Ludwig - Maximilians Universität München

Fakultät für Informatik

Lehr- und Forschungseinheit Informatik I

Prototyp zur Visualisierung von leistungsbezogenen Meßwerten bausteinbasierter Anwendungen

Fortgeschrittenen-Praktikum

Thomas Deiler

Themensteller: Univ.-Prof. Dr.H.-G.Hegering

Betreuer: Rainer Hauck

Inhaltsverzeichnis

1 Einleitung **5**

1.1 Basis . . . 5

1.2 Idee zu diesem FoPra . . . 6

1.3 Aufbau dieser Ausarbeitung . . . 6

2 Anforderungsanalyse **7**

2.1 Nicht-funktionale Anforderungen . . . 7

2.2 Funktionale Anforderungen . . . 8

2.3 Pseudo Anforderungen . . . 9

2.4 Szenarien . . . 9

3 System und Objekt Design **11**

3.1 Design Ziele . . . 11

3.2 Software Architektur . . . 11

3.2.1 Untersysteme . . . 12

3.2.1.1 MessObjekt Klasse . . . 12

3.2.1.2 StatsServer . . . 12

3.2.1.3 Swing Anwendung: StatisticView . . . 14

3.2.2 Datenverwaltung . . . 14

3.3 Objekt Design . . . 15

3.3.1 Package Organisation . . . 15

3.3.1.1 StatsServer . . . 15

3.3.1.2 StatisticView . . . 16

3.3.2 Klassenschnittstellen . . . 16

3.3.2.1 MessObjekt - StatsServer . . . 17

3.3.2.2 StatsServer - StatisticView . . . 17

3.4 XML Document Type Definition . . . 18

4 Benutzerhandbuch **19**

4.1 JVM 19

4.1.1 Installieren 19

4.1.2 Patchen 20

4.1.3 Übersetzen 20

4.2 JAXP 20

4.3 BeanBox 20

4.3.1 Installieren 20

4.3.2 Patchen und neu übersetzen 21

4.3.3 Modifizierte Beans installieren 21

4.3.4 BeanBox Manual (Quick Start) 21

4.4 StatsServer 22

4.4.1 Installieren und übersetzen 22

4.4.2 Starten 22

4.5 StatisticView 23

4.5.1 Installieren und übersetzen 23

4.5.2 StatisticView Manual 23

5 Weiterentwicklung **25**

Kapitel 1

Einleitung

1.1 Basis

In den Anfängen der EDV gab es riesige Großrechner, die viel Platz benötigten. Die Anwender hatten vergleichsweise mit heute über "dumme" Datenterminals Zugang zu diesen Ungetümen. Parallel zu den Großrechnern setzten sich immer mehr die PCs mit ihrer kontinuierlich ansteigenden Rechenleistung durch. Aufgaben für die früher ein Großrechner benötigt wurde, erledigte ein PC in Bruchteilen der früheren Zeit. Diese Dezentralisierung erlaubte eine exakte Anpassung der PCs an seine Aufgaben. Jedoch mußten in der Folge Systemadministatoren mit einem größeren Verwaltungsaufwand zurechtkommen. Anwendungen müssen bis heute für verschiedene Plattformen angepaßt und auf dem neuesten Stand gehalten werden. Darüberhinaus stellen die vielen Konfigurationsmöglichkeiten eines Rechners eine auf Dauer unlösbare Aufgabe dar. Der Trend bei Firmen geht daher wieder in Richtung Zentralisierung. Die Datenterminals sind nicht mehr ganz so "dumm" wie in den Anfängen der EDV und die Server, die Teilaufgaben von Großrechner übernehmen, werden ausgelagert. Dieser Dienst wird von den sogenannten Application Service Providern (ASP) angeboten. Zusätzlich können diese ASPs Speicherplatz zur Verfügung stellen. Eine Firma, die diesen Dienst nutzen will, braucht sich um die Aktualität der Anwendungen oder um ausreichende Speicherkapazität keine Sorgen zu machen. Dadurch entfallen die Produktupdate-, Lizenz- und Anschaffungskosten neuer Hardware. Die Firma mietet lediglich Anwendungen und Speicherplatz beim ASP, zahlt also nur für die erbrachte Leistung. An der Qualität der Leistungen sind Kunde und ASP gleichermaßen interessiert. Daher müssen die Anwendungen überwacht und analysiert werden, wozu die Anwendung selbst die nötigen Informationen liefern soll.

Eine Möglichkeit der Überwachung von Anwendungen ist die Anwendungsinstrumentierung. In der Dissertation von Rainer Hauck [Hau01] wird eine Architektur vorgestellt, die den enormen Aufwand einer Instrumentierung beseitigt - eines der Hauptprobleme heutiger Instrumentierungsansätze. Eine Überwachung bausteinbasierter delokalisierter Anwendungen, wie sie von ASPs angeboten werden, steht dabei im Vordergrund. Im Zuge dieser Arbeit entstand ein FoPra [Kali01], das die Java BeanBox[1] dahingehend erweitert, während der Ausführung JavaBeans basierter Anwendungen eine automatische

[1] experimentelle Plattform für bausteinbasierte Java Beans Anwendungen

Instrumentierung durchzuführen. Ergebnis dieses FoPras ist ein Programm (Java-Klasse), das Meßinformationen einer Java BeanBox Anwendung in einer Textdatei ausgibt.

1.2 Idee zu diesem FoPra

Die Ergebnisse aus dem vorgenannten FoPra sind auch für kompetente Fachleute schwer zu verstehen und das Ausgabeformat ist für die Weiterverarbeitung unhandlich. Die Idee war es, die ausgegebenen Managementinformationen zu sammeln, strukturiert in das XML[2] Format umzuwandeln und graphisch darzustellen, um die Interpretation zu vereinfachen, wobei eine zentrale Auswertung im Vordergrund stand. Die Visualisierung sollte qualitative und quantitative Rückschlüsse auf die verwendeten Bausteine zulassen. Ziel dieses FoPras war es ein Werkzeug zu bekommen, das eine einfache Analyse von Managementinformationen erlaubt, ohne mit der Arbeitsweise der bausteinbasierten Anwendungen vertraut zu sein.

1.3 Aufbau dieser Ausarbeitung

Der Aufbau dieser Ausarbeitung orientiert sich an dem Software Engineering Model von [BrDu00]. Es beschreibt objekt-orientierte Techniken, die die Umsetzung komplexer Softwareprojekte erleichtern.

In Kapitel zwei werden die Systemvoraussetzungen analysiert. Die gewünschte Funktionalität des Systems wird festgelegt, ohne dabei explizit auf Implementierungsdetails einzugehen.

Das dritte Kapitel setzt die Anforderungen aus dem zweiten Kapitel um. Die Untersysteme werden zerlegt und detailliert beschrieben.

Kapitel vier versteht sich als Handbuch für Studenten/Interessierte, die den Prototypen verwenden wollen. Es schafft auch eine Grundlage für die Weiterentwicklung des Prototypen.

Das Kapitel fünf bietet Lösungsansätze und Ideen für die Weiterentwicklung des Prototypen an.

[2] **EX**tensible **M**arkup **L**anguage

Kapitel 2

Anforderungsanalyse

Dieses Kapitel beschäftigt sich mit den Voraussetzungen, die das System erfüllen muß, um seiner Aufgabe gerecht zu werden. Auf die Analyse der Benutzer wurde bewußt verzichtet, weil das System nur einen aktiven Benutzertyp an einer genau definierten Stelle zuläßt.

2.1 Nicht-funktionale Anforderungen

Es sind Anforderungen, die der Benutzer an das System stellt und die tatsächliche Funktionalität des Systems nicht berühren.

Intuitive Benutzung

Für den Benutzer soll die Auswertung der Daten möglichst einfach sein. Es sollen weniger gewandte Anwender genauso zurecht kommen, wie versiertere. Ein grundlegend gleicher Aufbau, der sich an Schemata bekannter und häufig benutzter Anwendungen orientiert, ist erwünscht.

Stabilität bei großem Datenbestand

Das System soll in der Lage sein, auch bei der Speicherung großer Datenmengen stabil und ohne Fehlfunktion zu arbeiten, insbesondere auch dann, wenn auf der Seite des informationsliefernden Untersystems Fehler auftreten.

Anschauliche Visualisierung

Für den Benutzer soll durch die graphische Aufbereitung der Daten das Verstehen vereinfacht und anschaulich sein. Komplexe Vorgänge werden auf diese Weise leichter begriffen.

Statistische Informationen

Neben der Visualisierung soll das System statistische Informationen, die zur Auswertung großer Datenbestände nützlich sind, anzeigen können.

Ausgabe der strukturierten Informationen

Das System soll die Möglichkeit bieten, die dargestellten Informationen in einer Datei abzulegen, um sie mit anderen Anwendungen weiterverarbeiten zu können.

Flexibler Einsatzort

Die Auswertung der Daten soll auch auf einem anderen Gerät, als auf dem, das die Daten produziert, möglich sein.

Plattformunabhängig

Der Prototyp soll für verschiedene Betriebssystemplattformen verfügbar sein.

2.2 Funktionale Anforderungen

Die funktionalen Anforderungen beschreiben die Interaktion des Systems mit seiner Umgebung, ohne dabei näher auf Details einzugehen.

Anzeige der Benutzertransaktionen als Baum

Die BTAs[1] sollen als Baum dargestellt werden, damit der Zusammenhang zwischen der bausteinbasierten Anwendung und ihrer Darstellung klar wird.

Kommunikation der Subsysteme über Sockets

Die Übertragung der Daten soll mit Internettechnik geschehen, um möglichst auf jedem System zu funktionieren.

Graphische menügesteuerte Benutzeroberfläche

Die Visualisierung soll einfach mit der Maus als Eingabegerät steuerbar sein.

[1] Benutzertransaktionen

2.3 Pseudo Anforderungen

Die gesamte Software soll in Java geschrieben werden. Für die graphische Benutzeroberfläche soll die Java API Swing[2] verwendet werden. Das Ausgabeformat der Informationen soll XML sein.

2.4 Szenarien

Ein Szenario beschreibt einen möglichen Ablauf, wie das ganze System oder eine Menge von Untersystemen benutzt werden. Alle Szenarien zusammengefaßt beschreiben die komplette Funktionalität des Systems. Da die Aufführung aller Szenarien zu umfangreich wäre, werden nachfolgend nur zwei maßgebende beschrieben:

Generierung von Meßwerten

Beteiligte: *Rainer*: Anwender

Ereignisse:

1. Rainer lädt sich in seine BeanBox eine bausteinbasierte Anwendung der Firma XY. Sie besteht aus einem StartButton und einem SortierBaustein. Wird der Button gedrückt, so beginnt der Sortier-Baustein seine Arbeit.
2. Als Rainer den StartButton drückt, werden, ohne daß er es bemerkt, Meßwerte, die Auskunft über Art und Dauer der Aktionen geben, über das Netzwerk an einen Sammelserver verschickt.
3. Rainer ist mit dem Ergebnis zufrieden.

***Anmerkung**: Die Generierung von Meßwerten kann sich mehrmals ereignen; natürlich auch in abgewandelter Form.*

Suche nach Fehlern in den bausteinbasierten Anwendungen

Beteiligte: *Rainer*: Anwender; *Thomas*: Angestellter der ASP Firma XY

Ereignisse:

1. Thomas bekommt von Rainer einen Anruf. Dieser ist unzufrieden über eine Anwendung, die er bei der Firma XY geleased hat.
2. Thomas startet sein Visualisierungswerkzeug und lädt sich die Meßwerte eines ihm bekannten Sammelservers in das Werkzeug.
3. Nach einer eventuellen Zeitangabe von Rainer findet Thomas die Meßwerte der vermeintlich fehlerhaften Anwendung.

[2]Swing ist eine GUI Komponente der Java Foundation Classes (JFC)

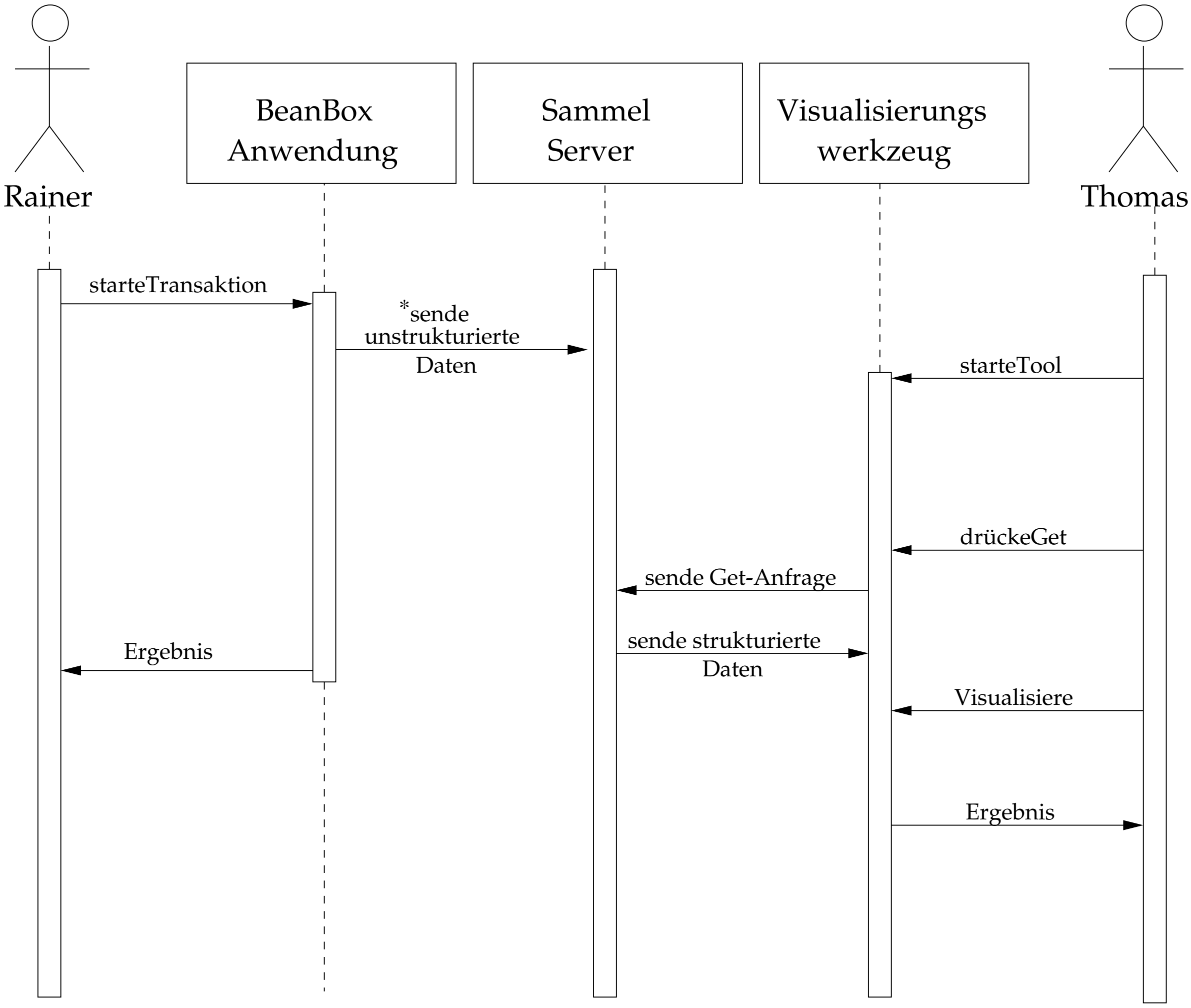

Abbildung 2.1: UML Sequenz Diagramm

4. In der Baumansicht stellt Thomas schnell fest, daß sich ein Baustein der Anwendung nicht korrekt verhält.

5. Thomas gibt die Meldung an den Hersteller des Bausteins weiter.

Die Abb. 2.1 veranschaulicht den Ablauf der Ereignisse mit Hilfe eines UML Sequenzdiagramms.

Kapitel 3

System und Objekt Design

Dieses Kapitel erläutert den kompletten Aufbau des Systems, den Datenaustausch einzelner Untersysteme und deren Kommunikationschnittstellen sowie die Syntax der Ausgabestruktur.

3.1 Design Ziele

Ziel war es die Anforderungen aus dem zweiten Kapitel umzusetzen. Während der Implementierung können die Anforderungen verfeinert und erweitert werden. Die Leistungfähigkeit des Systems stand nicht im Vordergrund, da es sich um einen Prototypen handelt. Dennoch wurde darauf geachtet, mit Speicherplatz und CPU Leistung sparsam umzugehen. Die Zuverlässigkeit des Systems wurde unter Testbedingungen überprüft. Ein im großen Maßstab angesetzter Feldversuch könnte die Zuverlässigkeit des Systems weiter verbessern.

3.2 Software Architektur

Das System als Ganzes betrachtet funktioniert nach dem Client/Server Prinzip. Es sind mehrere Clients möglich, die für den Moment der Datenübertragung eine Verbindung zum Server herstellen. Dabei wird zwischen zwei Clients unterschieden. Nach Abb. 3.1 befindet sich auf der linken Seite des Servers der Daten liefernde (push) Client und auf der rechten Seite der Daten ziehende (pull) Client.

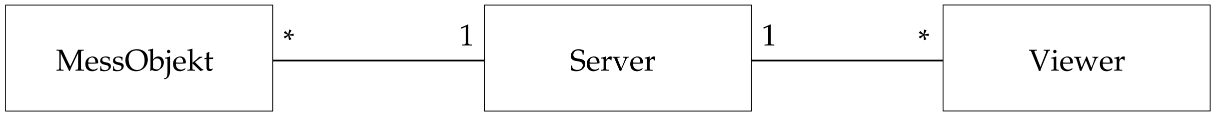

Abbildung 3.1: UML Class Diagram Multi-Client-Single-Server

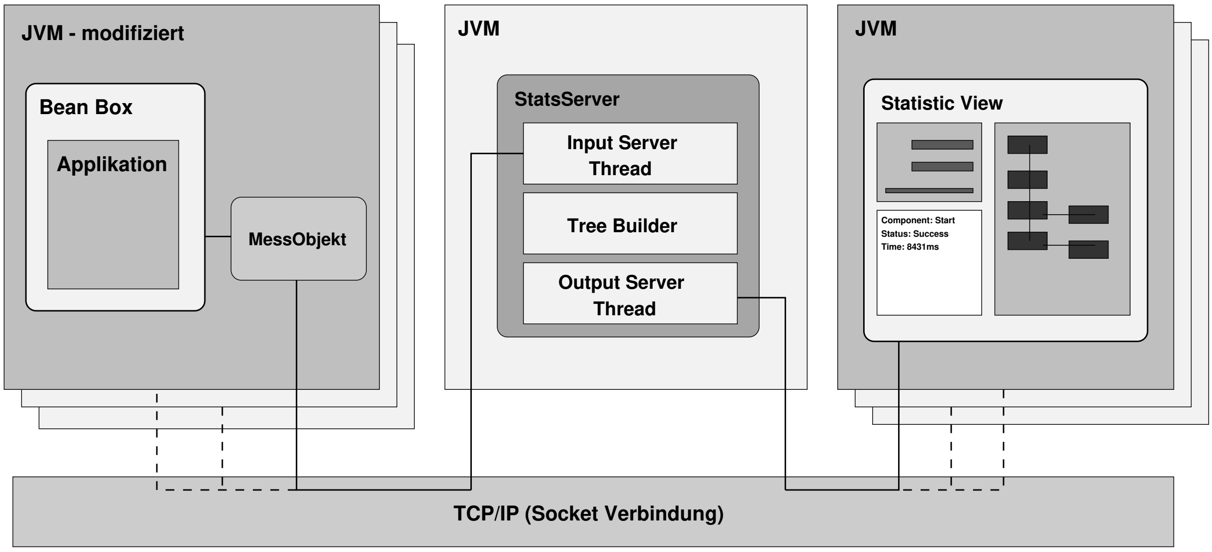

Abbildung 3.2: Kommunikation der Untersysteme

3.2.1 Untersysteme

Die Architektur der Untersysteme läßt es zu, daß das ganze System nicht an einen Ort gebunden ist (Abb. 3.2). Greift man den Fall der Firma XY aus Kapitel 2.4 auf, wird der Vorteil gegenüber einer lokalen Lösung verständlich. Moderne Unternehmen, die das Internet als Medium benutzen, können schnell auf die Wünsche ihrer Kunden eingehen.

3.2.1.1 MessObjekt Klasse

Das MESSOBJEKT steht der JVM als Java Package (*de.unimuenchen.informatik.mnm.monitor*) zur Verfügung. Es beinhaltet eine Klasse MESSPUNKT zur Speicherung eines Meßpunktes und Funktionen zur Bestimmung von Meßpunkten. Das MESSOBJEKT zusammen mit der modifizierten JVM von Rainer Hauck schreibt die Meßpunkte zyklisch mit der Funktion *doLog()* über eine Socket[1] Verbindung auf den STATSSERVER, sofern dieser existiert. Kommt keine Verbindung zustande, wird der zu sendende Meßpunkt verworfen.

3.2.1.2 StatsServer

Der STATSSERVER ist das zentrale Element des Systems. Er sammelt sämtliche Meßpunkte, die er von verschiedenen MESSOBJEKTEN erhalten kann und baut daraus einen strukturierten Transaktionsbaum im Speicher auf. Der Server ist multi-threaded, d.h. ein Programm teilt sich nach dem Start in mehrere mehr oder weniger selbstständig arbeitende Unterprogramme auf. Dadurch ist gewährleistet, daß mehrere Clients, ohne sich gegenseitig zu stören, auf den Server zugreifen können. Das UML State Chart in Abb. 3.3 veranschaulicht das Zusammenspiel der einzelnen Threads.

[1]TCP - Die korrekte Übertragung der Pakete wird sichergestellt

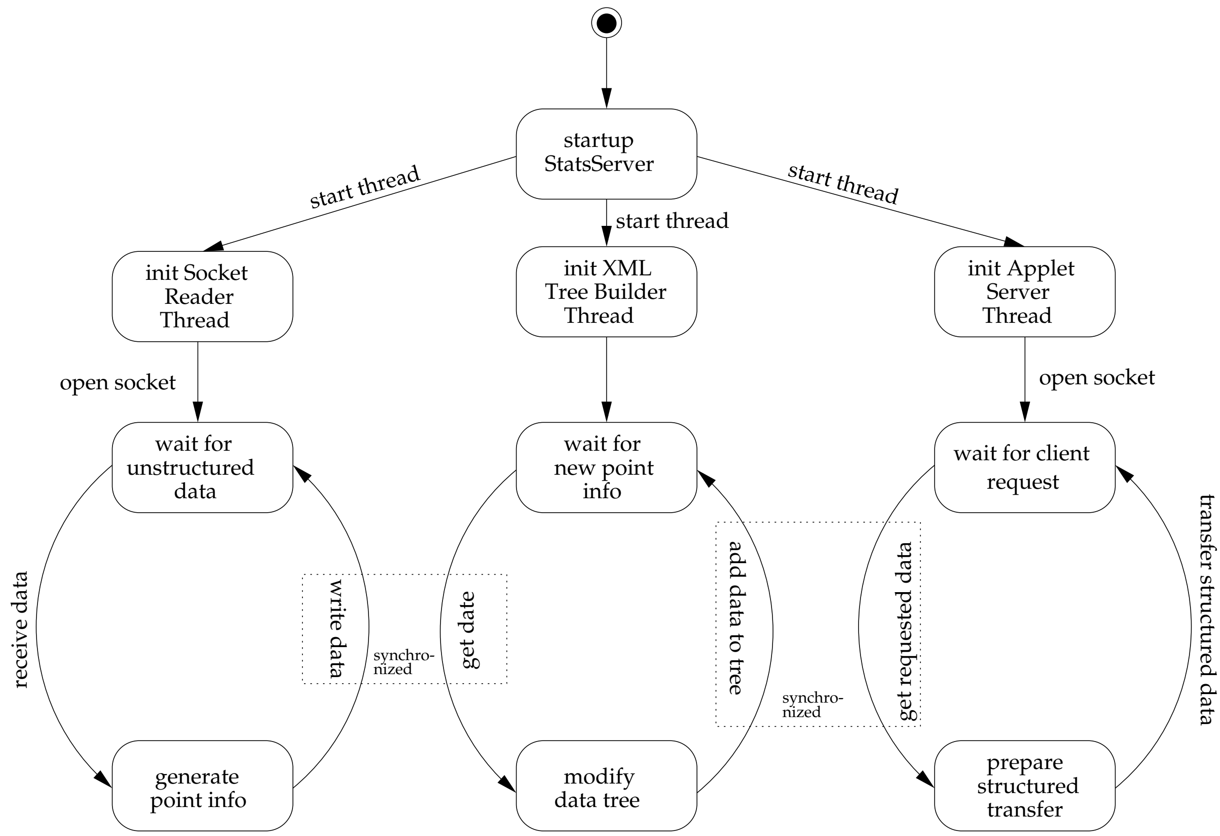

Abbildung 3.3: Multi-Threaded StatsServer

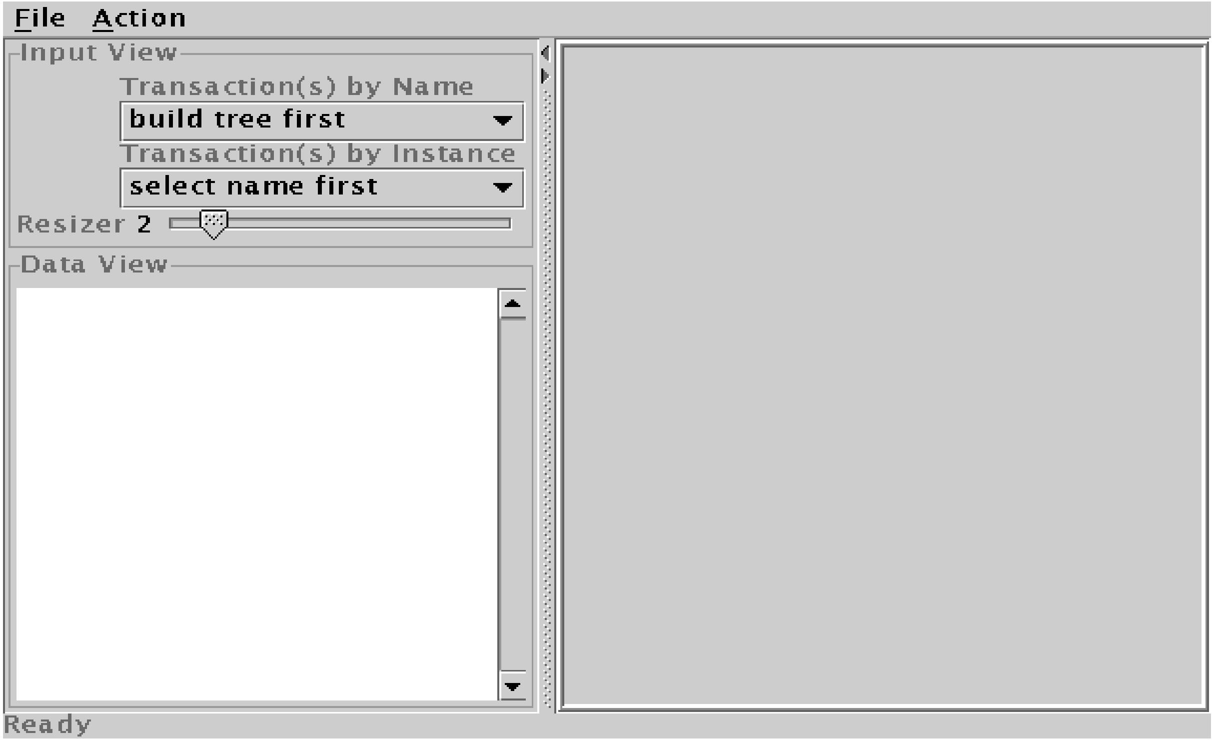

Abbildung 3.4: StatisticView nach dem Start

3.2.1.3 Swing Anwendung: StatisticView

Das Darstellungswerkzeug STATISTICVIEW ist, wie die meisten Programme mit graphischer Benutzeroberfläche, ein MVC[2] System [BrDu00], d.h. es besteht aus Kontrollelementen (Menüs, Knöpfe, u.s.w), die eine Interaktion mit dem Benutzer erlauben und aus Ansichtselementen (Textfelder, Pixelfelder), die dem Benutzer Informationen liefern. Für den Benutzer unsichtbar ist das Modell-Untersystem. Es verarbeitet Informationen und koordiniert alle Kontroll- und Ansichtselemente. In Abb. 3.4 ist ein Screeenshot des Programmes zu sehen. Der rechte große Bereich ist für die graphische Visualisierung der Daten bestimmt und stellt nur eine Instanz einer Transaktion dar, der linke untere für exakte Bausteininformationen. Hinter dem Menü "Action" verbirgt sich die Funktion, die die Daten vom STATSSERVER abholt. Der einfache Befehl "sendXML" veranlaßt den Server, alle momentan gespeicherten Informationen an den Viewer zu schicken. An dieser Stelle wären auch noch weitere Befehle möglich, wie z.B. alle in einer begrenzten Zeitspanne angefallenen Informationen zu übertragen oder nur fehlerhafte Informationen zu senden.

3.2.2 Datenverwaltung

Die Verwaltung der anfallenden Daten geschieht ausschließlich im STATSSERVER. Er speichert sie im flüchtigen Hauptspeicher der jeweiligen Maschine, d.h. wird das Programm beendet, gehen alle gesammelten Informationen verloren. Der Server terminiert sich nur im Falle einer schwerwiegenden Ausnahme. Das können belegte Socketports während des Serverstarts sein oder die Erschöpfung der

[2] **M**odel, **V**iewer, **C**ontrol Subsystem

BTA-Speicherkapazität, die zum Zeitpunkt der Initiierung festgelegt wurde. Momentan liegt der Wert bei 128 BTAs. Da bei intensiver Nutzung der reservierte Speicher schnell verbraucht sein wird, ist ein Mechanismus eingebaut, der nach einem festgelegten Verfalldatum abgelaufene BTAs entfernt, um so Platz für neue BTAs zu schaffen. Der Wert hierfür liegt bei 24 Stunden, sofern kein anderer Wert bei der Initiierung des Servers angegeben wurde.

3.3 Objekt Design

Das Objekt Design dient bei größeren Projekten zur besseren Absprache der Systementwickler untereinander. Wird objektorientiert programmiert, wie in diesem Fall, sind die Abhängigkeiten der verschiedenen Klassen und deren Schnittstellen zu dokumentieren. Dieser Abschnitt ist als Hilfe für jene gedacht, die den Prototypen weiterentwickeln wollen.

3.3.1 Package Organisation

3.3.1.1 StatsServer

StatsServer.java

Das zentale Hauptprogramm des Servers verarbeitet eventuell angegebene Programmparameter und startet drei Threads (SOCKETREADER, XMLTREEBUILDER, APPLETSERVER).

SocketReader.java

Diese Klasse beinhaltet den Thread, der die Daten vom MESSOBJEKT entgegennimmt. Er 'horcht' an einem festgelegten Socketport und gibt die gewonnenen Daten in Form einer EXCHANGECLASS an die SYNCCLASS weiter.

ExchangeClass.java

Diese Klasse ist nur ein Behälter für MESSPUNKT Daten. Im Konstruktor kann man gleich bei der Generierung des Objekts das Verfalldatum setzen.

SyncClass.java

SYNCCLASS synchronisiert den Austausch des EXCHANGECLASS Behälters mit dem XMLTREEBUILDER Thread.

XmlTreeBuilder.java

Dieser Thread fügt die Daten im EXCHANGECLASS Behälter der Klasse SPACECLASS zu. Außerdem wird nach jeder Aktion überprüft, ob ein Datensatz schon verfallen ist.

SpaceClass.java

Diese synchronisierte Struktur fügt die EXCHANGECLASS Daten zu einer Listen-/Baumstruktur zusammen. Sie beinhaltet alle wichtigen Funktionen, um die Struktur zu bearbeiten.

NodeClass.java

Diese Struktur speichert die eigentlichen Transaktionen ab. Sie wird allein von SPACECLASS genutzt. In ihr befinden sich alle Algorithmen zum Bau des Baumes.

AppletServer.java

Dieser Thread bezieht seine Daten aus der SPACECLASS Struktur und bedient damit den Client (STATISTICVIEW).

3.3.1.2 StatisticView

StatisticView.java

Das Swing Hauptprogramm. Hier werden alle GUI Elemente inklusive Funktionalität generiert und formatiert zusammengefügt.

MessageClass.java

Eine kleine Hilfsklasse, die die Ausgabe von Meldungen, wie Status-, Erfolgs- oder Mißerfolgsmeldungen, vereinfacht ausgibt.

AdapterNode.java

Eine weitere Hilfsklasse, die den Umgang mit einem eingelesenen XML Dokument vereinfacht. Außerdem stellt sie die Funktion *cleanup()* bereit, die einen Fehler im JAXP beseitigt.

BTAContainer.java

In dieser Klasse werden die einzelnen BTAs aus dem XML Dokument für die Visualisierung aufbereitet und abgespeichert. Sie ist das Gegenstück zur SPACECLASS.

ComponentContainer.java

Der COMPONENTCONTAINER speichert die einzelnen Bausteininformationen ab und beinhaltet Informationen für die graphische Darstellung.

TANameContainer.java

Diese Hilfsklasse ist notwendig, um die ComboBoxen[3] der Anwendung mit eindeutigen Namen zu belegen.

ScaleData.java

Das Kernstück der skalierbaren graphischen Darstellung steckt in dieser Klasse. Werte für das Koordinatensystem werden hier berechnet.

SumBta.java

Eine statistische Funktion ist in dieser Klasse enthalten.

3.3.2 Klassenschnittstellen

Dieser Abschnitt beschränkt sich auf die Beschreibung der externen Schnittstellen, die mit der Kommunikation über Socket-Verbindungen in Bezug stehen.

[3]Menüelement mit wechselnder Funktionalität

3.3.2.1 MessObjekt - StatsServer

Die Verbindung zwischen den beiden Untersystemen ist unidirektional, d.h. es existiert nur ein Datenstrom in Richtung Server. Die einzelnen Daten eines Meßpunktes werden als Zeichenketten über das Socket verschickt. Der Server kann anhand der Nummer der gesendeten Zeichenketten dem Inhalt eine Bezeichnung zuordnen. Auf das Versenden der Daten mit einer eindeutigen Bezeichnung wurde wegen der einfachen Beschaffenheit bewußt verzichtet. Auch wurden Methoden für gesicherte Übertragung weggelassen. Ein Meßpunkt besteht aus dreizehn möglichen Werten. Um leere Zeichenketten zu vermeiden, wird ein nicht gesetzter Wert mit "null" belegt.

btaInstance: Eine eindeutige Nummer einer BTA Instanz. Sie wird aus einer Zufallszahl und dem Datum erstellt.

timestamp: Ein Zeitstempel der während der Generierung des Meßpunktes erstellt wird. Er trägt einen Wert in Millisekunden, die seit dem 1.1.1970 vergangen sind[4].

action: Die Aktion, durch die der Meßpunkt enstanden ist. "startBTA", "stopBTA", "startTA", "stopTA", "initiatedTA", "startControlFlow" oder "logInfo" sind als Wert erlaubt (s. [Kali01])

currentThread: Der Name des momentanen Threads.

user: Der Benutzername (Auslöser der BTA).

componentClass: Der Java Klassenname des ausgeführten Bausteins

componentInstanceID: Eindeutige ID des Bausteins als Hashcode

taName: Der Name der ausgeführten Transaktion ist in der BeanBox frei wählbar.

status: Dieser Wert gibt Auskunft über den Erfolg einer ausgeführten Transaktion. Er kann den Wert "Success" oder "Failure" haben.

initiatedTaskID: Wird gesetzt, wenn die Action "initiatedTA" ist und erlaubt eine Zuordnung des Meßpunktes.

isAsynchronous: Feld um eine Nebenläufigkeit anzuzeigen.

newThread: Name des neuen Threads, wenn die Action "startControlFlow" ist. Ist notwendig, um den Thread trotz wechselnder Instanz zuordnen zu können.

info: Zusatzinformationen, die in dem Baustein erstellt wurden.

3.3.2.2 StatsServer - StatisticView

Die Verbindungen sind bidirektional. Der Client sendet einen Befehl als Zeichenkette[5] zum Server und erhält daraufhin ein vollständiges XML Dokument als Datenstrom[6] zurück. Das XML Dokument muß die gleiche DTD, wie in 3.4 besitzen, andernfalls wird das Dokument verworfen.

[4] Zeitmessung aus der Unix-Welt
[5] String
[6] Stream

3.4 XML Document Type Definition

Die Document Type Definition (DTD) ist die Grammatik eines XML Dokuments. Sie beschreibt die Abhängigkeiten der einzelnen XML-Tags[7] zueinander. Die folgende DTD beschreibt das Format des Datenstroms zwischen STATSSERVER und STATISTICVIEW :

```
<!ELEMENT btaList (Transaction*)>

<!ELEMENT Transaction
  (btaInstance,Action,Time,StartTime,Status,
  componentInstanceID,componentClass,currentThread,
  User,taName,Info,ExtraInfo,closed, syncTA*,
  asyncTA)
>

<!ELEMENT btaInstance (#PCDATA)>
<!ELEMENT Action (#PCDATA)>
<!ELEMENT Time (#PCDATA)>
<!ELEMENT StartTime (#PCDATA)>
<!ELEMENT Status (#PCDATA)>
<!ELEMENT componentInstanceID (#PCDATA)>
<!ELEMENT componentClass (#PCDATA)>
<!ELEMENT currentThread (#PCDATA)>
<!ELEMENT User (#PCDATA)>
<!ELEMENT taName (#PCDATA)>
<!ELEMENT Info (#PCDATA)>
<!ELEMENT ExtraInfo (#PCDATA)>
<!ELEMENT closed (#PCDATA)>

<!ELEMENT syncTA (Transaction)>
<!ELEMENT asyncTA (Transaction)>
```

Vergleicht man die DTD mit dem Format der Meßpunkte in 3.3.2.1 kann man Ähnlichkeiten erkennen. Werte, die zur Auswertung unnötig sind, wurden weggelassen. Neue sind:

StartTime: Das genaue Datum in menschen-lesbarer Form.

ExtraInfo: Ergänzende Informationen.

closed: Gibt Auskunft darüber, ob die Transaktion beendet wurde.

syncTA: Synchron aufgerufene Transaktionen

asyncTA: Asynchron aufgerufene Transaktionen

[7]eine Art Etikett, das einer Information einen Sinn gibt

Kapitel 4

Benutzerhandbuch

Dieses Kapitel ist das Handbuch für den Prototypen. Es beschreibt alle Arbeitsschritte die nötig sind, um das System in Betrieb nehmen zu können und damit umzugehen.

***Anmerkung**: Die JVM wurde unter SuSE 7.0 Linux (Kernel 2.2.16) übersetzt. Es sind einige Pakete notwendig, um zum Ziel zu gelangen: automake, flex, gcc, glibc, glibdev, diff, gtk, gtkdev, um nur einige Beispiele aufzuzählen. Verschiedene Kernel Versionen spielen bei der Ausführung der JVM keine Rolle, zumindest nicht bei Kernel 2.2.x.*

4.1 JVM

4.1.1 Installieren

Die erste und einfachere Möglichkeit besteht darin, die übersetzte JVM auf der CD zu benutzen. Dazu muß die beiliegende CD gemountet werden. Als root wechselt man anschließend in das Verzeichnis */usr/lib*. Normalerweise liegt hier schon der symbolische Link *java,* der z.B. auf */usr/lib/jdk1.1.8* zeigt. Man ersetzt ihn mit:

ln -s <path to CDROM>/java-modified /usr/lib/java

Die installierte Java Version bleibt so erhalten. Will man wieder wechseln, muß lediglich der Link neu gesetzt werden. Ist noch kein Java installiert gewesen, muß noch folgender Eintrag an die *PATH* Variable gehängt werden: *:/usr/lib/java/bin .* Ist man mit dieser Lösung einverstanden, kann man bei 4.3 fortfahren.

Die zweite Möglichkeit ist nicht trivial und sehr zeitaufwendig. Man muß eine Binär Version vom JDK 1.2 installieren, um die modifizierte JVM übersetzen zu können.

Wenn man nicht schon eine Binär Version des JDK 1.2.2 hat, kann man diese z.B. von der CD installieren. Sie liegt als RPM-Datei bereit.

rpm -i <path to CDROM>/jdk/java2.rpm

Danach legt man das Verzeichnis */usr/lib/jdk1.2.2-src*[1] an und entpackt die Quellen in diesem Verzeichnis:

bzcat <path to CDROM>/jdk/jdk1_2_2-src-linux.tar.bz2 | tar xf -

[1] wegen des Patches muß es *jdk1.2.2-src* heißen

4.1.2 Patchen

Nun werden die Quellen des JDK 1.2.2 für die Zwecke des FoPra gepatched. Man wechselt in das Verzeichnis /usr/lib und führt folgenden Befehl aus:

bzcat <path to CDROM>/patches/patch-jdk1.2.2-src-mod.bz2 | patch -p0

Nun muß auch die Binär Version gepatched werden

bzcat <path to CDROM>/patches/patch-jdk1.2.2-mod.bz2 | patch -p0

(Diese Version sollte in /usr/lib/jdk1.2.2 stehen)

4.1.3 Übersetzen

Man folge den Anweisungen in *jdk1.2.2-src/README.linux*. Nach dem Übersetzen befindet sich die JVM unter *jdk1.2.2-src/bulid/linux* in den Verzeichnissen */bin* und */lib*. Man kann die Klassen noch in einer JAR Datei zusammenfassen:

jar cvf /usr/lib/jdk1.2.2-src/build/linux/lib/rt.jar . (vorausgesetzt man befindet sich in */classes*)

Nun kopiert man sich die JVM in ein neues Verzeichnis, z.B. */usr/lib/jdk1.2.2-mod* und setzt den symbolischen Link darauf.

4.2 JAXP

Die *J*ava *A*PI for *X*ML *P*arsing wird benötigt, um mit dem XML Format hantieren zu können. Verwendet man die modifizierte Java Version auf der CD, sind keine weiteren Schritte nötig - JAXP ist bereits installiert.

Hat man die JVM selbst übersetzt, wechselt man in deren Verzeichnis, z.B. */usr/lib/java* und führt folgenden Befehl aus:

bzcat <path to CDROM>/jaxp/jaxp-1.0.1.tar.bz2 | tar xf -

4.3 BeanBox

4.3.1 Installieren

Die Java BeanBox 1.1 (BB) wird von Sun als ausführbare Datei angeboten. Als 'normaler' Benutzer gibt man folgenden Befehl in der Konsole (oder xterm) ein:

<path to CDROM>/bdk/bdk1_1-unix.bin

Man folge den Anweisungen.

4.3.2 Patchen und neu übersetzen

Auch die BB muß modifiziert werden. Man wechselt in das Installationsverzeichnis des BDKs, z.B. *~/BDK1.1* und führt folgenden Befehl aus:

bzcat <path to CDROM>/patches/patch-bdk1.1-mod.bz2 | patch -p0

Anschließend muß die BB neu übersetzt werden. Dazu wechselt man in das Verzeichnis */beanbox* und führt folgenden Befehl aus:

gmake clean ; gmake

4.3.3 Modifizierte Beans installieren

Die angepaßten Beans[2] installiert man im Installationsverzeichnis des BDKs mit:

bzcat <path to CDROM>/bdk/beans.tar.bz2 | tar xf -

4.3.4 BeanBox Manual (Quick Start)

Damit die BB mit dem STATSSERVER korrekt zusammenarbeiten kann, muß die Konfigurationsdatei *~/.MessObjektrc*[3] auf den richtigen Server zeigen. Läuft das System auf nur einer Maschine unter Verwendung der Grundeinstellungen, reicht der Eintrag

```
localhost
27333
```

in der Konfigurationsdatei. Enthält diese Datei falsche Werte, werden die Daten nicht versendet. Es können auch IP-Adressen statt Hostnamen angegeben werden.

Um die Ausgaben der BB auffangen zu können, muß der STATSSERVER gestartet sein. Man fahre zunächst bei 4.4 fort.

Die BB startet man im Verzeichnis */beanbox* mit dem Kommando

sh run.sh

Anschließend erscheinen, wie in Abb. 4.1 zu sehen, mehrere Fenster auf dem Bildschirm. Das rechte Fenster heißt Toolbox. Darin werden alle verfügbaren Java Beans angezeigt. Per Drag&Drop lassen sie sich im großen Hauptfenster zu einer Anwendung zusammensetzen. Das linke obere Fenster heißt Properties. Hier lassen sich die Eigenschaften der Beans verändern. Das darunterliegende Fenster liefert Statusinformationen; für diesen Fall darf es ignoriert werden.

Nun wird eine kleine Demo Anwendung zusammengebaut. Sie soll aus zwei SorterBeans und einem StartButton bestehen. Dazu zieht man zweimal die SorterBean in die BB und stellen bei beiden Sortern die Eigenschaft “sortBTA” von ’False’ auf ’True’ und bei einem der beiden Sorter den “algorithm” von ’BubbleSort’ auf ’QSort’. Dadurch wird ein Sorter schneller fertig als der andere. Um die Sorter starten

[2] Java Bausteine
[3] Config Datei muß im Benutzerverzeichnis plaziert werden

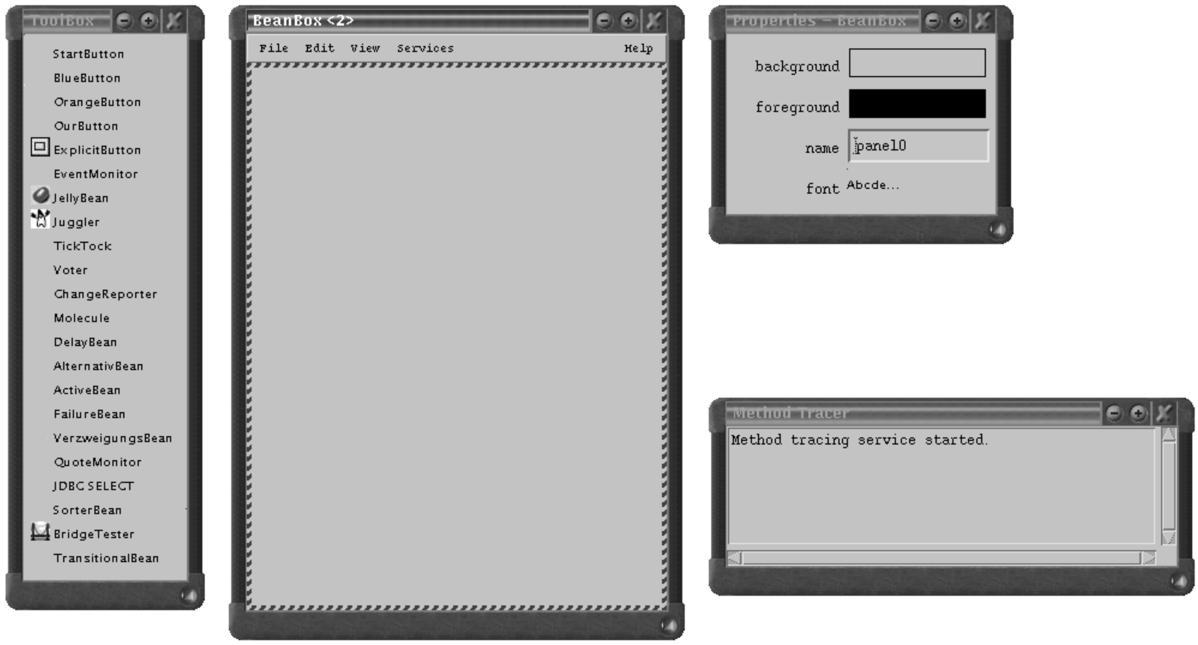

Abbildung 4.1: Die BeanBox nach dem Start

zu können, wird ein StartButton in die BB gezogen. Seine Eigenschaft "BTA_name" wird in 'DoubleSorter' abgeändert. Nun muß man die Beans miteinander verknüpfen. Der StartButton sollte dazu selektiert sein. Im Menü wählt man *Edit -> Events -> action -> actionPerformed* und ziehen den Link auf den ersten Sorter. Es erscheint der 'EventTagDialog'. Es wird 'starteSort' ausgewählt - die Beans sind verknüpft. Beim zweiten Sorter verfährt man genauso. Dazu selektiert man erneut den StartButton. Tip: Ein Klick knapp neben den Button selektiert ihn, ohne ihn auszulösen. Die BB sollte dann ähnlich aussehen, wie in Abb. 4.2. Löst man den Sortiervorgang aus, werden die Meßwerte an den Server weitergegeben. Das Ergebnis dieses Beispiels wird in 4.5.2 wieder aufgegriffen.

4.4 StatsServer

4.4.1 Installieren und übersetzen

In einem beliebigen Verzeichnis als normaler Benutzer mit:

bzcat <path to CDROM>/src/StatsServer_<Version>_<Date>.tar.bz2 | tar xf -

und anschließend im neu entstandenen Verzeichnis */server* mit

gmake

übersetzen.

4.4.2 Starten

Am einfachsten gelingt das im */server* Verzeichnis mit:

Abbildung 4.2: BeanBox mit DoubleSorter

java StatsServer

Der Server öffnet dann den Input Port auf *27333*, den Output Port auf *27334* und die Verfallzeit der BTAs wird auf *1440* Minuten (24 h) gesetzt. Ist man mit den Einstellungen nicht zufrieden, gibt man seine Wünsche als Parameter an:

java StatsServer 1111 9999 11

In diesem Fall wäre der Input Port *1111*, der Output Port *9999* und die BTAs würden nach *11* Minuten verfallen.

4.5 StatisticView

4.5.1 Installieren und übersetzen

In einem beliebigen Verzeichnis als normaler Benutzer mit:

bzcat <path to CDROM>/src/StatisticView_<Version>_<Date>.tar.bz2 | tar xf -

und anschließend im neu entstandenen Verzeichnis */viewer* mit

gmake

übersetzen.

4.5.2 StatisticView Manual

Der Aufruf

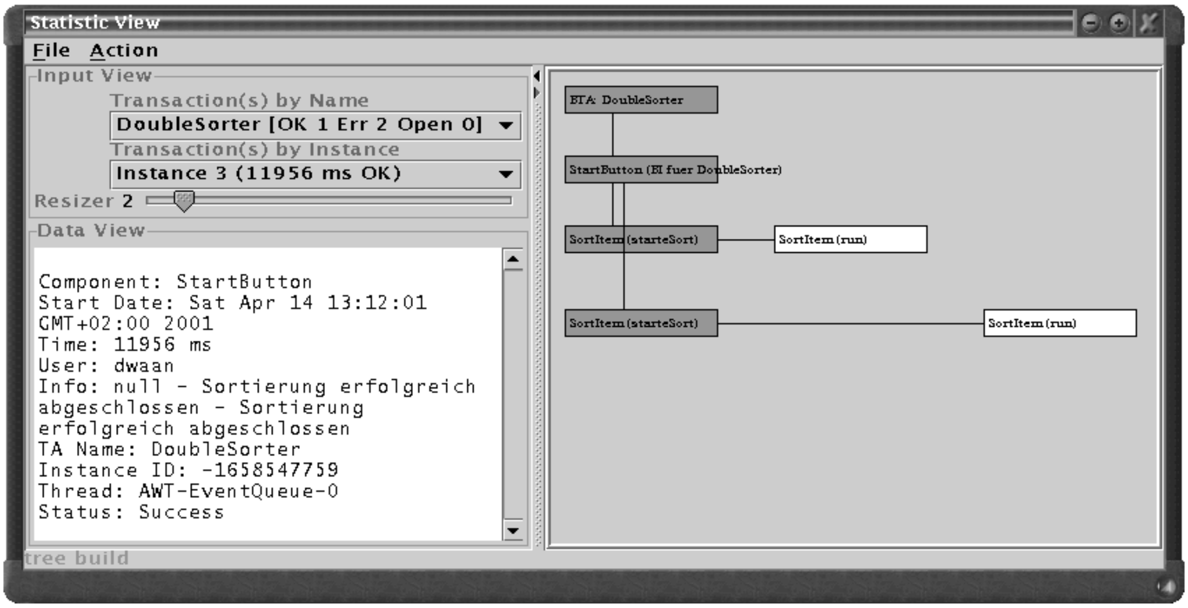

Abbildung 4.3: Transaktions Baum

java StatisticView

im gleichen Verzeichnis startet die graphische Benutzeroberfläche (s. Abb. 3.4). Im Menü “File” kann man Meßwerte als XML speichern oder laden. Letzteres erlaubt auch den Viewer ohne STATSSERVER auszuprobieren.

Nun sollen die gewonnenen Daten aus 4.3.4 ausgewertet werden. Dazu wählt man *Action -> Get XML Data* aus. Namen und Port des Servers sind anzugeben. Im gleichen Beispiel ist das *localhost* und *27334*. Anschließend müssen die Daten im Speicher mit *Action -> Build Component Tree* aufbereitet werden. In der ersten ComboBox erscheint “DoubleSorter ...”. Alle BTAs mit gleichem Namen werden hier zusammengefaßt. Der Sorter wird nocheinmal explizit ausgewählt. In der darunterliegenden ComboBox erscheint “Instance x ...”. Selektiert man auch diese, baut sich ein bunter graphischer Baum im rechten Bereich, wie er in Abb. 4.3 zu sehen ist, auf. Durch einen Mausklick auf die Rechtecke, kann man näheres über die Transaktionen erfahren. Der ’Resizer’ erlaubt den Maßstab des Baumes zu verändern.

Das erste Rechteck steht für die Benutzertransaktion. Alle weiteren vertikal verbundenen Rechtecke stehen für synchrone Transaktionen in einer Bean. Die horizontal verbundenen signalisieren asynchron aufgerufene Transaktionen. Klickt man in die beiden weißen Sorter Felder, kann man feststellen, daß ein Sorter schneller fertig war, aber später als erster gestartet wurde. Die Gesamtzeit der BTA richtet sich nach der letzten Transaktion, die beendet wird. Grüne Felder signalisieren erfolgreiche Transaktionen, gelbe unvollständige, rote fehlerhafte und weiße abgeschlossene mit unbekanntem Status.

Kapitel 5

Weiterentwicklung

Die prototypische Anwendung STATISTICVIEW läßt einfache Aussagen über Qualität und Dauer aufgerufener Komponenten zu. Die gewünschte Stabilität und Funktionalität entsprechen der Anforderungsanalyse. Mehr Zeitaufwand ist jedoch erforderlich, um den Prototyp zu einer einsatzbereiten Lösung zu erweitern. Die Anforderungen müssen für den tatsächlichen Einsatzort der Anwendung neu angepaßt werden. In Kapitel 3.2.1.3 wurde bereits darauf hingewiesen, die Abfragen des STATSSERVERS zu verfeinern. Auch ließe sich der Server noch verbessen, wenn es um die Bedienung von vielen Clients ginge. Eine Möglichkeit, den Datentransport zwischen MESSOBJEKT und Server nahezu 100% zu sichen wäre, über einen speziellen Port auf einen freien Port zu wechseln, ähnlich wie es einige Internetdienste[1] handhaben. Für jeden dieser Ports startet man einen neuen SOCKETREADER Thread. Ähnlich kann man auch am anderen Ende verfahren (APPLETSERVER). Jedoch ist es unwahrscheinlich, daß bei n MESSOBJEKTEN und einem Server auch n Viewer auf den Server zugreifen werden. Mit wachsendem Datenbaum dauert jede Übertragung im Vergleich zu kleinen Datenbäumen andererseits auch länger, die den einzigen Übertragunsport blockieren. Eine Zugangskontrolle und Verschlüsselung könnte den XML Datenstrom vor Unberechtigten schützen. Ein Aspekt, der beim Einsatz des Systems über das Internet sehr wichtig ist.

Es gibt viele Ideen den Protypen zu verbessern, zu beschleunigen oder zu erweitern. Vielleicht setzt an dieser Stelle ein anderer Praktikant meine Arbeit fort ? ...

[1] ftpd, telnetd

Literaturverzeichnis

[Hau01] Hauck, R.: Architektur für die Automation der Managementinstrumentierung bausteinbasierter Anwendungen, Dissertation, Ludwig-Maximilians-Universität München, to be published 2001.

[Kali01] Kalix, E.: Prototyp zur Automation von Performanz-Messungen in JavaBeans-basierten Anwendungen, Ludwig-Maximilians-Universität München, FoPra, 2001.

[BrDu00] Brügge, B.;Dutoit, A.: Object-Oriented Software Engineering - Conquering Complex Changing Systems. Prentice Hall, Upper Saddle River, NJ 07458, 2000.

[XML00] Bray, T.; Paoli, J.: Extensible Markup Language 1.0, W3C Recommendation, *http://www.w3c.org/TR/2000/REC-xml-20001006.pdf*, 2000.