.getSelectedItem());
    }
    //Series zeigen/verbergen
    if (series >= 0)
    {
      boolean visible = chart.getXYPlot().getRenderer().getItemVisible(series, 0);
      deviationRenderer.setSeriesVisible(series, new Boolean(!visible));
      smoothDeviationRenderer.setSeriesVisible(series, new Boolean(!visible));
      xyLineAndShapeRenderer.setSeriesVisible(series, new Boolean(!visible));
      smoothXYLineAndShapeRenderer.setSeriesVisible(series, new Boolean(!visible));
    }
   }
  }
}
```

JdbcMeanAndStandardDeviationXYDataset

```
package de.christianbell.apmfw.jdragicdatamining.charts;
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;
import java.sql.Types;
import java.util.ArrayList;
import org.jfree.data.statistics.Statistics;
import org.jfree.data.xy.AbstractIntervalXYDataset;
import org.jfree.data.xy.XYDataset;
/**
 * This class provides an implementation over a database
 * JDBC result set.  The dataset is populated via a call to executeQuery with
 * the string sql query.  The sql query must return two columns.
 * The first column will be the x-axis and the second column y-axis values.
 * The database connection is read-only.
 */
public class JDBCMeanAndStandardDeviationXYDataset extends AbstractIntervalXYDataset
{
    private static final long serialVersionUID = -2193438017373790725L;
    /** The database connection. */
    private transient Connection connection;
    /** Data. */
    private ArrayList<ClearableYIntervalSeries> data;
    /**
     * Creates a new JDBCXYDataset (initially empty) with no database
     * connection.
     */
    private JDBCMeanAndStandardDeviationXYDataset()
    {
        this.data = new ArrayList<ClearableYIntervalSeries>();
    }
    /**
     * Creates a new dataset (initially empty) and establishes a new database
     * connection.
     * @param  url  URL of the database connection.
     * @param  driverName  the database driver class name.
     * @param  user  the database user.
     * @param  password  the database user's password.
     * @throws ClassNotFoundException if the driver cannot be found.
     * @throws SQLException if there is a problem connecting to the database.
     */
    public JDBCMeanAndStandardDeviationXYDataset(String url,
          String driverName, String user, String password)
          throws SQLException, ClassNotFoundException
    {
        this();
        Class.forName(driverName);
        this.connection = DriverManager.getConnection(url, user, password);
    }
    /**
```

```
 * Creates a new dataset (initially empty) using the specified database
 * connection.
 * @param  con  the database connection.
 */
public JDBCMeanAndStandardDeviationXYDataset(Connection con)
{
    this();
    this.connection = con;
}
/**
 * ExecuteQuery will attempt execute the query passed to it against the
 * existing database connection.  If no connection exists then no action
 * is taken.
 * The results from the query are extracted and cached locally, thus
 * applying an upper limit on how many rows can be retrieved successfully.
 * @param  query  the query to be executed.
 * @throws SQLException if there is a problem executing the query.
 */
public void executeQuery(String query) throws SQLException
{
    executeQuery(this.connection, query);
}
/**
 * ExecuteQuery will attempt execute the query passed to it against the
 * provided database connection.  If connection is null then no action is
 * taken.
 * The results from the query are extracted and cached locally, thus
 * applying an upper limit on how many rows can be retrieved successfully.
 * @param  query  the query to be executed.
 * @param  con  the connection the query is to be executed against.
 * @throws SQLException if there is a problem executing the query.
 */
public void executeQuery(Connection con, String query) throws SQLException
{
    if (con == null)
        throw new SQLException("There is no database to execute the query.");
    ResultSet resultSet = null;
    Statement statement = null;
    try
    {
        statement = con.createStatement();
        resultSet = statement.executeQuery(query);
        ResultSetMetaData metaData = resultSet.getMetaData();
        int numberOfColumns = metaData.getColumnCount();
        if(numberOfColumns != 2)
            throw new SQLException("wrong number of columns
                    where generated by query.");
        for (int column = 0; column < numberOfColumns; column++)
        {
            try
            {
                int type = metaData.getColumnType(column + 1);
                switch (type)
                {
```

```
                case Types.NUMERIC:
                case Types.REAL:
                case Types.INTEGER:
                case Types.DOUBLE:
                case Types.FLOAT:
                case Types.DECIMAL:
                case Types.BIT:
                case Types.BIGINT:
                case Types.SMALLINT:
                        break;
                default :
                   throw new SQLException("Not enough valid
                     columns where generated by query.");
            }
        }
        catch (SQLException e)
        {
            throw e;
        }
    }
    // Might need to add, to free memory from any previous result sets
    if (this.data != null)
    {
        for (int column = 0; column < this.data.size(); column++)
        {
                ClearableYIntervalSeries  series = this.data.get(column);
                series.clear();
        }
        this.data.clear();
    }
    int count = 0;
    // Create value series
    //CJb3LL - autSort auf false, MySql ist schneller
    ClearableYIntervalSeries value =
       new ClearableYIntervalSeries ("Value", false, true);

    //CJb3LL - Nur beim letzten Punkt sollen die Listener
    //über ein SeriesChangedEvent benachrichtigt werden
    while (resultSet.next())
    {
        count++;
    }
    resultSet.first();
    int i = 1;
    while (i < count)
    {
        Double x = resultSet.getDouble(1);
        Double y = resultSet.getDouble(2);
        resultSet.next();
        value.add(x, y, y, y, false );
        i++;
    }
    if (count > 0)
    {
```

```
            Double x = resultSet.getDouble(1);
            Double y = resultSet.getDouble(2);
            value.add(x, y, y, y, true );
        }
        this.data.add(value);
        //Create mean value series
        ClearableYIntervalSeries meanValue =
          new ClearableYIntervalSeries ("Mean value", true, true);
        double mean = calculateMean();
        double stdDev = calculateStdDev();
        if(resultSet.first())
        {
            meanValue.add(((Double)value.getX(0)).doubleValue(),
                 mean, mean-stdDev, mean+stdDev);
           meanValue.add(((Double)value.getX(
               value.getItemCount()-1)).doubleValue(),
                 mean, mean-stdDev, mean+stdDev);
            this.data.add(meanValue);
            //Create moving average series
            ClearableYIntervalSeries movingAverage =
              new ClearableYIntervalSeries ("Moving average", true, true);
            this.data.add(movingAverage);
            updateMovingAverage(3);
        }
        // Tell the listeners that data changed.
        fireDatasetChanged();
    }
    finally
    {
        if (resultSet != null)
        {
            try
            {
                resultSet.close();
            }
            catch (Exception e)
            {
            }
        }
        if (statement != null)
        {
            try
            {
                statement.close();
            }
            catch (Exception e)
            {
            }
        }
    }
}
/**
 * Returns the mean value of data
 *
```

```
 * @return The mean value.
 */
private double calculateMean()
{
    ClearableYIntervalSeries  value = data.get(0);
    Number [] field = new Double[value.getItemCount()];
    for(int i=0; i<field.length; i++)
            field[i] = Double.valueOf(value.getYValue(i));
    return Statistics.calculateMean(field);
}
/**
 * Returns the standard deviation of data
 *
 * @return The standard deviation.
 */
private double calculateStdDev()
{
    ClearableYIntervalSeries  value = data.get(0);
    Number [] field = new Double[value.getItemCount()];
    for(int i=0; i<field.length; i++)
            field[i] = Double.valueOf(value.getYValue(i));
    if(field.length == 0)
            return -1.0;
    return Statistics.getStdDev(field);
}
/**
 * Returns a data set for a moving average on the data set passed in.
 * @param period  the number of data points to average
 * @return A double[][] the length of the data set in the first dimension,
 * with two doubles for x and y in the second dimension
 */
public double[][] calculateMovingAverage(int period)
{
    ClearableYIntervalSeries  value = data.get(0);
    int count = value.getItemCount();
    Number [] x = new Double[count];
    for(int i=0; i<x.length; i++)
        x[i] = value.getX(i);
    Number [] y = new Double[count];
    for(int i=0; i<y.length; i++)
        y[i] = Double.valueOf(value.getYValue(i));
    //CJb3LL
    if(period > x.length)
            return Statistics.getMovingAverage(x, y, 0);
    return Statistics.getMovingAverage(x, y, period);
}
public void updateMovingAverage(int period)
{
    ClearableYIntervalSeries  value = data.get(2);
    value.clear();
    double [][] field = calculateMovingAverage(period);
    for(int i=0; i<field.length; i++)
            value.add(field[i][0], field[i][1], field[i][1], field[i][1]);
    //Tell the listeners that data changed.
```

```
    fireDatasetChanged();
}
/**
 * Returns the number of items in the specified series.
 * @param  seriesIndex  the series (zero-based index).
 * @return The itemCount value
 */
public int getItemCount(int seriesIndex)
{
    return getSeries(seriesIndex).getItemCount();
}
/**
 * Returns a series from the collection.
 * @param series  the series index (zero-based).
 * @return The series.
 * @throws IllegalArgumentException if <code>series</code> is not in the
 *     range <code>0</code> to <code>getSeriesCount() - 1</code>.
 */
public ClearableYIntervalSeries  getSeries(int series)
{
    if ((series < 0) || (series >= getSeriesCount()))
            throw new IllegalArgumentException("Series index out of bounds");
    return this.data.get(series);
}
/**
 * Returns the number of series in the dataset.
 * @return The seriesCount value
 * @see XYDataset
 */
public int getSeriesCount()
{
    return this.data.size();
}
/**
 * Returns the key for the specified series.
 * @param seriesIndex  the series (zero-based index).
 * @return The seriesName value
 */
public Comparable getSeriesKey(int seriesIndex)
{
    return getSeries(seriesIndex).getKey();
}
/**
 * Returns the end x-value for an item within a series.
 * @param series  the series index.
 * @param item  the item index.
 * @return The x-value.
 */
public Number getEndX(int series, int item)
{
    return getX(series, item);
}
/**
 * Returns the end y-value for an item within a series.
```

```
         * @param series  the series index.
         * @param item  the item index.
         * @return The end y-value.
         */
        public Number getEndY(int series, int item)
        {
            ClearableYIntervalSeries s = this.data.get(series);
            return new Double(s.getYHighValue(item));
        }
        /**
         * Returns the start x-value for an item within a series.
         * @param series  the series index.
         * @param item  the item index.
         * @return The x-value.
         */
        public Number getStartX(int series, int item)
        {
            return getX(series, item);
        }
        /**
         * Returns the start y-value for an item within a series.
         * @param series  the series index.
         * @param item  the item index.
         * @return The start y-value.
         */
        public Number getStartY(int series, int item)
        {
            ClearableYIntervalSeries s = this.data.get(series);
            return new Double(s.getYLowValue(item));
        }
            /**
         * Returns the x-value for an item within a series.
         * @param series  the series index.
         * @param item  the item index.
         * @return The x-value.
         */
        public Number getX(int series, int item)
        {
            ClearableYIntervalSeries s = this.data.get(series);
            return s.getX(item);
        }
        /**
         * Returns the y-value for an item within a series.
         * @param series  the series index.
         * @param item  the item index.
         * @return The y-value.
         */
        public Number getY(int series, int item)
        {
            ClearableYIntervalSeries s = this.data.get(series);
            return new Double(s.getYValue(item));
        }
    }
```

7.1.4 Das Paket CSV

CSV

```
package de.christianbell.apmfw.jdragicdatamining.csv;
import com.csvreader.*;
import de.christianbell.apmfw.jdragicdatamining.charts.JdbcChartListPanel;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
import java.io.File;
import java.io.IOException;
import java.sql.BatchUpdateException;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
/*
 * Diese Klasse schreibt die CSV-Datei in die Datenbank
 * @author  Christian Bell
 */
public class CSV
{
        private String csvFile;
        /**
         * @return  the csvFile
         */
        public String getCsvFile()
        {
                return csvFile;
        }
        public void setCsvFile(String csvFileT)
        {
                        this.csvFile = csvFileT;
        }
        /** Diese Methode merkt sich die Timestamps
        *des Start- und des Stop-Labels, wichtig fuer Shared Memory
        *
        * @return String[] 1. Argument ist Timestamp von Start,
        * 2. Argument ist Timestamp von Stop
        * @exception IOException
        */
        private String[] getStartAndStop()
        {
                try
                {
                        CsvReader csvRead = new CsvReader(getCsvFile());
                        String mfwPoint;
                        String startTime = "";
                        String stopTime = "";
                        int abbruch = 0;
                        while (csvRead.readRecord())
                        {
```

```
                                mfwPoint = csvRead.get(1);
                                if (mfwPoint.equals("__START__"))
                                {
                                        startTime = csvRead.get(2);
                                        if (abbruch == 2)
                                        {
                                                break;
                                        }
                                        if (abbruch < 2)
                                        {
                                                abbruch++;
                                        }
                                }
                                if (mfwPoint.equals("__STOP__"))
                                {
                                        stopTime = csvRead.get(2);
                                        if (abbruch == 2)
                                        {
                                                break;
                                        }
                                        if (abbruch < 2)
                                        {
                                                abbruch++;
                                        }
                                }
                        }
                        String[] times = {startTime, stopTime };
                        return times;
                }
                catch(java.io.IOException fex)
                {
                        fex.printStackTrace();
                        String[] error = {"", ""};
                        return error;
                }
        }
    /** Diese Methode liest eine CSV ein und speichert sie in einer Datenbank
    *
    * @exception IOException
    * @exception BatchUpdateException
    * @exception SQLException
    */
        public void loadCSVIntoDatabase(String origTable)
        {
                String mfwPoint;
                String tStamp;
                String state;
                String time;
                String pid;
                String tid;
                String insertFields = "mfwPoint, tStamp, state, time, pid, tid";
                String insertValues = "";
                int count = 0;
                String[] times = getStartAndStop();
```

```
String startTimeS = times[0];
String stopTimeS = times[1];
GUI.konsoleArea.append("startTime\t" + startTimeS + "\n");
GUI.konsoleArea.append("stopTime\t" + stopTimeS + "\n");
GUI.autoScroll();
long startTime = Long.valueOf(startTimeS).longValue();
long stopTime = Long.valueOf(stopTimeS).longValue();
try
{
        CsvReader csvRead = new CsvReader(getCsvFile());
        DBConnection db = new DBConnection();
        Connection theConnection = db.getConnection();
        Statement statement = theConnection.createStatement();
        while (csvRead.readRecord())
        {
          tStamp = csvRead.get(2);
          if(Long.valueOf(tStamp).longValue() <= stopTime
            && Long.valueOf(tStamp).longValue() >= startTime)
          {
            mfwPoint = csvRead.get(1);
            state = csvRead.get(3);
            time = csvRead.get(4);
            pid = csvRead.get(5);
            tid = csvRead.get(6);
            insertValues = "'" + mfwPoint + "', '" + tStamp
                           + "', '" + state + "', '" + time
                           + "', '" + pid + "', '" + tid + "'";
            try
            {
             statement.addBatch("INSERT INTO " + origTable
                      + " ("+ insertFields + ") VALUES ("
                      + insertValues + ")");
             count++;
             if(count >= 10000)
             {
               statement.executeBatch();
               GUI.konsoleArea.append(".");
               count = 0;
             }
            }
            catch (BatchUpdateException e)
            {
              System.out.println("Fehler in CSV.loadCSVIntoDatabase(): "
                     + e.getMessage());
              e.printStackTrace();
            }
            catch (SQLException sqle)
            {
             System.out.println("Fehler in CSV.loadCSVIntoDatabase(): "
                     + sqle.getMessage());
             sqle.printStackTrace();
            }
          }
        }
```

```
                        statement.executeBatch();
                        GUI.konsoleArea.append("!\n");
                        GUI.autoScroll();
                        statement.close();
                        db.closeConnection(theConnection);
                        csvRead.close();
        }
        catch (java.io.IOException fex)
        {
         fex.printStackTrace();
        }
        catch(SQLException e)
        {
            e.printStackTrace();
        }
        catch(InstantiationException e)
        {
            e.printStackTrace();
        }
        catch(IllegalAccessException e)
        {
            e.printStackTrace();
        }
        catch (ClassNotFoundException e)
        {
            e.printStackTrace();
        }
    }
}
```

CSVTableGenerator

```
package de.christianbell.apmfw.jdragicdatamining.csv;
import java.io.File;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Vector;
import de.christianbell.apmfw.jdragicdatamining.mfw.ValueComputation;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
/*
  * Diese Klasse generiert eine Tabelle, in der die Daten
  * aus der CSV-Datei gespeichert werden
  * @author Christian Bell
  */
public class CsvTableGenerator
{
         public static String path = null;
         public static PropertiesSingleton ps = null;
         public static void generateTablesFromOrigCSV() throws SQLException
         {
```

```
ps = PropertiesSingleton.getInstance();
DBConnection db = new DBConnection();
Connection connection = null;
Statement statement = null;
ResultSet result = null;
ValueComputation calc = new ValueComputation();
CSV csvFile = new CSV();
//TODO In Config aufnehmen
path = "Apache/origs/";
File dir = new File(path);
File[] entries = dir.listFiles();
if(entries == null)
  System.out.println(dir + " ist kein gueltiges Verzeichnis!");
else if(entries.length <= 0)
  System.out.println(dir + " enthaelt keine Dateien
                     oder Verzeichnisse!");
else
{
        try
        {
                connection = db.getConnection();
                statement =  connection.createStatement();
                calc.setConnection(connection);
        }
        catch (InstantiationException e)
        {
                e.printStackTrace();
        }
        catch (IllegalAccessException e)
        {
                e.printStackTrace();
        }
        catch (ClassNotFoundException e)
        {
                e.printStackTrace();
        }
        for( int i=0; i<entries.length; i++ )
        {
         if( !entries[i].isDirectory()
           && entries[i].toString().endsWith(".csv"))
         {
          GUI.konsoleArea.append("Tabelle reinigen...");
          db.cleanTable("orig", statement);
          GUI.konsoleArea.append(" gereinigt!\n");
          GUI.autoScroll();
          statement.execute("CREATE TABLE IF NOT EXISTS orig (ID
             BIGINT(20) UNSIGNED AUTO_INCREMENT, PRIMARY KEY (ID),
             MFWPOINT VARCHAR(100) NOT NULL, TSTAMP BIGINT(20) UNSIGNED
             NOT NULL, STATE VARCHAR(10) NOT NULL, TIME BIGINT(20) NOT
             NULL, PID INT(10) UNSIGNED NOT NULL, TID INT(10) UNSIGNED
             NOT NULL) ENGINE=MyISAM CHARACTER SET utf8 COLLATE
             utf8_general_ci");
          GUI.konsoleArea.append("CSV wird in Datenbank
             geschrieben...\n");
```

```
                            GUI.autoScroll();
                            csvFile.setCsvFile(entries[i].toString());
                            csvFile.loadCSVIntoDatabase("orig");
                            GUI.konsoleArea.append("CSV wurde in Datenbank
                               geschrieben!\n");
                            GUI.konsoleArea.append("Berechne Werte fuer "
                               + entries[i].toString() + "\n");
                            GUI.autoScroll();
                            result = statement.executeQuery("SELECT DISTINCT
                               mfwpoint FROM orig");
                            Vector<String> v = new Vector<String>();
                            while(result.next())
                            {
                             if(!result.getString("mfwPoint").contains("__ST"))
                              v.add(result.getString("mfwpoint"));
                            }
                            calc.calculateAndSaveDifferences(entries[i].toString(), v);
                            GUI.konsoleArea.append("Werte wurden berechnet
                               und abgespeichert!\n");
                            GUI.autoScroll();
                           }
                         }
                         result.close();
                         statement.close();
                        db.closeConnection(connection);
                    }
            }
}
```

MySQLToCSV

```
package de.christianbell.apmfw.jdragicdatamining.export;
import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.sql.Statement;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.TimeZone;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
/*
 * Diese Klasse wandelt MySQL-Daten nach CSV
 * @author Christian Bell
 */
public class MySQLtoCSV
{
    /** Diese Methode uebertraegt den Inhalt der Datenbank in eine CSV-Datei
    * @param arg Nummer der Messung
    * @exception SQLException
    */
        public void exportToCSV(String table)
        {
                DBConnection db = new DBConnection();
```

```
                Connection connection = null;
                Statement statement = null;
                try {
                        connection = db.getConnection();
                        statement = connection.createStatement();
                } catch (InstantiationException e1) {
                        e1.printStackTrace();
                } catch (IllegalAccessException e1) {
                        e1.printStackTrace();
                } catch (ClassNotFoundException e1) {
                        e1.printStackTrace();
                } catch (SQLException e1) {
                        e1.printStackTrace();
                }
                String fileName = table + ".csv";
                try
                {
                        File file = new File(fileName);
                        if(file.exists())
                                file.delete();
                        String fileStr = file.getAbsolutePath();
                        //MySQL kennt nur / und nicht \
                        if(fileStr.contains("\\"))
                          fileStr = fileStr.replaceAll("\\\\", "/");
                        statement.execute("SELECT * FROM " + table
                                 + " INTO OUTFILE '" + fileStr
                                 + "' FIELDS TERMINATED BY ','");
                        statement.close();
                        connection.close();
                }
                catch (SQLException e)
                {
                  System.out.println("Fehler in loadDatabaseIntoCSV(): fileName := "
                           + fileName + "\nMessage: " + e.getMessage());
                  e.printStackTrace();
                }
        }
        /**
         * @param args
         */
        public static void main(String[] args)
        {
                MySQLtoCSV exporter = new MySQLtoCSV();
                exporter.exportToCSV("results_814preforktest003_5_20_5_1000_50_100_0");
        }
}
```

MySQLToXLS

```
package de.christianbell.apmfw.jdragicdatamining.export;
import java.io.FileNotFoundException;
import java.io.FileOutputStream;
import java.io.IOException;
import java.sql.Connection;
```

```
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Vector;
import org.apache.poi.hssf.usermodel.HSSFCell;
import org.apache.poi.hssf.usermodel.HSSFCellStyle;
import org.apache.poi.hssf.usermodel.HSSFDataFormat;
import org.apache.poi.hssf.usermodel.HSSFFont;
import org.apache.poi.hssf.usermodel.HSSFRichTextString;
import org.apache.poi.hssf.usermodel.HSSFRow;
import org.apache.poi.hssf.usermodel.HSSFSheet;
import org.apache.poi.hssf.usermodel.HSSFWorkbook;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
/*
 * Diese Klasse exportiert MySQL-Daten in
 * eine Excel-Datei
 * @author Christian Bell
 */
public class MySQLtoXLS
{
        public void exportToXLS(String table)
        {
                DBConnection db = new DBConnection();
                Connection connection = null;
                FileOutputStream out = null;
                try
                {
                        connection = db.getConnection();
                        System.out.println("Verbunden");
                        Statement statement = connection.createStatement();
                        System.out.println("Statement erzeugt!");
                        ResultSet results = statement.executeQuery("SHOW COLUMNS
                              FROM " + table);
                        System.out.println("SHOW COLUMNS erledigt");
                        out = new FileOutputStream(table + ".xls");
                        //Neues Workbook erzeugen
                        HSSFWorkbook wb = new HSSFWorkbook();
                        //Neues Sheet erzeugen
                        HSSFSheet s = wb.createSheet();

                        HSSFRow r = null;

                        HSSFCell c = null;
                        //Erzeuge Zellenstil
                        HSSFCellStyle cs = wb.createCellStyle();
                        HSSFDataFormat df = wb.createDataFormat();
                        //Schriftobjekt
                        HSSFFont f = wb.createFont();
                        //Schriftgroesse
                        f.setFontHeightInPoints((short)12);
                        //Setze den Zellenstil
                        cs.setFont(f);
                        //Setze das Zellenformat
                        cs.setDataFormat(df.getFormat("#,##0.0"));
                        wb.setSheetName(0, "Seite 1");
```

```
r = s.createRow(0);
int cellnum = 0;
Vector<String> v = new Vector<String>();
HSSFRichTextString string = null;
//Tabellenkopf schreiben
while(results.next())
{
 c = r.createCell(cellnum);
 string = new HSSFRichTextString(results.getString(1));
 c.setCellValue(string);
 c.setCellStyle(cs);
 s.setColumnWidth(cellnum,
   (short) ((50 * 8) / ((double) 1 / 20)));
 v.add(results.getString(1));
  cellnum++;
}
int rownum = 1;
r = s.createRow(1);
cellnum = 0;
results = statement.executeQuery("SELECT * FROM " + table);
int i = 0;
int sheetNumber = 1;

//Tabelleneintraege schreiben
while(results.next())
{
        System.out.println(rownum + " :: ");
        if(rownum >= 65535)
        {
         s = wb.createSheet();
         sheetNumber++;
         wb.setSheetName(sheetNumber-1, "Seite "
            + sheetNumber);
         r = s.createRow(0);
         rownum = 0;
        }
        while(i < v.size())
        {
         c = r.createCell(i);
         string = new HSSFRichTextString(
                  results.getString(v.get(i)));
         c.setCellValue(string);
         cellnum = i;
         c.setCellStyle(cs);
         s.setColumnWidth(cellnum,
          (short) ((50 * 8) / ((double) 1 / 20)));
         //System.out.print(results.getString(v.get(i)) + " ");
         i++;
        }
        //System.out.println();
        i = 0;
        r = s.createRow(++rownum);
}
results.close();
```

```
                        statement.close();
                        connection.close();
                        //Das Workbook in den Ausgabestream schreiben
                        try {
                                  wb.write(out);
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                        try {
                                out.close();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
                                //System.out.println("Fertig!");
                }
                catch(FileNotFoundException exFileNotFound)
                {
                        System.out.println("Kann Datei nicht oeffnen:");
                }
                catch(Exception eAllg)
                {
                        System.out.println("Allgemeiner Fehler");
                        eAllg.printStackTrace();
                }
        }
        public static void main(String[] args)
        {
          MySQLtoXLS exporter = new MySQLtoXLS();
          exporter.exportToXLS("results_814preforktest003_5_20_5_1000_50_100_0");
        }
}
```

7.1.5 Das Paket GUI

ActionListenerClass

```
package de.christianbell.apmfw.jdragicdatamining.gui;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.io.File;
import java.io.IOException;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Vector;
import javax.swing.JOptionPane;
import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.axis.NumberAxis;
import org.jfree.chart.plot.DatasetRenderingOrder;
import org.jfree.chart.plot.PlotOrientation;
```

```
import org.jfree.chart.plot.XYPlot;
import org.jfree.chart.renderer.xy.XYLineAndShapeRenderer;
import org.jfree.data.xy.XYDataset;
import org.jfree.data.xy.XYSeries;
import org.jfree.data.xy.XYSeriesCollection;
import org.jfree.ui.RectangleInsets;
import de.christianbell.apmfw.jdragicdatamining.auswertung.Statistics;
import de.christianbell.apmfw.jdragicdatamining.charts.
       JDBCMeanAndStandardDeviationXYDataset;
import de.christianbell.apmfw.jdragicdatamining.charts.JdbcChartListPanel;
import de.christianbell.apmfw.jdragicdatamining.charts.JdbcChartPanel;
import de.christianbell.apmfw.jdragicdatamining.export.MySQLtoCSV;
import de.christianbell.apmfw.jdragicdatamining.export.MySQLtoXLS;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
import de.christianbell.apmfw.jdragicdatamining.threads.*;
/*
 * Dies ist eine Abhoerer-Klasse, welche verschiedene Events der GUI handelt
 * @author Christian Bell
 */
public class ActionListenerClass implements java.awt.event.ActionListener
{
        MessungThread messungThread = null;
        public static Vector<String> listData = null;
        public void actionPerformed(ActionEvent event)
        {
                if(event.getSource() == GUI.startButton)
                {
                        GUI.getServerParamsList();
                        GUI.getStartWertField();
                        GUI.getStopWertField();
                        GUI.getStepField();
                        GUI.autoScroll();
                        messungThread = new MessungThread();
                        try
                        {
                                messungThread.start();
                        }
                        catch(IllegalThreadStateException e)
                        {
                                e.printStackTrace();
                        }
                }
                else if(event.getSource() == GUI.abbruchButton)
                {
                        try
                        {
                                messungThread.interrupt();
                        }
                        catch(IllegalThreadStateException e)
                        {
                                e.printStackTrace();
                        }
                }
```

```
else if(event.getSource() == GUI.modulComboBox)
{
        GUI.getServerParamsList();
        GUI.getStartWertField();
        GUI.getStopWertField();
        GUI.getStepField();
        GUI.konsoleArea.append("Modul gewechselt!\n");
        GUI.autoScroll();
}
else if(event.getSource() == GUI.speichereParamsButton)
{
 PropertiesSingleton ps = PropertiesSingleton.getInstance();
 ps.setProperty(GUI.modulComboBox.getSelectedItem().toString()
    + GUI.ServerParamsList.getSelectedValue().toString()
    + "Start", GUI.startWertField.getText());
 ps.setProperty(GUI.modulComboBox.getSelectedItem().toString()
    + GUI.ServerParamsList.getSelectedValue().toString()
    + "Stop", GUI.stopWertField.getText());
 ps.setProperty(GUI.modulComboBox.getSelectedItem().toString()
    + GUI.ServerParamsList.getSelectedValue().toString()
    + "Step", GUI.stepField.getText());
 ps.setProperty(GUI.jmeterParamsList.getSelectedValue().toString()
    + "Start", GUI.jmeterStartWertField.getText());
 ps.setProperty(GUI.jmeterParamsList.getSelectedValue().toString()
    + "Stop", GUI.jmeterStopWertField.getText());
 ps.setProperty(GUI.jmeterParamsList.getSelectedValue().toString()
    + "Step", GUI.jmeterStepField.getText());
 ps.setProperty("cgiStart", GUI.cgiStartField.getText());
 ps.setProperty("cgiStop", GUI.cgiStopField.getText());
 ps.setProperty("cgiStep", GUI.cgiIntervall.getText());
 ps.setProperty("processingDelayStart",
    GUI.processingDelayStartField.getText());
 ps.setProperty("processingDelayStop",
    GUI.processingDelayStopField.getText());
 ps.setProperty("processingDelayStep",
    GUI.processingDelayStepField.getText());
 ps.setProperty("messungsname",
    GUI.messungsnameField.getText().toLowerCase());
 ps.setProperty("anzahlWiederholungen",
    GUI.wiederholenField.getText());
 ps.setProperty("autoAuswerten",
    Boolean.valueOf(GUI.auswertenCheckBox.isSelected()).toString());
 ps.setProperty("keepAlive",
    Boolean.valueOf(GUI.keepAliveCheckBox.isSelected()).toString());
 ps.setProperty("saveResponseData",
    Boolean.valueOf(GUI.responseDataCheckBox.isSelected()).toString());
 GUI.konsoleArea.append("Server-Parameter wurde gespeichert!\n");
}
//Alle Bilder speichern
else if(event.getActionCommand().equals(
       JdbcChartListPanel.getPngAlleSpeichernButton().
         getActionCommand()))
{
  AlleMessungenAuswertenThread alleMessungenAuswertenThread =
```

```
        new AlleMessungenAuswertenThread();
        try
        {
                alleMessungenAuswertenThread.start();
        }
        catch(IllegalThreadStateException exception)
        {
                exception.printStackTrace();
        }
}
else if(event.getActionCommand().equals(
      JdbcChartListPanel.getTableComboBox().getActionCommand()))
{
        try
        {
          JdbcChartListPanel.fillList(
            JdbcChartListPanel.getTableComboBox().
            getSelectedItem().toString());
        } catch (SQLException e1)
        {
                e1.printStackTrace();
        }
}
else if(event.getActionCommand().equals(
     JdbcChartListPanel.getPngSpeichernButton().getActionCommand()))
{
        TabelleAuswertenThread tabelleAuswertenThread =
            new TabelleAuswertenThread();
        try
        {
                tabelleAuswertenThread.start();
        }
        catch(IllegalThreadStateException exception)
        {
                exception.printStackTrace();
        }
}
else if(event.getActionCommand().equals(
   GUI.getTableQuerComboBox().getActionCommand()))
{
        try
        {
                GUI.fillList(GUI.getTableQuerComboBox().
                     getSelectedItem().toString());
        } catch (SQLException e1)
        {
                e1.printStackTrace();
        }
}
else if(event.getSource() == GUI.addenButton)
{
        if(listData == null)
                listData = new Vector<String>();
        try
```

```
        {
          if(GUI.getTableQuerComboBox().getSelectedItem().
             toString().contains("results"))
            listData.add(GUI.getTableQuerComboBox().
            getSelectedItem().toString() + "split"
            + GUI.getMesspunkteQuerList().
              getSelectedValue().toString());
           else listData.add(GUI.getMesspunkteQuerList().
             getSelectedValue().toString());
             GUI.quantileList.setListData(listData);
             GUI.quantileList.setSelectedIndex(0);
             GUI.quantileList.repaint();
        }
        catch (Exception e) {
                System.out.println(e.getMessage());
                e.printStackTrace();
        }
}
else if(event.getActionCommand().equals(
      JdbcChartListPanel.getExcelExportButton().getActionCommand()))
{
        MySQLtoXLS exporter = new MySQLtoXLS();
         exporter.exportToXLS(
          JdbcChartListPanel.getTableComboBox().
          getSelectedItem().toString());
}
else if(event.getActionCommand().equals(
      JdbcChartListPanel.getCsvExportButton().getActionCommand()))
{
        MySQLtoCSV exporter = new MySQLtoCSV();
        exporter.exportToCSV(
          JdbcChartListPanel.getTableComboBox().
          getSelectedItem().toString());
}
else if(event.getSource() == GUI.graphQuerButton)
{
        Connection con = null;
        try
        {
                DBConnection db = new DBConnection();
                con = db.getConnection();
        }
        catch (Exception e1)
        {
                System.out.println(e1.getMessage());
                e1.printStackTrace();
                return;
        }
        //Pruefen, ob die Auswahlliste mindestens ein Element
        //enthaelt, falls nicht -> Auswahlliste fuellen
        if(GUI.quantileList.getModel().getSize() == 0)
        {
                GUI.konsoleArea.append("Leider keine Daten angegeben!");
                return;
```

```
                        }
                        int i = 0;
                        XYSeries series = new XYSeries("90%-Quantile");
                        XYSeriesCollection dataset = new XYSeriesCollection();
                        double quantile;
                        Statistics valComp = new Statistics();
                        int count = 0;
                        while(i < GUI.quantileList.getModel().getSize())
                        {
                                GUI.quantileList.setSelectedIndex(i);
                                count++;
                                try
                                {
if(GUI.quantileList.getSelectedValue().toString().contains("results"))
{
 String[] strArr = GUI.quantileList.getSelectedValue().toString().split("split");
 quantile = valComp.calculateQuantile(0.9, strArr[0], strArr[1]);
 series.add(count, quantile);
}
else
{
   quantile = valComp.calculateQuantile(0.9,
       GUI.quantileList.getSelectedValue().toString(), null);
   series.add(count, quantile);
}}
catch (Exception e1)
{
    e1.printStackTrace();
}
i++;
}
   dataset.addSeries(series);
   JFreeChart chart = ChartFactory.createXYLineChart("90%-Quantile", "", "",
            dataset, PlotOrientation.VERTICAL, true, true, false);
   chart.setBackgroundPaint(Color.white);
   XYPlot plot = (XYPlot) chart.getPlot();
   plot.setBackgroundPaint(Color.lightGray);
   plot.setAxisOffset(new RectangleInsets(5.0, 5.0, 5.0, 5.0));
   plot.setDomainGridlinePaint(Color.white);
   plot.setRangeGridlinePaint(Color.white);
   XYLineAndShapeRenderer renderer = (XYLineAndShapeRenderer) plot.getRenderer();
   renderer.setShapesVisible(true);
   renderer.setShapesFilled(true);
   NumberAxis rangeAxis = (NumberAxis)plot.getRangeAxis();
   File file = new File("home/studpro/auswertung.png");
   try {
     ChartUtilities.saveChartAsPNG(file, chart, 1000, 640);
     //BrowserLauncher launcher = new BrowserLauncher();
     //launcher.openURLinBrowser("http://" + file.getAbsolutePath());
   } catch (IOException e) {
        e.printStackTrace();
   }
   /*              catch(BrowserLaunchingInitializingException e1)
                        {
```

```
                                e1.printStackTrace();
                        }
                        catch(UnsupportedOperatingSystemException e3)
                        {
                                e3.printStackTrace();
                        }
*/
                }
        }
}
```

GUI

```
package de.christianbell.apmfw.jdragicdatamining.gui;
import de.christianbell.apmfw.jdragicdatamining.charts.JdbcChartListPanel;
import de.christianbell.apmfw.jdragicdatamining.server.HttpdMPMConfReader;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.*;
import de.christianbell.apmfw.jdragicdatamining.jmeter.JMXReader;
import javax.swing.JPanel;
import javax.swing.JFrame;
import javax.swing.JCheckBox;
import java.awt.Rectangle;
import java.io.IOException;
import java.sql.Connection;
import java.sql.SQLException;
import java.util.Iterator;
import java.util.Vector;
import javax.swing.BorderFactory;
import javax.swing.ButtonGroup;
import javax.swing.JOptionPane;
import javax.swing.JTabbedPane;
import javax.swing.JTextField;
import javax.swing.JLabel;
import javax.swing.JComboBox;
import javax.swing.JTextArea;
import javax.swing.JButton;
import javax.swing.JScrollPane;
import javax.swing.JList;
import java.awt.Color;
import java.awt.GridBagLayout;
import java.awt.GridBagConstraints;
import javax.swing.JRadioButton;
/*
  * Diese Klasse ist die GUI des Tools, sie sollte zum Starten
  * des Programmes aufgerufen werden
  * @author Christian Bell
  * @version 1.0.0beta
  */
public class GUI extends JFrame
{
   private static Connection con;
   private static DBConnection db;
   private static ActionListenerClass anActionListener = new ActionListenerClass();
   private static ListSelectionListenerClass anListSelectionListener =
```

```
      new ListSelectionListenerClass();
   public static PropertiesSingleton ps = null;
   private static final long serialVersionUID = 1L;
   public static JCheckBox responseDataCheckBox = null;
   private JTabbedPane jTabbedPane = null;
   private JPanel messungenTab = null;
   private JdbcChartListPanel ergebnisseTab = null;
   private JPanel configTab = null;
   public static JTextField messungsnameField = null;
   private JLabel messungsnameLabel = null;
   private JLabel responseDataLabel = null;
   public static JCheckBox keepAliveCheckBox = null;
   private JLabel keepAliveLabel = null;
   public static JComboBox modulComboBox = null;
   private JLabel modulLabel = null;
   public static JTextArea konsoleArea = null;
   private JLabel konsoleLabel = null;
   private JLabel ServerParams = null;
   private JLabel stepLabel = null;
   private JLabel startWertLabel = null;
   private JLabel cgiLabel = null;
   public static JTextField cgiStartField = null;
   public static JTextField cgiIntervall = null;
   public static JButton startButton = null;
   public static JButton abbruchButton = null;
   public static JCheckBox auswertenCheckBox = null;
   private JLabel auswertenLabel = null;
   public static JTextField wiederholenField = null;
   private JLabel wiederholenLabel = null;
   private JLabel filepathLabel = null;
private JTextField filepathField = null;
   private JLabel origtableLabel = null;
   private JTextField origtableField = null;
private JLabel hostLabel = null;
private JTextField hostField = null;
private JLabel dbnameLabel = null;
private JTextField dbnameField = null;
private JLabel BenutzernameLabel = null;
private JTextField benutzernameField = null;
private JLabel passwortLabel = null;
private JTextField passwortField = null;
private JLabel maxHeapSpaceLabel = null;
private JTextField maxHeapSpaceField = null;
private JButton speichernConfigButton = null;
private static JScrollPane konsoleScrollPane = null;
private JLabel pfadConfLabel = null;
private JTextField pfadConfField = null;
private JLabel jmeterXMLLabel = null;
private JTextField jmeterXMLField = null;
private JLabel jmeterhomeLabel = null;
private JTextField jmeterhomeField = null;
private JLabel resultlogLabel = null;
private JTextField resultlogField = null;
private JLabel stopWertLabel = null;
```

```
public static JTextField cgiStopField = null;
public static JList ServerParamsList = null;
public static JTextField startWertField = null;
public static JTextField stopWertField = null;
public static JTextField stepField = null;
public static JButton speichereParamsButton = null;
public static JScrollPane serverParamsScrollPane = null;
private JLabel serverLabel = null;
private JTextField serverTextField = null;
private JLabel jtlPathLabel = null;
private JTextField jtlPathField = null;
private JLabel logsPathLabel = null;
private JTextField logsPathField = null;
private JLabel jmeterVorlageLabel = null;
private JTextField jmeterVorlageField = null;
private JScrollPane jmeterParamsScrollPane = null;
public static JList jmeterParamsList = null;
public static JTextField jmeterStartWertField = null;
public static JTextField jmeterStopWertField = null;
public static JTextField jmeterStepField = null;
private JLabel jmeterParamsLabel = null;
private JLabel startWertLabel1 = null;
private JLabel stopWertLabel1 = null;
private JLabel stepLabel1 = null;
private JLabel startWertLabel2 = null;
private JLabel stopWertLabel2 = null;
private JLabel stepLabel2 = null;
private JPanel querAuswertungTab = null;
private static JComboBox tableQuerComboBox = null;
public static JList messpunkteQuerList = null;
public static JButton addenButton = null;
public static JList quantileList = null;
public static JButton graphQuerButton = null;
private JLabel jLabel = null;
private JTextField jLoadtestsField = null;
private JLabel processingDelayLabel = null;
public static JTextField processingDelayStartField = null;
public static JTextField processingDelayStopField = null;
public static JTextField processingDelayStepField = null;
private JRadioButton jRadioButton1 = null;
private JRadioButton jRadioButton2 = null;
private ButtonGroup buttonGroup = null;
private JLabel jLabelMethode = null;
private JLabel jLabel1 = null;
private JLabel jLabel2 = null;
private JLabel jLabel3 = null;
private JLabel jLabel4 = null;
/**
 * This method initializes jCheckBox
 *
 * @return javax.swing.JCheckBox
 */
private JCheckBox getResponseDataCheckBox() {
if (responseDataCheckBox == null) {
```

```
        responseDataCheckBox = new JCheckBox();
responseDataCheckBox.setSelected(Boolean.valueOf(ps.getProperty("saveResponseData")));
        responseDataCheckBox.setBounds(new Rectangle(15, 135, 24, 21));
        }
        return responseDataCheckBox;
}
/**
 * This method initializes jTabbedPane
 *
 * @return javax.swing.JTabbedPane
 */
private JTabbedPane getJTabbedPane() {
        if (jTabbedPane == null) {
        jTabbedPane = new JTabbedPane();
                jTabbedPane.addTab("Messungen", null, getMessungenTab(), null);
                jTabbedPane.addTab("Ergebnisse", null, getErgebnisseTab(), null);
        jTabbedPane.addTab("Querauswertung", null, getQuerAuswertungTab(), null);
        jTabbedPane.addTab("Einstellungen", null, getConfigTab(), null);
        }
return jTabbedPane;
}
/**
 * This method initializes jPanel
 *
 * @return javax.swing.JPanel
 */
private JPanel getMessungenTab()
{
        if (messungenTab == null)
        {
        jLabel2 = new JLabel();
        jLabel2.setBounds(new Rectangle(240, 240, 31, 16));
                jLabel2.setText("2");
        jLabel1 = new JLabel();
                jLabel1.setBounds(new Rectangle(165, 240, 16, 16));
                jLabel1.setText("1");
                jLabelMethode = new JLabel();
                jLabelMethode.setBounds(new Rectangle(15, 240, 106, 16));
                jLabelMethode.setText("Methode:");
                processingDelayLabel = new JLabel();
                processingDelayLabel.setBounds(new Rectangle(15, 210, 106, 16));
                processingDelayLabel.setText("Processing-Delay:");
                stepLabel2 = new JLabel();
                stepLabel2.setBounds(new Rectangle(345, 165, 106, 16));
                stepLabel2.setText("Schrittweite (ms):");
                stepLabel2.setToolTipText("Das Intervall gibt den
                     Abstand zwischen den zwei Werten zweier Messungen an.");
                stopWertLabel2 = new JLabel();
        stopWertLabel2.setBounds(new Rectangle(240, 165, 91, 16));
                stopWertLabel2.setText("Stopwert (ms):");
                startWertLabel2 = new JLabel();
                startWertLabel2.setBounds(new Rectangle(135, 165, 91, 16));
                startWertLabel2.setText("Startwert (ms):");
                stepLabel1 = new JLabel();
```

```
        stepLabel1.setBounds(new Rectangle(750, 105, 76, 16));
        stepLabel1.setText("Schrittweite:");
        stepLabel1.setToolTipText("Das Intervall gibt den Abstand
           zwischen den zwei Werten zweier Messungen an.");
stopWertLabel1 = new JLabel();
        stopWertLabel1.setBounds(new Rectangle(750, 75, 76, 16));
        stopWertLabel1.setText("Stopwert:");
        startWertLabel1 = new JLabel();
        startWertLabel1.setBounds(new Rectangle(750, 45, 76, 16));
        startWertLabel1.setText("Startwert:");
        jmeterParamsLabel = new JLabel();
        jmeterParamsLabel.setBounds(new Rectangle(585, 15, 151, 16));
        jmeterParamsLabel.setText("JMeter-Parameter:");
        stopWertLabel = new JLabel();
        stopWertLabel.setBounds(new Rectangle(480, 75, 76, 16));
        stopWertLabel.setText("Stopwert:");
        wiederholenLabel = new JLabel();
        wiederholenLabel.setBounds(new Rectangle(15, 45, 181, 16));
        wiederholenLabel.setText("Wiederholungen Messungen:");
        wiederholenLabel.setToolTipText("<HTML>Geben Sie hier eine ganze
             Zahl ein, welche angeben soll, wie oft eine Messung wiederholt
             werden soll.<BR>Wenn Sie keine Wiederholung wï¿½nschen, so
             schreiben Sie ein 0 in das Feld.</HTML>");
        auswertenLabel = new JLabel();
        auswertenLabel.setBounds(new Rectangle(45, 75, 151, 16));
        auswertenLabel.setText("Automatisch auswerten");
        auswertenLabel.setToolTipText("<HTML>Wenn nach einer Messung die
            Auswertung automatisch durchgefï¿½ht werden soll,<BR>so aktivieren
        cgiLabel = new JLabel();
        cgiLabel.setBounds(new Rectangle(15, 180, 61, 16));
        cgiLabel.setText("Wartezeit:");
cgiLabel.setToolTipText("Die in der CGI-Datei angegeben Wartezeit in ms.");
startWertLabel = new JLabel();
        startWertLabel.setBounds(new Rectangle(480, 45, 76, 16));
        startWertLabel.setText("Startwert:");
        stepLabel = new JLabel();
        stepLabel.setBounds(new Rectangle(480, 105, 76, 16));
        stepLabel.setText("Schrittweite:");
        stepLabel.setToolTipText("Das Intervall gibt den Abstand zwischen
             den zwei Werten zweier Messungen an.");
        ServerParams = new JLabel();
        ServerParams.setBounds(new Rectangle(315, 15, 151, 16));
        ServerParams.setText("Server-Parameter:");
        ServerParams.setToolTipText("Parameter des Servers aus der
                 Konfigurationsdtei.");
        konsoleLabel = new JLabel();
        konsoleLabel.setBounds(new Rectangle(380, 255, 61, 16));
        konsoleLabel.setText("Konsole:");
        konsoleLabel.setToolTipText("<HTML>Die Konsole liefert Ihnen
               Informationen ueber den aktuellen Zustand der laufenden
               Messungen.<HTML>");
modulLabel = new JLabel();
        modulLabel.setBounds(new Rectangle(480, 165, 46, 16));
modulLabel.setText("Modul:");
```

```
            modulLabel.setToolTipText("<HTML>Mit welchem Modul soll der Server
              laufen?</HTML>");
            keepAliveLabel = new JLabel();
            keepAliveLabel.setBounds(new Rectangle(45, 105, 151, 16));
            keepAliveLabel.setText("Keep-Alive");
            keepAliveLabel.setToolTipText("<HTML>Verbindungen sollen bestehen
               bleiben?<BR>Akivieren Sie dieses Kaestchen!</HTML>");
            responseDataLabel = new JLabel();
            responseDataLabel.setBounds(new Rectangle(45, 135, 151, 16));
            responseDataLabel.setText("Response Data speichern");
            responseDataLabel.setToolTipText("Die Antworten des Servers
                speichern.");
        messungsnameLabel = new JLabel();
            messungsnameLabel.setBounds(new Rectangle(15, 15, 106, 16));
            messungsnameLabel.setText("Messungsname:");
            messungsnameLabel.setToolTipText("<HTML>Geben Sie hier den Hauptnamen
                     der Messung ein.<BR>Das Programm ergaenzt automatisch eine
                         durchlaufende Nummer.");
            messungenTab = new JPanel();
            messungenTab.setLayout(null);
            messungenTab.add(getResponseDataCheckBox(), null);
            messungenTab.add(getMessungsnameField(), null);
            messungenTab.add(messungsnameLabel, null);
            messungenTab.add(responseDataLabel, null);
            messungenTab.add(getKeepAliveCheckBox(), null);
            messungenTab.add(keepAliveLabel, null);
            messungenTab.add(getModulComboBox(), null);
            messungenTab.add(modulLabel, null);
            messungenTab.add(konsoleLabel, null);
            messungenTab.add(ServerParams, null);
            messungenTab.add(stepLabel, null);
            messungenTab.add(startWertLabel, null);
            messungenTab.add(cgiLabel, null);
            messungenTab.add(getCgiStartField(), null);
            messungenTab.add(getCgiIntervall(), null);
            messungenTab.add(getStartButton(), null);
            messungenTab.add(getAbbruchButton(), null);
            messungenTab.add(getAuswertenCheckBox(), null);
            messungenTab.add(auswertenLabel, null);
            messungenTab.add(getWiederholenField(), null);
            messungenTab.add(wiederholenLabel, null);
            messungenTab.add(getKonsoleScrollPane(), null);
            messungenTab.add(stopWertLabel, null);
            messungenTab.add(getCgiStopField(), null);
            messungenTab.add(getServerParamsScrollPane(), null);
            messungenTab.add(getStartWertField(), null);
            messungenTab.add(getStopWertField(), null);
            messungenTab.add(getStepField(), null);
            messungenTab.add(getSpeichereParamsButton(), null);
            messungenTab.add(getJmeterParamsScrollPane(), null);
            messungenTab.add(getJmeterStartWertField(), null);
            messungenTab.add(getJmeterStopWertField(), null);
            messungenTab.add(getJmeterStepField(), null);
            messungenTab.add(jmeterParamsLabel, null);
```

```
        messungenTab.add(startWertLabel1, null);
messungenTab.add(stopWertLabel1, null);
        messungenTab.add(stepLabel1, null);
        messungenTab.add(startWertLabel2, null);
        messungenTab.add(stopWertLabel2, null);
        messungenTab.add(stepLabel2, null);
        messungenTab.add(processingDelayLabel, null);
        messungenTab.add(getProcessingDelayStartField(), null);
        messungenTab.add(getProcessingDelayStopField(), null);
        messungenTab.add(getProcessingDelayStepField(), null);
        messungenTab.add(getJRadioButton1(), null);
        messungenTab.add(getJRadioButton2(), null);
        messungenTab.add(jLabelMethode, null);
        messungenTab.add(jLabel1, null);
messungenTab.add(jLabel2, null);
        if (buttonGroup == null)
        {
                buttonGroup = new ButtonGroup();
                        buttonGroup.add(jRadioButton1);
                        buttonGroup.add(jRadioButton2);
                        jRadioButton1.setSelected(true);
                }
        }
        return messungenTab;
}
/**
 * This method initializes jPanel1
 *
 * @return javax.swing.JPanel
 */
private JdbcChartListPanel getErgebnisseTab() {
        if (ergebnisseTab == null) {
        ergebnisseTab = new JdbcChartListPanel();
//ergebnisseTab.setBounds(new Rectangle(3, 30, 785, 533));
        }
        return ergebnisseTab;
}
/**
 * This method initializes jPanel1
 *
 * @return javax.swing.JPanel
 */
private JPanel getConfigTab() {
        if (configTab == null) {
                jLabel = new JLabel();
                jLabel.setBounds(new Rectangle(15, 465, 151, 16));
                jLabel.setText("Loadtests-Pfad:");
                jmeterVorlageLabel = new JLabel();
                jmeterVorlageLabel.setBounds(new Rectangle(15, 435, 151, 16));
                jmeterVorlageLabel.setText("JMeter-Vorlage:");
                logsPathLabel = new JLabel();
                logsPathLabel.setBounds(new Rectangle(15, 405, 151, 16));
                logsPathLabel.setText("Logs-Verzeichnis:");
                jtlPathLabel = new JLabel();
```

```
                jtlPathLabel.setBounds(new Rectangle(15, 375, 151, 16));
                jtlPathLabel.setText("JTL-Verzeichnis:");
                serverLabel = new JLabel();
                serverLabel.setBounds(new Rectangle(15, 345, 151, 16));
                serverLabel.setText("Serveradresse:");
                resultlogLabel = new JLabel();
                resultlogLabel.setBounds(new Rectangle(15, 315, 151, 16));
                resultlogLabel.setText("Result-Log:");
resultlogLabel.setToolTipText("Wo sollen die Ergebnisse von JMeter
          gespeichert werden.");
                jmeterhomeLabel = new JLabel();
                jmeterhomeLabel.setBounds(new Rectangle(15, 285, 151, 16));
                jmeterhomeLabel.setText("JMeter-Home:");
jmeterhomeLabel.setToolTipText("Geben Sie den Ort von JMeter an.");
                jmeterXMLLabel = new JLabel();
                jmeterXMLLabel.setBounds(new Rectangle(15, 255, 151, 16));
                jmeterXMLLabel.setText("Dateiname Jmeter-XML:");
jmeterXMLLabel.setToolTipText("<HTML>Geben Sie hier den Namen evtl.
       mit Pfad zur JMeter-XML-Datei an.</HMTL>");
                pfadConfLabel = new JLabel();
                pfadConfLabel.setBounds(new Rectangle(15, 225, 151, 16));
                pfadConfLabel.setText("Konfigurationsdatei:");
                pfadConfLabel.setToolTipText("Geben Sie hier den Speicherort
               der Konfigurationsdatei des Apache Webservers an.");
                maxHeapSpaceLabel = new JLabel();
                maxHeapSpaceLabel.setBounds(new Rectangle(15, 195, 151, 16));
                maxHeapSpaceLabel.setText("max. Heap-Speicher:");
                konsoleLabel.setBounds(new Rectangle(15, 270, 104, 16));
                konsoleLabel.setText("Konsole:");
                konsoleLabel.setToolTipText("<HTML>Die Konsole liefert Ihnen
                  Informationen ueber den aktuellen Zustand der laufenden
                  Messungen.</HTML>");
                maxHeapSpaceLabel.setToolTipText("<HTML>Maximale Groesse des
                  Heap-Speichers.<BR>Bei vielen Messungen hoeheren Wert
                  waehlen.</HTML>");
                passwortLabel = new JLabel();
                passwortLabel.setBounds(new Rectangle(15, 165, 151, 16));
                passwortLabel.setText("Passwort:");
                passwortLabel.setToolTipText("Passwort zur Datenbank.");
                BenutzernameLabel = new JLabel();
                BenutzernameLabel.setBounds(new Rectangle(15, 135, 151, 16));
                BenutzernameLabel.setText("Benutzername:");
                BenutzernameLabel.setToolTipText("Benutzername fuer den Zugang
                    zur Datenbank.");
                dbnameLabel = new JLabel();
                dbnameLabel.setBounds(new Rectangle(15, 105, 151, 16));
                dbnameLabel.setText("Datenbankname:");
                dbnameLabel.setToolTipText("Name der Datenbank.");
                hostLabel = new JLabel();
                hostLabel.setBounds(new Rectangle(15, 75, 151, 16));
                hostLabel.setText("Host:");
                hostLabel.setToolTipText("Host der Datenbank.");
                origtableLabel = new JLabel();
                origtableLabel.setBounds(new Rectangle(15, 45, 151, 16));
```

```
                origtableLabel.setText("Temp-Tabelle:");
                origtableLabel.setToolTipText("Name der temporaeren Tabelle.");
                filepathLabel = new JLabel();
                filepathLabel.setBounds(new Rectangle(15, 15, 151, 16));
                filepathLabel.setText("Ergebnispfad:");
                filepathLabel.setToolTipText("Pfad, wo Ergebnisse gespeichert
                   werden.");
                configTab = new JPanel();
                configTab.setLayout(null);
                configTab.add(filepathLabel, null);
                configTab.add(getFilepathField(), null);
                configTab.add(origtableLabel, null);
                configTab.add(getOrigtableField(), null);
                configTab.add(hostLabel, null);
                configTab.add(getHostField(), null);
                configTab.add(dbnameLabel, null);
                configTab.add(getDbnameField(), null);
                configTab.add(BenutzernameLabel, null);
                configTab.add(getBenutzernameField(), null);
                configTab.add(passwortLabel, null);
                configTab.add(getPasswortField(), null);
                configTab.add(maxHeapSpaceLabel, null);
                configTab.add(getMaxHeapSpaceField(), null);
                configTab.add(getSpeichernConfigButton(), null);
                configTab.add(pfadConfLabel, null);
                configTab.add(getPfadConfField(), null);
                configTab.add(jmeterXMLLabel, null);
                configTab.add(getJmeterXMLField(), null);
                configTab.add(jmeterhomeLabel, null);
                configTab.add(getJmeterhomeField(), null);
                configTab.add(resultlogLabel, null);
                configTab.add(getResultlogField(), null);
                configTab.add(serverLabel, null);
                configTab.add(getServerTextField(), null);
                configTab.add(jtlPathLabel, null);
                configTab.add(getJtlPathField(), null);
                configTab.add(logsPathLabel, null);
                configTab.add(getLogsPathField(), null);
                configTab.add(jmeterVorlageLabel, null);
                configTab.add(getJmeterVorlageField(), null);
                configTab.add(jLabel, null);
                configTab.add(getLoadtestsField(), null);
        }
        return configTab;
}
/**
 * This method initializes messungsnameField
 *
 * @return javax.swing.JTextField
 */
private JTextField getMessungsnameField() {
        if (messungsnameField == null) {
                messungsnameField = new JTextField();
                messungsnameField.setText(ps.getProperty("messungsname"));
```

```
                messungsnameField.setBounds(new Rectangle(135, 15, 166, 16));
        }
        return messungsnameField;
}
/**
 * This method initializes keepAliveCheckBox
 *
 * @return javax.swing.JCheckBox
 */
private JCheckBox getKeepAliveCheckBox() {
        if (keepAliveCheckBox == null) {
                keepAliveCheckBox = new JCheckBox();
                keepAliveCheckBox.setSelected(Boolean.valueOf(
                      ps.getProperty("keepAlive")));
   keepAliveCheckBox.setBounds(new Rectangle(15, 105, 21, 21));
        }
        return keepAliveCheckBox;
}
/**
 * This method initializes modulComboBox
 *
 * @return javax.swing.JComboBox
 */
public static JComboBox getModulComboBox()
{
        if (modulComboBox == null)
        {
                modulComboBox = new JComboBox();
                modulComboBox.setBounds(new Rectangle(480, 180, 91, 16));
                Vector <String> v = new Vector <String>();
                try
                {
             HttpdMPMConfReader mpmConfReader = new HttpdMPMConfReader();
                        v = mpmConfReader.getModules();
                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }
                Iterator itr = v.iterator();
                while(itr.hasNext())
                {
                        modulComboBox.addItem(itr.next());
                }
                modulComboBox.addActionListener(anActionListener);
        }
        return modulComboBox;
}
/**
 * This method initializes konsoleArea
 *
 * @return javax.swing.JTextArea
 */
private JTextArea getKonsoleArea() {
```

```
        if (konsoleArea == null) {
                konsoleArea = new JTextArea();
                konsoleArea.setText("Bitte stellen Sie Ihre Werte nach Ihren
                  Wuenschen ein.\nSie koennen dann anschliessend eine Messung
                  starten.\n\n" + "Ueberpruefen Sie vor dem Start einer Messung
                  auch den Tab 'Config', ob dort alles richtig eingestellt
                  ist.\n\n" + "Nach einer Messung koennen Sie sich die
                  Auswertung im Tab 'Ergebnisse' ansehen bzw. erstellen.\n\n"
               + "Benoetigen Sie weitere Hilfe, so zeigen Sie mit dem
                  Mauszeiger auf ein Feld und es erscheint eine kurze
                  Beschreibung.\n" + " Oder Sie klicken auf den Tab 'Hilfe' um
                  Zugriff zum Handbuch zu erhalten.\n\n");
                konsoleArea.setEditable(false);
                konsoleArea.setLineWrap(true);
        }
        return konsoleArea;
}
/**
 * This method initializes cgiField
 *
 * @return javax.swing.JTextField
 */
private JTextField getCgiStartField() {
        if (cgiStartField == null) {
                cgiStartField = new JTextField();
                cgiStartField.setText(ps.getProperty("cgiStart"));
                cgiStartField.setBounds(new Rectangle(135, 180, 76, 16));
        }
        return cgiStartField;
}
/**
 * This method initializes cgiIntervall
 *
 * @return javax.swing.JTextField
 */
private JTextField getCgiIntervall() {
        if (cgiIntervall == null) {
                cgiIntervall = new JTextField();
                cgiIntervall.setText(ps.getProperty("cgiStep"));
                cgiIntervall.setBounds(new Rectangle(345, 180, 76, 16));
        }
        return cgiIntervall;
}
/**
 * This method initializes startButton
 *
 * @return javax.swing.JButton
 */
private JButton getStartButton() {
        if (startButton == null)
        {
                startButton = new JButton();
                startButton.setBounds(new Rectangle(675, 195, 151, 31));
                startButton.setText("Messung starten");
```

```
                startButton.addActionListener(anActionListener);
        }
        return startButton;
}
/**
 * This method initializes abbruchButton
 *
 * @return javax.swing.JButton
 */
private JButton getAbbruchButton() {
        if (abbruchButton == null) {
                abbruchButton = new JButton();
                abbruchButton.setBounds(new Rectangle(675, 240, 151, 31));
                abbruchButton.setText("Messung abbrechen");
                abbruchButton.addActionListener(anActionListener);
        }
        return abbruchButton;
}
/**
 * This method initializes auswertenCheckBox
 *
 * @return javax.swing.JCheckBox
 */
private JCheckBox getAuswertenCheckBox() {
        if (auswertenCheckBox == null) {
                auswertenCheckBox = new JCheckBox();
                auswertenCheckBox.setSelected(Boolean.valueOf(
                                ps.getProperty("autoAuswerten")));
                auswertenCheckBox.setBounds(new Rectangle(15, 75, 21, 21));
        }
        return auswertenCheckBox;
}
/**
 * This method initializes wiederholenField
 *
 * @return javax.swing.JTextField
 */
private JTextField getWiederholenField() {
        if (wiederholenField == null) {
                wiederholenField = new JTextField();
                wiederholenField.setText(ps.getProperty("anzahlWiederholungen"));
                wiederholenField.setBounds(new Rectangle(210, 45, 91, 16));
        }
        return wiederholenField;
}
/**
 * This method initializes filepathField
 *
 * @return javax.swing.JTextField
 */
private JTextField getFilepathField() {
        if (filepathField == null) {
                filepathField = new JTextField();
                filepathField.setText(ps.getProperty("filepath"));
```

```
                filepathField.setBounds(new Rectangle(180, 15, 316, 16));
        }
        return filepathField;
}
/**
 * This method initializes origtableField
 *
 * @return javax.swing.JTextField
 */
private JTextField getOrigtableField() {
        if (origtableField == null) {
                origtableField = new JTextField();
                origtableField.setText(ps.getProperty("origtable"));
                origtableField.setBounds(new Rectangle(180, 45, 316, 16));
        }
        return origtableField;
}
/**
 * This method initializes hostField
 *
 * @return javax.swing.JTextField
 */
private JTextField getHostField() {
        if (hostField == null) {
                hostField = new JTextField();
                hostField.setText(ps.getProperty("host"));
                hostField.setBounds(new Rectangle(180, 75, 316, 16));
        }
        return hostField;
}
/**
 * This method initializes dbnameField
 *
 * @return javax.swing.JTextField
 */
private JTextField getDbnameField() {
        if (dbnameField == null) {
                dbnameField = new JTextField();
                dbnameField.setText(ps.getProperty("dbname"));
                dbnameField.setBounds(new Rectangle(180, 105, 316, 16));
        }
        return dbnameField;
}
/**
 * This method initializes benutzernameField
 *
 * @return javax.swing.JTextField
 */
private JTextField getBenutzernameField() {
        if (benutzernameField == null) {
                benutzernameField = new JTextField();
                benutzernameField.setText(ps.getProperty("user"));
                benutzernameField.setBounds(new Rectangle(180, 135, 316, 16));
        }
```

```
        return benutzernameField;
}
/**
 * This method initializes passwortField
 *
 * @return javax.swing.JTextField
 */
private JTextField getPasswortField() {
        if (passwortField == null) {
                passwortField = new JTextField();
                passwortField.setText(ps.getProperty("password"));
                passwortField.setBounds(new Rectangle(180, 165, 316, 16));
        }
        return passwortField;
}
/**
 * This method initializes maxHeapSpaceField
 *
 * @return javax.swing.JTextField
 */
private JTextField getMaxHeapSpaceField() {
        if (maxHeapSpaceField == null) {
                maxHeapSpaceField = new JTextField();
                maxHeapSpaceField.setText(ps.getProperty("maxheap"));
                maxHeapSpaceField.setBounds(new Rectangle(180, 195, 316, 16));
        }
        return maxHeapSpaceField;
}
/**
 * This method initializes speichernConfigButton
 *
 * @return javax.swing.JButton
 */
private JButton getSpeichernConfigButton() {
        if (speichernConfigButton == null) {
                speichernConfigButton = new JButton();
          speichernConfigButton.setBounds(new Rectangle(15, 495, 121, 27));
                speichernConfigButton.setText("Speichern");
                speichernConfigButton.addActionListener(
                 new java.awt.event.ActionListener() {
                  public void actionPerformed(java.awt.event.ActionEvent e)
                   {
                       try
                      {
                       ps.setProperty("filepath", filepathField.getText());
                       ps.setProperty("origtable", origtableField.getText());
                       ps.setProperty("host", hostField.getText());
                       ps.setProperty("dbname", dbnameField.getText());
                       ps.setProperty("user", benutzernameField.getText());
                       ps.setProperty("password", passwortField.getText());
                       ps.setProperty("maxheap", maxHeapSpaceField.getText());
                       ps.setProperty("confPfad",pfadConfField.getText());
                       ps.setProperty("jmeterxml", jmeterXMLField.getText());
                       ps.setProperty("jmeterhome", jmeterhomeField.getText());
```

```
                        ps.setProperty("resultlog", resultlogField.getText());
                        ps.setProperty("server", serverTextField.getText());
                        ps.setProperty("jtlpath", jtlPathField.getText());
                        ps.setProperty("logspath", logsPathField.getText());
                ps.setProperty("jmetervorlage", jmeterVorlageField.getText());
                       ps.setProperty("loadtests", jLoadtestsField.getText());
                  konsoleArea.append("actionPerformed(): Speichern\n");
                      }
                       catch(Exception e1)
                      {
                         konsoleArea.append("Fehler: " + e1.getMessage());
                         autoScroll();
                       }
                    }
                  });
        }
        return speichernConfigButton;
}
/**
 * This method initializes konsoleScrollPane
 *
 * @return javax.swing.JScrollPane
 */
private JScrollPane getKonsoleScrollPane() {
        if (konsoleScrollPane == null) {
                konsoleScrollPane = new JScrollPane(getKonsoleArea());
                konsoleScrollPane.setBounds(new Rectangle(15, 285, 811, 241));
                konsoleScrollPane.setViewportView(getKonsoleArea());
        }
        return konsoleScrollPane;
}
/**
 * This method initializes pfadConfField
 *
 * @return javax.swing.JTextField
 */
private JTextField getPfadConfField()
{
        if (pfadConfField == null)
        {
                pfadConfField = new JTextField();
                pfadConfField.setText(ps.getProperty("confPfad"));
                pfadConfField.setBounds(new Rectangle(180, 225, 316, 16));
        }
        return pfadConfField;
}

/**
 * This method initializes jmeterXMLField
 *
 * @return javax.swing.JTextField
 */
private JTextField getJmeterXMLField()
{
```

```
        if (jmeterXMLField == null)
        {
                jmeterXMLField = new JTextField();
                jmeterXMLField.setText(ps.getProperty("jmeterxml"));
                jmeterXMLField.setBounds(new Rectangle(180, 255, 316, 16));
        }
        return jmeterXMLField;
}
/**
 * This method initializes jmeterhomeField
 *
 * @return javax.swing.JTextField
 */
private JTextField getJmeterhomeField()
{
        if (jmeterhomeField == null)
        {
                jmeterhomeField = new JTextField();
                jmeterhomeField.setText(ps.getProperty("jmeterhome"));
                jmeterhomeField.setBounds(new Rectangle(180, 285, 316, 16));
        }
        return jmeterhomeField;
}
/**
 * This method initializes resultlogField
 *
 * @return javax.swing.JTextField
 */
private JTextField getResultlogField()
{
        if (resultlogField == null)
        {
                resultlogField = new JTextField();
                resultlogField.setText(ps.getProperty("resultlog"));
                resultlogField.setBounds(new Rectangle(180, 315, 316, 16));
        }
        return resultlogField;
}
/**
 * This method initializes cgiStopField
 *
 * @return javax.swing.JTextField
 */
private JTextField getCgiStopField() {
        if (cgiStopField == null) {
                cgiStopField = new JTextField();
                cgiStopField.setText(ps.getProperty("cgiStop"));
                cgiStopField.setBounds(new Rectangle(240, 180, 76, 16));
        }
        return cgiStopField;
}
/**
 * This method initializes ServerParamsList
 *
```

```
             * @return javax.swing.JList
             */
            public static JList getServerParamsList()
            {
                    if (ServerParamsList == null)
                    {
                            ServerParamsList = new JList();
                            ServerParamsList.addListSelectionListener(anListSelectionListener);
                    }
                    try
                    {
                            HttpdMPMConfReader mpmConfReader = new HttpdMPMConfReader();
                            ServerParamsList.setListData(mpmConfReader.getParams(
                            modulComboBox.getSelectedItem().toString()));
                            ServerParamsList.setSelectedIndex(0);
                    } catch (Exception e) {
                            System.out.println("Fehler in GUI.getServerParamsList: "
                             + e.getMessage());
                            e.printStackTrace();
                    }
                    return ServerParamsList;
            }
            /**
             * This method initializes startWertField
             *
             * @return javax.swing.JTextField
             */
            public static JTextField getStartWertField() {
                    if (startWertField == null) {
                            startWertField = new JTextField();
                            startWertField.setBounds(new Rectangle(480, 60, 76, 16));
                    }
                    try
                    {
    startWertField.setText(ps.getProperty(modulComboBox.getSelectedItem().toString()
     + ServerParamsList.getSelectedValue().toString() + "Start"));
                    }
                    catch(Exception e)
                    {
                            e.printStackTrace();
                            System.out.println("Fehler in GUI.getStartWertField: "
                               + e.getMessage());
                    }
                    return startWertField;
            }
            /**
             * This method initializes stopWertField
             *
             * @return javax.swing.JTextField
             */
            public static JTextField getStopWertField() {
                    if (stopWertField == null) {
                            stopWertField = new JTextField();
                            stopWertField.setBounds(new Rectangle(480, 90, 76, 16));
```

```
                }
                stopWertField.setText(ps.getProperty(modulComboBox.getSelectedItem().toString()
+ ServerParamsList.getSelectedValue().toString() + "Stop"));
                return stopWertField;
        }
        /**
         * This method initializes stepField
         *
         * @return javax.swing.JTextField
         */
        public static JTextField getStepField() {
                if (stepField == null) {
                        stepField = new JTextField();
                        stepField.setBounds(new Rectangle(480, 120, 76, 16));
                }
                stepField.setText(ps.getProperty(modulComboBox.getSelectedItem().toString()
+ ServerParamsList.getSelectedValue().toString() + "Step"));
                return stepField;
        }
        /**
         * This method initializes speichereParamsButton
         *
         * @return javax.swing.JButton
         */
        private JButton getSpeichereParamsButton() {
                if (speichereParamsButton == null) {
                        speichereParamsButton = new JButton();
                        speichereParamsButton.setBounds(
                           new Rectangle(675, 150, 151, 31));
                        speichereParamsButton.setText("Speichern");
                        speichereParamsButton.addActionListener(anActionListener);
                }
                return speichereParamsButton;
        }
        /**
         * This method initializes serverParamsScrollPane
         *
         * @return javax.swing.JScrollPane
         */
        private JScrollPane getServerParamsScrollPane() {
                if (serverParamsScrollPane == null)
                {
                        serverParamsScrollPane = new JScrollPane(getServerParamsList());
                        serverParamsScrollPane.setBounds(
                           new Rectangle(315, 45, 151, 91));
                        //serverParamsScrollPane.setViewportView(getServerParamsList());
                }
                return serverParamsScrollPane;
        }
        /**
         * This method initializes serverTextField
         *
         * @return javax.swing.JTextField
         */
```

```
private JTextField getServerTextField() {
        if (serverTextField == null) {
                serverTextField = new JTextField();
                serverTextField.setText(ps.getProperty("server"));
                serverTextField.setBounds(new Rectangle(180, 345, 316, 16));
        }
        return serverTextField;
}
/**
 * This method initializes jtlPathField
 *
 * @return javax.swing.JTextField
 */
private JTextField getJtlPathField() {
        if (jtlPathField == null) {
                jtlPathField = new JTextField();
                jtlPathField.setText(ps.getProperty("jtlpath"));
                jtlPathField.setBounds(new Rectangle(180, 375, 316, 16));
        }
        return jtlPathField;
}
/**
 * This method initializes logsPathField
 *
 * @return javax.swing.JTextField
 */
private JTextField getLogsPathField() {
        if (logsPathField == null) {
                logsPathField = new JTextField();
                logsPathField.setText(ps.getProperty("logspath"));
                logsPathField.setBounds(new Rectangle(180, 405, 316, 16));
        }
        return logsPathField;
}
/**
 * This method initializes jmeterVorlageField
 *
 * @return javax.swing.JTextField
 */
private JTextField getJmeterVorlageField() {
        if (jmeterVorlageField == null) {
                jmeterVorlageField = new JTextField();
                jmeterVorlageField.setText(ps.getProperty("jmetervorlage"));
                jmeterVorlageField.setBounds(new Rectangle(180, 435, 316, 16));
        }
        return jmeterVorlageField;
}
private JTextField getLoadtestsField() {
        if (jLoadtestsField == null) {
                jLoadtestsField = new JTextField();
                jLoadtestsField.setText(ps.getProperty("loadtestspath"));
                jLoadtestsField.setBounds(new Rectangle(180, 465, 316, 16));
        }
        return jLoadtestsField;
```

```
}
/**
 * This method initializes jmeterParamsScrollPane
 *
 * @return javax.swing.JScrollPane
 */
private JScrollPane getJmeterParamsScrollPane() {
        if (jmeterParamsScrollPane == null)
        {
                jmeterParamsScrollPane = new JScrollPane(getJmeterParamsList());
                jmeterParamsScrollPane.setBounds(new Rectangle(585, 45, 151, 91));
        }
        return jmeterParamsScrollPane;
}
/**
 * This method initializes jmeterParamsList
 *
 * @return javax.swing.JList
 */
public JList getJmeterParamsList()
{
        if (jmeterParamsList == null)
        {
                jmeterParamsList = new JList();
                jmeterParamsList.addListSelectionListener(anListSelectionListener);
        }
        try
        {
                JMXReader jmxReader = new JMXReader();
                jmeterParamsList.setListData(jmxReader.getParams());
                jmeterParamsList.setSelectedIndex(0);
        } catch (Exception e) {
                System.out.println("Fehler in GUI.getServerParamsList: "
                  + e.getMessage());
                e.printStackTrace();
        }
        return jmeterParamsList;
}
/**
 * This method initializes jmeterStartWertField
 *
 * @return javax.swing.JTextField
 */
public static JTextField getJmeterStartWertField() {
        if (jmeterStartWertField == null) {
                jmeterStartWertField = new JTextField();
                jmeterStartWertField.setBounds(new Rectangle(750, 60, 76, 16));
        }
        try
        {
            jmeterStartWertField.setText(ps.getProperty(
               jmeterParamsList.getSelectedValue().toString() + "Start"));
        }
        catch(Exception e)
```

```
        {
                e.printStackTrace();
                System.out.println("Fehler in GUI.getJmeterStartWertField: "
                    + e.getMessage());
        }
        return jmeterStartWertField;
}
/**
 * This method initializes jmeterStopWertField
 *
 * @return javax.swing.JTextField
 */
public static JTextField getJmeterStopWertField()
{
        if (jmeterStopWertField == null)
        {
                jmeterStopWertField = new JTextField();
                jmeterStopWertField.setBounds(new Rectangle(750, 90, 76, 16));
        }
        try
        {
           jmeterStopWertField.setText(ps.getProperty(
             jmeterParamsList.getSelectedValue().toString() + "Stop"));
        }
        catch(Exception e)
        {
                System.out.println("Fehler in GUI.getJmeterStopWertField: "
                   + e.getMessage());
        }
        return jmeterStopWertField;
}
/**
 * This method initializes jmeterStepField
 *
 * @return javax.swing.JTextField
 */
public static JTextField getJmeterStepField()
{
        if (jmeterStepField == null)
        {
                jmeterStepField = new JTextField();
                jmeterStepField.setBounds(new Rectangle(750, 120, 76, 16));
        }
        try
        {
         jmeterStepField.setText(ps.getProperty(
          jmeterParamsList.getSelectedValue().toString() + "Step"));
        }
        catch(Exception e)
        {
                System.out.println("Fehler in GUI.getJmeterStepField: "
                  + e.getMessage());
        }
        return jmeterStepField;
```

```
}
/**
 * This method initializes querAuswertungTab
 *
 * @return javax.swing.JPanel
 */
private JPanel getQuerAuswertungTab() {
        if (querAuswertungTab == null) {
                GridBagConstraints gridBagConstraints22 =
                     new GridBagConstraints();
                gridBagConstraints22.gridx = 1;
                gridBagConstraints22.gridy = 0;
                jLabel4 = new JLabel();
                jLabel4.setText("Messpunkte der Tabelle:");
                GridBagConstraints gridBagConstraints12 =
                   new GridBagConstraints();
                gridBagConstraints12.gridx = 0;
                gridBagConstraints12.gridy = 0;
                jLabel3 = new JLabel();
                jLabel3.setText("Tabelle der anzuzeigenden Quantile:");
                GridBagConstraints gridBagConstraints11 =
                  new GridBagConstraints();
                gridBagConstraints11.gridx = 1;
                gridBagConstraints11.gridy = 3;
                GridBagConstraints gridBagConstraints21 =
                    new GridBagConstraints();
                gridBagConstraints21.fill = GridBagConstraints.BOTH;
                gridBagConstraints21.gridy = 1;
                gridBagConstraints21.weightx = 1.0;
                gridBagConstraints21.weighty = 1.0;
                gridBagConstraints21.gridx = 0;
                GridBagConstraints gridBagConstraints1 =
                        new GridBagConstraints();
                gridBagConstraints1.gridx = 0;
                gridBagConstraints1.gridy = 3;
                GridBagConstraints gridBagConstraints2 =
                     new GridBagConstraints();
                gridBagConstraints2.fill = GridBagConstraints.BOTH;
                gridBagConstraints2.gridy = 1;
                gridBagConstraints2.weightx = 1.0;
                gridBagConstraints2.weighty = 1.0;
                gridBagConstraints2.gridx = 1;
                GridBagConstraints gridBagConstraints =
                    new GridBagConstraints();
                gridBagConstraints.fill = GridBagConstraints.VERTICAL;
                gridBagConstraints.gridy = 2;
                gridBagConstraints.weightx = 1.0;
                gridBagConstraints.gridx = 0;
                querAuswertungTab = new JPanel();
                querAuswertungTab.setLayout(new GridBagLayout());
                querAuswertungTab.add(getTableQuerComboBox(),
                       gridBagConstraints);
                querAuswertungTab.add(getMesspunkteQuerList(),
                   gridBagConstraints2);
```

```
                querAuswertungTab.add(getAddenButton(), gridBagConstraints1);
                querAuswertungTab.add(getQuantileList(), gridBagConstraints21);
                querAuswertungTab.add(getGraphQuerButton(),
                      gridBagConstraints11);
                querAuswertungTab.add(jLabel3, gridBagConstraints12);
                querAuswertungTab.add(jLabel4, gridBagConstraints22);
        }
        return querAuswertungTab;
}
/**
 * This method initializes tableQuerComboBox
 *
 * @return javax.swing.JComboBox
 */
public static JComboBox getTableQuerComboBox() {
        if (tableQuerComboBox == null)
        {
                tableQuerComboBox = new JComboBox();
                Vector <String> v = new Vector <String>();
                if(con == null)
                {
                        try
                        {
                                if(db == null)
                                {
                                        db = new DBConnection();
                                }
                                con = db.getConnection();
                        }
                        catch (Exception e1)
                        {
                          JOptionPane.showMessageDialog( null,
                          "Fehler beim Verbinden mit der Datenbank",
                          "", JOptionPane.ERROR_MESSAGE);
                          e1.printStackTrace();
                        }
                }
                try
                {
                                v = db.getTables(con);
                }
                catch (SQLException e)
                {
                        e.printStackTrace();
                }
                Iterator itr = v.iterator();
                tableQuerComboBox.removeAllItems();
                while(itr.hasNext())
                {
                        tableQuerComboBox.addItem(itr.next());
                }
                tableQuerComboBox.addActionListener(anActionListener);
                tableQuerComboBox.setActionCommand("changeQuerTable");
        }
```

```
                return tableQuerComboBox;
        }
        public static void fillList(String table) throws SQLException
        {
                if(messpunkteQuerList != null && con != null)
                {
                        if(db == null)
                        {
                                db = new DBConnection();
                        }
                        messpunkteQuerList.setListData(
                            db.getMesspunktNamen(con, table));
                        messpunkteQuerList.setSelectedIndex(0);
                }
        }
        /**
         * This method initializes messpunkteQuerList
         *
         * @return javax.swing.JList
         */
        public static JList getMesspunkteQuerList()
        {
                if (messpunkteQuerList == null)
                {
// Auswahlliste erzeugen und mit Werten aus der Datenbank fuellen
                        messpunkteQuerList = new JList();
                        try
                        {
                                getTableQuerComboBox();
                                fillList(tableQuerComboBox.getSelectedItem().toString());
                        }
                        catch (SQLException e)
                        {
                         JOptionPane.showMessageDialog( null,
                          "Fehler beim Lesen der Messpunknamen aus der Datenbank",
                          "", JOptionPane.ERROR_MESSAGE);
                         e.printStackTrace();
                        }
                        JScrollPane listPane = new JScrollPane(
                          messpunkteQuerList,
                          JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
                         JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED );
                        listPane.setBorder(BorderFactory.createLineBorder(Color.lightGray, 2));
                }
                return messpunkteQuerList;
        }
        /**
         * This method initializes addenButton
         *
         * @return javax.swing.JButton
         */
        private JButton getAddenButton() {
                if (addenButton == null) {
                        addenButton = new JButton("Hinzufuegen");
```

```
                        addenButton.addActionListener(anActionListener);
                }
                return addenButton;
        }
        /**
         * This method initializes quantileList
         *
         * @return javax.swing.JList
         */
        public static JList getQuantileList()
        {
                if (quantileList == null)
                {
                        quantileList = new JList();
                }
                return quantileList;
        }
        /**
         * This method initializes graphQuerButton
         *
         * @return javax.swing.JButton
         */
        public static JButton getGraphQuerButton() {
                if (graphQuerButton == null) {
                        graphQuerButton = new JButton("Graph zeichnen");
                        graphQuerButton.addActionListener(anActionListener);
                }
                return graphQuerButton;
        }
        /**
         * This method initializes processingDelayStartField
         *
         * @return javax.swing.JTextField
         */
        private JTextField getProcessingDelayStartField() {
                if (processingDelayStartField == null) {
                        processingDelayStartField = new JTextField();
    processingDelayStartField.setText(ps.getProperty("processingDelayStart"));
                        processingDelayStartField.setBounds(
                           new Rectangle(135, 210, 76, 16));
                }
                return processingDelayStartField;
        }
        /**
         * This method initializes processingDelayStopField
         *
         * @return javax.swing.JTextField
         */
        private JTextField getProcessingDelayStopField() {
                if (processingDelayStopField == null) {
                        processingDelayStopField = new JTextField();
processingDelayStopField.setText(ps.getProperty("processingDelayStop"));
                        processingDelayStopField.setBounds(
                               new Rectangle(240, 210, 76, 16));
```

```
        }
        return processingDelayStopField;
}
/**
 * This method initializes processingDelayStepField
 *
 * @return javax.swing.JTextField
 */
private JTextField getProcessingDelayStepField() {
        if (processingDelayStepField == null) {
                processingDelayStepField = new JTextField();
                processingDelayStepField.setText(ps.getProperty("processingDelayStep"));
                processingDelayStepField.setBounds(
                     new Rectangle(345, 210, 76, 16));
        }
        return processingDelayStepField;
}
/**
 * This method initializes jRadioButton1
 *
 * @return javax.swing.JRadioButton
 */
private JRadioButton getJRadioButton1() {
        if (jRadioButton1 == null) {
                jRadioButton1 = new JRadioButton("Methode 1");
                jRadioButton1.setBounds(
                   new Rectangle(135, 240, 21, 16));
        }
        return jRadioButton1;
}
/**
 * This method initializes jRadioButton2
 *
 * @return javax.swing.JRadioButton
 */
private JRadioButton getJRadioButton2() {
        if (jRadioButton2 == null) {
                jRadioButton2 = new JRadioButton("Methode 2");
                jRadioButton2.setBounds(new Rectangle(210, 240, 21, 16));
        }
        return jRadioButton2;
}
/**
 * @param args
 */
public static void main(String[] args)
{
        // TODO Auto-generated method stub
        try
        {
                ps = PropertiesSingleton.getInstance("config/config.properties");
        }
        catch(IOException e)
        {
```

```
                        e.getMessage();
                        e.printStackTrace();
                }
                GUI thisClass = new GUI();
                thisClass.setDefaultCloseOperation(EXIT_ON_CLOSE);
                thisClass.setVisible(true);
        }
        /**
         * This is the default constructor
         */
        public GUI() {
                super();
                initialize();
        }
        /**
         * This method initializes this
         *
         * @return void
         */
        private void initialize()
        {
                this.setSize(849, 600);
                this.setContentPane(getJTabbedPane());
                this.setTitle("JDragicDataMining");
                this.validate();
        }
        /*
         * Diese Methode sorgt fuer den Autoscroll der Konsole
         * Sie muss immer nach dem hinzufuegen von Text aufgerufen werden
         */
        public static void autoScroll()
        {
                konsoleScrollPane.validate();
                int max = konsoleScrollPane.getVerticalScrollBar().getMaximum();
                konsoleScrollPane.getVerticalScrollBar().setValue(max);
                konsoleScrollPane.repaint();
        }
}
```

ListSelectionListenerClass

```
package de.christianbell.apmfw.jdragicdatamining.gui;
import javax.swing.event.ListSelectionEvent;
/*
  * Diese Klasse repraesentiert einen Listen-Abhoerer
  * Sie handelt Events von Listen
  * @author Christian Bell
  */
public class ListSelectionListenerClass implements javax.swing.event.ListSelectionListener
{
        public void valueChanged(ListSelectionEvent event)
        {
                if(event.getSource() == GUI.ServerParamsList)
                {
```

```
                        if(event.getValueIsAdjusting() == true)
                        {
                                GUI.getStartWertField();
                                GUI.getStopWertField();
                                GUI.getStepField();
                        }
                }
                else if(event.getSource() == GUI.jmeterParamsList)
                {
                        if(event.getValueIsAdjusting() == true)
                        {
                                GUI.getJmeterStartWertField();
                                GUI.getJmeterStopWertField();
                                GUI.getJmeterStepField();
                        }
                }
        }
}
```

7.1.6 Das paket Hilfsklassen

```
package de.christianbell.apmfw.jdragicdatamining.hilfsklassen;
import java.util.Date;
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Locale;
/*
 * Diese Klasse wandelt ein Datum in einen Timestamp um
 * @author Christian Bell
 */
public class DateConverter
{
   public long convertLogDate (String dateStr)
   {
     Locale locale;
     long ret = -1L;
     dateStr = dateStr.replace("Mär", "Mrz");
     //dateStr = dateStr.replace("Mo", "Mon");
     //dateStr = dateStr.replace("Di", "Die");
     SimpleDateFormat sDate;
    if(dateStr.startsWith("Mon") || dateStr.startsWith("Tue")
        || dateStr.startsWith("Wed") || dateStr.startsWith("Thu")
        || dateStr.startsWith("Fri") || dateStr.startsWith("Sut")
        || dateStr.startsWith("Sun"))
     {
       locale = Locale.ENGLISH;
       sDate = new SimpleDateFormat("EEE MMM dd HH:mm:ss zzzz yyyy", locale);
     }
     else
     {
       locale = Locale.GERMAN;
       sDate = new SimpleDateFormat("EE MMM dd HH:mm:ss z yyyy", locale);
     }
```

```
      Date dt;
      try
      {
        dt = sDate.parse(dateStr);
        ret = dt.getTime();
      }
      catch (ParseException e)
      {
        e.printStackTrace();
      }
      return ret;
    }
}
```

DBConnection

```
package de.christianbell.apmfw.jdragicdatamining.hilfsklassen;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Vector;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
/** Diese Klasse laedt eine CSV in eine Datenbank, fuehrt Berechnungen aus,
 * speichert die Berechnungen in einer neuen Datenbank, und erzeugt eine weitere CSV
 *
 * @author Christian Bell - http://www.christianbell.de
 * @version 1.0beta
 */
public class DBConnection
{
        public static PropertiesSingleton ps = null;
        /**Name der Tabelle, in welche die CSV geschrieben werden soll
         *
         */
        private String origTable;
        /**
         * Host der Datenbank
         */
        private String host;
        /**
         * Datenbankname
         */
        private String dbname;
        /**
         * Username der Datenbank
         */
        private String user;
        /**
         * Passwort der Datenbank
         */
        private String password;
        public DBConnection()
        {
```

```
            try
            {
             ps = PropertiesSingleton.getInstance("config/config.properties");
            }
            catch(Exception e)
            {
              System.out.println("GetInstance: " + e.getMessage());
              e.printStackTrace();
            }
            try
            {
              setDbname(ps.getProperty("dbname"));
              setHost(ps.getProperty("host"));
              setOrigTable(ps.getProperty("origtable"));
              setPassword(ps.getProperty("password"));
              setUser(ps.getProperty("user"));
            }
            catch(Exception e)
            {
              System.out.println("Setters: " + e.getMessage());
              e.printStackTrace();
            }
        }
/** Diese Funktion gibt alle Messpunkte sortiert in einem ResultSet zurueck
*
* @param messpunkt Der Messpunkt, dessen Werte ins ResultSet geschrieben werden sollen
* @return ResultSet enthaelt die Werte der Datenbank, welche zu messpunkt gehoeren
* @exception SQLException
*/
public ResultSet getAllSections(String messpunkt, Statement statement,
            String state, String modul)
{
  ResultSet result        = null;
  try
  {
    GUI.konsoleArea.append("Sortierte Auswahl von " + messpunkt + "!");
    GUI.autoScroll();
    if(modul.contains("worker"))
      result  = statement.executeQuery("Select * FROM " + origTable
         + " WHERE MFWPOINT='" +messpunkt + "' AND STATE='" + state
         + "' ORDER BY 'TID', 'PID', 'TSTAMP'");
    else result = statement.executeQuery("Select * FROM " + origTable
              + " WHERE MFWPOINT='" +messpunkt + "' AND STATE='" + state
              + "' ORDER BY 'PID', 'TSTAMP'");
       return result;
  }
  catch (SQLException e)
  {
   System.out.println("Ein Fehler ist aufgetreten in DBConnection.getAllSections(): "
            + e.getMessage());
   return null;
  }
}
    /** Diese Methode loescht die uebergebenen Tabelle
```

```
*
* @param table Der Name der Tabelle, welche geloescht werden soll
* @exception SQLException
*/
   public void cleanTable(String table, Statement statement)
   {
           try
           {
             statement.executeUpdate("DROP TABLE IF EXISTS " +table);
           }
           catch (SQLException e)
           {
            System.out.println("Ein Fehler ist aufgetreten in
               DBConnection.cleanTable(): " + e.getMessage());
           }
   }
/** Diese Methode stellt eine Verbindung zur Datenbank her
*
* @param theConnection Die Verbindung
* @return Statement Statement wird zurueck geliefert
* @exception SQLException
* @exception ClassNotFoundException
* @exception InstantiationException
* @exception IllegalAccessException
*/
   public Statement connect(Connection theConnection)
   {
           if (theConnection == null)
           {
                   try
                   {
                     Class.forName("org.gjt.mm.mysql.Driver").newInstance();
                     theConnection = java.sql.DriverManager.getConnection(host
                           + dbname, user , password);
                   }
                   catch (ClassNotFoundException e1)
                   {
                      System.out.println("ClassNotFoundException in
                       DBConnection.connect(): " + e1.getMessage());
                      return null;
                   }
                   catch(InstantiationException e2)
                   {
                     System.out.println("InstantiationException in
                       DBConnection.connect(): " + e2.getMessage());
                     return null;
                   }
                   catch(SQLException e3)
                   {
                     System.out.println("SQLException in DBConnection.connect(): "
                        + e3.getMessage());
                     System.out.println("SQLException in DBConnection.connect(): "
                        + e3.getStackTrace());
                     return null;
```

```
                }
                catch(IllegalAccessException e4)
                {
                 System.out.println("IllegalAccessException in
                     DBConnection.connect(): " + e4.getMessage());
                 return null;
                }
        }
        try
        {
                theConnection.setAutoCommit(false);
                return theConnection.createStatement();
        }
        catch(SQLException sqlex)
        {
          System.out.println("SQLException in DBConnection.connect(): "
                    + sqlex.getMessage());
          return null;
        }
}
/**
 * Stellt eine Verbindung zur DB her
 *
 * @return Verbundung zu DB
 * @throws InstantiationException
 * @throws IllegalAccessException
 * @throws ClassNotFoundException
 * @throws SQLException
 */
public Connection getConnection() throws InstantiationException,
              IllegalAccessException,
          ClassNotFoundException, SQLException
{
        Connection connection = null;
        Class.forName("org.gjt.mm.mysql.Driver").newInstance();
  connection = java.sql.DriverManager.getConnection(host + dbname,
            user , password);
        connection.setAutoCommit(false);
        return connection;
}
/**
 * Liefert alle in der DB vorhandenen Messpunktennamen
 *
 * @param con Verbindung zu DB
 * @return Vector mit den Namen der Messpunkten
 * @throws SQLException
 */
public Vector <String> getMesspunktNamen(Connection con,
               String table) throws SQLException
{
        if(con == null)
                return null;
        ResultSet rs = null;
        Vector <String> v = new Vector <String>();
```

```
                Statement statement = con.createStatement();
                if(table.contains("results"))
                {
                rs = statement.executeQuery("SELECT DISTINCT MFWPOINT FROM " + table);
                while(rs.next())
                        v.add(rs.getString(1));
                }
                else
                {
                        v.add(table);
                }
                return v;
        }
        /*
         * Diese Funktion gibt alle Tabellen der Datenbank zurueck
         * @param Connection die Verbindung
         * @return Vector Ein Vektor mit den Tabellennamen
         */
        public Vector <String> getTables(Connection connection) throws SQLException
        {
                if(connection == null)
                        return null;
                Statement statement = connection.createStatement();
                ResultSet rs = statement.executeQuery("SHOW TABLES");
                Vector <String> v = new Vector <String>();
                while(rs.next())
                        v.add(rs.getString(1));
                return v;
        }
        /** Diese Methode beendet die Verbindung zur Datenbank
    *
    * @param theConnection Die Verbindung
    * @exception SQLException
    */
        public void closeConnection(Connection theConnection)
        {
                if (theConnection != null)
                {
                        try
                        {
                                theConnection.close();
                        }
                        catch (SQLException e)
                        {
                         System.out.println("Ein Fehler ist in
                          DBConnection.closeConnection() aufgetreten: "
                             + e.getMessage());
                        }
                }
        }
        public String getDbname() {
                return dbname;
        }
        public void setDbname(String dbname) {
```

```
                this.dbname = dbname;
        }
        public String getHost() {
                return host;
        }
        public void setHost(String host) {
                this.host = host;
        }
        public String getPassword() {
                return password;
        }
        public void setPassword(String password) {
                this.password = password;
        }
        public String getUser() {
                return user;
        }
        public void setUser(String user) {
                this.user = user;
        }
        public String getOrigTable() {
                return origTable;
        }
        public void setOrigTable(String origTable) {
                this.origTable = origTable;
        }
}
```

ParameterParser

```
package de.christianbell.apmfw.jdragicdatamining.hilfsklassen;
/*
 * SplitParameter.java
 *
 * Created on 6. April 2002, 18:16
 */
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;
import java.util.Vector;
import java.util.regex.Pattern;
import java.util.regex.Matcher;
import java.util.Hashtable;
/**
 *Diese Klasse ist fuer das Ersetzen von Platzhaltern fuer Parameter
 * durch richtige Werte zustaendig
 * @author  André Westhoff, modified by Christian Bell
 */
public class ParameterParser {
    private Pattern einPattern;
    /** Creates a new instance of SplitParameter */
```

```
public ParameterParser(String CodeParamAnfang, String CodeParamEnde) {
    einPattern = Pattern.compile(ersetzeSonderzeichen(CodeParamAnfang)
       +  "(\\w+?)" + ersetzeSonderzeichen(CodeParamEnde));
}
public Vector<String> getParameterAlsVector(String input)
{
    return getParameterAlsVector(input,null);
}
private Vector<String> getParameterAlsVector(String input, Vector<String> V){
    if (V == null)
        V = new Vector<String>();
    Matcher m = einPattern.matcher(input);
    while (m.find())
        addParameterZuVector(V,m.group(1));
    return V;
}
public String[] getParameters(String input){
    String Ergebnis[] = null;
    Vector<String> V = getParameterAlsVector(input,null);
     if (!V.isEmpty()){
        Ergebnis = new String[V.size()];
        V.toArray((Object[])Ergebnis);
        }
    return Ergebnis;
}
public String[] getParameter (String[] Zeilen){
    String Ergebnis[] = null;
    Vector<String> V = new Vector<String>();
    if (Zeilen !=null){
        for (int i = 0; i < Zeilen.length; i++)
            V = getParameterAlsVector(Zeilen[i],V);
    }
    if (!V.isEmpty()){
        Ergebnis = new String[V.size()];
        Ergebnis = (String[])V.toArray((Object[])Ergebnis);
    }
    return Ergebnis;
}
public String ersetzeParameter(Hashtable<String, String> Ersetzungsliste,
           String input){
    String tmp;
    StringBuffer sb = new StringBuffer();
    Matcher m = einPattern.matcher(input);
    while (m.find()){
        tmp = (String)Ersetzungsliste.get(m.group(1));
        if (tmp == null)
            tmp = "";
        m.appendReplacement(sb, tmp);
    }
    m.appendTail(sb);
    return sb.toString();
}
public String[] ersetzeParameter(Hashtable<String, String> Ersetzungsliste,
        String[] input){
```

```
        String Ergebnis[] = new String[input.length];
        for (int i = 0; i < input.length; i++)
            Ergebnis[i] = ersetzeParameter(Ersetzungsliste,input[i]);
        return Ergebnis;
    }
    private void addParameterZuVector(Vector<String> V, String Parameter){
        for (int i = 0; i < V.size(); i++){
            if (V.elementAt(i).equals(Parameter))
                return;
        }
        V.add(Parameter);
    }
    private String ersetzeSonderzeichen(String input){
        input = input.replaceAll("[*]","[*]");
        return input;
    }
    public static void main(String[] args)
    {
           Hashtable<String,String> PErsetzliste = new Hashtable<String, String>();
           String Parametername="StartServers";
           String Wert="5";
           PErsetzliste.put(Parametername, Wert);
           File file = new File("Apache/httpd-mpm.conf");
           try
           {
                   FileReader fileReader = new FileReader(file);
                   BufferedReader buffReader = new BufferedReader(fileReader);
                   StringBuffer sb = new StringBuffer();
                   String line;
                   line = buffReader.readLine();
                   sb.append(line + "\n");
                   while(line != null)
                   {
                           line = buffReader.readLine();
                           sb.append(line +"\n");
                   }
                   buffReader.close();
                   ParameterParser paramPars = new ParameterParser("!--PARAM::", "--!");
                   String newString = paramPars.ersetzeParameter(PErsetzliste,
                     sb.toString());
                   File newFile = new File("Apache/new-mpm.conf");
                   FileWriter fileWriter = new FileWriter(newFile);
                   BufferedWriter buffWriter = new BufferedWriter(fileWriter);
                   buffWriter.append(newString);
                   buffWriter.close();
           }
           catch(IOException e)
           {
                   e.printStackTrace();
           }
    }
}
```

PropertiesSingleton

```
package de.christianbell.apmfw.jdragicdatamining.hilfsklassen;
import java.io.*;
import java.util.*;
/**
 * A <pattern> Singleton</pattern> for the management of 'global'
 * configuration data. <br/> Most applications need only a single
 * configuration file. They either reach a reference from object to
 * object or reload it many times. Here it is loaded and created only
 * once, but can be accessed by many objects.
 * @author      stephan@stephanwiesner.de, modified by Christian Bell
 * @date        1. Dezember 2002
 */
public class PropertiesSingleton implements java.io.Serializable
{
   private static final long serialVersionUID = 1L;
   /**  The singleton reference */
    private static PropertiesSingleton config;
    /**  The Properties object */
    private Properties props;
    /**  The path to the file containing the configuration */
    private String configFile = "";
    /**
     *  Constructor is a Singleton. <br>
     *
     *
     *@param  configFile  The path to the file containing the configuration
     *@exception  IOException  Description of the Exception
     *@throws  IOException     if the file can not be loaded
     */
    private PropertiesSingleton(String configFile) throws IOException
    {
       this.configFile = configFile;
       props = new Properties();
       DataInputStream in = new DataInputStream(
           new BufferedInputStream(new FileInputStream(configFile)));
       props.load(in);
    }
    /**
     *  Constructor is a Singleton. <br>
     *
     *
     *@param  baseClass The class is used to load the property file with
     *                  getClass().getResource()
     *@param  configFile The name of the file containing the configuration
     *@exception  IOException  If the file can not be loaded
     */
    private PropertiesSingleton(Class baseClass, String configFile)
           throws IOException
    {
       this.configFile = configFile;
       java.io.InputStream in = baseClass.getResourceAsStream(configFile);
       props = new Properties();
       props.load(in);
```

```
      System.out.println("Number:" + props.size());
   }
   /**
 * @return     The path to the file containing the configuration
 * @uml.property  name="configFile"
 */
   public String getConfigFile()
   {
      return this.configFile;
   }
   /**
    *@param  configFile    The path to the file containing the configuration
    *@return               Returns the Singleton instance <br>
    *
    *@throws  IOException  If a new instance is to be created and the
    *                      configuration file can not be loaded.
    */
   public static PropertiesSingleton getInstance(String configFile)
          throws IOException
   {
      if (config == null) { config = new PropertiesSingleton(configFile);  }
      return config;
   }
   /**
    *@param  baseClass  The class is used to load the property file with
    *                   getClass().getResource()
    *@param  configFile The name of the file containing the configuration
    *@return Returns the Singleton instance
    *
    *@throws  IOException  If a new instance is to be created and the
    *      configuration file can not be loaded.
    */
   public static PropertiesSingleton getInstance(Class baseClass,
      String configFile)  throws IOException
   {
      if (config == null)
      {
         config = new PropertiesSingleton(baseClass, configFile);
      }
      return config;
   }
   /**
    * This method throws a RuntimeException if the Singleton instance is not
    * yet instantiated.
    *@return Returns the Singleton instance
    *
    */
   public static PropertiesSingleton getInstance()
   {
      if (config == null)
      {
         throw new RuntimeException("No configuration file was provided, yet.");
      }
      return config;
```

```
    }
    /**
     *@param  key  The key whose value is to be returned.
     *@return      Returns the asked for Property. Null if it doesn't exist.
     *
     */
    public String getProperty(String key)
    {
       return props.getProperty(key);
    }
    /*
     * Diese Funktion speichert eine Property
     */
    public String setProperty(String key, String value)
    {
            String prevValue = getProperty(key);
            props.setProperty(key, value);
            try
            {
                    DataOutputStream out = new DataOutputStream(
                         new BufferedOutputStream(
                         new FileOutputStream(getConfigFile())));
                    props.store(out, "Wichtige Einstellungsparameter");
            }
            catch(IOException e)
            {
                    System.out.println(e.getMessage());
            }
                    return prevValue;
    }
}
```

7.1.7 Das Paket JMeter

JMeterTableGenerator

```
package de.christianbell.apmfw.jdragicdatamining.jmeter;
import java.util.ArrayList;
import core.files.FileGetter;
import core.parser.special.JTLParser;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
/*
  * Diese Klasse generiert eine Tabelle zu einer JTL-Datei
  * @author Christian Bell
  */
public class JMeterTableGenerator
{
         public static String tablePrefix = null;
         public static String path = null;
         public static PropertiesSingleton ps = null;
         /*
          * @auhtor Martin Gasse, modified by Christian Bell
          */
```

```
        public static void generateTablesFromJTLFiles()
        {
                ps = PropertiesSingleton.getInstance();
                //tablePrefix = ps.getProperty("messungsname");
                path = ps.getProperty("jtlpath");
                ArrayList<String> lJTLQuerys = new ArrayList<String>();
                JTLParser lJTLParser = new JTLParser();
                // Sucht nach Dateien im aktuellen Verzeichniss
                FileGetter lFileGetter = new FileGetter(path);
                lJTLParser.setFilesToParse(lFileGetter.getAllLogs());
                //lJTLQuerys =
                lJTLParser.generateSQLQueryList("");
                //lJTLParser.commit(lJTLQuerys);
                //lJTLQuerys = lJTLParser.generateSQLforaktivRequests("");
                //lJTLParser.commit(lJTLQuerys);
        }
}
```

JMXReader

```
package de.christianbell.apmfw.jdragicdatamining.jmeter;
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Iterator;
import java.util.Vector;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
/*
 * Diese Klasse liest eine JMX-Datei und sucht dort die Parameter raus
 * @author Christian Bell
 */
public class JMXReader
{
        public Vector <String> getParams() throws IOException
        {
            PropertiesSingleton ps =
               PropertiesSingleton.getInstance("config/config.properties");
                File jmeterFile = new File(ps.getProperty("jmetervorlage"));
                RandomAccessFile raFile = new RandomAccessFile(jmeterFile, "r");
                String currentLine = "";
                Vector <String> v = new Vector <String>();
                while(null != (currentLine = raFile.readLine()))
                {
                  if(currentLine.contains("!--PARAM::")
                  && !currentLine.contains("cgizeit")
                  && !currentLine.contains("processingDelay"))
                        {
                                String[] strArr = currentLine.split("::");
                                String[] strArr2 = strArr[1].split("--!");
                                v.add(strArr2[0]);
                        }
                }
                return v;
        }
```

```
}
```

7.1.8 Das Paket Logs

LogsTableGenerator

```
package de.christianbell.apmfw.jdragicdatamining.logs;
import core.files.FileGetter;
import core.parser.special.IOStatLogParser;
import core.parser.special.NetStatLogParser;
import core.parser.special.SarLogParser;
import core.parser.special.VMStatLogParser;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
/*
 * Diese Klasse generiert Tabellen zu den Log-Dateien
 * @author Martin Gasse, modified by Christian Bell
 */
public class LogsTableGenerator
{
        public static String tablePrefix = "";
        public static String path = null;
        public static PropertiesSingleton ps = null;
        public static void generateTablesFromNetStatLog()
        {
                ps = PropertiesSingleton.getInstance();
                path = ps.getProperty("logspath");
                //tablePrefix = ps.getProperty("messungsname") + "_";
                System.out.println("generateTablesFromNetStatLog");
                NetStatLogParser lNetStatLogParser = new NetStatLogParser();
                FileGetter lFileGetter = new FileGetter(path);
                lNetStatLogParser.setFilesToParse(lFileGetter.getAllLogs());
                lNetStatLogParser.commit(lNetStatLogParser.generateSQLQueryList(tablePrefix));
        }
        public static void generateTablesFromVMStatLog()
        {
                ps = PropertiesSingleton.getInstance();
                path = ps.getProperty("logspath");
                //tablePrefix = ps.getProperty("messungsname") + "_";
                System.out.println("generateTablesFromVMStatLog");
                VMStatLogParser lVMStatLogParser = new VMStatLogParser();
                FileGetter lFileGetter = new FileGetter(path);
                lVMStatLogParser.setFilesToParse(lFileGetter.getAllLogs());
                lVMStatLogParser.commit(
                  lVMStatLogParser.generateSQLQueryList(tablePrefix));
        }
        public static void generateIOTablesfromIOLogFiles()
        {
                ps = PropertiesSingleton.getInstance();
                path = ps.getProperty("logspath");
                //tablePrefix = ps.getProperty("messungsname") + "_";
                System.out.println("generateIOTablesfromIOLogFiles");
                IOStatLogParser lIOStatLogParser = new IOStatLogParser();
                FileGetter      lFileGetter = new FileGetter(path);
```

```
                lIOStatLogParser.setFilesToParse(lFileGetter.getAllLogs());
                lIOStatLogParser.commit(
                   lIOStatLogParser.generateSQLQueryList(tablePrefix));
        }
        public static void generateTablesFromSarLogFiles()
        {
                ps = PropertiesSingleton.getInstance();
                path = ps.getProperty("logspath");
                //tablePrefix = ps.getProperty("messungsname") + "_";
                System.out.println("generateTablesFromSarLogFiles");
                SarLogParser lSarLogParser = new SarLogParser();
                FileGetter      lFileGetter = new FileGetter(path);
                lSarLogParser.setFilesToParse(lFileGetter.getAllLogs());
                lSarLogParser.commit(lSarLogParser.generateSQLQueryList(tablePrefix));
        }
}
```

7.1.9 Das Paket MFW

Cut

```
/*
 * @author Christian Bell
 */
package de.christianbell.apmfw.jdragicdatamining.mfw;
/**
 * Diese Klasse erweitert die Klasse MFWPoint um einen Status und einen Zeitpunkt
 * @author  CJb3LL
 * @version  1.0
 */
public class Cut extends MFWPoint
{
        /**
         * Status des Eintrages, entweder on oder off
         */
        private String  state;
        /**
         * Zeitpunkt
         */
        private long    time;

        /**
         * @return  the state
         * @uml.property  name="state"
         */
        public String getState() {
                return state;
        }
        /**
         * @param state  the state to set
         * @uml.property  name="state"
         */
        public void setState(String state) {
```

```
                this.state = state;
        }
        /**
         * @return  the time
         * @uml.property  name="time"
         */
        public long getTime() {
                return time;
        }
        /**
         * @param time  the time to set
         * @uml.property  name="time"
         */
        public void setTime(long time) {
                this.time = time;
        }
}
```

MFWPoint

```
package de.christianbell.apmfw.jdragicdatamining.mfw;
/**
  * Diese Klasse stellt die Struktur eines Messpunktes dar
  * @author  CJb3LL
  * @version  1.0
  */
public class MFWPoint {

        /**
         * Die ID des Messpunktes
         */
        private int     id;
        /**
         * Name eines Messpunktes
         */
        private String  mfwPoint;
        /**
         * Zeitstempel
         */
        private long    tStamp;
        /**
         * Prozess-ID
         */
        private int     pid;
        /**
         * Thread-ID
         */
        private long    tid;

        /**
         * @return  the id
         * @uml.property  name="id"
         */
        public int getId() {
```

```
        return id;
}
/**
 * @param id  the id to set
 * @uml.property  name="id"
 */
public void setId(int id) {
        this.id = id;
}
/**
 * @return  the mfwPoint
 * @uml.property  name="mfwPoint"
 */
public String getMfwPoint() {
        return mfwPoint;
}
/**
 * @param mfwPoint  the mfwPoint to set
 * @uml.property  name="mfwPoint"
 */
public void setMfwPoint(String mfwPoint) {
        this.mfwPoint = mfwPoint;
}
/**
 * @return  the pid
 * @uml.property  name="pid"
 */
public int getPid() {
        return pid;
}
/**
 * @param pid  the pid to set
 * @uml.property  name="pid"
 */
public void setPid(int pid) {
        this.pid = pid;
}
/**
 * @return  the tid
 * @uml.property  name="tid"
 */
public long getTid() {
        return tid;
}
/**
 * @param tid  the tid to set
 * @uml.property  name="tid"
 */
public void setTid(long tid) {
        this.tid = tid;
}
/**
 * @return  the tStamp
 * @uml.property  name="tStamp"
```

```
         */
        public long getTStamp() {
                return tStamp;
        }
        /**
         * @param tStamp  the tStamp to set
         * @uml.property  name="tStamp"
         */
        public void setTStamp(long stamp) {
                tStamp = stamp;
        }
}
```

Results

```
/*
 * @author Christian Bell
 */
package de.christianbell.apmfw.jdragicdatamining.mfw;
/**
 * Diese Klasse erweitert die Klasse MFWPoint um einen Differenzwert
 * @author  CJb3LL
 * @version  1.0
 */
public class Results extends MFWPoint
{
        /**
         * Differenzwert zwischen Messpunkt-ON und Messpunkt-OFF
         */
        private long    value;

        /**
         * @return  the value
         * @uml.property  name="value"
         */
        public long getValue() {
                return value;
        }
        /**
         * @param value  the value to set
         * @uml.property  name="value"
         */
        public void setValue(long value) {
                this.value = value;
        }
}
```

ValueComputation

```
package de.christianbell.apmfw.jdragicdatamining.mfw;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
```

```
import java.sql.Statement;
import java.util.Iterator;
import java.util.Vector;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
/*
 * Diese Klasse berechnet die Zeitwerte des MFW
 */
public class ValueComputation
{

        static DBConnection db;
        public Connection theConnection;
        public Connection getConnection() {
                return theConnection;
        }
        public void setConnection(Connection theConnection) {
                this.theConnection = theConnection;
        }

    /** Diese Methode berechnet die Werte der Messpunkte
    * und speichert sie in einer Tabelle
    * @param name Name der zu erstellenden und zu fuellenden Tabelle
    * @param messpunkte Ein Vektor aus Messpunkten
    * @param theConnection Eine Verbindung zur Datenbank
    * @param statement ein Statement
         * @return
    * @exception SQLException
    */
        public void calculateAndSaveDifferences(String name, Vector<String> messpunkte)
        {
                ResultSet       resultOn                = null;
                ResultSet       resultOff               = null;
                Cut             pointOn                 = new Cut();
                Cut             pointOff                = new Cut();
                Results         resultTable             = new Results();
                long    value;
                String  mfwPoint;
                long    tStamp;
                int     pid;
                long    tid;
                int count = 0;
                try
                {
                        if(db == null)
                        {
                                db = new DBConnection();
                        }
                        if(theConnection == null)
                        {
                                theConnection = db.getConnection();
                        }
                        Statement statement1 = theConnection.createStatement();
                        Statement statement2 = theConnection.createStatement();
```

```
Statement statement3 = theConnection.createStatement();
GUI.konsoleArea.append("Erzeuge neue Tabelle...\n");
GUI.autoScroll();
String[] nameArr;
//Je nach Betriebssystem enthaelt es entweder / oder \
if (name.contains("/"))
        nameArr = name.split("/");
else nameArr = name.split("\\\\");
int length = nameArr.length;
name = nameArr[length-1];
String[] nameArr2 = name.split("\\.");
name = nameArr2[0];
GUI.konsoleArea.append(name + "\n");
GUI.autoScroll();
statement2.execute("DROP TABLE IF EXISTS " + name);
if(name.contains("worker"))
{
   statement2.execute("CREATE TABLE IF NOT EXISTS "
   + name + " (ID BIGINT(20) UNSIGNED AUTO_INCREMENT,
   PRIMARY KEY (ID), MFWPOINT VARCHAR(100) NOT NULL,
   TSTAMP BIGINT(20) UNSIGNED NOT NULL, VALUE BIGINT(20)
   NOT NULL, PID INT(10) UNSIGNED NOT NULL, TID INT(10)
   UNSIGNED NOT NULL) ENGINE=MyISAM CHARACTER SET utf8
   COLLATE utf8_general_ci");
        GUI.konsoleArea.append("Tabelle erzeugt!\n");
        GUI.autoScroll();
        Iterator itr = messpunkte.iterator();
        String itrStr;
        while(itr.hasNext())
        {
          itrStr = itr.next().toString();
          resultOn = db.getAllSections(itrStr,
              statement1, "on", "worker");
          resultOff = db.getAllSections(itrStr,
              statement3, "off", "worker");

          resultOn.next();
          resultOff.next();
          while(!resultOn.isAfterLast()
                && !resultOff.isAfterLast())
          {
           pointOn.setMfwPoint(
              resultOn.getString("mfwPoint"));
           pointOn.setTStamp(
              resultOn.getLong("tStamp"));
           pointOn.setTime(
              resultOn.getLong("time"));
           pointOn.setPid(resultOn.getInt("pid"));
           pointOn.setTid(resultOn.getLong("tid"));
           pointOn.setState(
              resultOn.getString("state"));
           pointOff.setMfwPoint(
              resultOff.getString("mfwPoint"));
           pointOff.setTStamp(
```

```
   resultOff.getLong("tStamp"));
pointOff.setTime(
   resultOff.getLong("time"));
pointOff.setPid(
   resultOff.getInt("pid"));
pointOff.setTid(
   resultOff.getLong("tid"));
pointOff.setState(
   resultOff.getString("state"));
if(pointOn.getPid() < pointOff.getPid()
   || pointOn.getTid() < pointOff.getTid())
{
  if(resultOn.next());
}
else if(pointOn.getPid() > pointOff.getPid()
    || pointOn.getTid() > pointOff.getTid())
{
  if(resultOff.next());
}
else if(pointOn.getPid() == pointOff.getPid()
    && pointOn.getTid() == pointOff.getTid())
{
 if(pointOn.getTStamp() <= pointOff.getTStamp())
 {
  if(pointOn.getTime() <= pointOff.getTime()
     && (pointOff.getTStamp()
     - pointOn.getTStamp()) <= 200000 )
  {
   value =
     pointOff.getTime() - pointOn.getTime();
   value = (long)(value * 0.55555);
  }
  else
  {
   value =
    pointOff.getTStamp() - pointOn.getTStamp();
   value = value * 1000;
  }
  resultTable.setValue(value);
  mfwPoint = pointOn.getMfwPoint();
  resultTable.setMfwPoint(mfwPoint);
  tStamp = pointOn.getTStamp();
  resultTable.setTStamp(tStamp);
  pid = pointOn.getPid();
  resultTable.setPid(pid);
  tid = pointOn.getTid();
  resultTable.setTid(tid);
  String insertFields = "mfwPoint, tStamp,
         value, pid, tid";
  String insertValues = "'" + mfwPoint
         + "', '" + tStamp + "', '"
         + value + "', '" + pid + "', '"
         + tid + "'";
  statement2.addBatch("INSERT INTO " + name
```

```
                    + " (" + insertFields + ") VALUES ("
                    + insertValues + ")");
              count++;
              if(count >= 10000)
              {
               statement2.executeBatch();
               GUI.konsoleArea.append(".");
               count = 0;
              }
              if(resultOn.next() && resultOff.next());
            }
            else
            {
              if(resultOff.next());
            }
          }
        }
        statement2.executeBatch();
        GUI.konsoleArea.append("!\n");
        GUI.autoScroll();
     }
}
else
{
  statement2.execute("CREATE TABLE IF NOT EXISTS " + name
     + " (ID BIGINT(20) UNSIGNED AUTO_INCREMENT, PRIMARY
     KEY (ID), MFWPOINT VARCHAR(100) NOT NULL, TSTAMP
     BIGINT(20) UNSIGNED NOT NULL, VALUE BIGINT(20) NOT
     NULL, PID INT(10) UNSIGNED NOT NULL) ENGINE=MyISAM
     CHARACTER SET utf8 COLLATE utf8_general_ci");
  GUI.konsoleArea.append("Tabelle erzeugt!\n");
  GUI.autoScroll();
  Iterator itr = messpunkte.iterator();
  String itrStr;
  while(itr.hasNext())
  {
   itrStr = itr.next().toString();
   resultOn = db.getAllSections(itrStr, statement1,
       "on", "prefork");
   resultOff = db.getAllSections(itrStr, statement3,
       "off", "prefork");
   while(resultOn.next() && resultOff.next())
   {
     pointOn.setMfwPoint(resultOn.getString("mfwPoint"));
     pointOn.setTStamp(resultOn.getLong("tStamp"));
     pointOn.setTime(resultOn.getLong("time"));
     pointOn.setPid(resultOn.getInt("pid"));
     pointOn.setState(resultOn.getString("state"));
     pointOff.setMfwPoint(resultOff.getString("mfwPoint"));
     pointOff.setTStamp(resultOff.getLong("tStamp"));
     pointOff.setTime(resultOff.getLong("time"));
     pointOff.setPid(resultOff.getInt("pid"));
     pointOff.setState(resultOff.getString("state"));
     if(pointOn.getPid() < pointOff.getPid())
```

```
      {
        if(resultOn.next());
      }
      else if(pointOn.getPid() > pointOff.getPid())
      {
        if(resultOff.next());
      }
      else if(pointOn.getPid() == pointOff.getPid())
      {
        if(pointOn.getTStamp() <= pointOff.getTStamp())
        {
         if(pointOn.getTime() <= pointOff.getTime()
            && (pointOff.getTStamp() - pointOn.getTStamp())
            <= 200000)
         {
          value = pointOff.getTime() - pointOn.getTime();
          value = (long)(value * 0.55555);
         }
         else
         {
          value = pointOff.getTStamp() - pointOn.getTStamp();
          value = value * 1000;
         }
         resultTable.setValue(value);
         mfwPoint = pointOn.getMfwPoint();
         resultTable.setMfwPoint(mfwPoint);
         tStamp = pointOn.getTStamp();
         resultTable.setTStamp(tStamp);
         pid = pointOn.getPid();
         resultTable.setPid(pid);
         String insertFields = "mfwPoint, tStamp, value, pid";
         String insertValues = "'" + mfwPoint + "', '"
           + tStamp + "', '" + value + "', '" + pid + "'";
         statement2.addBatch("INSERT INTO " + name + " ("
             + insertFields + ") VALUES (" + insertValues
             + ")");
         count++;
         if(count >= 10000)
         {
          statement2.executeBatch();
          GUI.konsoleArea.append(".");
          count = 0;
         }
        }
        else
        {
         if(resultOff.next());
        }
       }
     }
     statement2.executeBatch();
     GUI.konsoleArea.append("!\n");
     GUI.autoScroll();
   }
```

```
                        }
                        if (resultOn != null)
                          resultOn.close();
                        if (resultOff != null)
                          resultOff.close();
                        statement1.close();
                        statement2.close();
                    }
                    catch (SQLException e)
                    {
                      System.out.println("Ein Fehler ist bei
                          ValueComputation.calculateAndSaveDifferences() aufgetreten: "
                          +e.getMessage() );
                      e.printStackTrace();
                    }
                    catch (IllegalAccessException e)
                    {
                      System.out.println("Ein Fehler ist bei
                          ValueComputation.calculateAndSaveDifferences() aufgetreten: "
                          +e.getMessage() );
                      e.printStackTrace();
                    }
                    catch(InstantiationException e)
                    {
                     System.out.println("Ein Fehler ist bei
                        ValueComputation.calculateAndSaveDifferences() aufgetreten: "
                        +e.getMessage() );
                     e.printStackTrace();
                    }
                    catch(ClassNotFoundException e)
                    {
                     System.out.println("Ein Fehler ist bei
                        ValueComputation.calculateAndSaveDifferences() aufgetreten: "
                        +e.getMessage() );
                     e.printStackTrace();
                    }
            }
}
```

7.1.10 Das Paket Server

HttpdMPMConfReader

```
package de.christianbell.apmfw.jdragicdatamining.server;
import java.io.File;
import java.io.IOException;
import java.io.RandomAccessFile;
import java.util.Iterator;
import java.util.Vector;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
/*
  * Diese Klasse liest die Konfigurationsdatei des Servers ein
  * @author Christian Bell
```

```
 */
public class HttpdMPMConfReader
{
        /*
         * Diese Methode durchsucht die Confdatei nach allen Modulen
         * und liefert die gefundenen Moudle in einem Vektor aus
         * @return Vektor mit allen Modulnamen
         * @throws IOException
         */
        public Vector<String> getModules() throws IOException
        {
          PropertiesSingleton ps =
                PropertiesSingleton.getInstance("config/config.properties");
                File confFile = new File(ps.getProperty("confVorlage"));
                RandomAccessFile raFile = new RandomAccessFile(confFile, "r");
                String prefix = "<IfModule mpm_";
                String currentLine = "";
                Vector <String> v = new Vector <String>();
                String tempLine = "";
                while(null != (currentLine = raFile.readLine()))
                {
                        if(currentLine.contains(prefix))
                        {
                                String[] strArr = currentLine.split("_");
                                if(strArr.length == 3)
                                        v.add(strArr[1]);
                                else if(strArr.length > 3)
                                {
                                        for(int i = 1; i < strArr.length - 2; i++)
                                        {
                                                tempLine += strArr[i] + "_";
                                        }
                                        tempLine += strArr[strArr.length-2];
                                        v.add(tempLine);
                                        tempLine = "";
                                }
                        }
                }
                return v;
        }
        public Vector <String> getParams(String modul) throws IOException
        {
         PropertiesSingleton ps =
                PropertiesSingleton.getInstance("config/config.properties");
                File confFile = new File(ps.getProperty("confVorlage"));
                RandomAccessFile raFile = new RandomAccessFile(confFile, "r");
                String modulStartLine =
                    "<IfModule mpm_" + modul.toLowerCase() +"_module>";
                String modulStopLine = "</IfModule>";
                //System.out.println(modulStartLine);
                String currentLine = "";
                Vector <String> v = new Vector <String>();
                int isParamName = 2;
                while(!currentLine.equals(modulStartLine))
```

```
                {
                        currentLine = raFile.readLine();
                }
                currentLine = raFile.readLine();
                while(!currentLine.equals(modulStopLine))
                {
                        String[] strArr = currentLine.split(" ");
                        for(int i = 0; i < strArr.length; i++)
                        {
                                if(!strArr[i].equals(""))
                                {
                                        if((isParamName % 2) == 0)
                                        {
                                                v.add(strArr[i]);
                                        }
                                        isParamName++;
                                }
                        }
                        currentLine = raFile.readLine();
                }
                return v;
        }
        public static void main(String[] args)
        {
                try
                {
                        HttpdMPMConfReader mpmConfReader = new HttpdMPMConfReader();
                        Vector <String> v = mpmConfReader.getModules();
                        Iterator itr = v.iterator();
                        while(itr.hasNext())
                        {
                                System.out.println(itr.next().toString());
                        }
                }
                        catch(IOException e)
                        {
                                e.printStackTrace();
                        }
        }
}
```

Parameter

```
package de.christianbell.apmfw.jdragicdatamining.server;
import org.jdom.Element;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
/*
  * Diese Klasse repraesentiert einen Parameter
  * Zusaetzlich wurden rekursive Methoden zur Bildung einer Parameterliste aller
  * moeglichen Werte implementiert, welche jedoch noch nicht verwendet werden!
  * @author Christian Bell
  */
public class Parameter
{
```

```
private PropertiesSingleton ps = PropertiesSingleton.getInstance();
private int aktuellerWert;
private int schrittweite;
private int startWert;
private int stopWert;
private String name;
private Parameter naechstesElement;
private Parameter vorherigesElement;
public Parameter getVorherigesElement() {
        return vorherigesElement;
}
public void setVorherigesElement(Parameter vorherigesElement) {
        this.vorherigesElement = vorherigesElement;
}
public Parameter(int startWert, int stopWert, int schrittweite, String name)
{
        this.aktuellerWert = startWert;
        this.schrittweite = schrittweite;
        this.startWert = startWert;
        this.stopWert = stopWert;
        this.name = name;
}
public int inkrementieren(Element target)
{
        Element sshexec = new Element("sshexec");
        sshexec.setAttribute("host",ps.getProperty("server"));
        sshexec.setAttribute("username", ps.getProperty("user"));
        sshexec.setAttribute("password", ps.getProperty("password"));
        //TODO Paramter schreiben, also edit von conf
        sshexec.setAttribute("command", "ls");
        target.addContent(sshexec);
        System.out.println(aktuellerWert);
        aktuellerWert += schrittweite;
        if (aktuellerWert > stopWert)
        {
                if (naechstesElement == null)
                        return -1;
                else
                {
                        wertZuruecksetzen();
                        return naechstesElement.inkrementieren(target);
                }
        }
        return aktuellerWert;
}
public void wertZuruecksetzen()
{
        aktuellerWert = startWert;
}
public int getAktuellerWert()
{
        return aktuellerWert;
}
public void setAktuellerWert(int aktuellerWert)
```

```
        {
                this.aktuellerWert = aktuellerWert;
        }
        public Parameter getNaechstesElement()
        {
                return naechstesElement;
        }
        public void setNaechstesElement(Parameter naechstesElement)
        {
                this.naechstesElement = naechstesElement;
        }
        public int getSchrittweite()
        {
                return schrittweite;
        }
        public void setSchrittweite(int schrittweite)
        {
                this.schrittweite = schrittweite;
        }
        public int getStartWert()
        {
                return startWert;
        }
        public void setStartWert(int startWert)
        {
                this.startWert = startWert;
        }
        public int getStopWert()
        {
                return stopWert;
        }
        public void setStopWert(int stopWert)
        {
                this.stopWert = stopWert;
        }
        public String getName() {
                return name;
        }
        public void setName(String name) {
                this.name = name;
        }
}
```

Das Paket Threads

```
package de.christianbell.apmfw.jdragicdatamining.threads;
import java.io.File;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.Statement;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.Iterator;
import java.util.Vector;
```

```
import javax.swing.ComboBoxModel;
import javax.swing.ListModel;
import de.christianbell.apmfw.jdragicdatamining.auswertung.HTMLOverview;
import de.christianbell.apmfw.jdragicdatamining.auswertung.PNG;
import de.christianbell.apmfw.jdragicdatamining.charts.
       JDBCMeanAndStandardDeviationXYDataset;
import de.christianbell.apmfw.jdragicdatamining.charts.JdbcChartListPanel;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
/*
 * Dieser Thread wertet die Messungen aus
 * @author Christian Bell
 */
public class AlleMessungenAuswertenThread extends Thread
{
        Connection con;
        DBConnection db;
        PropertiesSingleton ps;
        //Quelle: http://www.javaworld.com/javaworld/jw-03-2001/jw-0330-time.html?page=2
        public void calcHMS(long timeInSeconds) {
              long hours, minutes, seconds;
              hours = timeInSeconds / 3600;
              timeInSeconds = timeInSeconds - (hours * 3600);
              minutes = timeInSeconds / 60;
              timeInSeconds = timeInSeconds - (minutes * 60);
              seconds = timeInSeconds;
              GUI.konsoleArea.append("Bisher verbrauchte Zeit: " + hours
                  + " hour(s) " + minutes + " minute(s) " + seconds + " second(s)\n");
           }
        public void run()
        {
                 GregorianCalendar gc1 = new GregorianCalendar();
              GregorianCalendar gc2 = new GregorianCalendar();
              // the above two dates are one second apart
              Date d1 = gc1.getTime();
              Date d2 = gc2.getTime();
              long l1 = d1.getTime();
              long l2 = d2.getTime();
              long difference = l2 - l1;
                calcHMS(difference/1000);
                GUI.konsoleArea.append("Auswertung:\n");
                try
                {
                        ps = PropertiesSingleton.getInstance();
                        if(con == null)
                        {
                                try
                                {
                                        if(db == null)
                                                {
                                                db = new DBConnection();
                                                }
                                        con = db.getConnection();
```

```
        }
        catch (Exception e1)
        {
          System.out.println("Connection konnte nicht
            erzeugt werden: " + e1.getMessage());
          e1.printStackTrace();
          return;
        }
}
try
{
        Statement statement = con.createStatement();
        ResultSet result = null;
        JDBCMeanAndStandardDeviationXYDataset dataset = null;
        int anzahlTabellen =
          JdbcChartListPanel.tableComboBox.getModel().getSize();
        String filepath = ps.getProperty("filepath");
        String measurementName = ps.getProperty("messungsname");
        boolean success = (new File(filepath
          + measurementName + "/").mkdirs());
        int i = 0;
        int j = 0;
        PNG png = new PNG();
        File file = null;
        ListModel listModel;
        ComboBoxModel comboBoxModel =
          JdbcChartListPanel.tableComboBox.getModel();
        Vector<String> tabellenNamen = new Vector<String>();
        Vector<String> messpunktNamen = new Vector<String>();
        boolean checkedResults = false;
        String currentTableName;
        String currentMesspunktName;
        for(int k = 0; k < comboBoxModel.getSize(); k++)
        {
             currentTableName =
               comboBoxModel.getElementAt(k).toString();
                if(currentTableName.contains(measurementName))
                {
                 tabellenNamen.add(currentTableName);
                 if(checkedResults == false
                  && currentTableName.contains("results"))
                 {
                  JdbcChartListPanel.tableComboBox.
                    setSelectedIndex(k);
                  listModel =
                   JdbcChartListPanel.list.getModel();
                  for(int l = 0; l < listModel.getSize(); l++)
                  {
                   messpunktNamen.add(
                    listModel.getElementAt(l).toString());
                    checkedResults = true;
                  }
                 }
                }
```

```
 }
 GUI.konsoleArea.append("Erzeuge Bilder...\n");
 GUI.autoScroll();
 Iterator tableItr = tabellenNamen.iterator();
 Iterator messpunktItr = messpunktNamen.iterator();
 while(tableItr.hasNext())
 {
         if(con.isClosed())
                 con = db.getConnection();
         currentTableName = tableItr.next().toString();
         if(currentTableName.contains("results"))
         {
          while(messpunktItr.hasNext())
          {
           currentMesspunktName =
            messpunktItr.next().toString();
           file = new File(filepath +measurementName
            + "/" + currentTableName + "_"
            + currentMesspunktName + ".png");
           GUI.konsoleArea.append(file + "\n");
           GUI.autoScroll();
           result = statement.executeQuery("SELECT
            MIN(tstamp) FROM " + currentTableName
            + " WHERE mfwpoint='"
            + currentMesspunktName + "'");
           result.next();
           long min = result.getLong("MIN(tstamp)");
           if(min != 0)
           {
            statement.executeUpdate("UPDATE "
             + currentTableName + " SET tstamp=tstamp-"
             + min + " WHERE mfwpoint='"
             + currentMesspunktName + "'");
           }
     dataset =
        new JDBCMeanAndStandardDeviationXYDataset(con);
     dataset.executeQuery(con, "SELECT tStamp, value
        FROM " + currentTableName + " where mfwPoint='"
        + currentMesspunktName + "' ORDER BY tstamp");
     png.createPNG(file, dataset, currentTableName);
   }
   messpunktItr = messpunktNamen.iterator();
}
else if(!currentTableName.contains("iostat")
   || !currentTableName.contains("sarlog"))
{
 file = new File(filepath +measurementName +"/"
     + currentTableName + ".png");
 GUI.konsoleArea.append(file + "\n");
 GUI.autoScroll();
 result = statement.executeQuery("SELECT MIN(tstamp)
     FROM " + currentTableName);
 result.next();
 long min = result.getLong("MIN(tstamp)");
```

```
      if(min != 0)
      {
       statement.executeUpdate("UPDATE " + currentTableName
           + " SET tstamp=tstamp-" + min);
      }
      statement.executeUpdate("UPDATE " + currentTableName
          + " SET value = REPLACE(value,',',',')");
      dataset =
         new JDBCMeanAndStandardDeviationXYDataset(con);
      dataset.executeQuery(con, "SELECT tStamp, CAST(value
         AS DECIMAL(5,2)) AS value FROM "
         + currentTableName + " ORDER BY tstamp");
      png.createPNG(file, dataset, currentTableName);
    }
    else
    {
     file = new File(filepath +measurementName +"/"
        + currentTableName + ".png");
     GUI.konsoleArea.append(file + "\n");
     GUI.autoScroll();
     result = statement.executeQuery("SELECT MIN(tstamp)
        FROM " + currentTableName);
     result.next();
     long min = result.getLong("MIN(tstamp)");
     if(min != 0)
     {
      statement.executeUpdate("UPDATE " + currentTableName
         + " SET tstamp=tstamp-" + min);
     }
     dataset =
        new JDBCMeanAndStandardDeviationXYDataset(con);
     dataset.executeQuery(con, "SELECT tStamp, value FROM "
        + currentTableName + " ORDER BY tstamp");
     png.createPNG(file, dataset, currentTableName);
    }
  }
  con.close();
  GUI.konsoleArea.append("Bilder erzeugt!\n");
  GUI.autoScroll();
  gc2 = new GregorianCalendar();
  d2 = gc2.getTime();
  l2 = d2.getTime();
  difference = l2 - l1;
  calcHMS(difference/1000);
  GUI.konsoleArea.append("Erzeuge HTML:\n");
  GUI.autoScroll();
  HTMLOverview aHtml = new HTMLOverview();
  aHtml.createHTML(tabellenNamen, messpunktNamen, filepath
    + measurementName);
  gc2 = new GregorianCalendar();
  d2 = gc2.getTime();
  l2 = d2.getTime();
  difference = l2 - l1;
  calcHMS(difference/1000);
```

```
                        GUI.konsoleArea.append("HTML fertig!\n");
                        GUI.autoScroll();
                    }
                    catch(Exception e1)
                    {
                            System.out.println(e1.getMessage());
                            e1.printStackTrace();
                    }
                    GUI.konsoleArea.append("Auswertung beendet!");
                    GUI.autoScroll();
                }
                catch(Exception e2)
                {
                        e2.printStackTrace();
                }
        }
}
```

MessungThread

```
package de.christianbell.apmfw.jdragicdatamining.threads;
import java.io.IOException;
import java.sql.SQLException;
import java.util.ArrayList;
import org.jdom.JDOMException;
import de.christianbell.apmfw.jdragicdatamining.ant.AntStarter;
import de.christianbell.apmfw.jdragicdatamining.ant.AntWriter;
import de.christianbell.apmfw.jdragicdatamining.charts.JdbcChartListPanel;
import de.christianbell.apmfw.jdragicdatamining.csv.CsvTableGenerator;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
import de.christianbell.apmfw.jdragicdatamining.logs.LogsTableGenerator;
import de.christianbell.apmfw.jdragicdatamining.jmeter.JMeterTableGenerator;
import core.*;
import core.files.FileGetter;
import core.parser.special.JTLParser;
/*
 * Diese Klasse erzeugt einen neuen Thread, in dem die Messuungen ablaufen,
 * so ist es moeglich, dass man auch waehrend einer Messung andere Dinge mit
 * dem Tool bewerkstelligen kann.
 * @author Christian Bell
 */
public class MessungThread extends Thread
{
        //TODO Auslagern
        //public static final String TABLE_PREFIX = "test";
        PropertiesSingleton ps = PropertiesSingleton.getInstance();
                public void run()
            {
                GUI.konsoleArea.append("Erzeuge Ant-File:\n");
     try
                        {
                                if(this.isInterrupted())
                                {
```

```
                                GUI.konsoleArea.append("Abgebrochen!\n");
                                return;
                        }
                        AntWriter antWriter = new AntWriter();
                        antWriter.createAntFile();
                 GUI.konsoleArea.append("Ant-File erfolgreich erzeugt.\n");
                        GUI.autoScroll();
                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }
                catch (JDOMException e)
                {
                        e.printStackTrace();
                }
                catch(Exception e)
                {
                        e.printStackTrace();
                }
        try
                {
                        if(this.isInterrupted())
                        {
                                GUI.konsoleArea.append("Abgebrochen!");
                                return;
                        }
                        GUI.konsoleArea.append("Starte Messung:\n");
                        GUI.autoScroll();
                        AntStarter.runAnt();
                        GUI.konsoleArea.append("Messung beendet!\n");
                        GUI.autoScroll();
                }
                catch (IOException e)
                {
                        e.printStackTrace();
                }
                catch (InterruptedException e)
                {
                        e.printStackTrace();
                }
                /*
                 * Dieser Block erzeugt die JTL-Tabellen
                 */
                if(this.isInterrupted())
                {
                        GUI.konsoleArea.append("Abgebrochen!\n");
                        return;
                }
                GUI.konsoleArea.append("Erzeuge JTL-Tabellen...\n");
                GUI.autoScroll();
                JMeterTableGenerator.generateTablesFromJTLFiles();
                GUI.konsoleArea.append("JTL-Tabellen erzeugt!\n");
                GUI.autoScroll();
```

```
if(this.isInterrupted())
{
        GUI.konsoleArea.append("Abgebrochen!\n");
        return;
}
/*
 * Dieser Block erzeugt die NetStatLog-Tabellen
 */
GUI.konsoleArea.append("Erzeuge NetStatLog-Tabellen...\n");
GUI.autoScroll();
LogsTableGenerator.generateTablesFromNetStatLog();
GUI.konsoleArea.append("NetStatLog-Tabellen erzeugt!\n");
GUI.autoScroll();
if(this.isInterrupted())
{
        GUI.konsoleArea.append("Abgebrochen!\n");
        return;
}
/*
 * Dieser Blog erzeugt die IOStatLog-Tabellen
 */
GUI.konsoleArea.append("Erzeuge IOStatLog-Tabellen...\n");
GUI.autoScroll();
LogsTableGenerator.generateIOTablesfromIOLogFiles();
GUI.konsoleArea.append("IOStatLog-Tabellen erzeugt!\n");
GUI.autoScroll();
if(this.isInterrupted())
{
        GUI.konsoleArea.append("Abgebrochen!\n");
        return;
}
/*
 * Dieser Blog erzeugt die SarLog-Tabellen
 */
GUI.konsoleArea.append("Erzeuge SarLog-Tabellen...\n");
GUI.autoScroll();
LogsTableGenerator.generateTablesFromSarLogFiles();
GUI.konsoleArea.append("SarLog-Tabellen erzeugt!\n");
GUI.autoScroll();
if(this.isInterrupted())
{
        GUI.konsoleArea.append("Abgebrochen!\n");
        return;
}
/*
 * Dieser Blog erzeugt die VMStatLog-Tabellen
 */
GUI.konsoleArea.append("Erzeuge VMStatLog-Tabellen...\n");
GUI.autoScroll();
LogsTableGenerator.generateTablesFromVMStatLog();
GUI.konsoleArea.append("VMStatLog-Tabellen erzeugt!\n");
GUI.autoScroll();
if(this.isInterrupted())
{
```

```
                                GUI.konsoleArea.append("Abgebrochen!\n");
                                return;
                        }
                        /*
                         * Dieser Blog erzeugt die CSV-Tabellen
                         */
                        GUI.konsoleArea.append("Erzeuge CSV-Tabellen...\n");
                        GUI.autoScroll();
                try
                        {
                                CsvTableGenerator.generateTablesFromOrigCSV();
                        }
                        catch (SQLException e)
                        {
                                // TODO Auto-generated catch block
                                e.printStackTrace();
                        }
                        GUI.konsoleArea.append("CSV-Tabellen erzeugt!\n");
                        GUI.autoScroll();
                        if(ps.getProperty("autoAuswerten").equals("true"))
                        {
                         AlleMessungenAuswertenThread pngSpeichernThread =
                               new AlleMessungenAuswertenThread();
                                try
                                {
                                        JdbcChartListPanel.getTableComboBox();
                                        pngSpeichernThread.start();
                                }
                                catch(IllegalThreadStateException exception)
                                {
                                        exception.printStackTrace();
                                }
                        }
            }
}
```

TabelleAuswertenThread

```
package de.christianbell.apmfw.jdragicdatamining.threads;
import java.io.File;
import java.sql.Connection;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
import java.util.Iterator;
import java.util.Vector;
import javax.swing.ListModel;
import de.christianbell.apmfw.jdragicdatamining.auswertung.HTMLOverview;
import de.christianbell.apmfw.jdragicdatamining.auswertung.PNG;
import de.christianbell.apmfw.jdragicdatamining.charts.
       JDBCMeanAndStandardDeviationXYDataset;
import de.christianbell.apmfw.jdragicdatamining.charts.JdbcChartListPanel;
import de.christianbell.apmfw.jdragicdatamining.gui.GUI;
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.DBConnection;
```

```
import de.christianbell.apmfw.jdragicdatamining.hilfsklassen.PropertiesSingleton;
/*
 * Diese Klasse wertet eine Tabelle aus
 * @author Christian Bell
 */
public class TabelleAuswertenThread extends Thread
{
        Connection con;
        DBConnection db;
        PropertiesSingleton ps;
        public void run()
        {
                ps = PropertiesSingleton.getInstance();
                String filepath = ps.getProperty("filepath");
                String measurementName = ps.getProperty("messungsname") +"/";
                if(con == null)
                {
                        System.out.println("Erzeuge Connection");
                        try
                        {
                                if(db == null)
                                        {
                                        db = new DBConnection();
                                        }
                                con = db.getConnection();
                        }
                        catch (Exception e1)
                        {
                               System.out.println("Connection konnte nicht
                                erzeugt werden: " + e1.getMessage());
                                e1.printStackTrace();
                                return;
                        }
                }
                try
                {
                        JDBCMeanAndStandardDeviationXYDataset dataset = null;
                        int anzahlMesspunkte =
                          JdbcChartListPanel.list.getModel().getSize();
                        System.out.println(anzahlMesspunkte);
                        System.out.println("Erzeuge Bilder...");
                        boolean success = (new File(filepath
                          + measurementName).mkdirs());
                        int i = 0;
                        JdbcChartListPanel.list.setSelectedIndex(i);
                        PNG png = new PNG();
                        File file = null;
                        String currentMesspunktName;
                        ListModel listModel = JdbcChartListPanel.list.getModel();
                        Vector<String> messpunktNamen = new Vector<String>();
                        for(int l = 0; l < listModel.getSize(); l++)
                        {
                         messpunktNamen.add(listModel.getElementAt(l).toString());
                        }
```

```
                        Iterator messpunktItr = messpunktNamen.iterator();
                        while(messpunktItr.hasNext())
                        {
                                currentMesspunktName = messpunktItr.next().toString();
                                try
                                {
                                 if(JdbcChartListPanel.tableComboBox.
                                  getSelectedItem().toString().contains("results"))
                                 {
                                  file = new File(filepath +measurementName + "/"
                                     + JdbcChartListPanel.tableComboBox.
                                     getSelectedItem().toString() + "_"
                                     +currentMesspunktName  + ".png");
                                  Statement statement = con.createStatement();
                                  ResultSet result = statement.executeQuery("SELECT
                                     MIN(tstamp) FROM " + JdbcChartListPanel.
                                     tableComboBox.getSelectedItem().toString()
                                     + " WHERE mfwpoint='" + currentMesspunktName
                                     + "'");
                                    result.next();
                                    long min = result.getLong("MIN(tstamp)");
                                    if(min != 0)
                                    {
                                     statement.executeUpdate("UPDATE "
                                        + JdbcChartListPanel.
                                        tableComboBox.getSelectedItem().toString()
                                        + " SET tstamp=tstamp-" + min + " WHERE
                                        mfwpoint='" + currentMesspunktName + "'");
                                    }
                                    dataset =
                                     new JDBCMeanAndStandardDeviationXYDataset(con);
                                    dataset.executeQuery(con, "SELECT tStamp,
                                      value FROM "
                                      + JdbcChartListPanel.tableComboBox.
                                      getSelectedItem().toString() + " where
                                      mfwPoint='" + currentMesspunktName
                                      + "' ORDER BY tstamp");
                                    png.createPNG(file, dataset,
                                      JdbcChartListPanel.
                                       tableComboBox.getSelectedItem().toString());
                                  }
                                  else
if(JdbcChartListPanel.tableComboBox.getSelectedItem().toString().contains("iostat")
|| JdbcChartListPanel.tableComboBox.getSelectedItem().toString().contains("sarlog"))
        {
          file = new File(filepath + measurementName + "/"
            + JdbcChartListPanel.tableComboBox.getSelectedItem().toString() + ".png");
          Statement statement = con.createStatement();
          statement.executeUpdate("UPDATE "
            + JdbcChartListPanel.tableComboBox.getSelectedItem().toString()
            + " SET value = REPLACE(value,',','.')");
          ResultSet result = statement.executeQuery("SELECT MIN(tstamp) FROM "
            + JdbcChartListPanel.tableComboBox.getSelectedItem().toString());
          result.next();
```

```
     long min = result.getLong("MIN(tstamp)");
     if(min != 0)
     {
      statement.executeUpdate("UPDATE "
        + JdbcChartListPanel.tableComboBox.getSelectedItem().toString()
        + " SET tstamp=tstamp-" + min);
     }
      dataset = new JDBCMeanAndStandardDeviationXYDataset(con);
      dataset.executeQuery(con, "SELECT tStamp, CAST(value AS DECIMAL(5,2))
        AS value FROM "
        + JdbcChartListPanel.tableComboBox.getSelectedItem().toString()
        + " ORDER BY tstamp");
      png.createPNG(file, dataset,
        JdbcChartListPanel.tableComboBox.getSelectedItem().toString());                }
else
{
 file = new File(filepath + measurementName + "/"
   + JdbcChartListPanel.tableComboBox.getSelectedItem().toString() + ".png");
 dataset = new JDBCMeanAndStandardDeviationXYDataset(con);
 Statement statement = con.createStatement();
 ResultSet result = statement.executeQuery("SELECT MIN(tstamp) FROM "
   + JdbcChartListPanel.tableComboBox.getSelectedItem().toString());
 result.next();
 long min = result.getLong("MIN(tstamp)");
 if(min != 0)
 {
  statement.executeUpdate("UPDATE "
    + JdbcChartListPanel.tableComboBox.getSelectedItem().toString()
    + " SET tstamp=tstamp-" + min);
 }
 dataset.executeQuery(con, "SELECT tStamp, value FROM "
   + JdbcChartListPanel.tableComboBox.getSelectedItem().toString()
   + " ORDER BY tstamp");
 png.createPNG(file, dataset,
   JdbcChartListPanel.tableComboBox.getSelectedItem().toString());
 }
 }
            catch (SQLException e1)
            {
                e1.printStackTrace();
            }
         }
        }
         catch(Exception ex)
          {
                  System.out.println("Fehler: " + ex.getMessage());
                   ex.printStackTrace();
           }
           ListModel listModel;
           HTMLOverview aHtml = new HTMLOverview();
           Vector<String> tabellenName = new Vector<String>();
           String tabellenNameStr =
            JdbcChartListPanel.tableComboBox.getSelectedItem().toString();
           tabellenName.add(tabellenNameStr);
```

```
                Vector<String> messpunktNamen = new Vector<String>();
                GUI.konsoleArea.append("Erzeuge HTML:\n");
                if(tabellenNameStr.contains("results"))
                {
                        listModel = JdbcChartListPanel.list.getModel();
                        for(int l = 0; l < listModel.getSize(); l++)
                        {
                         messpunktNamen.add(listModel.getElementAt(l).toString());
                        }
                }
                aHtml.createHTML(tabellenName, messpunktNamen, filepath
                   + measurementName);
                GUI.konsoleArea.append("HTML fertig!\n");
                GUI.konsoleArea.append("Auswertung beendet!");
                GUI.autoScroll();
        }
}
```

7.1.11 Das Messframework apmfw.c

```
/*
 * Apache httpd Measurement Framework
 * apmfw
 * Simon Olofsson <simon@olofsson.de>
 * modified by Christian Bell <info@christianbell.de>
 */
#include <sys/time.h>
#include <string.h>
#include <stdio.h>
apr_pool_t *apmfw_memory_pool = NULL;
apr_file_t *FileRes = NULL;
static int apmfw_init = 0;
uint64_t x = 0;
/*
 * Start the measurement
 * called by accessing apmfw/start.html
 */
int apmfw_start()
{
    apmfw_write("__START__", "on");
    apmfw_error("Start: %d,%lu", getpid(), pthread_self());
    return APR_SUCCESS;
}
/*
 * Stop the measurement
 * called by accessing apmfw/stop.html
 */
int apmfw_stop()
{
    if (FileRes != NULL) {
        apmfw_write("__STOP__", "on");
        apr_file_close(FileRes);
        apmfw_error("Stop: %d,%lu", getpid(), pthread_self());
```

```
    }
    else {
        apmfw_error("Stop: Fehler");
    }
    return APR_SUCCESS;
}
/*
 * Write the results
 */
int apmfw_write(char *mpoint, char *state)
{
    struct timeval apmfw_timestamp;
    if (apmfw_init == 0) {
        if (apmfw_memory_pool == NULL) {
            apr_pool_create(&apmfw_memory_pool, NULL);
        }
        else {
            apmfw_error("ERROR: Memory Pool");
        }
        if (apr_file_open
            (&FileRes, "/usr/local/apache2/htdocs/apmfw/orig.csv",
             APR_WRITE | APR_CREATE | APR_APPEND | APR_XTHREAD |
             APR_SHARELOCK, 0x666, apmfw_memory_pool) != APR_SUCCESS) {
            apmfw_error("ERROR: Could not open File");
        }
        apmfw_init = 1;
    }
    gettimeofday(&apmfw_timestamp, NULL);
    //TODO Ueberpruefen, ob Zeitwerte richtig
    apr_file_printf(FileRes, ",%s,%lu,%s,%u,%d,%lu\n", mpoint,
                    ((apmfw_timestamp.tv_sec % 10000000) * 1000000) +
                    apmfw_timestamp.tv_usec, state, apmfw_rdtsc(), getpid(),
                    pthread_self());
    return APR_SUCCESS;
}
int apmfw_flush()
{
    apmfw_error("INFO: apmfw_flush aufgerufen");
    return APR_SUCCESS;
}
/*
 * Check URL for start or stop file
 */
int apmfw_check_url(char *url, char *query)
{
    if (apr_strnatcasecmp(url, "/apmfw/start.html") == 0) {
        apmfw_start();
    }
    else if (apr_strnatcasecmp(url, "/apmfw/stop.html") == 0) {
        apmfw_stop();
    }
    else if (strstr(url, "Testseite.html") && query != NULL)
    {
    apmfw_wait_and_process(query);
```

```
    }
    return APR_SUCCESS;
}
/*
 * Read Time Stamp Counter
 * DON'T use this on multi core/processor systems
 */
/*uint64_t apmfw_rdtsc()
{
    x = 0;
    __asm__ volatile ("rdtsc\n\t":"=A" (x));
    return x;
}*/
/*
 * Read Time Stamp Counter
 * DON'T use this on multi core/processor systems
 */
__inline__ uint64_t apmfw_rdtsc()
{
    uint32_t lo, hi;
    __asm__ __volatile__ ("rdtsc" : "=a" (lo), "=d" (hi));
    return (uint64_t)hi << 32 | lo;
}
int apmfw_wait_and_process(char *query)
{
        char* s = query;
    char* res;
    char* res1;
    res = strsep(&s,"=");
    res = strsep(&s,"&");
    apmfw_write("waitTime", "on");
    apr_sleep(atoi(res) * 1000);
    apmfw_write("waitTime", "off");
    res = strsep(&s,"=");
    res = strsep(&s, "&");
    res1 = strsep(&s, "=");
    apmfw_processing_delay(apr_atoi64(res), atoi(s));
    return APR_SUCCESS;
}
int apmfw_processing_delay(uint64_t processingDelay, int method)
{
        apmfw_write("processingDelay", "on");
        uint64_t startTime = apmfw_rdtsc();
        uint64_t currentTime = apmfw_rdtsc();
        uint64_t time = currentTime - startTime;
        //Methode 1
        if(method == 1)
        {
                int count = 0;
                processingDelay *= 1000000;
                while(time < processingDelay)
                {
                        count++;
                        int x = 0;
```

```
                        int j = 0;
                        for(j = 0; j < 893310; j++)
                        {
                                x = x + 1;
                                x = x - 1;
                        }
                        currentTime = apmfw_rdtsc();
                        time = currentTime - startTime;
                }
                apmfw_write("processingDelay", "off");
                int i;
                char str[15];
                i = sprintf(str, "%d", count);
                apmfw_write("Methode1Durchlaeufe", str);
        }
        else if (method == 2) //Methode 2
        {
                int count;
                for(count = 0; count < processingDelay; count++)
                {
                        int x = 0;
                        int j = 0;
                        for(j = 0; j < 893310; j++)
                        {
                                x = x + 1;
                                x = x - 1;
                        }
                }
                apmfw_write("processingDelay", "off");
        }
        else
        {
                        apmfw_write("Schleifendauer", "on");
                        int x = 0;
                        int j = 0;
                        for(j = 0; j < 893310; j++)
                        {
                                x = x + 1;
                                x = x - 1;
                        }
                        apmfw_write("Schleifendauer", "off");
                        apmfw_write("processingDelay", "off");
        }
        return APR_SUCCESS;
}
/*
 * Write into the error log
 */
int apmfw_error(const char *fmt, ...)
{
    va_list args;
    va_start(args, fmt);
    char Buffer[1000];
    vsprintf(Buffer, fmt, args);
```

```
    ap_log_error(APLOG_MARK, APLOG_NOTICE, 0, NULL, Buffer);
    va_end(args);
    return APR_SUCCESS;
}
```

7.1.12 Die Makefile

```
# $Id: Makefile 700 2007-04-04 10:41:44Z simon $
# Makefile for Apache MFW
# Simon Olofsson <simon@olofsson.de>
SVN=            https://rr14.iis.rub.de/cjba
USER=           simon
PWD=            studpro
MINORVER=       4
CC=             "ccache gcc"
help:
        @echo '*****************************************************'
        @echo '* Makefile fuer Apache httpd Messframework mit       *'
        @echo '*                                                   *'
        @echo '*               *****$$Rev: 700 $$****              *'
        @echo '*                                                   *'
        @echo '* Bei Erstinstallation:                             *'
        @echo '* # make workerinst                                 *'
        @echo '* oder:                                             *'
        @echo '* # make preforkinst                                *'
        @echo '*                                                   *'
        @echo '* Simon Olofsson <simon@olofsson.de>                *'
        @echo '* modified by Christian Bell <info@christianbell.de> *'
        @echo '*****************************************************'
        @echo ''
        @echo 'Installation nur als root moeglich!'
        @echo 'Bitte das Verzeichnis /usr/local/apache2 vorher loeschen!'
        @echo 'ccache muss installiert sein'
        @echo ''
workerinst: fetchworker configworker build conf
preforkinst: fetchprefork configprefork build conf
workerinst-old: fetchworker patch-old configworker build conf
fetchworker:
        @svn co --username ${USER} --password ${PWD} \
                ${SVN}/httpd-2.2.${MINORVER}.${worker} httpd-2.2.${MINORVER}
        @svn co --username ${USER} --password ${PWD} \
                ${SVN}/apmfw-patch
fetchprefork:
        @svn co --username ${USER} --password ${PWD} \
                ${SVN}/httpd-2.2.${MINORVER}.${prefork} httpd-2.2.${MINORVER}
        @svn co --username ${USER} --password ${PWD} \
                ${SVN}/apmfw-patch
patch-old:
        @(cd httpd-2.2.${MINORVER}/include && \
                cp -f apmfw-old.c apmfw.c && \
                cp -f apmfw-old.h apmfw.h)
configworker:
        @(cd httpd-2.2.${MINORVER}/ && CC=${CC} \
```

```
                ./configure --with-mpm=${worker} --with-included-apr)
configprefork:
        @(cd httpd-2.2.${MINORVER}/ && CC=${CC} \
                ./configure --with-mpm=${prefork} --with-included-apr)
build:
        @(cd httpd-2.2.${MINORVER}/ && make && make install && make clean)
conf:
        @cp apmfw-patch/favicon.ico /usr/local/apache2/htdocs/
        @cp apmfw-patch/index.html /usr/local/apache2/htdocs/
        @cp apmfw-patch/cgi-bin/results /usr/local/apache2/cgi-bin/
        @chmod 755 /usr/local/apache2/cgi-bin/results
        @gcc apmfw-patch/cgi-bin/time.c -o apmfw-patch/cgi-bin/time
        @mv apmfw-patch/cgi-bin/time /usr/local/apache2/cgi-bin/
        @mkdir -p -m 777 /usr/local/apache2/htdocs/apmfw/
        @cp -R apmfw-patch/apmfw/* /usr/local/apache2/htdocs/apmfw/
        @touch /usr/local/apache2/htdocs/apmfw/orig.csv
        @chmod 666 /usr/local/apache2/htdocs/apmfw/orig.csv
        @cp apmfw-patch/httpd.conf.patch /usr/local/apache2/conf/
        @(cd /usr/local/apache2/conf/ && \
        patch -p0 < httpd.conf.patch)
        @rm -f /usr/local/apache2/conf/httpd.conf.patch
clean:
        @rm -rf httpd-2.2.${MINORVER} apmfw-patch
```

7.2 Inhalte der DVDs

7.2.1 DVD 1

- Apache_Quellcodes: Instrumentalisierter Quellcode des Apache Webservers in der Version 2.2.3 für das Prefork MPM und in der Version 2.2.4 für das Prefork MPM und das Worker MPM
- Dokumentation: Diese Arbeit im PDF-Format, alle Kapitel als Lyx-Dateien, alle Bilder der Dokumentation, das Literaturverzeichnis in Form einer Bib-Datei, Messungsübersicht als Excel-Datei
- Ergebnisse: Alle Ergebnisse aller Messungen in Form von HTML-Übersichtsseiten mit statistischen Größen und Graphen
- JDataMining: Programm, Quelltexte, Javadoc
- MFW: Makefile, apmfw-patch
- Messpläne: Messpläne (JMX-Dateien) aller Messungen
- Plugins: Für die Installation benötigte Plugins

- Vorlagen: Vorlagen für Stresstests, Langzeittests, Langzeittests mit Zusatzlast, Serverkonfigurationsdatei

7.2.2 DVD 2

- MySQLDaten: Alle Messdaten in Form von MySQL-Tabellen

7.2.3 DVD 3

- Rohdaten: Gemessene Daten in Form von CSV-Dateien, JTL-Dateien, Log-Dateien