Projektgruppe 470

jETI - Electronic Tool Integration Plattform

Endbericht

11. Mai 2006

Veranstalter
Lehrstuhl 5, Universität Dortmund

Betreuer
Prof. Dr. Bernhard Steffen
Ralf Nagel
Harald Raffelt

Autoren

Hatim Kenzi	Iris Paternoster
Jiong Yu	Akif Köse
Fifame Pascale Judith Metozounve	Moctar Zaidane
Markus Kenkmann	Norman Braß
Zhongyue Guan	Armelle-Claude Ekoto Nyangono
Hunervan Barzani	

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

Das Handy ist immer mehr zum Alltagsgegenstand geworden und ist so für manche nicht mehr wegzudenken. Bei der Wahl des Handy´s geht man in den Handyladen, guckt sich die verschiedenen Geräte an, testet die Menüführung, die Optik, das Gewicht und die Maße der Geräte. Am Ende kann man sich für ein Handy entscheiden oder auch nicht. Beim Aussuchen einer Software ist es nicht so einfach. Man muss jede zu testende Software erst einmal installieren, um sie überhaupt testen zu können. Im Allgemeinen testet man mehrere Programme von verschiedenen Herstellern, nach der obigen Prozedur. Man installiert ein Programm, das entweder in der Zeit oder der Funktionalität limitiert ist. Dieses testet man nun ausgiebig und kommt zum Entschluß die Software nicht zu erwerben, was auch immer der Grund hierfür ist. Man will die Software nicht mehr auf der Festplatte haben, also wird die Software deinstalliert. Dies ist aber nicht gerade so, wie man es sich wünschen würde, denn mit einer Deinstallation ist nicht alles wieder in dem Zustand, wie vor der Installation der Software, sondern es sind oft noch Teile der Software auf der Festplatte und in der Registry (Microsoft Betriebssysteme) und das will man bestimmt nicht. Genau hier kommt die jETI Platform zur Anwendung. Die jETI-Plattform stützt sich auf die Ideen des ETI-Systems und nutzt die neuen Technologien des Webservices. Das Electronic Tool Integration kurz ETI System ist eine Online Plattform zum Testen und Experimentieren von Software über das Internet. Es wurde im Jahr 1996 entwickelt und ist seit Anfang 1997 verfügbar. Es bestand aus einem zentralen Server, der über einen Java Applet Client bedient wurde. Seine Grundidee ist es, eine große Anzahl von verschiedenen Softwaretools einer breiten Masse von Benutzern, mit geringem Aufwand für die einzelnen Benutzer verfügbar zu machen. Dieses war besonders für neue Benutzer sehr einfach, da Sie eine Orientierung bzgl. der Vielzahl von existierenden Tools bekommen haben. Bei anwendungsspezifischen Problemen, für deren Lösung ein ETI Neuling keines seiner lokalen Tools einsetzen konnte, ermöglichte es ETI, die passenden Tools zu identifizieren. Mit Hilfe dieser ausgewählten Tools konnte der ETI Benutzer nun sein spezielles Problem lösen. Es ist seitdem für einen Benutzer des ETI Systems möglich geworden

- Tools einzeln auszuführen
- Tools mit verschiedenen Funktionalitäten in geeigneten Sequenzen zu kombinieren und diese Sequenzen auszuführen
- Informationen über Tools abzufragen.

Das ETI System stellte durch seinen Online Service ein vollständiges Spektrum an Koordinierungsfähigkeiten und Auswertungsfähigkeiten dar, welche auf der MetaFrame open tool [SMCB96] Koordinationsumgebung basierten. Zum Teil, wurden einfache und komplexe Kombinationen von Tool-Funktionalitäten automatisch oder auf interaktiver Basis zusammengebaut und getestet. Dieses entstand durch Anwendung einer einfachen Spezifikationssprache, die für lose Beschreibungen von Tool Sequenzen entworfen wurde. Der Erfolg der ETI Plattform war stets von zwei wesentlichen Faktoren abhängig:

- eine breite Tool-Palette musste im Tool Repository angeboten werden, damit die Mitglieder der ETI Online Community diese für Test- und Experimentierzwecke benutzen konnten
- der Integrationsprozess für einen Tool Provider sollte so einfach wie möglich gemacht werden, um die Attraktivität des ETI Systems zu erhöhen.

Allerdings gab es Faktoren, die für den ETI Integrationsansatz nachteilige Folgen hatten. Die Trennung der Aufgaben- und Verantwortungsbereiche zwischen dem ETI-Team, insbesondere dem Tool Integrator und dem Tool Provider war ein Nachteil. Ein zweiter nachteiliger Aspekt war, dass der Tool Integrator und der Tool Provider genügend technische Kenntnisse über die Tool Eigenschaften, sowie Hintergrundwissen über die Funktionalität der ETI Plattform besitzen mussten. Die Webservice Technologie ist eine Basis für dauerhafte herstellerunabhängige Schnittstellen zwischen Methoden und wird vom W3C Konsortium standardisiert. Wenn Datenformate auf Basis von XML Schemata und Protokolle auf Basis des standardisierten SOAP definiert werden, kann eine herstellerübergreifende Kommunikation zwischen Anwendungen ermöglicht werden. Diese Kombination (ETI und Technologie des Webservices) hat viele Vorteile für das jETI-System mit sich gebracht. Im jETI-System wird ein verteilter Webservice basierter Ansatz durchgeführt - die Tools werden vom Server des Tool-Providers aus über das Internet ausgeführt und nicht mehr von dem zentralen Server des ETI-System. Im folgenden wird auf den Aufbau und die Funktionalität des jETI-Systems eingegangen.

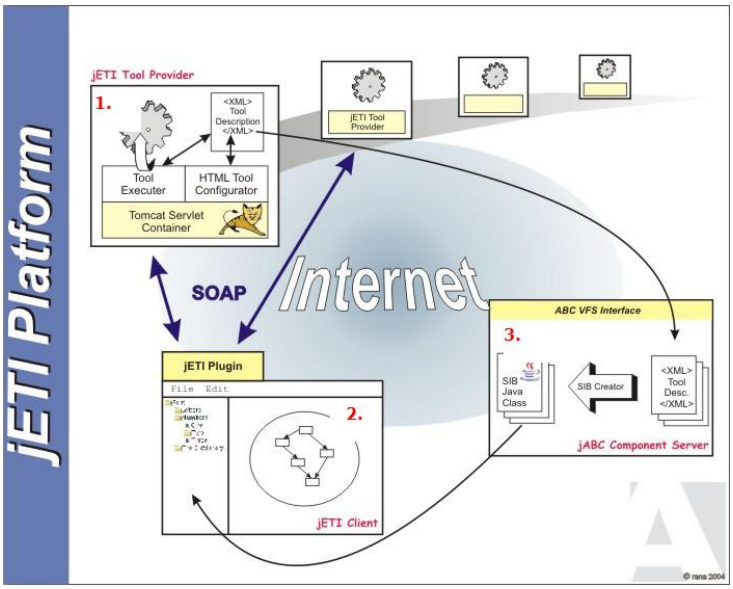

Abbildung 1: jETI Platform

Die jETI Platform besteht aus den folgenden Teilen:

- jETI Client
- jETI Tool Provider
- jABC Component Server

1.1 Der jETI Client

Der jETI Client ist die grafische Oberfläche des Java ABC-Framework, kurz jABC Framework genannt. Der jABC Client ist ein Bestandteil des jETI Systems (siehe Abbildung 2).

Das System verwendet eine graphische high-level Programmierschicht, in der aus Komponenten (SIB's) hierarchische Graphen erstellt werden. Durch diese abstrakte Modellierungsebene braucht der Anwender keinerlei Programmierkenntnisse. SIB's (Service Indepedents Building

Blocks) sind die wiederverwendbaren Komponenten, die jABC zur Modellierung eines Systems braucht. Jedes SIB hat eine Definition durch eine SIB Klasse.

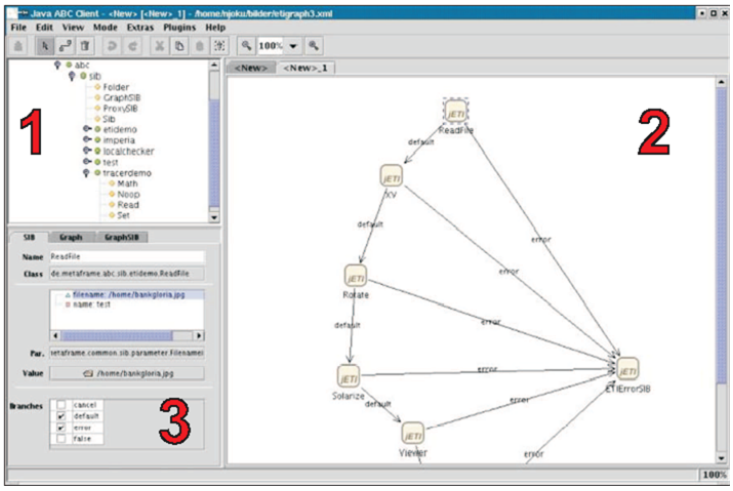

Abbildung 2: jETI-Client

Eine minimale SIB Klasse ist eine einfach Java Klasse, ohne jegliche Parameter und Branches. SIBClass ist ein Interface, mit dem man die SIB Klassen erkennt. Das zweite Interface ist Serializable. Die SIB Klasse enthält auch die Onlinehilfe zum jeweiligen SIB. Diese wird bei der Übertragung des SIB's zwischen Server und Client gebraucht. Es muss darauf geachtet werden, dass jedes SIB diese Interfaces implementiert. Jedes SIB hat eine weltweit eindeutige Identifikationsnummer, die Universal ID. Die UID eines SIB's muss immer gleich bleiben, auch wenn die SIB Klasse umbenannt oder in ein anderes Package verschoben wird. Nur wenn eine nicht rückwärts kompatible Erweiterung vorgenommen wird, muss eine neue UID vergeben werden. Jedes SIB darf ein eigenes Icon haben. Dieses Icon kann sogar unter wechselnden Zuständen des SIB's unterschiedlich sein. Die Einsatzgebiete des Systems sind z.B: Entwicklung von Web Services, Beschreibung und Ausführung von Workflows. Das Gesamtsystem hat drei grobe Bestandteile: jABC Client, jABC Server und Eclipse; die Entwicklungsumgebung. Diese Entwicklungsumgebung ähnelt mit ihrem Plug-In Konzept dem Java ABC. Diese Plugins ermöglichen eine enge Integration der Eclipse IDE in das Gesamtsystem. Der jABC Client ermöglicht die Modellierung von Systemen mit Hilfe von Graphen. Der Anwender kann nach einer kurzen Schulung durch Benutzung der einfachen Benutzeroberfläche sein eigenes System modellieren.

Die Grundidee des jETI Clients für den Benutzer besteht darin, die von jABC Komponenten-Server zur Verfügung gestellten Komponenten (SIB's), zur Modellierung von Systemen zu verwenden. In dem dargestellten Graphen sind die Knoten SIBs, die lokale oder verteilte Anwendungen sein können, die Kanten sind gerichtet und beschriftet und heißen Branches. Bei Ausführung eines Graphen, der ein Tool eines Tool-Anbieters enthält, wird eine Kommunikation mit Hilfe des jETI Plugins über SOAP mit dem Tool Executor stattfinden. Der Tool-Executor wertet die Eingabe des jETI Clients aus und gibt ein Ergebnis zurück. Es existieren zwei verschiedene Arten von SIB's:

- Tool SIB's, die die bereits verfügbaren externen jETI Tools darstellen. Ein Tool SIB beinhaltet die nötigen Parameter, die später an den Adapter eines entfernten jETI Tools übertragen werden. Die Tool SIB's werden von dem SIB Creator Programm im JABC Server generiert

- Spezielle I/O-SIB's, auch Input/Output-SIB's genannt, die eine lokale Aufgabe erfüllen

1.2 Der Tool-Provider

Der Tool-Provider ist der Tool-Server des Tool-Anbieters. Er besteht aus einem webservicebasierten Adapter, der direkt mit dem jETI Client anhand des jETI Plugins über SOAP kommuniziert. Dieser Adapter basiert auf dem Tomcat Servlet Container. Das jETI-System stellt dem Tomcat-Servlet Container zwei verschiedene Web-Applikationen, die Subsysteme des Tool-Providers bilden, zur Verfügung:

- HTML Tool-Konfigurator
  Der Tool-Anbieter kann mit Hilfe des HTML Tool-Konfigurators ein Tool einfach in der jETI Plattform anmelden. Nach Beschreibung eines Tools anhand des Tool-Konfigurators wird eine Steuer-Datei(XML-Datei) erzeugt. Eine Kopie dieser Steuer-Datei wird automatisch an den jABC Komponenten Server gesendet.

- Tool-Executor
  Der Tool-Executor erkennt über eine XML-Datei alle von Tool-Anbietern freigegebene lokale Tools. Nach Erkennung wird aus statischen und dynamischen Parametern entweder eine Kommandozeile, die vom Betriebssystem ausgeführt wird oder eine Java Klasse, die aufgerufen wird. Im ersten Semester der Projekt-Gruppe wird der Aufruf eines Tools durch einen CORBA-Client ermöglicht.

1.3 Der jABC Komponenten Server

Der jABC Komponenten Server stellt dem jETI Client alle benötigten Elemente (SIB's) zur Verfügung. Die Zugriffe auf die einzelnen SIB's werden durch Benutzer-Rechte geschützt und kontrolliert. Im jETI-System speichert der jABC Komponenten Server die vom HTML Tool-Konfigurator gesendete Datei. Daraus erzeugt er eine SIBClass durch den SIB-Creator und stellt dem jETI Client bereit. Der jABC-Server erlaubt das Speichern beliebiger externer Daten, die zu einem Modell gehören. Er dient auch zur Versionierung der SIB's und der Modelle.

2 Motivation

Am Anfang bestand die Vorgänger Version vom ETI aus einem zentralen Server, der über einen Java Applet Client bedient wurde. Dabei erwies sich aber die Integration der unterschiedlichen Tools, die zum Teil aus heterogenen Betriebssystem- und Hardwareplattformen stammten, als besonders schwierig, sowohl für die Tool-Provider als auch für die Tool-Integratoren.

Das jETI System setzt im wesentlichen auf dem Konzept der ursprünglichen ETI Plattform auf. Das jETI löst sich vom ETI durch die Einführung einer neuen jETI Architektur. Diese neue Architektur ist gekennzeichnet durch eine Verteilung der jETI-Tools. Die einzelnen Tools müssen nicht mehr an zentraler Stelle im ETI Server verfügbar sein, sondern können im Netz verteilt angeordnet werden (z.B. beim jeweiligen Software Hersteller des Tools). Praktisch ist man dabei zu einem verteilten Web Services Ansatz übergegangen. Die lokalen jETI-Tools werden mit einer Erweiterung versehen, die es ermöglicht, diese Tools dann per Netzwerk aufzurufen.

Das jETI ist eine Lösung zur Steigerung an Benutzerfreundlichkeit. Es setzt auf der Programmiersprache Java und Web Services Technologien auf, die es ermöglichen, ein einheitliches Software-Framework für unterschiedliche Plattformen und Anwendungen in einem heterogenen Netzwerk zur Verfügung zu stellen.

3 Zielsetzung

3.1 Aufgaben der PG

Im Rahmen dieser PG sollten vier Aufgaben gelöst werden. Die erste Aufgabe bestand darin, einen HTML Client zu realisieren. Dabei sollte es möglich sein, Graphen zu erstellen. Die zweite Aufgabe war darauf basiert ein Licence Manager zur Überwachung von JETI-Tools aufzubauen. Dazu sollte ein Corba Interface implementiert werden. Die Statistik von Webseiten definierte die dritte Aufgabe. Für die Statistik sollten Filter definiert und gespeichert werden, um die Web-Anwendung zu entwickeln. Die vierte Aufgabe war den JABC Client zu erweitern. Dafür sollte eine Parallele Ausführung ermöglicht werden.

3.2 GesamtZeitplan

Der Gesamtzeitplan sieht folgendermaßen aus:

Gruppe 1: HTML Client
Im jETI System gibt es den jETI Client, mit dem man sein System mit Hilfe eines Graphen modellieren kann. Die Aufgabe der Gruppe war nun, einen eingeschränkten HTML Client zu entwickeln. Hierbei sollte es möglich sein, einen Graphen zu zeichnen, dessen Kanten beschriftet werden können. Einfache Knotenparameter sollten verändert werden können, ferner sollte der Graph gespeichert und geladen werden können. Desweiteren sollte eine webbasierte Userverwaltung zusammen mit der Gruppe 2 eingerichtet werden. Im zweiten Semester ist dann die ETI Anbindung erfolgt. Spezielle Komponenten wie `ReadFile`, `ShowFile` oder `WriteFile`, die schon im ETI System vorhanden sind, wurden implementiert. Zum Schluss sollten die Graphsequenzen durch einen Interpreter ausgeführt werden können.

Gruppe 2: Tool Executor
Die Gruppe hat sich mit den zwei Web-Applikationen des Tool-Providers beschäftigt. Im ersten Semester wurde an dem Tool-Executor gearbeitet, welcher für die Toolaufrufe verantwortlich ist. In den Tool-Executor soll ein Lizenzmanager integriert werden, welcher die Toolaufrufe kontrolliert. Bisher konnte man keine Tools mit Lizenz in das System integrieren. Dies soll mit dem Lizenz-Manager ermöglicht werden. Ferner wurden alle Toolausführungen in eine Datenbank geschrieben werden, welche von Gruppe 3 ausgewertet wurde. Man könnte bisher Tools per Befehlszeile oder durch eine Java-Klasse aufrufen. Nun soll der Tool-Executor um ein CORBA Interface erweitert werden, so dass Tools mit CORBA-Schnittstelle aufgerufen werden können. Im zweiten Semester wurden verschiedene Lizenz-Module zur Verfügung gestellt. Des Weiteren wurde der HTML Tool-Konfigurator erweitert. Es sollte möglich sein die verschiedenen Lizenz-Module und die Aufrufmethode für CORBA einzustellen.

Gruppe 3: Statistik Web Applikation
Im ersten Semester wurde zusammen mit Gruppe 1 und 2, eine webbasierte Userverwaltung

erstellt. Die Gruppe hat sich hauptsächlich mit Statistiken beschäftigt und hierfür eine Web Applikation implementiert. Die von Gruppe 2 in die Datenbank geschriebenen Informationen über die Tools und deren Ausführung sollten zu einem Report zusammengefasst und gespeichert werden. Hierbei sollten die Art der Charts und die Zuordnungen der Achsen frei wählbar sein. Schließlich sollten die Reports aufgerufen und ausgeführt werden können. Im zweiten Semester wurde die Web Applikation erweitert. Es sollten Filter für die Reports definiert werden. Diese sollten auch gespeichert werden können. Ferner sollte es Zwangsfilter für bestimmte User geben, welche die Einsicht auf die Auswertungen beschränken. Die Filter sollten schließlich angewendet werden können.

Gruppe 4: Parallelausführung
In Kooperation mit Gruppe 2 wurde eine ETI Userabfrage ermöglicht. Ferner wurde der vorhandene Tracer für Threads erweitert. Des Weiteren wurden auch zwei spezielle Komponenten entwickelt, welche die Parallelausführung erst ermöglichen. Die Fork Komponente soll dazu dienen, mehrere Graphsequenzen, die unabhängig voneinander sind, parallel abzuarbeiten. Die durch Fork initialisierten Threads werden schließlich durch das Join SIB synchronisiert, das heißt es muss solange gewartet werden, bis alle vom Fork ausgehenden Graphsequenzen beendet werden und im JOIN eingehen. Nachdem die parallele Ausführung der Graphsequenzen implementiert wurde, sollte ein Mechanismus entwickelt werden, der das Erkennen der für die parallele Ausführung geeigneten Graph Situationen realisiert und den Graphen dementsprechend umbaut. Weiterhin werden Graphen nach festgelegten Fehlern untersucht. Die im ersten Semester gemachten Änderungen am Tracer wurden nun im Hinblick auf die Graph Hierarchie erweitert.

3.3 Gruppeneinteilung

Die 12 Teilnehmer wurden in die oben vorgestellten Untergruppen unterteilt.
Gruppe 1: HTML Client
Hatim Kenzi, Iris Paternoster, Jiong Yu
Gruppe 2: Tool Executor
Akif Köse, Fifame Pascale Judith Metozounve, Moctar Zaidane
Gruppe 3: Statistik Web Applikation
Markus Kenkmann, Norman Braß, Zhongyue Guan
Gruppe 4: Parallelausführung
Armelle-Claude Ekoto Nyangono, Hunervan Barzani

3.4 Vorbereitung der Projektgruppe

Die erste Sitzung fand am 26.01.2005 statt. Eine Auflistung der Seminar Themen wurde verteilt. Deadlines, die bei der Vorbereitung des Seminars eingehalten werden sollten, wurden aufgelistet. Um die oben beschriebenen Aufgaben zu bewältigen, war es nötig sich über verschiedene Themen zu informieren und diese zu verstehen. Jeder Teilnehmer der Projektgruppe hat ein Thema von 12 vorhandenen Themen ausgewählt und nach gründlicher Recherche eine Zusammenfassung erstellt. Ferner wurde von jedem eine Präsentation erarbeitet, welche vor der Projektgruppe vorgetragen wurde. Üblicherweise findet die Präsentation des Seminars in einem Tagungshotel statt. Nach einer Abstimmung wurde entschieden, die Präsentation im Seminarraum des Lehrstuhls vorzutragen. Die Seminar -Themen wurden wie folgt verteilt

	Seminar Thema	Vortragende
1	Eclipse CVS	Markus Kenkmann
2	Corba	Fifame Pascale Judith Metozounve
3	Parallele Algorithmen	Sergej Baranov
4	JFreeChart	Norman Braß
5	UML	Hunervan Barzani
6	SOAP	Zhongyue Guan
7	jABC	Iris Paternoster
8	jETI	Moctar Zaidane
9	Hibernate	Akif Köse
10	JSP / Servlet	Jiong Yu
11	Tomcat	Hatim Kenzi
12	jUNIT	Armelle-Claude Ekoto Nyangono

4 HTML Client

von Paternoster, Iris; Kenzi, Hatim; Yu, Jiong

Eine der Teilaufgaben der Projektgruppe war es, einen HTML basierten Client zu entwickeln, mit dem auf der Benutzerseite der Zugang zu einem jETI System ermöglicht wird. Ein solcher Client ist erforderlich um von der Benutzerseite aus auf das jETI System zuzugreifen. Der schon vorhandene jETI-Client ist eine Java-Applikation. Die voraussetzt, dass auf der Benutzerseite eine Java Virtual Machine (JVM) existiert.

Der HTML-Client soll es ermöglichen, auch ohne JVM auf das jETI System über eine Internetverbindung zuzugreifen. Er muss aber so beschaffen sein, dass der Zugriff auch von Rechnern aus möglich ist, auf denen die Ausführung der Java-Applikationen aus Sicherheitsgründen gesperrt ist.

4.1 Aufgaben

Das bestehenden jETI-System ermöglicht dem Benutzer die Darstellung seiner Graphen durch die “SIB“s genannten Knoten. Diese Knoten sind miteinander durch eine Kante verbunden. Damit an der Die ganze Grapherstellung und Bearbeitung soll HTML basiert sein.

Im jETI-System existiert ein Tracer-Plugin, dass die Ausführung des erstellten Graphen ermöglicht. Die Ausführung startet immer von einem Anfangsknoten aus. Danach wird der Nachfolgeknoten festgestellt und ausgeführt. Dabei wird die verarbeitete Datei an den Nachfolger weitergeleitet.

Diese Eigenschaft soll auch in HTML-Client vorhanden sein. Dafür mussten auch die lokale I/O-SIBs `ReadFile`-, `ViewFile`- und `WriteFile`-SIB’s entwickelt werden. `ReadFile`-SIB soll ermöglichen, Dateien, in denen der Benutzer arbeiten will, für das System zur Verfügung stellen. Mit der `ViewFile`-SIB wird die verarbeitete Datei angezeigt, mit der `WriteFile`-SIB kann der Benutzer die Datei auf seinem System speichern.

Das jETI-System soll es ermöglichen, verschiedene Programme ohne Installation auf der Benutzerseite auszuführen. Theoretisch ist es jetzt möglich, dass jeder mit Zugang zum Internet

das jETI-System benutzen könnte. Deshalb ist eine Userverwaltung erforderlich, die registrierten Benutzern verschiedene Rechte zu teilt, in deren Rahmen sie sich in das System einloggen können.

4.2 Benutzte Technologien

4.2.1 Servlets und JSP

HTML-Seiten sind statische Seiten. Um das jETI-System nutzen zu können, muss die Kommunikation zwischen dem Benutzer und dem jETI-System möglich sein. Das kann auf verschieden Weise ermöglicht werden.

Eine Möglichkeit wäre, das CGI (Common Gateway Interface) zu benutzen. CGI ist ein Standard für die Schnittstellen zwischen externe Applikationen und Information-Servern wie HTTP-, WEB-Server. Ein CGI-Programm kann in verschiedene Sprachen z.B. C/C++ , PERL, Visual Basic entwickelt werden. Dabei existiert leider die Unflexibilität des CGI-Programmierung, weil die Sprache vom System ausgewählt muss. Falls C oder Fortran benutzt worden ist, muss es vorher noch compiliert werden.

Anstelle der traditionellen CGI-Programmierung wurde bei der Entwicklung des HTML-Clients Java-Servlets bevorzugt. Die Vorteile von Java Servlets gegenüber der CGI-Programmierung sind folgende:

- Effizienz: Bei der CGI-Programmierung wird mit jedem HTTP-Aufruf ein neuer Prozess gestartet. Wenn ein Servlet aufgerufen wird, wird jede Anfrage durch einen Thread verwaltet und nicht durch einen Prozess. Falls Anfragen zu einer CGI n-mal aufgerufen werden, wird der Code des CGIs n-mal geladen. Beim Servlets dagegen existieren zwar n Threads, aber nur eine Kopie der Servlet-Klasse ist im Speicher.
- Praktisch: Servlets haben eine umfangreiche Infrastruktur für automatisches Parsen und Dekodieren der HTML-Dataforms sowie Lesen und Einstellen der HTTP-Headers, das Handeln von Cookies, das Nachführen von Sessions, usw.
- Mächtig: Java Servlets können direkte Verbindungen zum Web-Server aufnehmen, was einfache CGI Programme nicht können. Diese Eigenschaft vereinfacht die Suche nach gespeicherten Bildern oder Daten. Servlets können auch Daten miteinander austauschen.
- Portabel: Servlets werden in Java geschrieben und können ohne Änderung durch verschiedenen Servlet Containers wie z.B. Apache-Tomcat, IBM-WebSphere oder JBoss ausgeführt werden.

Neben Servlets wurde die JSP (Java Server Pages) Technologie benutzt, um die dynamische Darstellung der HTML-Seiten zu ermöglichen. Mit der JSP Technologie kann man statische HTML-Seiten mit den dynamisch generierten Seiten vermischen. Viele Webseiten, die mit CGI Programme erstellt wurden, werden als ganze Seite generiert. JSP ermöglicht die Trennung der beiden Teile. Die Vorteile der JSP Technologie gegenüber anderen Technologien können wie folgt aufgelistet werden:

- ASP (Active Server Pages): ASP ist eine der JSP ähnliche Technologie vom Microsoft. JSP ist in Java geschrieben und daher gegenüber dem in Visual Basic geschriebenen (oder andere Microsoft Spezifische Sprache) ASP unabhängiger und portabler.

- Reine Servlets: Theoretisch gesehen können JSP nicht mehr tun als ein Servlet. Es ist aber pragmatischer, normalen HTML-Code zu schreiben als Millionen "`println`" Befehle innerhalb der Servlets zu schreiben, um den HTML-Code zu generieren. Und es ist auch praktischer, Design und Inhalt zu trennen.

- Java Script: Java Script kann dynamische HTML-Seiten generieren. Es ist eine brauchbare Eigenschaft, aber macht nur dann Sinn, wenn die dynamische Information auf der Umgebung des Benutzers basiert. Er hat keinen Zugriff auf Server-Side Quellen wie Datenbanken. Außerdem ist die Anwendung von JavaScript in vielen Browsern aus Sicherheitsgründen vom Benutzer ausgestellt.

4.2.2 Hibernate

Die Relationale Datenbank Management Systeme (RDMS) sind nicht Java-Spezifisch. Die Relationale Datenbanken sind auch nicht für eine Applikation bestimmt. Die Relationale-Technologie ermöglicht den Datenaustausch zwischen verschiedenen Applikationen oder Technologien, die ein Teil derselben Applikation sind.

RDMSe haben eine SQL-basierte Applikation-Interface, daher werden sie auch SQL Datenbank Management Systeme oder SQL-Datenbanken genannt.

Wenn man innerhalb einer Java-Applikation auf einen SQL-Datenbank zugreift, wird der Java-Code benötige SQL-Befehle durch einen "Java DataBase Connectivity" JDBC-API an die Datenbank senden.

Man verwendet die JDBC-API um die Argumente zu Suchanfrageparametern einzubinden, um den Ausführung des Suchanfrages zu beginnen oder etwas bestimmtes von der Ergebnissen zu abfragen u.s.w.

Wenn man eine objektorientierte Software entwickelt, sollten komplexe Objekte, die Instanzen von Klassen sind, in der Datenbank gespeichert und wieder aufgerufen werden können. Das geschieht durch Objekt/Relational Mapping.

Unter dem Namen Object/Relational Mapping versteht man folgendes: Objekte werden mit Hilfe eines Persistenz Layers in eine Datenbank abgebildet, also gespeichert. Die Speicherung wird so vollzogen, dass man später diese Daten wieder verwenden kann. Man kann dieses Mapping in beide Richtungen ausführen, also man kann entweder aus einer bestehenden Datenbank die jeweiligen Java Dateien oder aus Java Dateien eine Datenbank (Tabellen) erhalten.

Hibernate ist ein Object/Relational Mapper, der wie oben schon erwähnt das Mapping in beide Richtungen ausführen kann. Die Vorteile des Hibernates kann wie folgt erwähnt werden:

- Hibernate unterstützt eine Vielzahl von Datenbanken wie z.B. MySQL [MySQL()], Oracle [ORACLE()], MSSQL [Microsoft()], PostgreSQL [PostgreSQL()] und mehrere andere.

- Es unterstützt auch mehrere Datenbanken auf einmal durch hinzufügen weiterer Konfigurationsdateien.

- Es unterstützt ausführliches Logging mittels Log4j [Foundation()]. Mit dessen Hilfe man sich alle internen vorgehensweisen, wie z.B. die generierten SQL Befehle, von Hibernate ausgeben lassen kann.

- Desweiteren hat es eine gute, objektorientierte Query-Sprache, die Hibernate Query Language(HQL) [hibernate.org()]. Die HQL hält die Applikation unabhängig vom SQL-Dialekt der verwendeten Datenbank.

4.3 Architektur des HTML-Clients

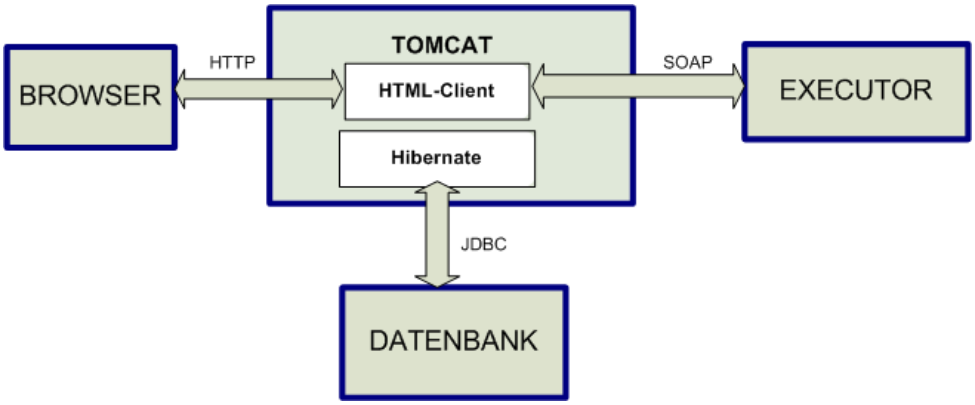

Abbildung 3: Kommunikation des jETI HTML-Clients

Eine der wichtigsten Eigenschaften des jETI-Systems ist seine Plattform-Unabhängigkeit. Es wurde Datenbank-Typ unabhängig entwickelt. Dafür wurde die Hibernate-Technologie benutzt. Die Datenbanken können PostgreSQL, MySQL, Oracle oder andere sein.

Wenn der Benutzer sich in das jETI-System einloggen will, werden seine Eingaben via HTTP an den WEB-Server geschickt. Die Anfrage wird durch Hibernate formuliert und via JDBC an die Datenbank gestellt. Diese Datenbank-Anfragen werden innerhalb der Servlets bearbeitet und die Ergebnisse innerhalb einer JSP-Seite dargestellt.

Falls der Benutzer sein Graph ausführen will, übermittelt er diesen via HTTP an den WEB-Server, der die Anforderung ausführt. Die Ausführung des Tools wird durch SOAP-Aufrufe innerhalb des Executor gemacht, und das Ergebnis wird wieder an den WEB-Server geliefert.

4.4 Modellierungsphase

Bei einem starken Internetauftritt, spielt ein schönes Design eine nicht zu unterschätzende Rolle. Aus diesem Grund wurde versucht, die Webseiten attraktiv und übersichtlich zu gestalten, aber auch die Benutzerführung ist wesentlich. Sie wird im folgenden beschrieben.

4.4.1 HTML-Design

Die erste Seite der jETI Webseite wurde als `index.jsp` angelegt. Nach den ersten Vorüberlegungen wurde entschieden, dass die Index Seite aus folgenden 4 Frames bestehen soll (Abbildung 4) :

- Frame 1: Logo und Titel Frame
- Frame 2: Menu Frame

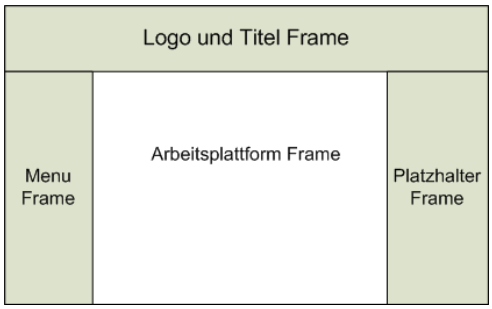

Abbildung 4: Index Seite

- Frame 3: Arbeitsplattform Frame
- Frame 4: Platzhalter Frame

Von der Index-Seite aus gelangt der Benutzer durch Anklicken der Menü-Buttons innerhalb des Menü-Frames: Impressum, Login, E-Mail und Links, auf die entsprechende Seite (Abbildung 5).

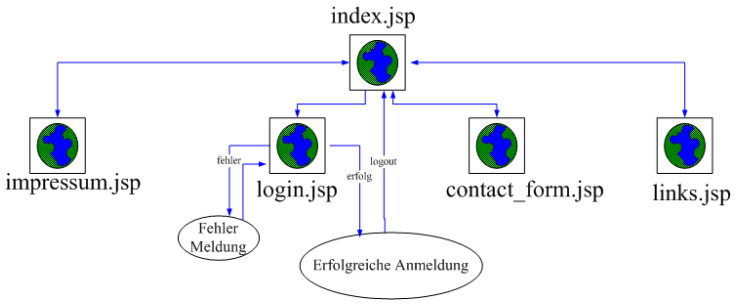

Abbildung 5: Index Seite Workflow

Auf der Links-Seite werden Informationen über die Firmen, die Ihre Tools in der jETI Plattform zur Verfügung stellen, angezeigt. Durch Kontakt kann der Benutzer über E-Mail direkten Kontakt mit dem Administrator aufnehmen, um sich z.B. für das jETI-System registrieren zu lassen. Falls der Benutzer schon registriert ist, kann er durch Login auf das System zugreifen. Falls die Informationen, die er eingegeben hat, falsch sind, werden drei verschiedene Fehlermeldungen angezeigt:

- Username und Passwort nicht korrekt
- Nicht angemeldet
- Leere Eingabe

Nach der Fehlermeldung wird der Benutzer wieder auf die Login-Seite zurückgeführt. Falls seine Angaben (Username und Password) korrekt sind, wird er auf die Webseiten weitergeleitet, für die er Zugangsberechtigungen hat. Es wurde von drei Hauptbenutzergruppen ausgegangen: Administrator, Tool Anbieter und normale Benutzer.

Falls sich ein Tool-Anbieter in das System eingeloggt hat, hat er die Möglichkeit den HTML-Konfigurator und Statistiken aufzurufen.

Normale Benutzer können über ein Menü Befehle wie **Graph Load**, **Graph New**, **Statistics New** aufrufen.

Durch diese Befehle wird die aufgerufene Seite innerhalb der Arbeitsplattform angezeigt. Wenn beim Laden eines Graphen ein Fehler auftritt, wird eine Fehlermeldung angezeigt. Falls beim Speichern ein schon verwendeter Name benutzt wird, wird ebenfalls eine Warnung angezeigt. Auch beim Löschen wird der Benutzer gefragt, ob er wirklich löschen möchte. Der Benutzer kann neue Graphen erstellen. Wenn er den Befehl dazu gibt, wird ein leerer Workspace innerhalb des Arbeitsplattform Frames angezeigt. Bei den ersten Vorüberlegungen waren wir uns noch nicht darüber klar, wie die zur Verfügung stehenden SIB's angezeigt werden sollten. Der Benutzer darf auch eigene Statistiken und Reports einsehen. Das Ergebnis wird wieder innerhalb des Arbeitsplattform Frames dargestellt.

Die Menüeinträge werden nicht aktiviert, wenn der Benutzer darauf keine Zugriffsrechte hat. Der Style des Menüs basiert auf der CSS Datei "navigation.css".

Die Untermenus sind wie folgt aufgebaut:

- Home:
  –Kein Untermenü. → Begrüßungsseite
- Login:
  –Kein Untermenü. → Login-Fenster wird innerhalb des Arbeitsplattform Frames angezeigt.
- Graph:
  –Laden, –Neuer Graph
- Statistiken:
  –Laden, –Löschen, –Anzeigen Erstellen, –Forced Filter
- Gruppenverwaltung:
  –Ändern Löschen, –Neue Gruppe
- Userverwaltung:
  –Ändern Löschen, –Neuer Benutzer
- Kontakt:
  –Kein Untermenü. → Das Formular wird im Arbeitsplattform Frame angezeigt.
- Links:
  –Kein Untermenü. → Links zu den Firmenseiten werden innerhalb des Arbeitsplattform Frames angezeigt.
- Impressum:
  –Kein Untermenü. Der Inhalt wird im Arbeitsplattform Frame angezeigt.

Als Darstellungsort für die Liste der SIB's und die Eigenschaften des ausgewählten SIB's ist der Arbeitsplattform Frame vorgesehen worden. Damit der Benutzer nicht vergisst, sich auszuloggen, wird ganz oben innerhalb des Logos und Titel Frame angezeigt, mit welchem Benutzername er eingeloggt ist. Das "Logout" soll an das Ausloggen erinnern.

4.4.2 Graph Erstellen

In diesem Projekt wurde vorausgesetzt, dass der Benutzer in seinem eigenen System keinerlei vorinstallierte JVM hat. Die größte Herausforderung war es die volle Funktionalität des bestehenden jETI-Systems eins zu eins in HTML-Form umzusetzen. Bei den ersten Überlegungen wurde der Workspace wie ein Schachbrett entworfen. In jeder Zelle darf der Benutzer entweder ein SIB oder eine Kante platzieren. Zuerst wurde der JSP Webseiten Flow erstellt (Abbildung 6).

4.5 Implementierungsphase

4.5.1 Administration

In diesem Abschnitt geht es darum, wie die Kontrolle des Systems durch Rechte Vorgabe in der Benutzer- und Gruppenverwaltung realisiert wird.

4.5.1.1 HTML-Seiten

Die Index Seite besteht aus 5 Frames (Abbildung 7):

- Logo und Titel Frame: `title.jsp`
- Menü Frame: `menu_closed.htm`
- Content Frame: `content.jsp`
- Edit-Cell Frame: `sibs.htm`
- Eigenschaften Frame: `properties.htm`

Die Seiten wurden zuerst als statische HTML-Seiten gebaut und danach mit JSP Funktionalität erweitert.

Die provisorischen statischen Menü-Seiten wurden durch eine `FeaturesMenu.jsp` Seite ersetzt. Auf der Index-Seite werden, innerhalb des Menu-Frames, jetzt die Links HOME, LOGIN, KONTAKT, LINKS, IMPRESSUM angezeigt (Abbildung 7).

Nach erfolgreichem LOGIN werden die Rechte des Benutzers kontrolliert. Davon abhängig werden die dynamischen Menüeinträge erzeugt und dargestellt. Diese Kontrolle wird zur Zeit über das Servlet `FeaturesServlet.java` durchgeführt. Die Kontrolle wird durch zwei Methode, `checkLoginname` und `pruefeLogin` innerhalb der Klasse `Usr.java` vorgenommen. Die Methode `checkLoginname` überprüft, ob überhaupt ein solcher Loginname in der Datenbank `usr` existiert. Falls er existiert, so vergleicht die `pruefeLogin` Methode das Passwort mit dem abgespeicherten Passwort und gibt ein entsprechendes boolean zurück.

Anfangs konnte der Inhalt des Content-Frames nicht gleichzeitig mit Menu-Frame aktualisiert werden. Dieses Problem wurde durch eine Aktualisierung des ganzen Framesets gelöst. Die Untermenüs werden unter den Haupteinträgen, z.B. GRAPH, STATISTIK angezeigt, wobei diese Haupeinträge keine aktiven Links sind, sondern nur zur optischen Ordnung dienen. In der Datenbank wurden z.B. der Benutzer **Ralf** mit Administrator Rechten eingefügt (das entsprechende Menü sieht man in Abbildung 8), sowie der Benutzer **Iris** mit Rechten für Statistik-Laden, Löschen und Neu eingefügt (das entsprechende Menü sieht man in Abbildung 9).

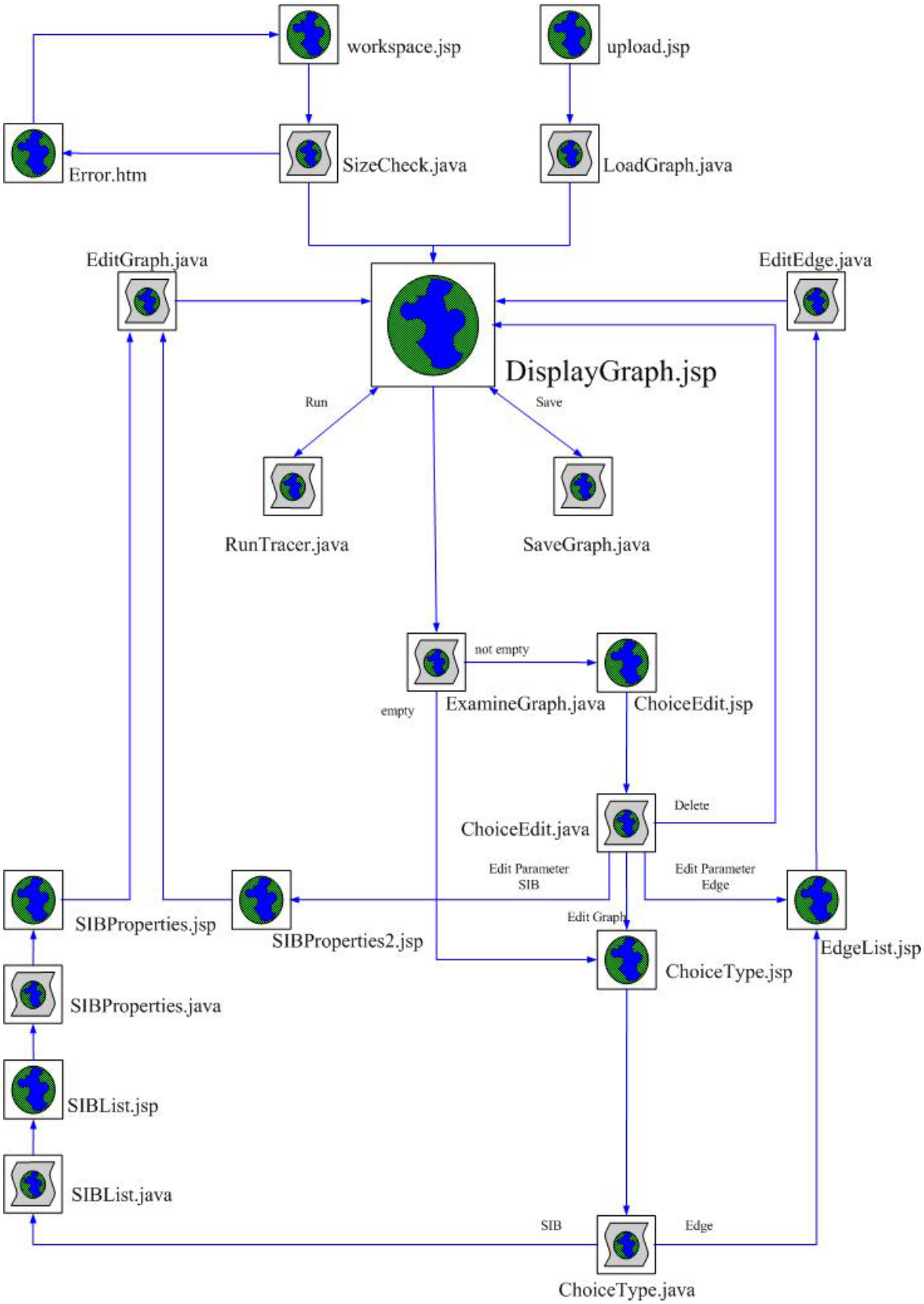

Abbildung 6: JSP Webseiten Flow bei der Erstellung eines Graphen

Abbildung 7: Die Index Seite des jETI HTML-Clients

4.5.1.2 Benutzerverwaltung

In der `user_new.htm`-Seite (Abbildung 10) werden die Daten eines neuen Benutzers eingegeben. Die Daten werden im Servlet `NewUserServlet.java` bearbeitet und in die Datenbank `usr` eingefügt.

Für die Bearbeitung der Daten in den Tabellen `usr`, `usrgroup`, und `features` gibt es die Klassen `Usr.java`, `Usrgroup.java` und `Feature.java`. Zu jedem Benutzer werden folgende Daten gespeichert:

Typ	Name	Bedeutung
Integer	PK	Primärschlüssel der Datenbank `usr`
String	password	Benutzerpasswort
String	enterprise	Firmenname wenn es sich um keinen normalen Benutzer handelt
String	firstname	Vorname des Benutzers, er darf nicht null sein
String	surname	Nachname des Benutzers, er darf nicht null sein
String	loginname	Loginname des Benutzers, er darf nicht null sein und muss eindeutig sein
String	email	Emailadresse des Benutzers
Date	lastlogin	Zeitpunkt des letzten Logins für einen Benutzer
Integer	logincount	Zähler für das erfolgreiche Einloggen
Integer	loginfailed	Zähler für das erfolglose Einloggen. Nach 5 Versuchen wird sein Konto gesperrt
boolean	accountlock	Zum Sperren des Kontos eines Benutzers, wenn sein loginfailed 5 erreicht
Usrgroup	ugroup	Eine Referenz auf Benutzer in der Klasse `Usrgroup.java`

Die Klasse `Usrgroup.java` hat die folgenden Attribute:

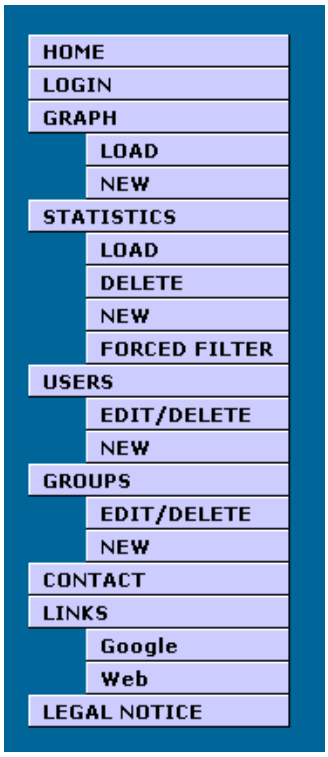

Abbildung 8: Das Menü für den Administrator

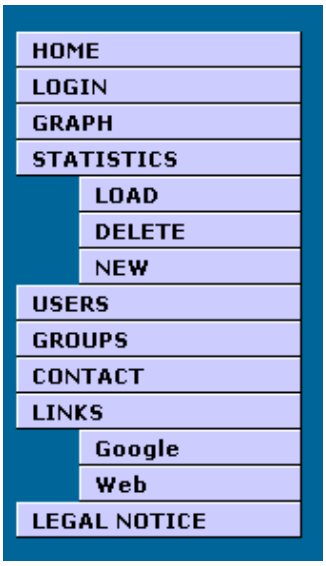

Abbildung 9: Das Menü für einen “normalen“ Benutzer

Typ	Name	Bedeutung
Integer	PK	Primärschlüssel der Datenbank `usrgroup`
String	name	Ist der Loginname für Benutzer und Gruppenname für Gruppen
boolean	isuser	Zum Unterscheiden von einem Benutzer und einer Gruppe
Set	users	Enthält die Gruppen, die einem Benutzer zugeordnet sind
Set	gruppen	Enthält die Gruppen, die einer Gruppe zugeordnet sind
Set	features	Enthält die Features, die einer Gruppe zugeordnet sind

Die Klasse `Feature.java` hat die Attribute:

Typ	Name	Bedeutung
Integer	PK	Primärschlüssel der Datenbank `Feature`
String	featurename	Name des Features
Set	gruppen	Enthält die Gruppen zu denen ein Feature gehört

Wenn man alle Daten eingibt und mindestens eine Gruppe auswählt und danach auf **Submit** klickt, wird die Anmeldung bestätigt.

Bei der Registrierung eines Benutzers wird überprüft, ob Vorname, Nachname, Loginname, Passwort, Email-Adresse und Gruppen eingegeben wurden. Wenn alles eingegeben worden ist, erfolgt die Anmeldung. Wenn einer der Eingabewerte fehlt, wird der Administrator aufgefordert

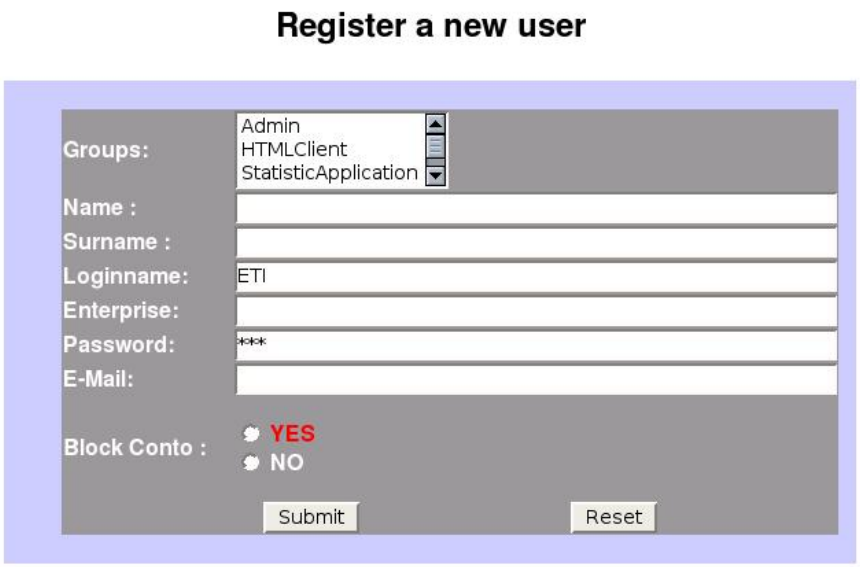

Abbildung 10: Einfügen eines neuen Benutzers in die User Datenbank

alle Daten einzugeben. Falls der Administrator einen schon vorhandenen Loginnamen eingibt, wird er gewarnt, dass zwei User mit dem selben Loginnamen nicht erlaubt sind.

Innerhalb der Benutzerverwaltung gibt es die Möglichkeit, dass der Administrator alle Benutzer aufgelistet bekommt. Abbildung 11 veranschaulicht diese Möglichkeit. Man bekommt diese Auflistung, wenn man auf dem Link **List of all users** innerhalb der `users_edit.htm`-Seite klickt (Abbildung 12). Damit wird das Servlet `AllUsersServlet.java` aufgerufen und die `AllUsers.jap`-Seite werden alle Benutzer und deren Daten aufgelistet.

Abbildung 11: Ausgabe der angemeldeten Benutzer

Wenn der Administrator bei der Userverwaltung Einträge von einem Benutzer ändern oder löschen will, muss er zuerst eine Suche durchführen (siehe Abbildung 12). Die Suche wird durch das Servlet `SearchServlet.java` durchgeführt und das Ergebnis wird innerhalb der `Search.jsp`- Seite angezeigt. Man kann nach Loginname, Vorname, Nachname oder Vorname und Nachname suchen. Die Abbildung 13 zeigt ein Beispiel, wenn man für den Nachnamen den Suchbegriff “Yu“ eingibt. Als Ergebnis bekommt man alle Benutzer deren Nachnamen mit “Yu“ beginnt.

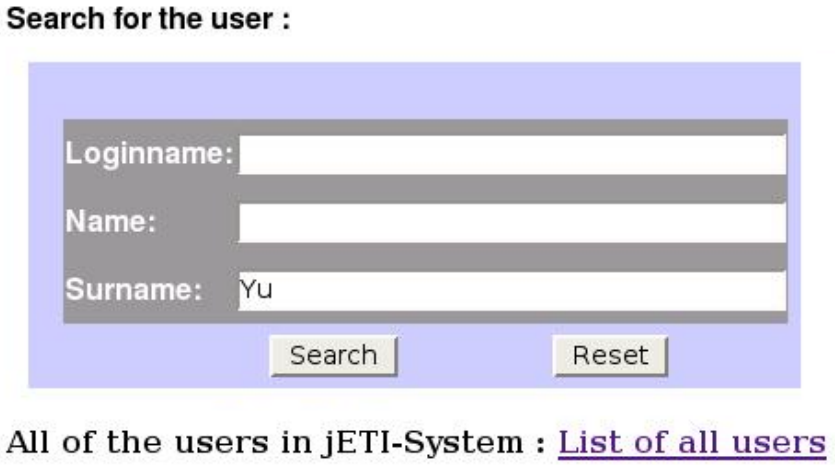

Abbildung 12: Userverwaltung -Ändern /Löschen

The list of all users :

Loginname	Vorname	Nachname	Email	Enterprise	Rechte	
Jiong	Jiong	Yu	jiong.yu@web.de		graphnew statisticssave conto usersedit statisticsdelete statisticsnew usersnew graphdelete statisticsload buycredits groupsedit buylicence groupsnew graphsave graphload	Delete/Edit

Abbildung 13: Ausgabe der Suche nach “Yu“ als Nachname

Falls es einen solchen Benutzer nicht gibt, wird eine entsprechende Meldung angezeigt.

Falls die Suche erfolgreich ist, werden neben den Benutzerdaten noch die Benutzerrechte angezeigt. Auf der Ergebnistabelle existiert für jeden Eintrag ein Link. Wenn man darauf klickt, wird das Servlet `UserDataServlet.java` aufgerufen und die Informationen über den ausgewählten Benutzer innerhalb eines Formulars angezeigt. Abbildung 14 illustriert das.

Abbildung 14: Bearbeiten oder Löschen eines Benutzers

Man kann die Daten des Benutzers bearbeiten und durch den **Update**-Button neue Informationen in die Datenbank `usr` speichern oder den Benutzer durch den **Delete**-Button, aus der DB entfernen.

4.5.1.3 Gruppenverwaltung

Auf der Seite `Groups_New.jsp` gibt es ein Formular, in das man den Gruppennamen eingeben soll. Man kann für die neue Gruppe aus der Auswahlliste mehrere Hauptgruppen bestimmen. Durch die Checkboxen werden die Rechte für die Gruppe ausgewählt 15). Die Daten werden im Servlet `GroupRegistrationServlet.java` bearbeitet und in die Datenbank `usrgroup` eingefügt.

Die zur Verfügung stehenden Rechte in der Tabelle `Feature` sind fest vordefiniert. Diese sind: graphload, graphsave, graphdelete, graphnew, statisticsload, statisticssave, statisticsdelete, statisticsnew, conto, buycredits, buylicence, useredit, usernew, groupsedit, groupsnew.

Ohne Hauptgruppen und Rechte wird die Gruppe nicht in die Datenbank eingetragen. Man muss auf jeden Fall mindestens eine Gruppe oder ein Recht auswählen.

Hier wird auch berücksichtigt, dass der Gruppenname eindeutig ist. Wenn man versucht, beispielsweise, die Gruppe2 einzutragen, und Gruppe2 schon in der Datenbank gespeichert ist, dann wird eine Warnung ausgegeben.

Um die Gruppen zu bearbeiten bzw. zu löschen, hat der Administrator erstmal die Möglichkeit alle Gruppen mit den zugehörigen Rechten in Form einer Tabelle zu sehen (Abbildung 16). Dies wird in der `AllGroups.jsp`-Seite angezeigt nach dem Aufruf des Servlets `AllGroupServlet.java`. In der ersten Spalte steht der Name der Gruppe. In der zweiten Spalte sieht man, ob die Gruppe einer anderen Gruppe zugeordnet ist. In der Dritten stehen die eigenen Rechte der Gruppe. Weil eine Gruppe mehreren Gruppen zugeordnet werden kann, ist in der vorletzten Spalte die gesamten Rechte der Gruppe zu sehen.

Die letzte Spalte enthält einen Link, von dem man zur `GroupDetails.jsp`-Seite gelangt. Man kann den Namen der Gruppe oder die Gruppen zu denen sie zugeordnet ist, durch Klicken

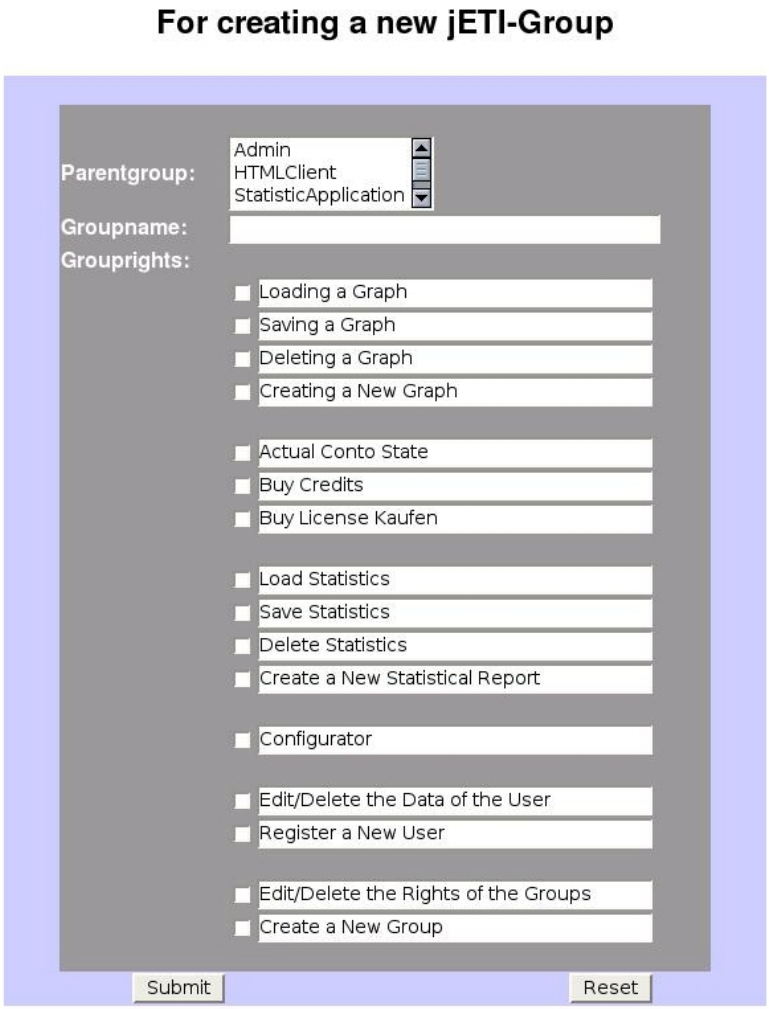

Abbildung 15: Eintragen einer Gruppe

auf den **Update**-Button ändern. Dies geschieht im Servlet `GroupUpdateServlet.java`. Es ist auch möglich durch Klicken auf den **Delete**-Button die Gruppe zu entfernen, indem das `GroupDeleteServlet.java`-Servlet aufgerufen wird. Allerdings wird bei Update beachtet, dass kein Kreis entstehen darf. Man will beispielsweise die Gruppe Gr.3 zu Gr.1 zuordnen. Es darf keine Kante von Gr.3 zu Gr.1 laufen, sonst entsteht ein Kreis. Dies geschieht durch eine Methode `sucheKreis(Gr.1, Gr.3)`, die als Parameter Gr.1 und Gr.3 hat. Sie benutzt die Tiefensuche im Fall eines gerichteten Graphen und nimmt als Startknoten die Gr.1. Sie sucht in der Menge der direkten Nachfolger von Gr.1 (Gruppen, zu denen Gr.1 zugeordnet ist) nach Gr.3. Ist Gr.3 in dieser Menge, dann kann ein Kreis entstehen, sonst wird die Methode rekursiv aufgerufen und zwar mit den beiden Parameter: ein Nachfolger aus der gerade erwähnten Menge und Gr.3. Dies wird wiederholt, solange alle Knoten, die Von Gr.1 erreichbar sind, durchlaufen sind oder es wurde vorher einen Kreis gefunden.

The list of the all groups

Group	Parentgroup	Rights of the group	All of the rights	
Admin		graphdelete buylicence graphsave statisticssave groupsnew statisticsnew buycredits graphload statisticsdelete graphnew conto statisticsload usersedit usersnew groupsedit	graphdelete buylicence graphsave statisticssave groupsnew statisticsnew buycredits graphload statisticsdelete graphnew conto statisticsload usersedit usersnew groupsedit	Edit / Delete
HTMLClient		graphload graphdelete graphnew graphsave groupsnew usersnew usersedit groupsedit	graphload graphdelete graphnew graphsave groupsnew usersnew usersedit groupsedit	Edit / Delete
StatisticApplication		statisticsdelete statisticssave statisticsload statisticsnew	statisticsdelete statisticssave statisticsload statisticsnew	Edit / Delete
License		buylicence conto buycredits	buylicence conto buycredits	Edit / Delete

Abbildung 16: Liste aller Gruppen

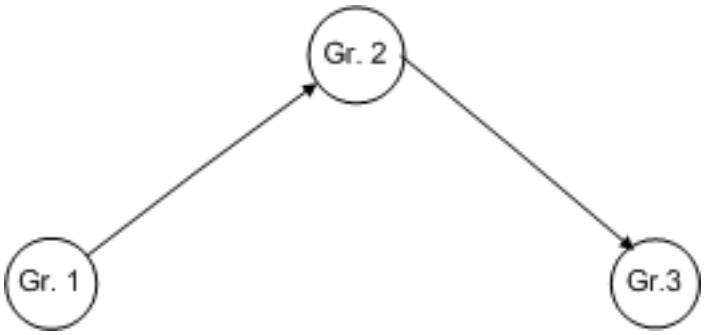

Abbildung 17: Graphische Zuordnung der Gruppen

4.5.2 Erstellen und Bearbeiten eines Graphen

Um SIB's auszuführen hat der Benutzer die Möglichkeit sie in Form eines Graphen zu erstellen, speichern, laden und bearbeiten.

4.5.2.1 Erstellen eines Graphen durch ein Servlet

Der Benutzer, der einen Graphen erstellen will, klickt nach dem Einloggen auf den **NEW**-Button unter dem Hauptmenüeintrag GRAPH. Danach öffnet sich `workspace.jsp`-Seite innerhalb des Content-Frames.

In `workspace.jsp` sollen zwei ganze Zahlen zwischen 1-50 eingegeben werden (Abbildung 18).

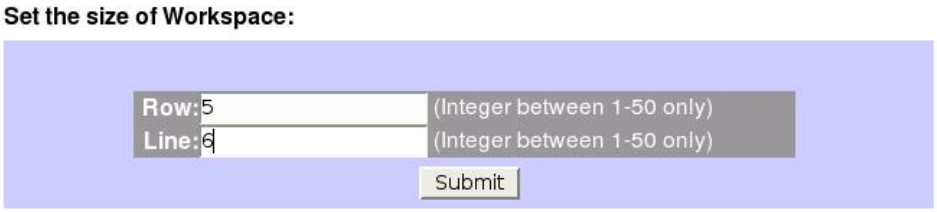

Abbildung 18: Die Grösse der Graphzeichenfläche eingeben

Die eingegebenen Werte werden in Servlet `SizeCheck.java` geprüft:

- Falls Werte keine ganzen Zahlen zwischen 1 und 50 sind, wird eine Seite `Error.htm` innnerhalb des Content-Frames ausgegeben und der Benutzer zur `workspace.jsp`-Seite zurückgeführt.

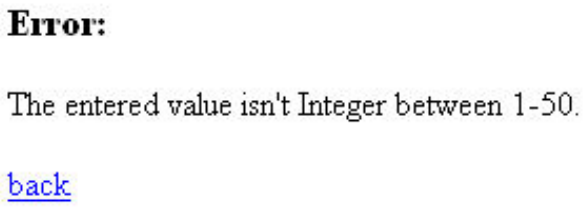

Abbildung 19: Error.htm

- Falls die Werte ganze Zahlen zwischen 1 und 50 sind, werden die eingegebenen Werte im Servlet `SizeCheck.java` aus dem Formular in `Workspace.jsp` gelesen um das Array der SIB-Objekte und das Array der Kantennamen zu initialisieren mit diesen Werten. Diese beiden Arrays werden als Session Attributte gespeichert und nach `DisplayGraph.jsp` weitergeleitet um die Grosse der Zeichenfläsche festzulegen.

Durch Klicken auf die Graphzeichenfläche wird zu `ExamineGraph.java`-Servlet gegangen, welche überprüft, ob die gerade vom Benutzer geklickte Zelle leer ist.

- Falls die Zelle leer ist, wird die `ChoiceType.jsp` Seite aufgerufen (Abbildung 20). Der Benutzer soll hier den Typ des Objekts (`SIB` oder `Edge`), den er einfügen will, auswählen. Der ausgewählte Typ wird zum Servlet `ChoiceType.java` weitergeleitet.

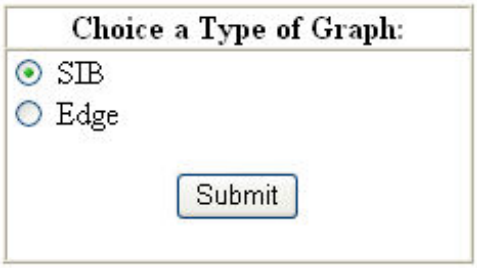

Abbildung 20: Entscheiden, ob eine Kante oder ein SIB eingefügt wird

– Falls der Typ `SIB` ausgewählt ist, wird eine SIB-Liste vom Servlet `SIBList.java` gemäß der Datei `tools.xml` erzeugt. Die SIB-Liste wird in der `SIBList.jsp`-Seite angezeigt. Durch die Auswahl eines SIB's wird der Name dieses SIB's nach `SIBProperties.java` weitergeleitet. Dort wird per Reflection[1] auf die Eigenschaften (Attribute der SIB-Klassen) der SIB's zugegriffen. Die Eigenschaften, die für den Benutzer wichtig sind, werden in einem Array gespeichert und an die `SIBProperties.jsp` weitergeleitet. Dort werden in einem Formular angezeigt (siehe Abbildung 21). Die vom Benutzer eingegebenen Werte sind Eigenschaften eines SIB-

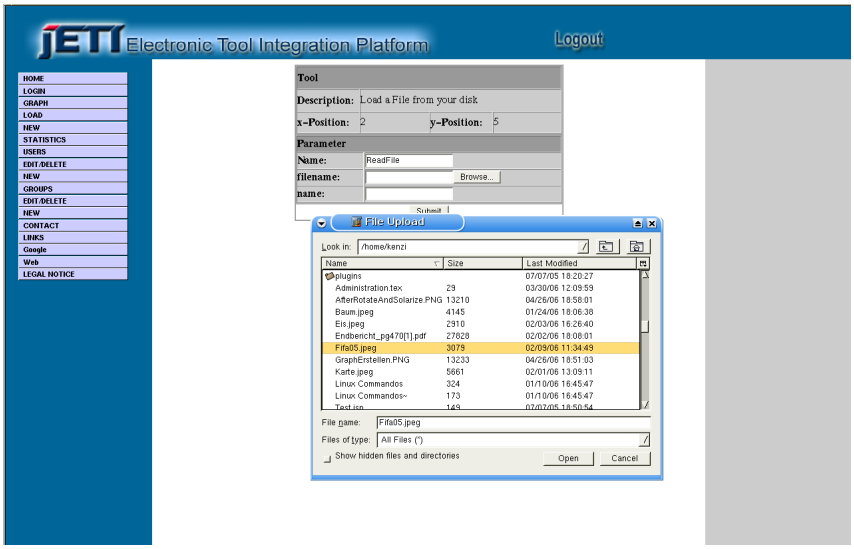

Abbildung 21: Die Eigenschaften des ausgewählten SIB's

Objekt in `EditGraph.java` Servlet. Diese Eigenschaften werden in einem Array gespeichert und ein Bild dieses SIB's wird auf der Graphzeichenfläche gezeigt.

– Falls der Typ `Edge` ausgewählt ist, wird `EdgeList.jsp`-Seite aufgerufen und die 12 Kantentypen werden gelistet (Abbildung 22).

  Durch ein Klick auf einer Kante werden die Positionen und das ausgewählte Bild der Kante in Sesssion-Attribute unter den Namen `xPos`, `yPos` und `edgename` gespeichert. Die Attributen werden zum `EditEdge.java`-Servlet weitergeleitet und innerhalb des Servlets gespeichert. Das Bild der ausgewählten Kante wird auf der Graphzeichenfläche angezeigt.

[1] Die Reflection ermöglicht den Zugriff auf Eigenschaften und Methoden der Klassen zur Laufzeit, deren Existenz zur Zeit der Programmerstellung nicht bekannt war.

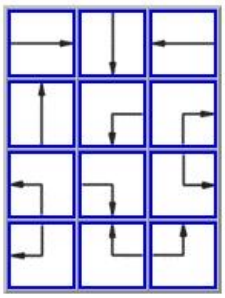

Abbildung 22: Die Liste der Kanten

- Falls die Zelle nicht leer ist, d.h., der Benutzer will den Inhalt der Zelle ändern, dann wird die `ChoiceEdit.jsp`-Seite aufgerufen (Abbildung 23).

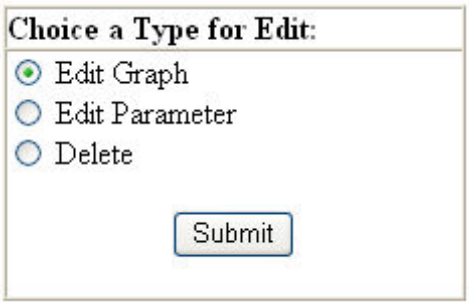

Abbildung 23: Die belegte Zelle ändern oder löschen

Der Benutzer kann hier auswählen, was er ändern will.

  - Falls **Edit Graph** ausgewählt wird, wird die `ChoiceType.jsp`-Seite aufgerufen (Abbildung 20).
  - Falls **Edit Parameter** ausgewählt wird, wird die `SIBProperties2.jsp (oder die EdgeList.jsp)`-Seite aufgerufen, die eingegebenen Parameter des SIB's werden angezeigt und der Benutzer kann die Parameter weiter bearbeiten(oder das Bild von Kante ändern).
  - Falls **Delete** ausgewählt wird, wird das SIB oder die Kante von Graphzeichenfläche gelöscht.

4.5.2.2 Graph Speichern

Falls der Benutzer seinen Graphen speichern will, hat er die Möglichkeit den Button **Save** unter dem Workspace anzuklicken. Das Speichern des Graphen geschieht im `SaveGraph.java` Servlet. Hier wird das XML Format des Graphen erstellt und danach wird die `do_download.jsp`-Seite aufgerufen. Weil das Speichern des Graphen auf der Client-Seite durch einen Download durchgeführt wird, wird ein Fenster zum Speichern geöffnet. Dort kann der Benutzer in seinem eigenen System, mit einem selbst bestimmten Namen seinen Graphen in einem XML Format speichern. Für diesen Download wurde die Bibliothek `cos.jar` von Oreilly benutzt. Die Struktur der XML-Datei, sieht z.B. wie in der (Abbildung 24) aus:

Die Variablen `Columns` und `Rows` zeigen die Größe des Workspaces. Jede Zelle ist durch das Element `Cell` dargestellt und die Attributen `Typ`, `xPosition`, `yPosition` und `bild`.

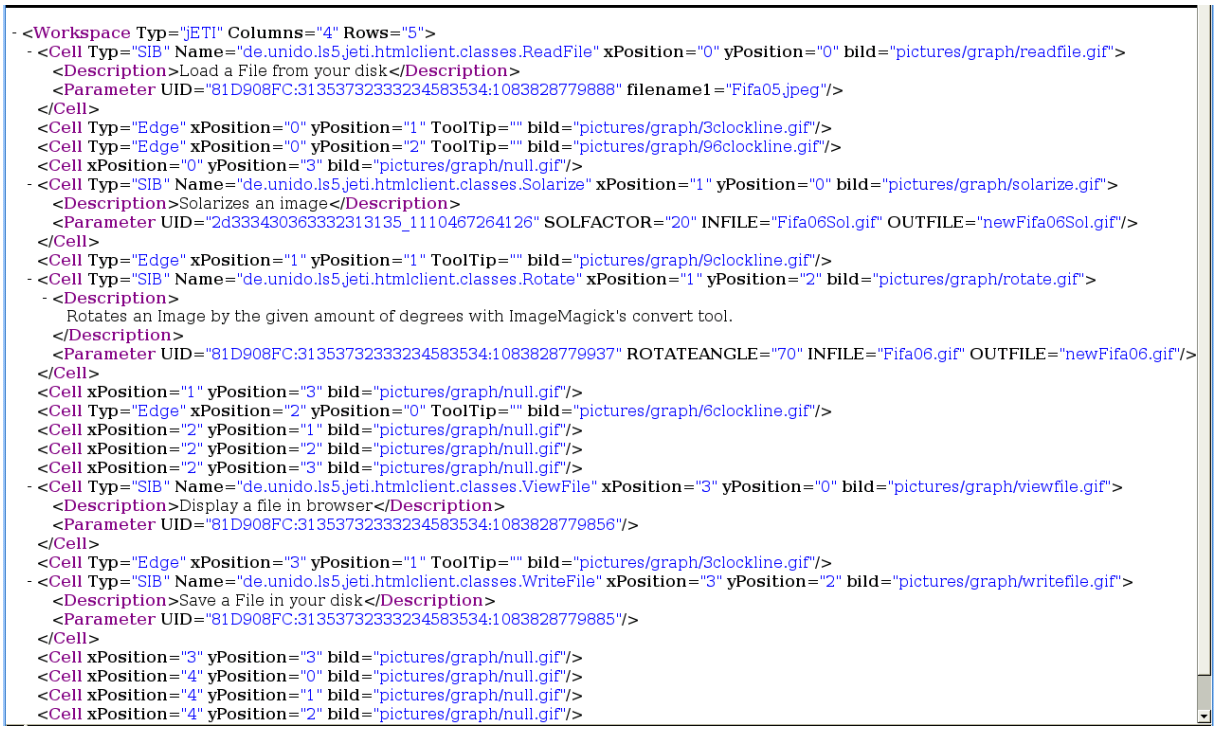

Abbildung 24: XML-Struktur des gespeicherten Graphen

- Typ: zum Unterscheiden der SIB's (Knoten) von Kanten. Der Wert "SIB" steht für SIB's und "Edge" für Kanten.
- xPosition: stellt die Position der X-Achse der Zelle
- yPosition: stellt die Position der Y-Achse der zelle
- bild: Für den Pfad der Bilder. `null.gif` bedeutet, dass die Zelle leer ist.

Falls die Zelle ein SIB enthält, dann sind zusätzlich zu den oben erwähnten Attributen noch die folgenden Attribute und Kindelemente:

- Name: stellt den Name der SIB-Klasse dar
- Description: kurze Beschreibung des SIB's
- Parameter: Kindelement und hat auf jeden Fall das Attribute UID[2] und die speziellen Eigenschaften wie ROTATEANGLE, INFILE und OUTFILE, deren Werte vom Benutzer eingegeben worden sind.

4.5.2.3 Graph Laden

Wenn der Benutzer seine schon in seinem System gespeicherten Graphen weiter bearbeiten will, hat er die Möglichkeit, sie über den Button **LOAD** unter dem Hauptmenüeintrag **GRAPH**, wieder auf die Server-Seite zu übertragen.

[2]UID: Ist eine weltweit eindeutige Identifikationsnummer, die die SIB's eindeutig bestimmt

Nach dem Anklicken des Buttons **Upload** wird die `upload.jsp`-Seite innerhalb des Content-Frames angezeigt (Abbildung 25). Von der `upload.jsp`-Seite aus kann der Benutzer jede beliebige Datei aus seinem System wählen. Dadurch wird das `LoadGraph.java` Servlet aufgerufen. Innerhalb dieses Servlets wird überprüft, ob die ausgewählte Datei der vereinbarten Struktur entspricht (siehe Abbildung 24). Die XML-Dateien, die als Typ-jETI bezeichnet sind, sind erlaubte Dateien.

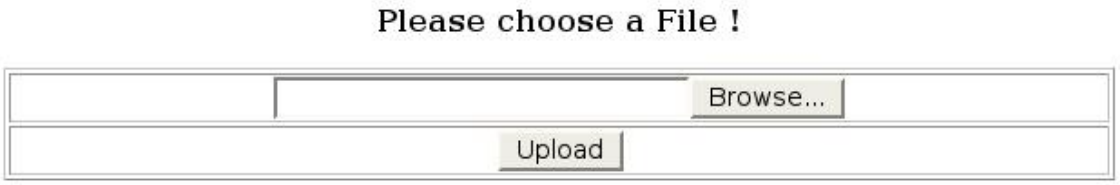

Abbildung 25: Gespeicherten Graph aus einer XML Datei lesen

Für dieses Upload an der Server-Seite wurde als Package cos.jar benutzt. Es ermöglicht das Laden von Dateien aus dem lokalen Dateisystem des Benutzers. Allerdings sollte im Formular das Attribut `enctype=multipart/form-data` vorhanden sein.

4.5.3 ETI-Anbindung

Nach dem Erstellen bzw. Laden des Graphen kann der Benutzer ihn ausführen. Die Ausführung wird schrittweise nach dem Anklicken des **Run**-Buttons durchgeführt. Es wird das Servlet `Tracer.java` aufgerufen. Dieses Servlet ruft zwei Methoden namens `findStartNode` und `searchSuccessor` auf. Die erste ist zuständig für die Suche nach dem Start-SIB[3] und die zweite hat als Parameter zwei Arrays und ein SIB-Objekt, für das der Nachfolger gesucht wird. Das erste Array enthält die Knoten (SIB's) des Graphen und das zweite enthält die Namen der Kanten. Diese Methoden sind in der Klasse `ForTracer.java` implementiert. Wie es gerade erwähnt wurde, wird erstens das Start-SIB gesucht und danach mit `searchSuccessor` der Nachfolger festgelegt. Im Fall, dass das Start-SIB ein **ReadFile** ist, wird die zu bearbeitende Datei aus einem lokalen Verzeichnis des Benutzers ausgewählt. Diese Datei wird zuerst mit Hilfe der beiden Methoden `addForStore` und `Store` an den **ETI**-Server gesendet.

Die erste fügt die Datei zu einer Liste hinzu und beim nächsten Aufruf von `Store` wird sie mit einem Namen, der vom Benutzer beim Auswählen des SIB's in einem Formular im Feld `INFILE` eingegeben wurde, an den Server geschickt. Dann wird das SIB (Nachfolger von **ReadFile**), indem die Methode `exec` aufgerufen wird, ausgeführt. Diese ruft eine Kommandozeile, die vom Betriebssystem ausgeführt wird, oder eine Java Klasse auf.

Bei der Ausführung spielt die Datei `tools.xml` eine wichtige Rolle. Sie wird vom Tool-Konfigurator (dieser wird in Abschnitt 7 ausführlich beschrieben) erstellt und daraus werden die nötigen Parameter für die Ausführung wie z.B Toolname, die auszuführende Methode, ihre Klasse, usw. ausgelesen.

Danach wird nach der Ausgabedatei gesucht, aus dem Server abgeholt und dem Benutzer zur Verfügung gestellt. Die Suche geschieht in drei Schritten:

[3] Start-SIB ist einfach der Startknoten im linear gerichteten Graphen

- Als erstens wird der Name, der zu suchenden Datei, als Parameter an die Methode `addForRetrieval` übergeben, die sie in der Liste der abzuholenden Dateien hinzufügt.
- Dann wird die Methode `retrieve` aufgerufen, um die Ausgabedatei vom Server abzuholen und in einer Liste zu speichern.
- Als letztes wird die Ausgabedatei mit Hilfe der Methode `getRetrievedFiles` gelesen.

Um dem Benutzer zu ermöglichen, die Ausführung seines Graphen zu verfolgen, wird ein Fenster angezeigt und darin steht, von welchem SIB die Datei momentan bearbeitet wird.

Die Abbildung 26 illustriert ein kleines Beispiel mit den folgenden SIB's: **ReadFile**, **Rotate**, **Solarize**, **ViewFile** und **WriteFile**. Als Start-SIB in diesem Graphen ist **ReadFile**. Dadurch

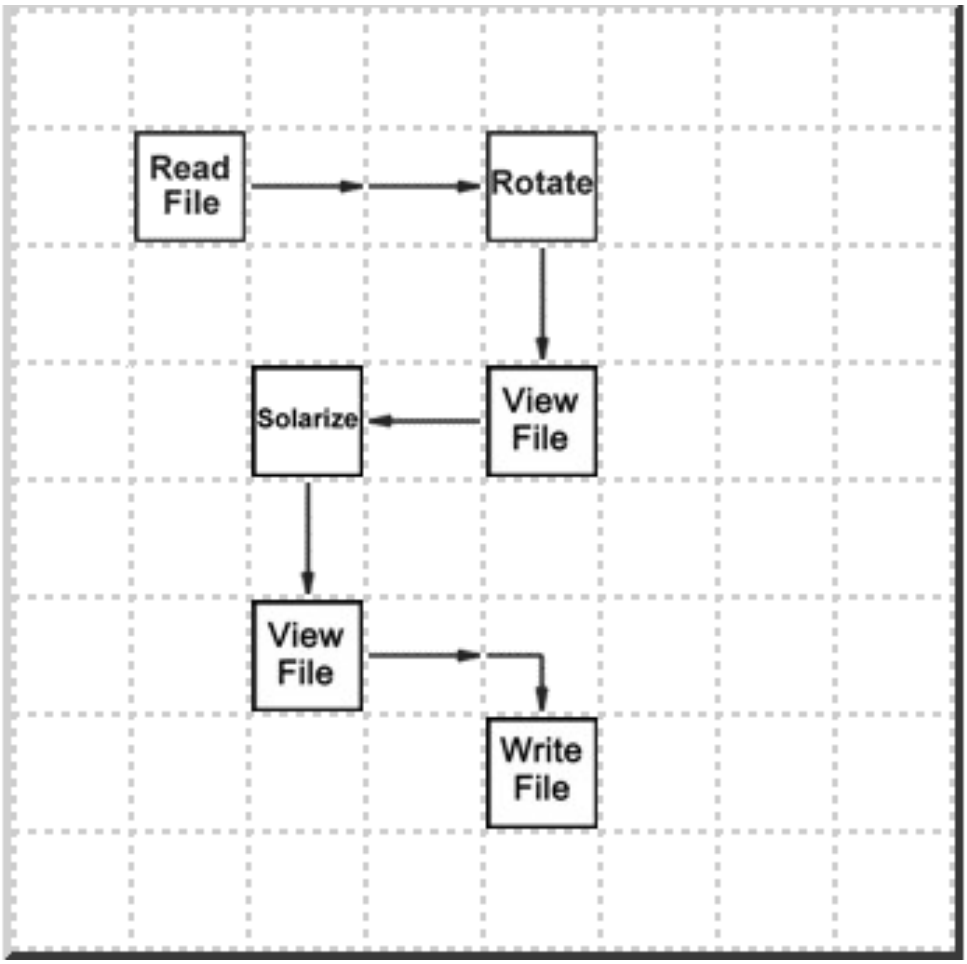

Abbildung 26: Ausführung von SIB's

hat der Benutzer die Möglichkeit eine Bilddatei (das Originalbild sieht man in Abbildung 27) auszuwählen. Der erste Nachfolger ist ein **Rotate**-SIB, es rotiert das Bild um 90 Grad (ROTATEANGLE = 90) und gibt die rotierte Datei weiter an **ViewFile**. Dieses zeigt das Bild im Browser an, und gibt es an **Solarize** weiter. **Solarize** solarisiert das Bild (SOLFACTOR = 20) und gibt es wiederum an **ViewFile** weiter. Das Ergebnis ist in der Abbildung 28 zu sehen.

Das **WriteFileSIB** dient zum Speichern der Ausgabedatei in einem lokalen Verzeichnis des Benutzers. Es öffnet sich ein Fenster, in dem der Benutzer den Dateinamen und den Speicherort eingeben kann.

Abbildung 27: Das Originalbild

Abbildung 28: Nach der Ausführung von `Rotate` und `Solarize`

Falls der Benutzer irrtümlich seine Eingabeparameter nicht wie erwartet eingegeben hat, beispielsweise wenn er statt eine Zahl ein Zeichen oder statt ein Bild- ein Textdatei eingegeben hat, wird eine Fehlermeldung ausgegeben.

4.5.4 Testen der Servlets

Das Testen wurde clientseitig durchgeführt. EJB, Servlets oder JSP's werden im allgemein serverseitig aufgerufen. Falls nur JUnit beim Testen angewendet wird, sind die Umgebungen zwischen Client und Server unterschiedlich, es führt dazu, dass unterschiedliche Testergebnisse eintreten. Cactus ist ein Test-Framework, welches von der Apache Software Foundation im Jakarta-Projekt entwickelt wurde. Es baut auf dem JUnit-Test-Framework auf, wird jedoch für serverseitige Tests benutzt, z.B. zum Testen von Servlets. Cactus erweitert JUnit und liefert alles was zum Testen von Servlets, JSP-Seiten und Filtern gebraucht wird:

- Cactus läuft selbst auf dem Server (dort wo das Servlet auch läuft).
- Cactus bildet alle wichtigen APIs nach, so dass alle Parameter, die ein Servlet normalerweise über die Konfiguration oder den Request erreichen, vom TestCase gesteuert werden.
- Cactus kann das Testergebnis im Browser oder direkt in Eclipse anzeigen

Installation
Nach dem Download des Cactus-Archivs muss man zunächst dafür sorgen, dass alle Jar-Files, die auf der Server-Seite gebraucht werden im jeweiligen Container gefunden werden. Mit Tomcat geht das am einfachsten, so dass man die JAR's `aspectjrt.jar`, `cactus.jar`, `commons-logging.jar`, `commons-httpclient.jar` und `junit.jar`(jeweils mit Versionsnummer) in den `.../common/lib/`-Pfad von Tomcat kopiert. Alternativ kann man all diese JAR's auch jeweils ins Lib-Verzeichnis der Web-Applikation kopieren. Dann steht die Testmöglichkeit mit Cactus eben nicht allen Web-Applikationen automatisch zur Verfügung, sondern immer nur der einen, die die JAR's auch mitbringt.

Konfiguration
Anschließend wird ein spezielles Servlet, der "ServletTestRunner", in der globalen `web.xml` konfiguriert. Die globale web.xml von Tomcat findet sich in `.../conf/web.xml`. Zwei Abschnitte sind einzufügen(siehe Anhang "web.xml").

Ein erster Test
Anhang "Ein erster Test" ist ein einfacher Servlet zum Testen mit Cactus.

Die Funktionalität des Servlets “ExamizeGraph.java“ ist schon in Quellcode beschrieben. Folgend ist die Testklasse dafür veranschaulicht. Die Beschreibung ist auch in Quellcode enthalten(siehe Anhang “Testklasse“).

Der letzter Schritt ist der Aufruf des Containers.Dieser ist im Browser wie folgend einzugeben:

http://localhost:8080/ServletTestRunner?suite=de.unido.ls5.jeti.htmlclient.servlet.graph.TestExamizeGraph

Das Testergebnis wird als XML-Datei angezeigt:

```
<?xml version="1.0" encoding="UTF-8" ?>
-<testsuites>
    -<testsuite name="de.unido.ls5.jeti.htmlclient.servlet.graph.TestExamizeGraph" tests="1"
            failures="0" errors="0" time="0.016">
        <testcase name="testservice" time 0.016>
    </testsuite>
</testsuites>
```

Dabei bedeutet failures=“0“ errors=“0“, mit den gegebenen Eingaben hat das Servlet die richtige Antwort zurückgegeben. Beim Testen ist es erlaubt (teilweise sogar notwendig), falsche Eingaben zu simulieren, um zu testen, ob das Servlet die Exception generiert.

5 HTML Statistik

von Kenkmann, Markus; Braß, Norman; Guan, Zhongyue

5.1 Modellierungsphase

Statistiken sind in der heutigen Zeit für alle Dienstleister sehr wichtig, da Sie dadurch viele Daten über ihre Dienstleitungen bekommen. Im bisherigen ETI System gab es keine Statistiken. Um zu erfahren wie oft bzw. ob die Tools von den Usern überhaupt benutzt werden, musste man mühsam Logs lesen. Dies war für die Toolprovider nicht zumutbar. Außerdem gab es dabei das Problem, das alle Toolprovider die selben Logs lesen mussten und somit auch die Beliebtheit von den Tools der anderen Toolprovidern erfahren hätten. Diese Statistiken sollen nun für die Toolprovider aufbereitet werden. Toolprovider sollen in der Statistik Web Applikation umfangreiche Möglichkeiten bekommen, die gesammelten Statistiken zu ihren Produkten optisch anzuzeigen.

In diesem Abschnitt findet man die Vorüberlegungen und generellen Festlegungen zur Statistik Web Applikation, sowie den darin enthaltenen Teilstrukturen. Dabei werden die Designvorschläge einmal kurz erklärt. Die Umsetzung der Designvorschläge wird in Abschnitt 5.3 behandelt.

5.1.1 Reports

Die generelle Vorgehensweise der Gruppe zum Bearbeiten der Teilaufgaben war zuerst das Analysieren der anfallenden Arbeiten. Dabei wurden die entsprechenden Ideen immer mitgezeichnet. Diese wurden dann als Workflows auf dem Computer umgesetzt. Danach wurde mit Hilfe des Workflows eine Probeversion der Statistik-Web-Applikation in statischem HTML Code (der nicht die volle Funktionalität hatte) umgesetzt. Auf Basis des HTML Codes wurde dann ein entsprechender Statistik JSP-Webseiten-Flow erstellt. Auf diesem waren erstmals auch die

benötigten Servlets eingezeichnet. Danach begann die Implementierungsphase mit dem Umsetzen dieses JSP Flows in die JSP Seiten und der Erstellung der Servlets. Während der Implementierung wurde der JSP Flow dann mehrmals umgeändert, nachdem Probleme beim eigentlichen Entwurf uns zu Änderungen zwangen. Dies war meistens der Fall, wenn wir merkten das die Methoden in den Servlets für mehrere Aufgaben gebraucht werden und wir diese dann aus den urspünglichen Servlets Klassen ausgegliedert haben.

Um die Teilaufgaben **Reports definieren und speichern** und **Reports aufrufen und speichern** zu bewältigen, wurde innerhalb der Gruppe zuerst besprochen, welche Statistiken die Statistik Web Applikation überhaupt anbieten soll. Bei diesen Vorbesprechungen wurde festgestellt, dass Statistiken immer aus 2 der 3 Werten von User, Zeit und Tools bestehen.

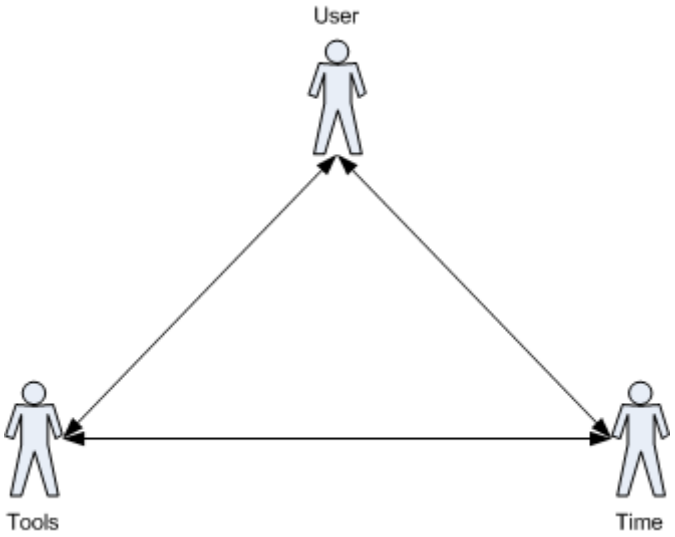

Abbildung 29: Aufbau der Statistiken

In Abbildung 29 kann man genau diese Wechselbeziehung der 3 Werte in einem Diagramm sehen.

Dann wurde überlegt, welche Statistiken in der Web Applikation angeboten werden. Die Statistik Web Applikation wird folgende Statistiken anbieten (in der 2ten Ebene steht immer welche Daten man für diese Statistik braucht):

- Wie lange bleibt ein User in unserem System?
  - User, Loginzeit, Logoutzeit, Session-ID
- Wie oft wird ein bestimmtes Tool ausgeführt?
  - Tool, Anzahl der Toolaufrufe
- Wie lange wird ein Tool ausgeführt?
  - Tool, Startzeitpunkt, Endzeitpunkt, (Aggregation der Zeiten)
- Welche Tools werden ausgeführt?
  - Auswahl aus Tools, Anzahl der Toolaufrufe oder Toolausführungszeit

- Wann werden die Tools ausgeführt?
  - Tools, (Aggregation über den Zeitstempel)
- Wie oft wurde ein Tool ausgeführt bis die Lizenz erworben wurde?
  - User, Anzahl der Toolaufrufe ohne Lizenz

Nachdem klar war, welche Statistiken angezeigt werden sollen, wurde entschieden mit welchen Diagrammtypen diese Statistiken am besten dargestellt werden können. Dabei hat sich die Gruppe 3 auf folgende Typen geeinigt:

- Balkendiagramm 2D
- Balkendiagramm 3D
- Flächendiagramm
- Kreisdiagramm 2D
- Kreisdiagramm 3D
- Liniendiagramm 2D
- Liniendiagramm 3D

Diese Diagramme schienen uns die geeignetsten Diagramme zu sein, um die Statistiken vernünftig grafisch aufzubereiten.

Die Statistik Web Applikation soll dem Benutzer des Systems das größtmögliche Maß an Freiheit geben und trotzdem noch handelbar bleiben. Der Benutzer soll sich seine Diagramme komplett selbst aus vorher ausgewählten Parametern erstellen können. Denn nur der Benutzer selbst, weiß wirklich wie sein Diagramm aussehen muss, damit er alle geforderten Informationen, die er benötigt, auch bekommt. Des Weiteren soll er auch das Layout der Diagramme komplett selbst gestalten können.

Die Gruppe hat daraufhin entschieden, dass es das Beste ist den Benutzer durch die verschiedenen Möglichkeiten zu führen. Es soll also ein **Wizard** (auch Assistent genannt, ist eine Oberfläche, die für eine ergonomische Dateneingabe sorgt) für die Statistiken implementiert werden. Dieser **Wizard** leitet den Benutzer von einer Seite zur nächsten und hilft ihm dabei sich eigene Diagramme zu erstellen. Mit Hilfe des **Wizards** sollen Fehleingaben und Probleme vermieden werden und der Benutzer einfach zu einem eigenen Diagramm geleitet werden. Falls der Benutzer Probleme hat, so bietet er dem Benutzer Hilfen, um die Probleme zu lösen.

Bei der weiteren Planung des **Wizards** wurde festgestellt, dass bei den ausgewählten Diagrammen, meistens nur minimale Änderungen nötig sind. Das beruht darauf, das sie auf 2 Grundmustern basieren. Deswegen wurden auch nur 2 verschiedene Arten des **Wizards** für alle Diagramme erstellt. Der erste **Wizard** ist für alle Diagramme die auf 2 Achsen aufgebaut sind. Dies betrifft: Balkendiagramm 2D, 3D, Flächendiagramm, Liniendiagramm 2D und 3D. Der zweite **Wizard** ist ausschließlich für Diagramme die auf einem Kreis aufgebaut sind, dies betrifft also nur Kreisdiagramm 2D und 3D.

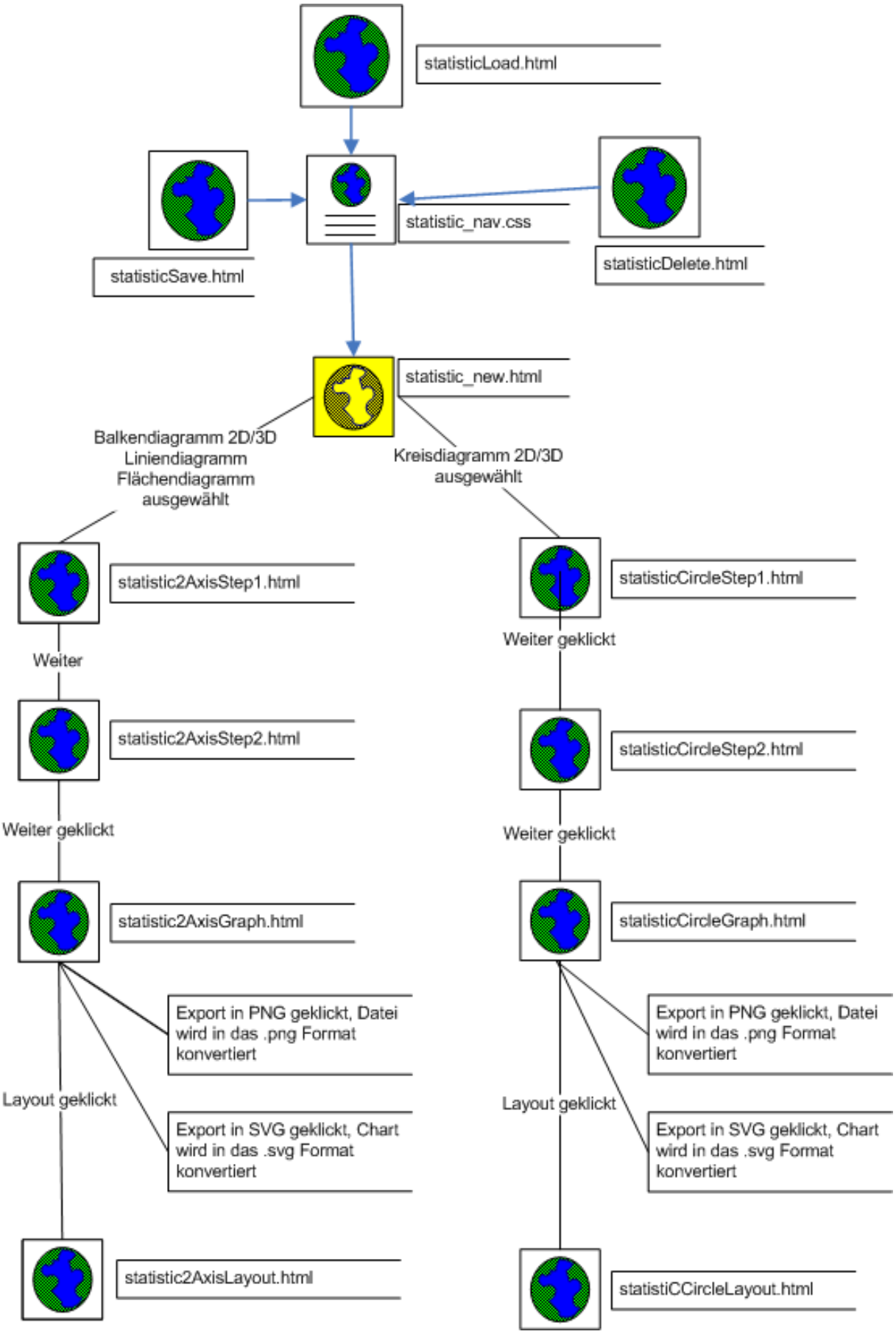

Abbildung 30: Der Statistik Webseiten Flow

In Abbildung 30 kann man den kompletten Flow der Statistik-Webseiten-Applikation sehen. Die Startseite des **Wizards** für die Statistik Web Applikation soll `statistic_new.html` sein. Auf dieser Seite bekommt der Benutzer die Möglichkeit, per DropDown-Menü, einen passenden Diagrammtyp für seine benötigten Statistiken auszuwählen. Außerdem bekommt der Benutzer auf der Seite nach einem Druck auf den **Help** Button die Hilfe zu der Seite angezeigt.

Nachdem der Benutzer sich nun für ein Diagramm entschieden hat, geht es im Wizard, je nach Auswahl des Benutzers, auf die nächste Seite. Sollte der Benutzer sich für eins der folgenden Diagramme (Balkendiagramm 2D, Balkendiagramm 3D, Liniendiagramm, Flächendiagramm) entschieden haben, ist der zweite Schritt des **Wizards** die Seite `statistic2AxisStep1.html`. Sollte der Benutzer sich für ein Kreisdiagramm (egal ob 2D oder 3D) entschieden haben, so ist der zweite Schritt des Wizards die Seite `statisticCircleStep1.html`.

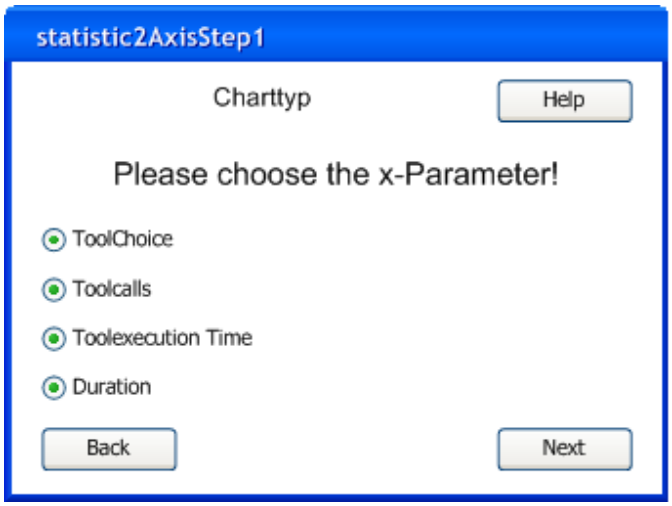

Abbildung 31: Designvorschlag für die statistic2AxisStep1.jsp Seite

Beim ersten Schritt des 2Achsen-Teils des Wizards soll der Benutzer die x-Achse seines Diagramms auswählen (siehe Abbildung 31). Dazu muss er zuerst entscheiden, ob er nur Daten seines Accounts oder von allen Accounts haben möchte. Diese Abfrage soll mit CheckBoxen realisiert werden. Unten auf der Seite soll der Benutzer dann den eigentlichen Parameter für die x-Achse wählen. Die Abfrage soll durch eine RadioButtonGroup abgefragt werden. Im oberen Teil der Seite soll immer eine Art Status des **Wizards** erscheinen. Dort soll angezeigt werden, welche Parameter man bereits ausgewählt hat. Im unteren Teil werden die Buttons für die Navigation durch den **Wizard** angezeigt. Der Benutzer soll hier die Möglichkeit bekommen wieder zurück zur Vorgänger-Seite zu kommen (durch klicken auf **Back**). Er soll zu einer Hilfe-Seite kommen können (durch den **Help** Button) und natürlich soll er, wenn er den benötigten Parameter ausgewählt hat, mit einem Klick auf **Next** zur nächsten Seite des Wizards kommen.

Nach einem Klick auf **Next** soll der Benutzer die zweite Seite des 2Achsen-Teils des Wizards angezeigt bekommen (`statistic2AxisStep2.html` siehe auch Abbildung 32). Auf dieser Seite sind im oberen Bereich, zur besseren Übersicht, wieder die bisher ausgewählten Parameter aufgelistet. Darunter soll wie schon beim ersten Schritt des Wizards wieder mit Hilfe einer RadioButtonGroup der Parameter für die y-Achse ausgewählt werden können. Dem Benutzer werden nur noch die Parameter in der RadioButtonGroup angezeigt, die sinnvoll zu seinem ersten Parameter sind. Im unteren Teil der Seite soll es dann wieder so aussehen wie auf der Seite zuvor. Es wird wieder je einen **Help**, **Back** und **Next** Button mit gleichen Funktionen geben, wie vorher.

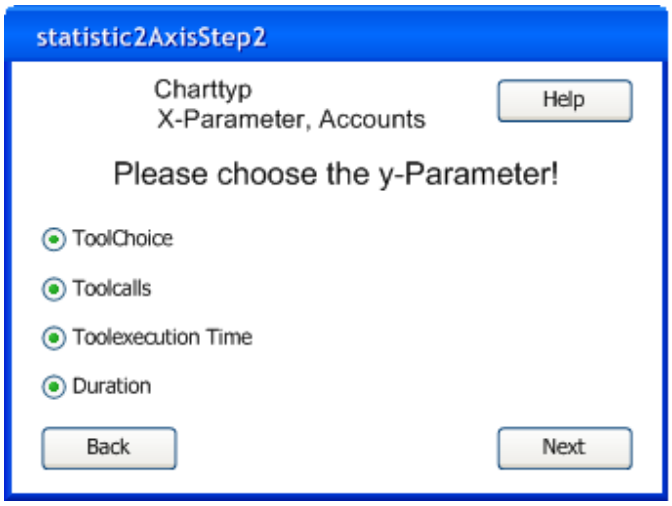

Abbildung 32: Designvorschlag für die statistic2AxisStep2 Seite

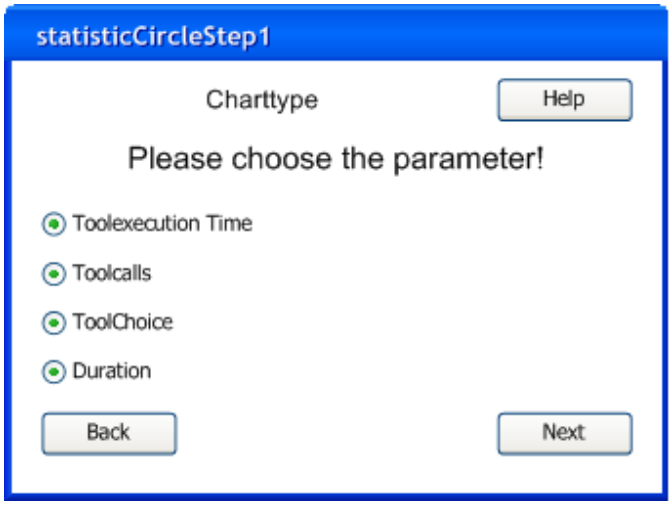

Abbildung 33: Designvorschlag für die statisticCircleStep1.jsp Seite

Im ersten Schritt des **Wizards**, für die Diagramme die auf einen Kreis basieren, wählt der Benutzer einen Parameter für sein Diagramm aus (siehe Abbildung 33). Zuerst muss der Benutzer wie vorher auch auswählen, ob er nur Daten seines Accounts oder von allen Accounts haben möchte. Die Abfrage wird wieder mit CheckBoxen realisiert werden. Weiter unten soll der Benutzer dann den eigentlichen Parameter wählen. Mit einer RadioButtonGroup wird die Abfrage realisiert. Im oberen Teil der Seite wird wieder der aktuelle Status des Wizards angezeigt. Im unteren Teil werden, wie beim 2Achsen Teil, die Buttons für die Navigation durch den Wizard angezeigt.

Sollte der Benutzer für einen seiner Parameter die Toolauswahl gewählt haben, so kommt er (egal ob vom 2Achsen oder Kreis) auf eine neue Seite, auf der er die Tools, die er für seine Statistiken benutzen möchte, auswählen kann. Die Toolauswahl (siehe auch Abbildung 34) kann wie folgt aussehen: Je ein Listenfeld links und rechts, zwischen den beiden Listenfeldern 4 Buttons zum Hinzufügen und Entfernen, ein **Help** Button, ein **Next** Button und ein **Back** Button. Dabei muss der Benutzer in dem Listenfeld links die verschiedenen Tools auswählen und danach auf **Add** klicken. Wenn der Benutzer die Auswahl danach wieder ändern möchte, so kann er einfach auf der rechten Seite das nicht mehr benötigte Tool auswählen und danach auf **Delete** klicken. Mit einem Klick auf **Next** kommt der Benutzer dann zum nächsten Schritt, entweder auf die `Statistic2AxisStep2.jsp` oder die `StatistticGraph.jsp` Seite. Mit einem Klick auf **Back** kommt er wieder auf die Seite zurück, von der er zur Toolauswahl gekommen ist.

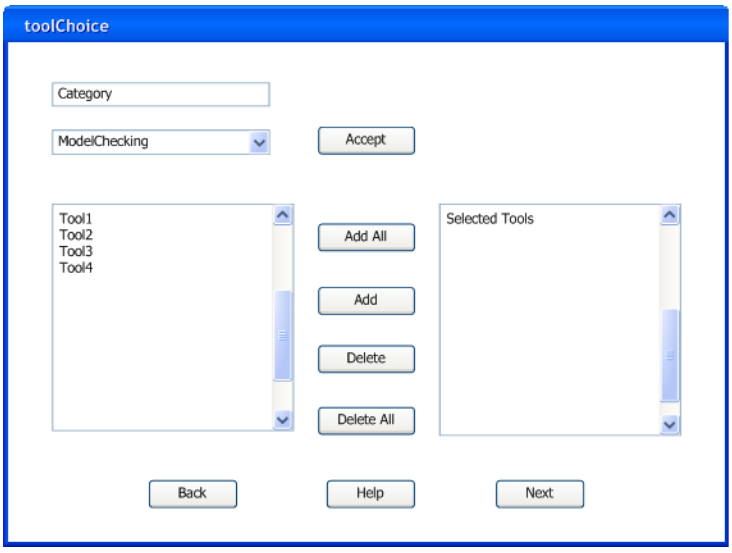

Abbildung 34: Die statisticToolChoice.jsp Seite

Nach dem Klick auf **Next** kommt der Benutzer dann zum letzten Punkt des Wizards. Der Benutzer bekommt hier das fertige Diagramm angezeigt. Auf der Seite sind zusätzliche Auswahlmöglichkeiten die Graphparameter zu speichern, ihn als Bild zu speichern oder um Hilfe zu bekommen. Wie die Parameter des Graphen gespeichert werden sollen, wurde später entschieden. Der Graph soll als `*.png` und als `*.svg` Bilder speicherbar sein. Außerdem soll er die Möglichkeit haben, das Layout des Graphen komplett zu ändern. Dazu muss er auf **Layout** klicken.

In den Layout Seiten (Abbildung 35), egal ob von 2Achsen oder vom Kreis, ist geplant, dass der User dort die angezeigten Diagramme seinen eigenen Wünschen anpassen kann. Er soll die

Möglichkeit bekommen, die Hintergrundfarbe seines Diagrammes, die Farben der verschiedenen Sektionen und die Achsenbeschriftung selbst wählen zu können.

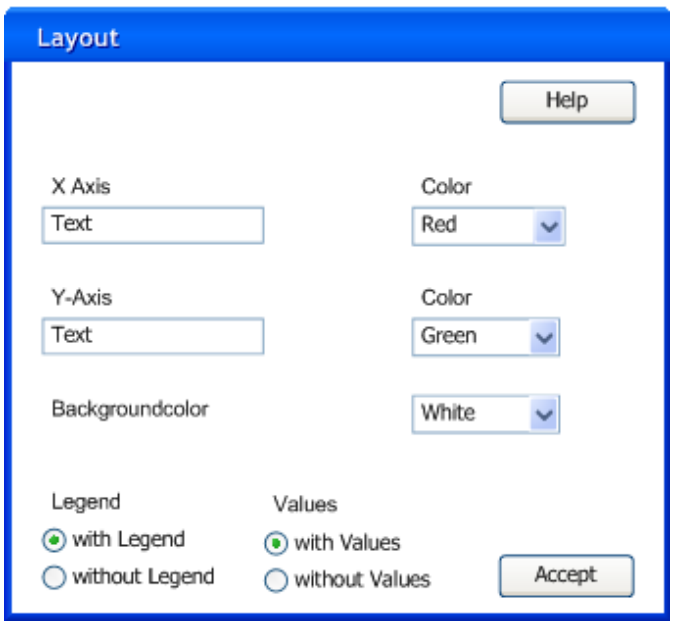

Abbildung 35: Die statisticLayout Seite

Danach musste entschieden werden, wie die Diagramme der User abgespeichert werden sollen. Die Gruppe hat sich dafür entschieden, die verschiedenen Diagramme in einer Datenbank, die dann alle erforderlichen Parameter enthält um das Diagramm neu zu erstellen, abzuspeichern. Das entsprechende Datenbank Layout kann man sich in Abschnitt B.1 genauer ansehen.

Zum Speichern, Laden und Löschen sind die Seiten `statistic_save.html, statistic_load.html` und `statistic_delete.html` zuständig. Auf der `statistic_save.html` Seite soll der Benutzer einen Namen für sein Diagramm angeben können und danach soll dieses gespeichert werden. Auf den `statistic_load.html` und `statistic_delete.html` Seiten bekommt der Benutzer angezeigt, welche Diagramme er bereits abgespeichert hat und kann dann auswählen, ob er ein Diagramm lädt oder es löscht. Dazu muss er nur auf den entsprechenden Namen klicken, denn es ist so geplant, dass der Name in einem Button erscheint. Nach einem Klick soll dann die entsprechende Aktion einfach ausgeführt werden..

5.1.2 Filter

Der Einsatz von Filtern im Bereich von Statistiken ist sehr wichtig. Nur so bekommt der Anwender die Möglichkeit, die Menge der Statistiken auf seine Bedürfnisse anzupassen und die für ihn relevanten Daten angezeigt zu bekommen.

5.1.3 Hilfsfilter

Die Gruppe hat beschlossen dem Benutzer einen Hilfsfilter auf der Seite `toolChoice.jsp` anzubieten. Dieser Filter soll über den eigentlichen Auswahlmöglichkeiten für die Tools angezeigt werden. Der Benutzer kann dort eine Auswahl der Kategorie der aufgelisteten Tools vornehmen

um sich einen besseren Überblick über verschiedene Tools einer Kategorie machen zu können. Dies soll dem Benutzer helfen eine entsprechende Auswahl treffen zu können, auch wenn die Kategorie des Tools nicht aus dem Namen hervorgeht.

Den Designvorschlag für diesen Filter kann man in Abbildung 34 sehen. Der Filter soll mit Hilfe eines DropDown Menüs realisiert werden. Im DropDown Menü werden alle von den Tools verwendeten Kategorien aufgelistet.

5.1.4 Zeitfilter auswählen, speichern & löschen

Die Gruppe hat auf der `statisticGraph.jsp` Seite einen weiteren Filter eingebaut. Der Benutzer hat mit Hilfe des Buttons **Filter** die Möglichkeit Zeitfilter für seine Statistik auszuwählen. Dieser Filter gibt dem Benutzer die Möglichkeit seine Statistik so anzupassen, dass er nur noch die Daten eines bestimmten Zeitraum angezeigt bekommt. Bei einem Klick auf **Filter** gelangt der Benutzer auf die `statisticFilterChoice.jsp` Seite. Auf dieser Seite werden die bereits definierten Filter dynamisch ausgelesen und aufgelistet. Wenn der Benutzer sich für einen oder mehrere Filter entschieden hat, muss er vor den entsprechenden Filtern einen Haken setzen und dann auf **Next** klicken. Oben über der Tabelle soll der Benutzer noch die Möglichkeit bekommen, die ausgewählten Filter mit einem UND oder einem ODER zu verknüpfen. Dies ist Notwendig, da nicht immer ein Filter die Daten so auswerten kann, wie der Benutzer es braucht. Daher ist diese Möglichkeit der Verschärfung sehr wichtig um ihm die Möglichkeit zu geben die Filter so auszuwählen, dass er mit mehreren Filtern genau die Informationen kriegt, die er haben möchte. Einen Überblick über das geplante (und letztlich auch umgesetzte) Design bekommt man in Abbildung 36.

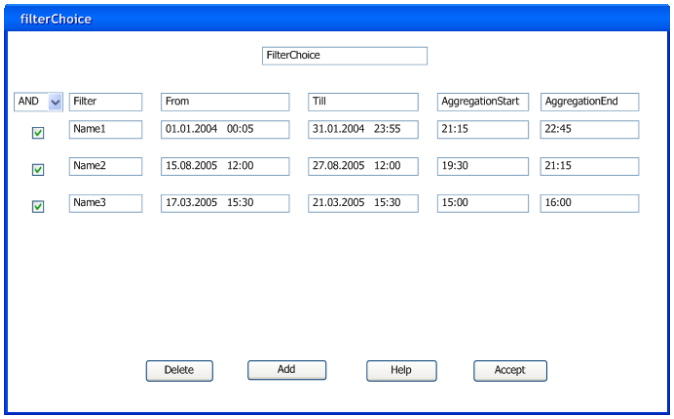

Abbildung 36: Designvorschlag für die `statisticFilterChoice.jsp` Seite

Bei einem Klick auf den Button **Help** wird dem Benutzer eine Hilfestellung zur aktuellen Seite gegeben.

Falls der Benutzer neue Filter hinzufügen möchte so hat er die Möglichkeit mit Hilfe des Buttons **Add** sich eigene Filter zu erstellen. Er gelangt dann auf die Seite `statisticFilter.jsp`, die man in Abbildung 37 sehen kann.

Die Auswahl des Datums und der Uhrzeit wird mit der Hilfe von DropDown Boxen, die alle Auswahlmöglichkeiten enthalten, realisiert. In diesem Filter kann man also den Startzeitpunkt und den Endzeitpunkt der Statistik einstellen. Darunter ist noch die Aggregation vorgesehen. Dort kann man einstellen, wenn man beispielsweise im Zeitraum vom 18.10.2005

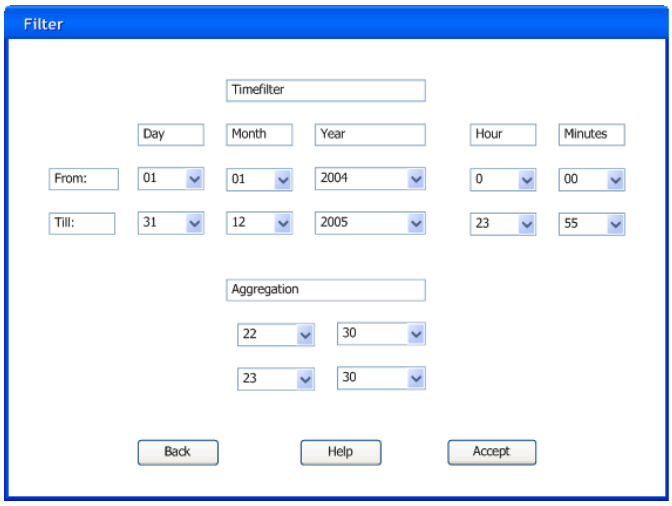

Abbildung 37: Auf dieser Seite kann der Benutzer Filter definieren

0:00 bis zum 31.10.2005 23:55 nur die Ergebnisse in der Zeit von 14 Uhr bis 15 Uhr sehen möchte. Dazu müsste man dann unter Aggregation 14:00 Uhr und darunter 15:00 einstellen. Durch die SQL Abfragen wird der Filter beim Auslesen aus der Datenbank nur noch die relevanten Daten lesen. Falls der Benutzer ein nicht existierendes Datum, wie z.B. 31.2.2005, auswählt, so muß die Seite noch mal aufgerufen werden, mit der Meldung das die Eingaben des Benutzers einen Fehler enthalten. Es soll erst weiter verlinkt werden, wenn der Benutzer ein korrektes Datum eingegeben hat. Der Filter wird nach der Definition und dem Klick auf **Next** automatisch in der Filter Datenbank abgespeichert. Der Benutzer gelangt dann zurück zur `statisticFilterChoice.jsp` Seite. Sollte der Benutzer auf **Back** klicken, gelangt er zwar auch zurück zur `statisticFilterChoice.jsp` Seite, allerdings wurde dann der Filter nicht in der Datenbank abgespeichert. Beim Wiederaufruf der Seite soll dann der neue Filter des Benutzers mit angezeigt werden, so dass er ihn auch auswählen kann. Das Datenbank Layout der filterDB kann man in Abbildung 79 sehen.

Möchte der Benutzer Filter wieder löschen, so muss er die entsprechenden Filter auswählen und danach auf **Delete** klicken. Die Filter werden dann aus der Datenbank entfernt.

5.1.5 Zwangsfilter

Toolprovider dürfen in der Statistik Web Applikation nur die Statistiken zu Ihren eigenen Tools sehen. Daher hat man beschlossen dafür Zwangsfilter einzusetzen. Nach einigen Überlegungen wurde beschlossen, dass die Zwangsfilter für Gruppen definiert werden sollen, da ja jeder Benutzer einer bestimmten Gruppe zugeordnet wird. Außerdem soll noch die Möglichkeit bestehen, das einzelnen Usern noch Sonderrechte hinzugefügt oder gelöscht werden können. In Abbildung 38 kann man den Designvorschlag für die Zwangsfilter sehen. Auf der linken Seite werden die verschiedenen Gruppen oder User angezeigt. Nachdem man eine Gruppe bzw. einen User ausgewählt hat, kann der Administrator die Rechte verändern. Dazu muss er in die Liste der erlaubten Toolprovider entweder neue Toolprovider aus der Gesamtliste hinzufügen oder er muss Toolprovider entfernen. Danach muss er die Änderungen noch speichern. Auf diese Seite hat nur der Admin Zugriffsrecht, da er die Zwangsfilter für die Gruppen bzw. User setzt.

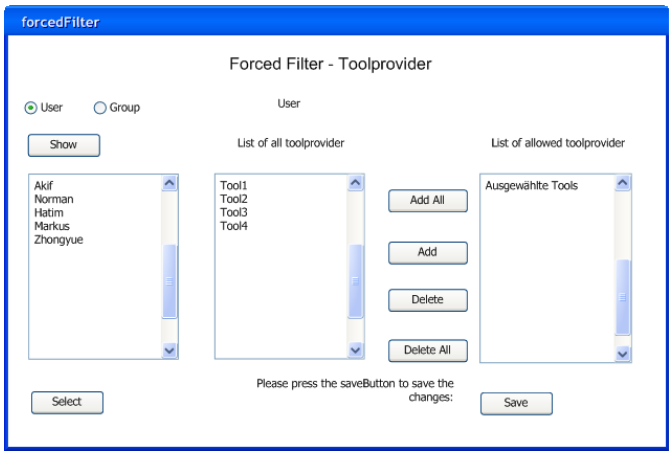

Abbildung 38: Der Designvorschlag für die ForcedFilter

5.2 Umsetzung

Nachdem die grobe Planung abgeschlossen war, mussten wir uns überlegen welche Technologien wir zum Umsetzen unserer Anforderungen benutzen. In diesem Abschnitt bekommt man einen kurzen Überblick über verschiedene Technologien, die wir zur Umsetzung der Aufgaben benutzt haben. Außerdem bekommt man noch zusätzliche Informationsquellen aufgezeigt.

5.2.1 Mehrsprachigkeit

Da die Statistik Web Applikation in mehreren Sprachen verfügbar sein soll, haben wir uns für den Einsatz von Resource Bundles entschieden. Ressource Bundles bieten aber noch weitere Vorteile, siehe dazu den nachfolgenden Abschnitt 5.2.2. Damit die nicht dynamischen Elemente der JSP-Seiten auch in mehreren Sprachen verfügbar sind, haben wir die Ordnerstruktur so angelegt, dass alle Seiten doppelt existieren. Einmal im Ordner de (Deutsch) und einmal im Ordner en (Englisch). Für die dynamischen Elemente wurden wie bereits beschrieben Resource Bundles eingesetzt. Um die Sprache jederzeit wechseln zu können, wurde auf jeder Seite die Deutsche sowie die Englische Fahne eingefügt, so dass man auf jeder Seite zwischen den beiden Sprachvarianten wechseln kann. Für das Wechseln selber ist das Servlet `SetLanguageServlet.java` zuständig. Es setzt den zu verwendenden Pfad für die Applikation dementsprechend neu. Wenn man noch eine dritte Sprache hinzufügen möchte, so funktioniert dieses durch die Resource Bundles sehr einfach.

5.2.2 Resource Bundles

Mit Hilfe der Ressource Bundles kann man den kompletten dynamischen Inhalt der Webseiten in dieser Datei festlegen. Meistens werden sie für eine Mehrsprachigkeit eingesetzt. In unserer Applikation werden sie für die Mehrsprachigkeit und die Auswahlmöglichkeiten für die Diagramme eingesetzt. Der Vorteil dieser Technik liegt darin, dass der dynamische Text der Oberfläche in ein Bundle ausgelagert wird. Dadurch wird eine sehr gute Erweiterbarkeit gewährleistet, da z.B. eine weitere Sprache oder ein weiterer Punkt in den Auswahlmöglichkeiten ohne Codeänderungen hinzugefügt werden kann. Im Bundle hinzugefügt, wird der Punkt ohne

weitere Codeänderung automatisch in die JSP Seite mit eingebunden. Weitere Informationen zu Resource Bundles findet man unter [Api()].

5.2.3 Custom Tags

Um eine bessere Lesbarkeit des Quellcodes zu erreichen wurden Custom Tags eingesetzt. Custom Tags sorgen dafür, dass kein Java Code mehr auf den JSP Seiten verwendet wird. Der Java Code wird in einem Tag ausgelagert. Dadurch ist eine strikte Trennung zwischen Design und Programmierung möglich, denn die Custom Tags können von anderen Personen (z.B. Programmierer) entwickelt werden und der JSP Webseiten Designer kann sie dann in seinem Editor verwenden.

Custom Tags bestehen im wesentlichen aus 3 Teilen:

- dem Java Bean, dabei handelt es sich um compilierten Java Code, der als .class oder .jar Datei vorhanden sein muss. Der Java Bean ist entweder von `javax.servlet.jsp.tagext.TagSupport` oder vom `Body.Tag.Support` abgeleitet. Sie enthält für alle Attribute Get und Set Methoden. Zusätzlich werden die Methoden `doStartTag()`, `doAfterBody()` und `doEndTag()` benötigt.
- die TLD-Datei (Tag Library Descriptor), diese definiert per XML die verschiedenen Tag Namen die man einsetzen möchte, die Referenz auf die zu benutzende JavaBean, diemöglichen Parameter, sowie die zusätzlichen Eigenschaften der Tags bzw. Parameter.
- den Code Fragmenten, die zur Einbettung in JSP Dateien benötigt werden. Die Code Fragmente definieren eine Referenz auf eine TLD-Datei und einen Taglib-Präfix (`<%@ taglib uri= "..."prefix="..."%>`). Die Code Fragmente können dann über HTML-Tags mit Hilfe des Taglib-Präfix und dem Tag-Namen (`<meinetaglib:MeinCustomTag ... >`) verwendet werden.

Weitere Informationen zu CustomTags findet man unter [Spielman(2001)].

5.2.4 Behandlung von Benutzer-Fehlern

Als weiteres wurde überlegt wie die Fehler, die durch die fehlerhafte Benutzung der Statistik Web Applikation entstehen, behandeln kann. Um Exceptions abzufangen kann man bei der JSP Programmierung natürlich, wie bei Java üblich, den Quellcode einfach in einer try und catch Umgebung schreiben. Dies funktioniert ohne Probleme. Wenn man jedoch auf verschiedenen Webseiten immer wieder die gleichen Exceptions bekommt oder es sehr viele unterschiedliche sind, ist es ratsam Exceptions etwas anders zu handhaben. Eine einfache Handlingweise ist die ErrorPage. Diese muss in der JSP Seite spezifiziert werden. Dies geschieht indem man `<% page errorPage="NameOfTheErrorPage.jsp"%>` vor dem HTML Tag schreibt. Die ErrorPage selbst muss mit `<% page isErrorPage="true" import="java.io.*" %>` gekennzeichnet werden. Der große Vorteil der ErrorPage ist, dass man für verschiedene JSP-Seiten die gleiche ErrorPage nehmen kann. Die ErrorPage kann so konstruiert sein, dass sie z.B. einfach nur anzeigt das ein Fehler aufgetreten ist. Man spart sich damit das Abfangen von vielen einzelnen Exceptions. In der Statistik Web Applikation wurde jede JSP Seite so implementiert, dass bei einer Exception die ErrorPage `statisticError.jsp` angezeigt wird. Die `statisticError.jsp` Seite zeigt dem Benutzer dann die Exception an, die aufgetreten ist.

5.2.5 Hibernate Query Language

Hibernate selbst wurde bereits in Abschnitt 4.2.2 ausführlich beschrieben. Die dort beschriebene Query Language, die Hibernate Query Language (HQL) sieht ähnlich wie SQL aus. Weiterführende Informationen zu HQL findet man z.B. unter: [hibernate.org()].

5.2.6 JFreeChart

Nachdem die Grundlagen nun geklärt sind, brauchten wir noch eine entsprechende Bibliothek mit dessen Hilfe die Statistiken graphisch dargestellt werden können. Die Gruppe entschied sich dabei für die jFreeChart Bibliothek. Diese Bibliothek ist für die Statistik Web Applikation sehr wichtig. Bei JFreeChart handelt es sich um eine Java-Bibliothek, die zur Erstellung von Diagrammen benutzt wird. Dabei gibt es viele verschiedene Ausgabemöglichkeiten. Es besteht die Möglichkeit Diagramme in Applikationen, Applets, Servlets und JSPs zu verwenden, aber auch ein Export in diverse Bildformate (PNG/ JPG) und Dokumentformate (PDF/ SVG) ist möglich. Die Bibliothek steht unter der GNU Lesser General Public Licence (LGPL). JFreeChart stellt dabei viele verschiedene Diagrammtypen wie z.B. Kreisdiagramme, Balkendiagramme oder Liniendiagramme und noch viele weitere zur Verfügung. JFreeChart verfügt über `create`-Methoden mit denen man die verschiedenen Diagramm-Instanzen erzeugt. Das Aussehen der Diagramme wird über `plot`- und `renderer`-Klassen erzeugt und geändert. Diagramme können in vielen Bereichen verändert werden, so ist z.B. das Anpassen von Schriftarten, Schriftgröße, das Achsenerscheinungsbild oder das ändern der Farben ohne größere Probleme möglich. Um komplexe grafische Diagramme auf Webseiten darstellen zu können wurde das Cewolf Projekt gegründet. JFreeChart dient bei diesem Projekt zum Rendern des Diagramms. Es werden dabei keine Daten auf der Serverseite erzeugt, da alles auf Session-Objekten und dynamischer Datenanalyse basiert. Das Servlet kümmert sich dabei um das Rendering des Diagramms während die Taglibrary die Diagrammdefinition in der JSP in ein HTML img Tag umwandelt, welches das Diagramm beim Servlet anfordert. Weitere Informationen zu JFreeChart findet man unter [Javadoc()] und [charting the web()].

5.3 Implementierungsphase

In diesem Abschnitt findet man die Umsetzungen der Designvorschläge, sowie die gesamte Implementierung der Statistik Web Applikation. Dabei wird immer auf die verschiedenen JSP Seiten und Servlets einzeln eingegangen.

5.3.1 Workflows

Bevor die Gruppe mit dem Implementieren angefangen hat, wurde noch basierend auf den bisherigen Erkenntnissen und Festlegungen ein Statistik JSP Webseiten Flow erstellt. In diesem Workflow sollte der genaue Ablauf der JSP Seiten, sowie die Anzahl und der Ablauf der benötigten Servlets genau geregelt sein. Die Überlegungen der Gruppe kann man in Abbildung 39 sehen. Dieser Workflow wurde im Laufe des Semesters immer wieder an neue Situationen und an Änderungen angepasst, so dass die hier abgebildete Version nicht mehr viel mit unserem ersten JSP Workflow zu tun hat.

Die Gruppe hat dabei konsequent darauf geachtet, dass JSP Seiten nur für die Ansicht und Servlets zum Verarbeiten der Daten zuständig sind. Dies hat den Vorteil, dass später, falls das JETI

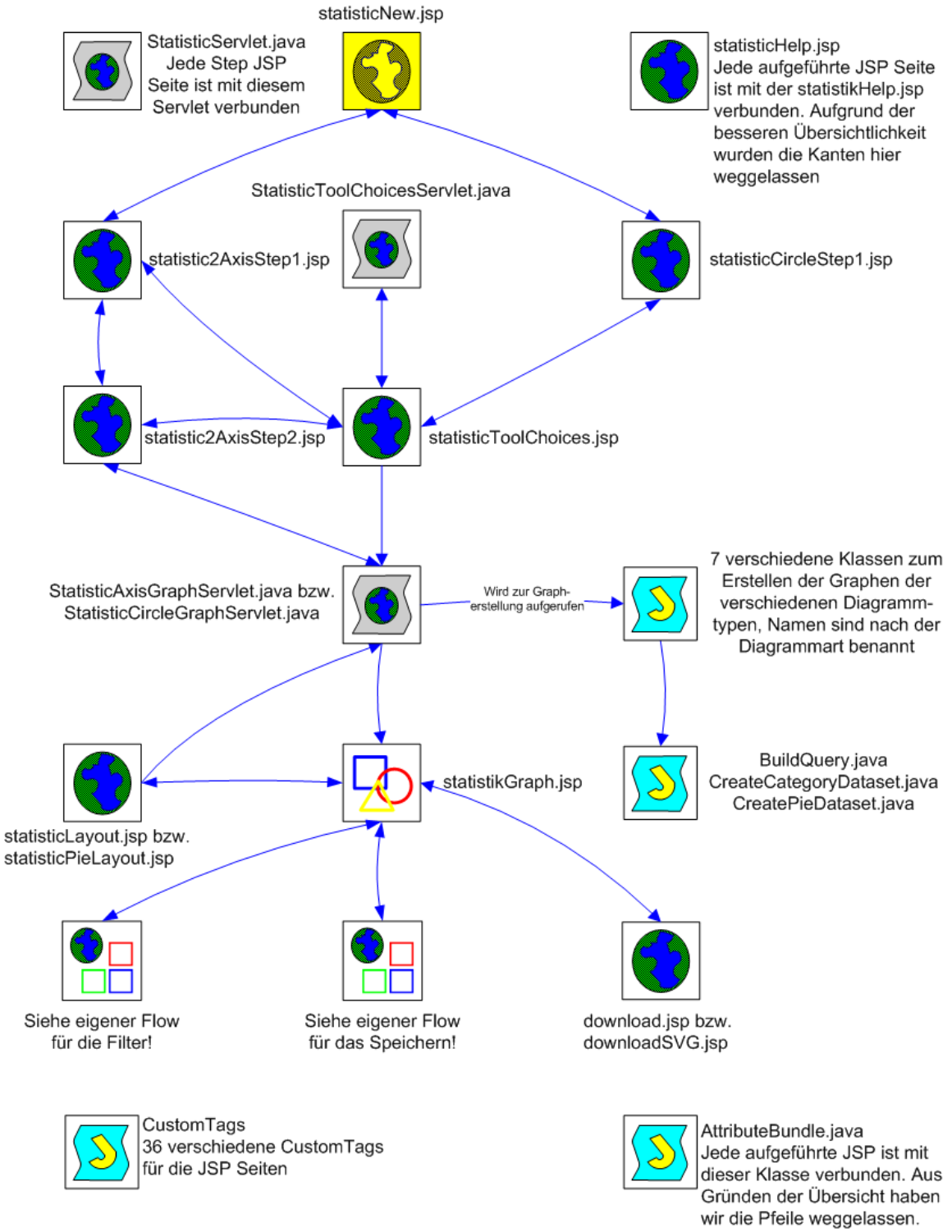

Abbildung 39: Der Statistik JSP Webseiten Flow

System noch verbessert werden soll, man Graphische Änderungen nur in den JSP Seiten ändern muss und Quellcode Änderungen immer in den Servlets macht. Dies bedeutet, eine Änderung des Aussehens könnte genauso gut von einem Webdesigner gemacht werden, obwohl dieser eventuell gar nicht mit Java umgehen kann. Falls die JSP Seiten doch Java Code benötigten, so wurde dieser mit Hilfe eines CustomTags ausgegliedert, so dass in den JSP Seiten bis auf kleine Ausnahmen kein Java Code enthalten ist.

Im Workflow kann man den Ablauf der Statistik Web Applikation sehen. Er zeigt wie man von der Startseite (`statisticNew.jsp`, Gelb markiert) mit Hilfe der Servlets und den weiteren JSP Seiten zur fertigen Statistik gelangt und wann die einzelnen Seiten benutzt werden. Der Workflow zeigt weiterhin, das wir insgesamt 36 Custom Tags für die Statistik Web Applikation entwickelt haben. Diese wurden dann in den JSP Seiten verwendet. Außerdem haben wir ein Servlet, das für die komplette Navigation zuständig ist, das Servlet `statisticServlet.java`. Es steuert die Navigation von JSP Seite zu JSP Seite. Das eben angesprochene Konzept, das Servlets für das Verarbeiten von Daten zuständig sein sollen, kann man im Workflow sehr schön erkennen. Immer wenn Daten verarbeitet oder aufbereitet werden müssen wurde ein Servlet eingebaut. Manche Servlets(z.B. die `...GraphServlet.java`) wurden dabei so groß und unübersichtlich, das verschiedene Klassen dieser Servlets nochmal in eigene Java Klassen ausgegliedert haben. Für den dynamischen Inhalt der Webseiten wurde, wie bereits beschrieben, die Technologie der ResourceBundles verwendet. Die ResourceBundle-Klasse heißt in der Statistik Web Applikation `AttributeBundle.java`. Zusätzlich zu der genannten Klasse, wurden noch 2 weitere Klassen (mit den Zusätzen de und en) implementiert, in denen dann der länderspezifische Text steht.

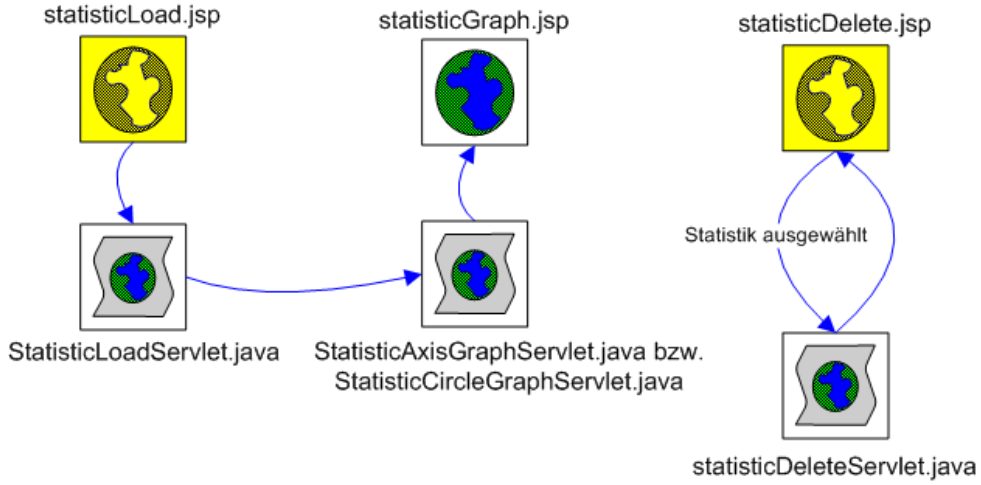

Abbildung 40: Der Statistic Load und Delete Flow

Für kompliziertere Abläufe, wie für das Laden, Löschen und Speichern einer Statistik sowie für den Ablauf der Filter wurden noch eigene Workflows erstellt. Die Überlegungen der Gruppe dazu, kann man in den Abbildungen 40 und 41 sehen.

Abbildung 40 zeigt den Ablauf des Ladens bzw. des Löschens. Beim Laden befindet sich der Benutzer zuerst auf der Seite `statisticLoad.jsp`. Dort muss er sich für ein bereits gespeichertes Diagramm entscheiden und es auswählen. Durch seine Auswahl werden mit dem Servlet: `statisticLoadServlet.java` alle abgespeicherten Daten wieder in die Session geladen und zugewiesen. Der Benutzer bekommt als nächstes die Seite `statisticGraph.jsp` mit dem Graphen der geladenen Diagrammparameter angezeigt.

Beim Löschen befindet sich der Benutzer zuerst auf der Seite `statisticDelete.jsp`. Dort muss der Benutzer das gespeicherte Diagramm, was er löschen möchte, auswählen. Mit Hilfe des Servlets, `statisticDeleteServlet.java`, wird das gespeicherte Diagramm gelöscht. Danach wird er zurück zur `statisticDelete.jsp` Seite geleitet und bekommt eine Meldung, ob die ausgewählte Statistik erfolgreich gelöscht wurde.

Abbildung 41 zeigt den Ablauf wenn man Daten eines Diagramms speichern möchte. Der Benutzer befindet sich zuerst auf der Seite `statisticGraph.jsp` und gelangt mit Hilfe des `statisticServlets` zur `statisticSave.jsp`. Dort muss er einen Namen für seine Statistik eingeben. Nach der Eingabe des Namens muss er mit **Save** bestätigen und die Daten werden durch das Servlet `statisticSaveServlet.java` gespeichert. Nachdem die Daten gespeichert wurden, wird der Benutzer wieder zurück auf `statisticSave.jsp` geleitet. Dort kann er mit Hilfe des **Back** Buttons wieder zur `statisticGraph.jsp` Seite zurückkehren. Oder durch **New** eine neue Statistik erstellen.

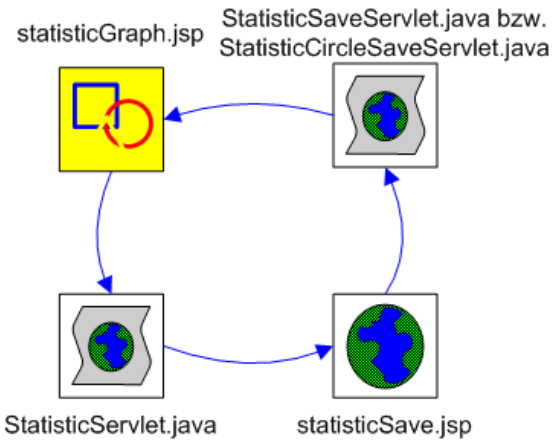

Abbildung 41: Der Statistik Save Flow

Abbildung 42 zeigt den Ablauf der Zeitfilter. Zuerst befindet sich der Benutzer auf der `statisticGraph.jsp` Seite. Mit Hilfe vom `statisticServlet.java` kommt er zur `statisticFilterChoice.jsp`. Auf dieser Seite kann der Benutzer sich Filter aussuchen. Nach seiner Auswahl gelangt er mit Hilfe des `statisticFilterServlet.java` entweder wieder zurück zur `statisticGraph.jsp` oder zur `statisticFilter.jsp`. Auf dieser Seite kann der Benutzer neue Filter erzeugen und diese werden dann mit Hilfe `statisticCreateFilterServlet.java` erzeugt. Dieses Servlet leitet ihn auch wieder zurück zur `statisticFilterChoice.jsp` Seite, wo er wieder neu entscheiden kann.

5.3.2 Statistik Erstellen

Die erste Seite die implementiert wurde, war die Startseite `statisticNew.jsp`. Diese sieht fertig implementiert wie Abbildung 43 aus.

Die Seite wurde mit Hilfe eines HTML Formularfeldes implementiert. Der Benutzer hat in den DropDown Menüs die Auswahl aus den Diagrammtypen: BarChart (2D/3D), LineChart (2D/3D), AreaChart und PieChart (2D/3D). Das für seine Zwecke entsprechende Diagram muss der Benutzer auswählen.

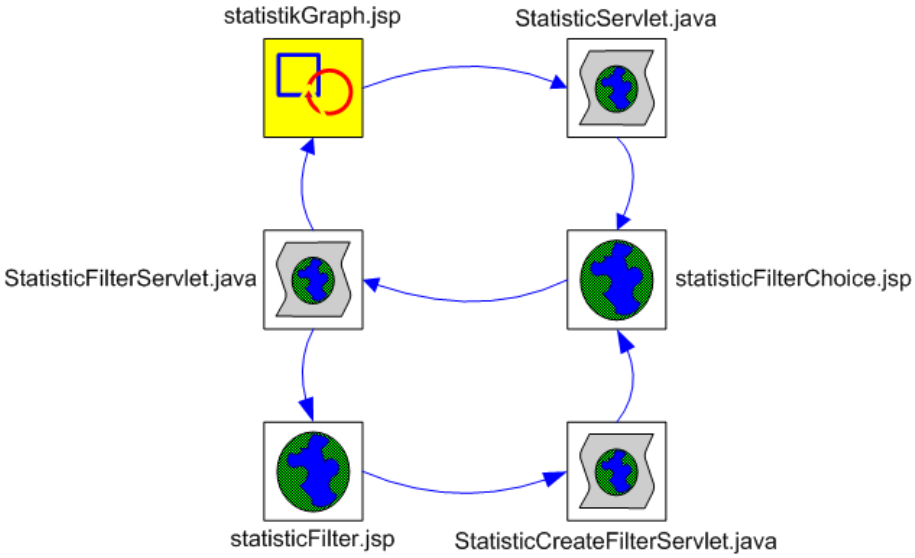

Abbildung 42: Der Statistik Filter Flow

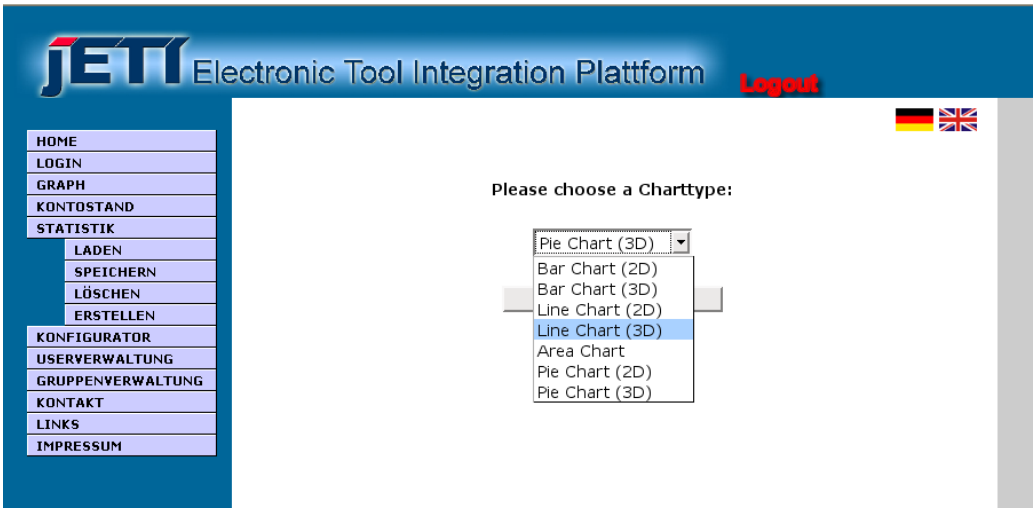

Abbildung 43: Die fertig implementierte `statisticNew.jsp` Seite

Der **Help**-Button leitet den Benutzer zur `StatisticHelp.jsp` Seite weiter. Auf der `StatisticHelp.jsp` Seite bekommt der Benutzer die Hilfe des Programms zur Diagrammauswahl angezeigt.

Die `StatisticHelp.jsp` Seite wurde so implementiert, dass in dieser Datei die Hilfe für alle JSP-Seiten, die für die Statistik Web Applikation benötigt werden, enthalten ist. Die `StatisticHelp.jsp` Seite erkennt mittels der Session-Variable `page`, von welcher Seite aus die Hilfe angewählt wurde und zeigt dann auch nur den entsprechenden Text für diese Seite an. Dies hat dann den Vorteil, das es eine zentrale Datei gibt, die jegliche Hilfe Texte enthält und man immer direkt auf die Hilfe zugreifen kann. Sicherlich hätte man die Hilfe auch innerhalb der einzelnen Seiten implementieren können, dies hätte aber den Nachteil gehabt, das die Hilfe nicht mehr zentral gewesen wäre. Um die Variablen der Session zu übergeben haben wir uns, damit der Benutzer nicht sehen kann wie die Variablen heißen, für die doPost Methode entschieden. Die Gruppe hat sich bei allen JSP Seiten für den Aufbau der Seiten mit Hilfe von Formularfeldern, sowie die Weiterleitung mit Hilfe der Session-Variablen und der doPost-Methode entschieden. Dabei wird auch immer mit einem hidden-input-type gearbeitet, um die Values für die Seitenzahlen zu setzen. Alle JSP-Seiten haben eine eigene Nummer, mit der sie sich bei den Servlets eindeutig identifizieren können. Die genauen Nummern kann man in der Tabelle in Abschnitt B.2 sehen. Die Überprüfung der Session-Variable erfolgt mit Hilfe von switch, wobei die cases dann den Seitenzahlen entsprechen.

Für die komplette Navigation durch die Statistik Web Applikation ist das statisticServlet.java Servlet zuständig. Dieses Servlet enthält, genauso wie die `statisticHelp.jsp`, eine switch case Anweisung über die Session Variable `page`. In den cases wird dann der einzelne Quellcode für die verschiedenen Fälle ausgelesen und zugewiesen. Am Ende jedes Falles steht im String `path` wie die nachfolgende Seite heißt. Für die Navigation wurde somit auf jeder JSP Seite ein **Next**, **Back** und **Help** Button implementiert. Mit **Next** gelangt der Benutzer immer zur nachfolgenden Seite, mit **Back** immer zur Vorgängerseite und mit **Help** immer zur Hilfe zur aktuellen Seite.

Auf jeder Seite wird überprüft ob der Benutzer eingeloggt ist und ob er die erforderlichen Rechte besitzt. Dies geschieht mit den Methoden `CheckLogin` und `CheckFeatures`. Sollte der Benutzer also mitten im Wizard ein Bookmark setzen und versuchen nach einer gewissen Zeit dieses dann wieder aufzurufen, bekommt er eine Fehlermeldung.

Die Gruppe hat in die Statistik Web Applikation einen **History Modus** implementiert. Dieser sorgt dafür, dass bei der Vorgängerseite alle Auswahlmöglichkeiten so gesetzt sind, wie der Benutzer sie vorher ausgewählt hatte. Dies erspart dem Benutzer bei einem kleinen Fehler, dass er alle Angaben neu machen muss. Für diesen **History Modus** wird die bis dahin gespeicherte Session-Variable wieder ausgelesen und bei den Auswahlmöglichkeiten genau so wieder umgesetzt.

Der Benutzer gelangt also mit Hilfe des `statisticServlet.java` Servlets zur nächsten Seite. Diese ist je nachdem welches Diagramm er ausgewählt hat, entweder bei einem Kreisdiagramm `statisticCircleStep1.jsp` oder bei einem anderen Diagramm `Statistic2AxisStep1.jsp`.

Dabei werden die Auswahlmöglichkeiten der Seite dynamisch mit Hilfe des String Arrays `paramter` aus der `AttributeBundle.java` aufgebaut In Abbildung 44 kann man die CheckBoxen für **Own Account** und **All Accounts** sehen. Diese waren in der Zeichnung der Analysephase noch nicht vorhanden. Diese CheckBoxen sind nötig geworden, damit man die Statistiken weiter eingrenzen kann. **Own Account** bedeutet man bekommt nur Daten von seinem eigenen Account in der Statistik angezeigt. **All Accounts** bedeutet, dass man die Daten von allen Usern

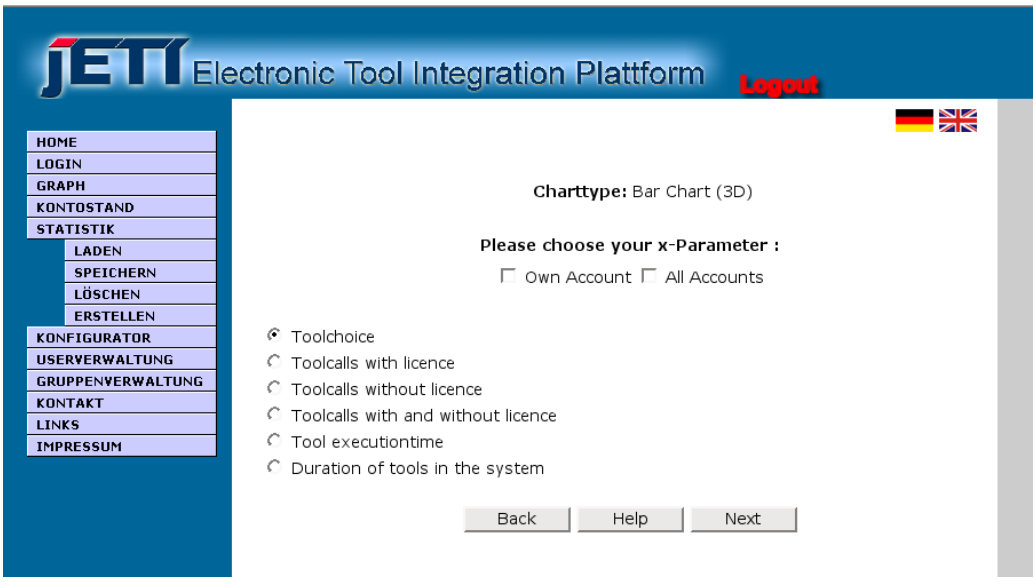

Abbildung 44: Die fertig implementierte `Statistic2AxisStep1.jsp` Seite

des Systems angezeigt bekommt (sofern kein Zwangsfilter gesetzt ist, dazu aber später mehr). Dabei spielt es keine Rolle ob der Benutzer nur eine Checkbox anwählt oder beide. Er muss aber mindestens eine anwählen. Wenn der Benutzer beide CheckBoxen anwählt, so bekommt er im Graphen später eine Graphic für seinen eigenen Account und eine für alle Accounts angezeigt. So ist es für den Benutzer möglich seine eigenen Daten mit denen von allen Usern zu vergleichen. Allerdings ist hier auch wichtig direkt die Benutzereingaben zu überprüfen. Die Seite wird mit Hilfe des `statisticServlet.java` Servlets validiert. Das bedeutet, die Seite wird überprüft, ob er mindestens einen Haken für die Accounts gesetzt hat. Sollte der Benutzer keinen Haken gesetzt haben, wird die Seite erneut aufgerufen und es erscheint für die fehlenden Parameter ein Hinweis, siehe dazu Abbildung 45, hier fehlt exemplarisch der erforderliche Haken.

Im oberen Bereich der Seite wird dem Benutzer angezeigt, welche Parameter er bislang ausgewählt hat. In Abbildung 45 ist es als Diagrammtyp das Balkendiagramm 2D. Als Auswahlmöglichkeiten wurden die Statistiken umgesetzt, die sich die Gruppe in Abschnitt 5.1.1 ausgedacht hatte, also hat der Benutzer die Auswahl aus einer Toolauswahl, Toolaufrufen mit Lizenz, Toolaufrufen ohne Lizenz, Toolaufrufen mit und ohne Lizenz, der Toolausführungsdauer in Stunden und der Zeitdauer im System in Tagen.

Die `statisticCircle.jsp` (siehe Abbildung 46) wurde genauso wie die `Statistic2AxisStep1.jsp` implementiert.

Ein Unterschied zwischen den beiden Seiten ist, dass es auf dieser Seite andere Auswahlmöglichkeiten gibt. Hier werden Sie dynamisch mit Hilfe des String Arrays `circleParameter` erzeugt. Allerdings gibt es hier noch einen weiteren kleinen Unterschied. Der Benutzer bekommt je nachdem welche Kombination er auswählt, später eine unterschiedliche Menge an Diagrammen angezeigt. Unser erster Entwurf sah vor, das nur ein einzelnes großes Kreisdiagramm angezeigt wird. In diesem sollten alle Attribute dargestellt werden. Nach der Implementierung und den ersten Tests wurde jedoch erkannt, dass ein einzelnes Diagramm viel zu unüberischtlich ist, um Informationen aus diesem zu bekommen. Also hat man sich für Multiple Pie Charts entschieden. Dies bedeutet, dass wenn der Benutzer z.B. **Own Account** und **Toolcalls with Li-**

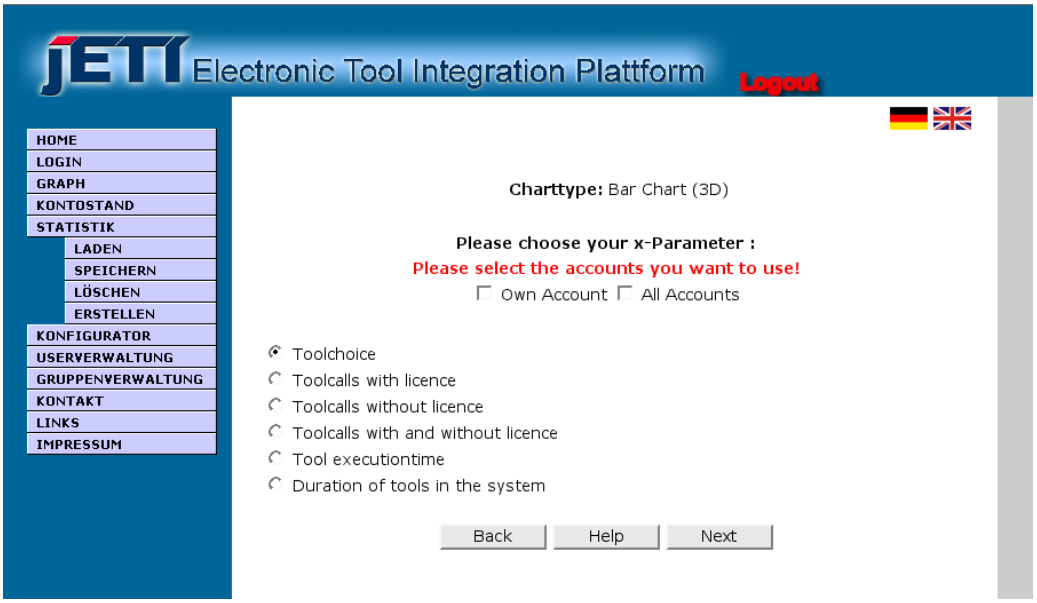

Abbildung 45: Die `Statistic2AxisStep1.jsp` Seite mit Hinweisen für den Benutzer

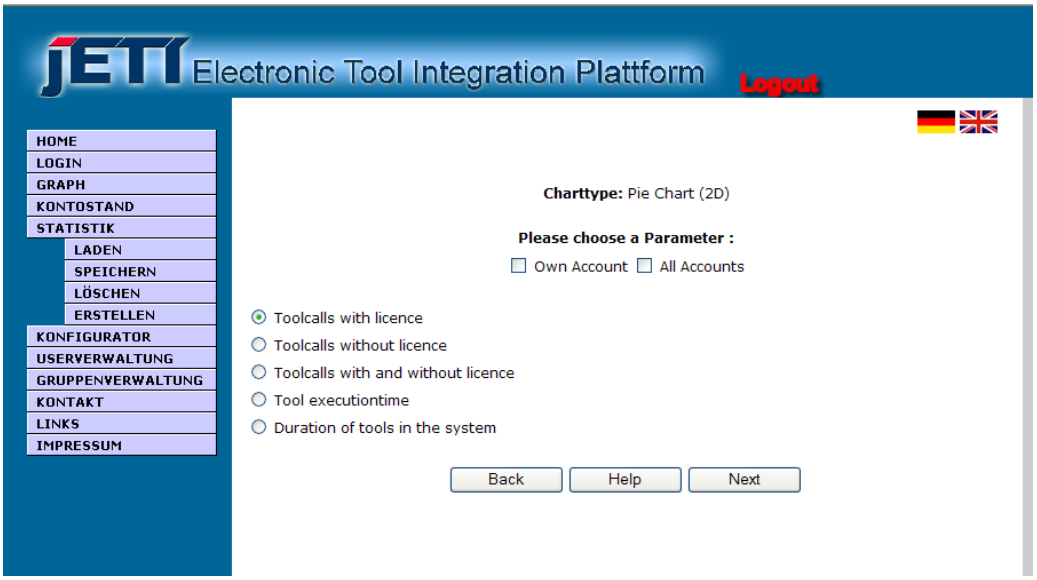

Abbildung 46: Die fertig implementierte `statisticCircleStep1.jsp` Seite

cence auswählt, bekommt er später ein Diagramm angezeigt. Wenn der Benutzer jedoch **Own Account** und **All Accounts**, sowie **Toolcalls with and without Licence** auswählt, bekommt er 4 Diagramme angezeigt. Er bekommt also für jede Kombination ein eigenes Diagramm angezeigt. Die Diagramme werden mit der Methode `createMultiplePieChart` erzeugt. Diese Methode ist Teil der jFreeChart Bibliothek und erzeugt die verschiedenen Diagramme.

Der nächste Schritt ist die `statistikToolauswahl.jsp` Seite. Diese kann man in Abbildung 47 sehen.

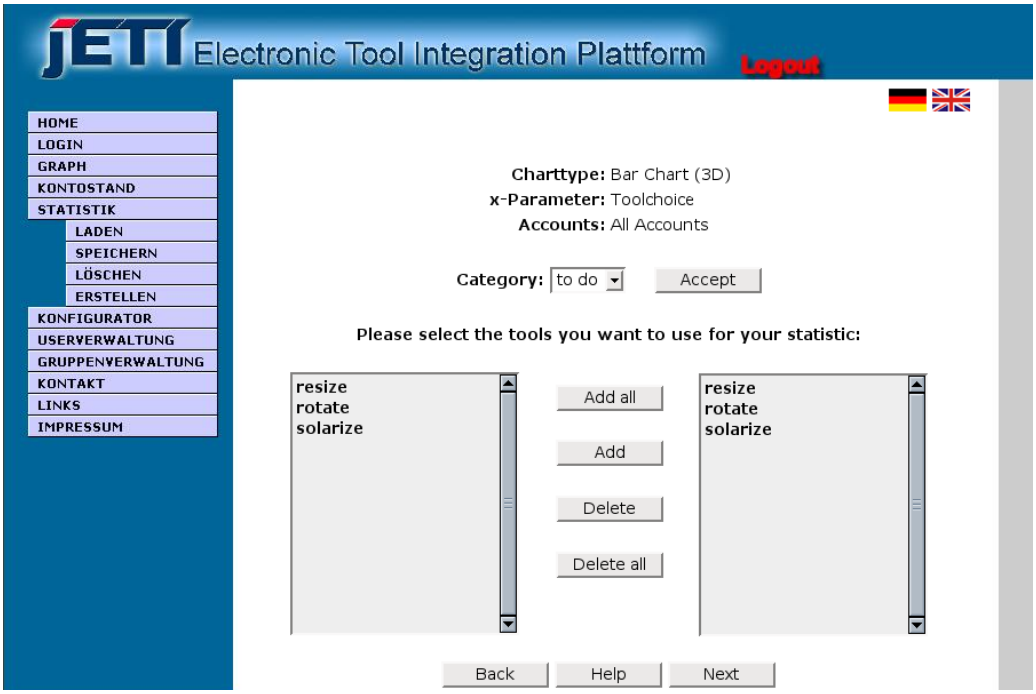

Abbildung 47: Die `statistikToolAuswahl.jsp` Seite

Auf der Seite sieht man, wie im **Wizard** üblich, die bisher ausgewählten Parameter. Darunter sind die beiden Auswahllisten. In der linken Liste sieht man alle Tools, die zur Auswahl stehen. Die Tools werden direkt aus der ToolLogging Tabelle der Datenbank ausgelesen. Sie werden mittels Hibernate (Abschnitt 5.2.5) aus der DB ausgelesen und dann in einem String Array gespeichert. In der rechten Spalte sind die bereits ausgewählten Tools aufgelistet. Der Benutzer kann mit Hilfe der 4 Buttons **Add**, **Add all**, **Delete all** und **Delete** seine Auswahl an Tools verändern.

Für die Auswahl und die damit verbundenen Möglichkeiten ist das `StatisticToolChoiceServlet.java` Servlet zuständig. Es schreibt die Auswahl zuerst in einem Vector (hat den Vorteil, das man vorher nicht wissen muss wie groß die Auswahl ist), wandelt diesen Vector nachher in ein String Array um und schreibt dieses Array mit der Auswahl dann in die Session. Die Toolauswahl ist mit einer Überprüfung, die Doppelauswahlen verhindert, implementiert, so dass es für den Benutzer unmöglich ist ein Tool doppelt auszuwählen. Dazu überprüft das Servlet ob das hinzu zufügende Element bereits in der Auswahl ist. Da der Benutzer mindestens ein Tool auswählen muss, wird natürlich auch überprüft ob die Auswahl leer ist, wenn dem so ist, bekommt der Benutzer eine Fehlermeldung angezeigt.

Im linken oberen Bereich sieht man den “Kategorie“ Filter. Um diesen Filter zu realisieren braucht man das Feld `category` in der ToolLogging Datenbank. Da die ToolLogging Daten-

bank aber nur Daten auswerten kann, die in den Tools stehen oder vom System angegeben werden und die Unterscheidung nach Kategorien am Anfang nicht vorgesehen war, kann die ToolLogging Datenbank diese Information noch nicht loggen. Nach Rücksprache mit unseren Betreuern wurde beschlossen, dass es eine sinnvolle Erweiterung ist, dieses Attribut in die ToolLogging Datenbank aufzunehmen und die Daten vorerst offen zu lassen. Der Filter ist hier bislang nur vorgesehen und noch nicht fertig implementiert! Dies ist ein sehr guter Ansatz um die Statistik Web Applikation später noch zu verbessern.

Im zweiten Schritt des **Wizards** für die 2Achsen-Diagramme kommt der Benutzer zur Seite `statistic2AAxisStep2.jsp`. Normalerweise hat der Benutzer hier die gleichen Auswahlmöglichkeiten, wie im ersten Schritt. Allerdings wird diese Seite dynamisch aufgebaut, da nicht jede Kombination mit dem vorher ausgewählten Parameter einen Sinn ergibt. Die Gruppe hat sich hier sinnvolle Kombinationen ausgedacht und der Benutzer bekommt so nur noch die Auswahlmöglichkeiten angezeigt, die mit dem ersten Parameter harmonieren. Dazu wurden alle möglichen Kombinationen in der Klasse `AttributeBundle.java` eingefügt und es wird dann die passende Kombination nach der Auswahl auf `statistic2AxisStep1.jsp` des Benutzers angezeigt. Dies geschieht durch das 2D Array **combination** indem die Kombinationen hinterlegt sind. Abbildung 48 zeigt eine dynamisch aufgebaute `statistic2AxisStep2.jsp` Seite.

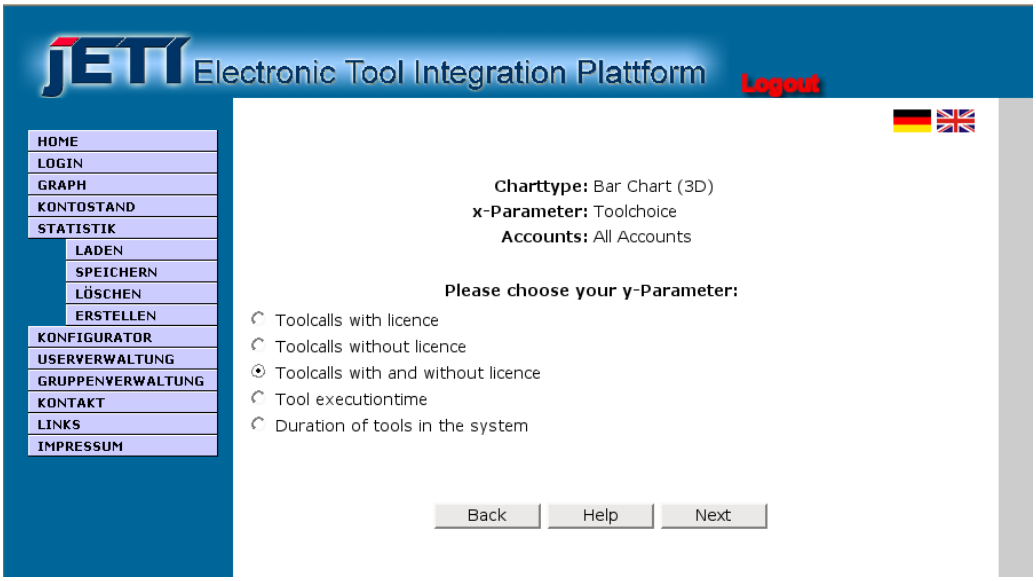

Abbildung 48: Die fertig implementierte `statistic2AxisStep2.jsp` Seite

Bei der Implementierung der ersten Servlets haben wir gemerkt, das bestimmte Bereiche in den Servlets immer gleich sind, und nur minimal an die jeweiligen Diagrammtypen angepasst werden müssen. Daher wurden 2 Servlets (`StatisticAxisGraphServlet.java` und `StatisticCircleGraphServlet.java`) zum Erstellen der Diagramme implementiert. Diese greifen dann auf die jeweiligen Klassen zum Erstellen der Diagramme zurück. Die Funktion zum Erstellen der Datasets wurde, für die beiden Versionen 2Achsen (`CreateDataset.java` und Kreis `CreatePieDataset.java`), wie auch eine eigene Klasse Dafür die Datenbanktabellenabfrage (`BuildQuery.java`) ausgegliedert).

Aus den bisher in die Session geschriebenen Daten wird nun mit Hilfe dieser beiden Servlets `StatisticAxisGraphServlet.java` bzw. des `StatisticCircleGraphServlet.java` das fertige Diagramm erzeugt. Dazu wird die zum Diagrammtyp passende Klasse z.B. `BarChart-2D.java` benutzt. Es gibt insgesamt 7 verschiedene Diagrammtypklassen. Diese heißen so wie

das Diagramm, für das sie zuständig sind. In dieser Klasse wird das Diagramm mit Hilfe der aus jFreeChart stammenden Methode `createChart` erzeugt. In dieser Methode werden zuerst die Daten, die für das Diagramm benötigt werden mit Hilfe der Klassen `CreateDataset.java` und `CreatePieDataset.java` erzeugt.

Die Dataset-Klassen setzen mit Hilfe der Methode `setParameters` die entsprechenden Parameter mit den Daten aus der Session. Dies wurde über eine switch case Anweisung über den `parametercase` implementiert. Durch diese Daten erstellt die Methode dann einen entsprechenden case. In der Methode `setDataset` werden die beiden Arrays columnKey und rowKey deklariert. Diese werden, abhängig vom entsprechenden Fall, mit Daten gefüllt. Im Array columnKey werden die ausgewählten Tools und im Array rowKey werden die passenden Schlüssel gespeichert.

Die BuildQuery Klasse führt mit der Methode `makeQuery` die entsprechenden Abfragen für den konkreten Fall an die Datenbank durch. Die Methode `setData` erzeugt die Abfragen für die Methode `makeQuery`. Die dazugehörigen Daten werden in dem 2DArray `data` gespeichert.

Als nächstes werden die Farben für das neue Diagramm definiert. Als letztes wird nun aus diesen Daten ein neues chart Objekt erzeugt.

Nachdem ein Chart erzeugt wurde, wird dem Benutzer mit Hilfe der `statisticGraph.jsp` Seite (Abbildung 49) angezeigt. Auf dieser Seite bekommt der Benutzer nun seine fertige Statistik mit den Parametern und dem gewählten Diagrammtyp angezeigt, die er vorher ausgewählt hat.

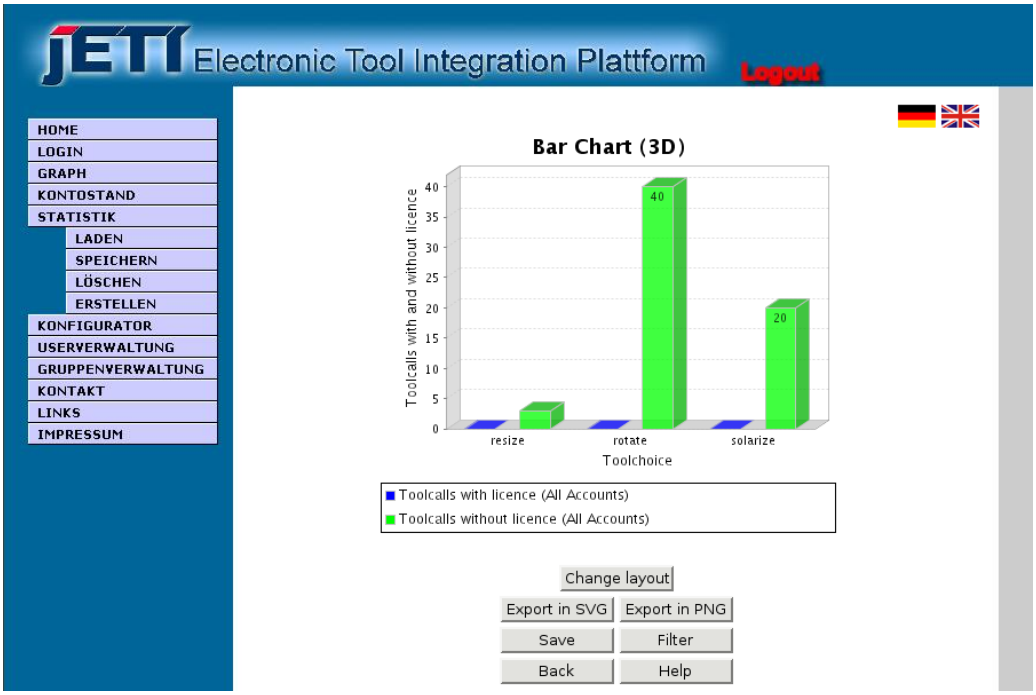

Abbildung 49: Eine Beispiel `statisticGraph.jsp` Seite

Das Diagramm sieht man im oberen Bereich der Seite. Unter dem Diagramm sind die Buttons für die Navigation. Der Benutzer hat hier mit **Export in PNG** die Möglichkeit sein Diagramm im Bilddateiformat PNG zu speichern. Dazu wird dann ein Fenster geöffnet, indem der Benutzer den Namen für das Diagramm und den Speicherort angeben muss. Das Bild wird nun mit der fest eingetragenen Größe 550 * 400 auf der Festplatte des Benutzers abgespeichert.

Die Schaltfläche **Export in SVG** ermöglicht dem Benutzer das Speichern des Diagramms im Vektorformat SVG. Auch hier erscheint wieder ein Fenster für Name und Speicherort. Diese Fenster kann man unter Abbildung 50 sehen.

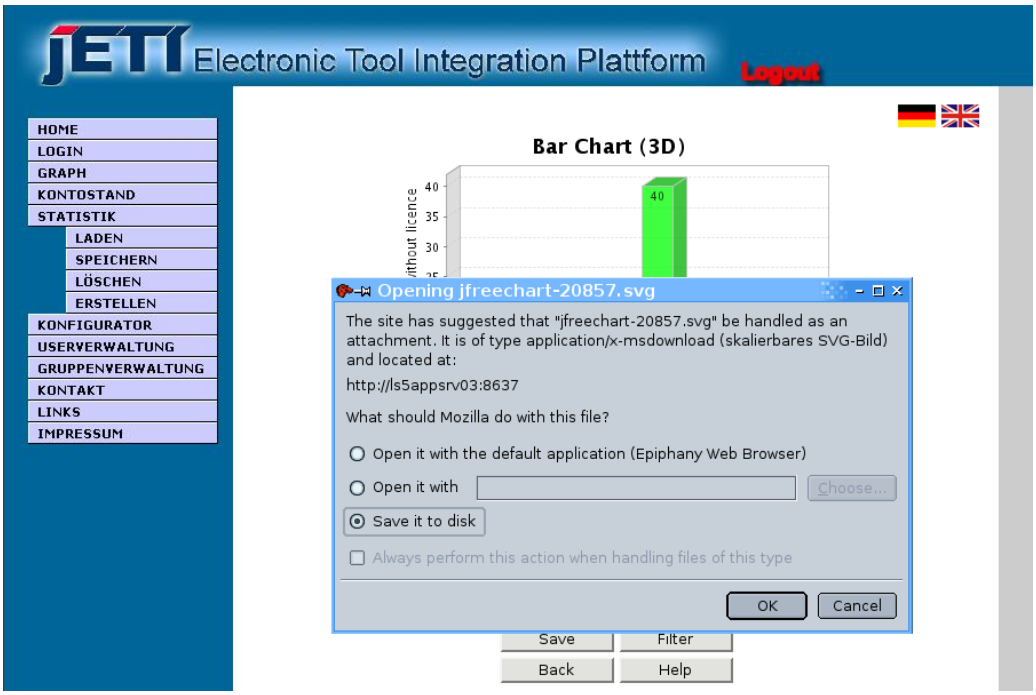

Abbildung 50: Das Fenster zum speichern nach einem Klick auf `Export in SVG`

Diese beiden Bilddateien werden bei der Erstellung des Graphen schon temporär erzeugt und nach der Speicherbestätigung mit Hilfe der Dateien `download.jsp` bzw. `downloadSVG.jsp` gespeichert.

Save ermöglicht dem Benutzer das Abspeichern seiner Statistik. Dies geschieht auf der `statisticSave.jsp` Seite. Der genaue Vorgang des Speicherns wird in Kapitel 5.3.4 noch ausführlich erklärt.

Change Layout, leitet den Benutzer, je nachdem welchen Diagrammtyp er ausgewählt hat, entweder auf die `statistic2AxisLayout.jsp` oder auf die `statisticCircleLayout.jsp` Seite. Auf der `statistic2AxisLayout.jsp` hat der Benutzer die Möglichkeit alle Einstellungen die das aktuelle 2 Achsen Diagramm betreffen, auszuwählen und zu ändern. Diese Seite (siehe Abbildung 51) ist so aufgebaut, dass der Benutzer oben die Namen für die x-Achse und die y-Achse ändern kann. Dazu muss er nur einen neuen Namen in die entsprechenden Felder eintragen. Darunter werden alle Elemente des Diagramms angezeigt. Diese Auflistung erfolgt dynamisch und wird aus den Sessionvariablen ausgelesen. Rechts neben den Namen ist die Farbe für die einzelnen Elemente angegeben. Die Farbe setzt sich aus den 3 Farben Rot, Grün und Blau zusammen. Hier kann man Werte zwischen 0-255 eintragen. Falls der Benutzer hier zu hohe oder zu niedrige Werte eingibt, gelangt der Benutzer zwar wieder zurück zum Graphen, aber seine eingetragenen Werte bei den Farben wurden jedoch nicht gespeichert. Links unten kann der Benutzer auswählen ob er in seinem Diagramm eine Legende angezeigt bekommen möchte, rechts daneben kann er entscheiden ob die einzelnen Werte der Statistik mit angezeigt werden sollen oder nicht. Ganz unten sind die Buttons für die Navigation.

Wenn man als Diagramm einen Kreis Typ ausgewählt hatte, gelangt man auf die `statistic-`

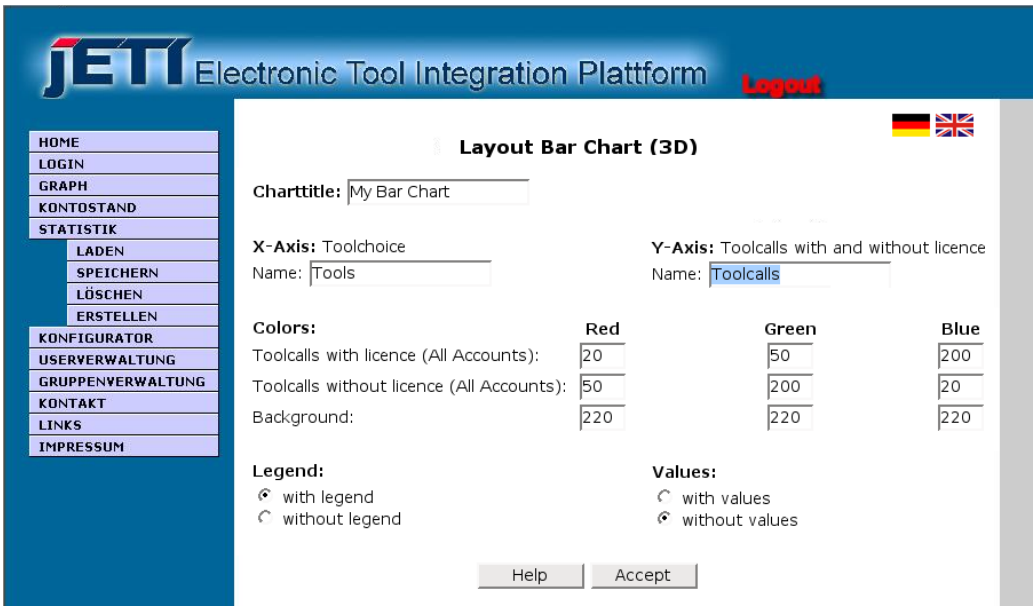

Abbildung 51: Ein Beispiel für eine `statistic2AxisLayout.jsp` Seite

`CircleLayout.jsp`. Diese ist im Prinzip ähnlich aufgebaut wie die Layout Seite von 2Achsen. Allerdings gibt es auf der CircleLayout Seite nicht so viele Auswahlmöglichkeiten wie für 2Achsen. Hier kann man nämlich nur die Hintergrundfarbe des Diagramms ändern. Auch hier werden wieder nur Farben zwischen 0-255 akzeptiert. Das es hier weniger Auswahlmöglichkeiten gibt, liegt einfach daran, das man bei den Multiple Charts, wenn man die größtmögliche Kombination von 4 verschiedenen Diagrammen ausgewählt hat, sonst sehr leicht den Überblick verlieren würde. Daher hat der Benutzer hier nur die Möglichkeit die Hintergrundfarbe zu ändern.

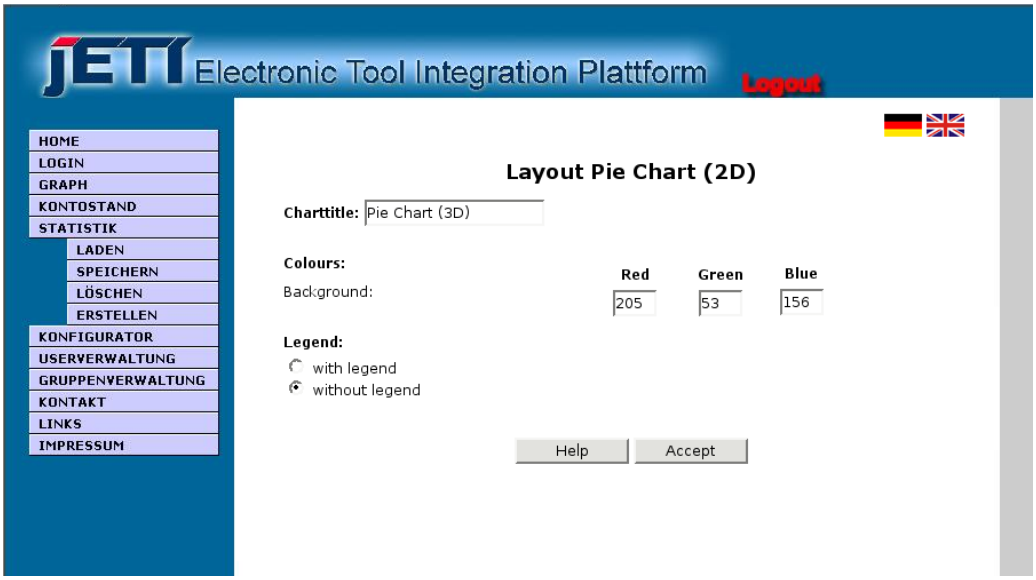

Abbildung 52: Ein Beispiel für eine `statisticCircleLayout.jsp` Seite

5.3.3 Zeitfilter

Auf der `statisticGraph.jsp` Seite gibt es noch den Button **Filter**. Hier hat man die Möglichkeit Filter für das Diagramm auszuwählen. Dies geschieht auf der Seite `statisticFilterChoice.jsp` (siehe Abbildung 53).

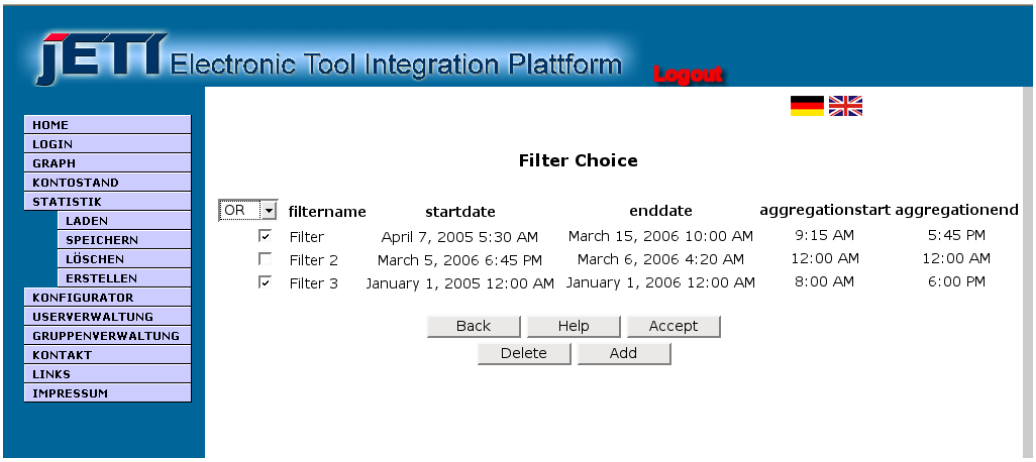

Abbildung 53: Die Umsetzung der `statisticFilterChoice.jsp` Seite

Auf dieser Seite werden alle bisher vom Benutzer abgespeicherten Filter mit dem CustomTag `FilterSelection.java` angezeigt. Das Custom Tag liest dazu die entsprechenden Datensätze aus der Filter Datenbank für den jeweiligen User aus und sortiert diese nach ihrem Namen. Die Auswahl eines Filters erfolgt durch die CheckBox vor dem entsprechenden Filter. Der Filter entspricht dann genau der WHERE Klausel bei der Datenbankabfrage. Außerdem enthält die Seite wieder die entsprechenden Buttons für die Navigation. Der Benutzer hat zusätzlich noch die Möglichkeit seine Filter mit Hilfe eines logischen UND oder einem ODER zu verknüpfen um eine bessere Auswahl zu erreichen. So sollte es ihm möglich sein, seine Statistiken exakt auf seine Bedürfnisse anzupassen.

Mit **Add** können neue Filter erstellt werden. Dies geschieht auf der `statisticFilter.jsp` Seite, die man in Abbildung 54 sehen kann.

Diese Seite wurde wie in der Designphase angedacht umgesetzt. Allerdings waren unsere Überlegungen, wie man verhindern kann dass der Benutzer ein falsches Datum eingibt, vollkommen umsonst. Die Umwandlung des eingetragenen Datums in ein Query erfolgt auch bei der Eingabe eines falschen Datums. Java wandelt dabei die Eingaben um und geht dabei so vor, dass falls die Eingabe der 31. Februar ist, der 28. Februar genommen wird und die restlichen Tage einfach dazu gezählt werden. Aus dem 31. Februar wird also der 3. März gemacht.

Nachdem alle Eingaben erfolgt sind, wird, bevor der Filter mit Hilfe des `StatisticCreateFilterServlet.java` Servlets in die Datenbank geschrieben wird, zuerst überprüft, ob der Benutzer beim Namen nur ASCII Code verwendet hat (diese Überprüfung ist extrem wichtig, da es sonst zu Problemen innerhalb der Datenbank kommen kann, da dort manche Zeichen nicht korrekt abgespeichert werden) und ob der eingegebene Name bereits existiert. Nach dieser Überprüfung findet eine weitere Überprüfung statt und zwar ob ein Enddatum eingegeben wurde, das vor dem Startdatum liegt. Dies geschieht alles automatisch innerhalb der `doPost` Methode. Falls alle Eingaben richtig sind, wird das Attribut filterChecked auf true gesetzt, ansonsten bleibt es auf false und der Benutzer bekommt eine entsprechende Fehlermeldung angezeigt. Die `doPost` Methode versucht nun mit den beiden Methoden createDate und createTime

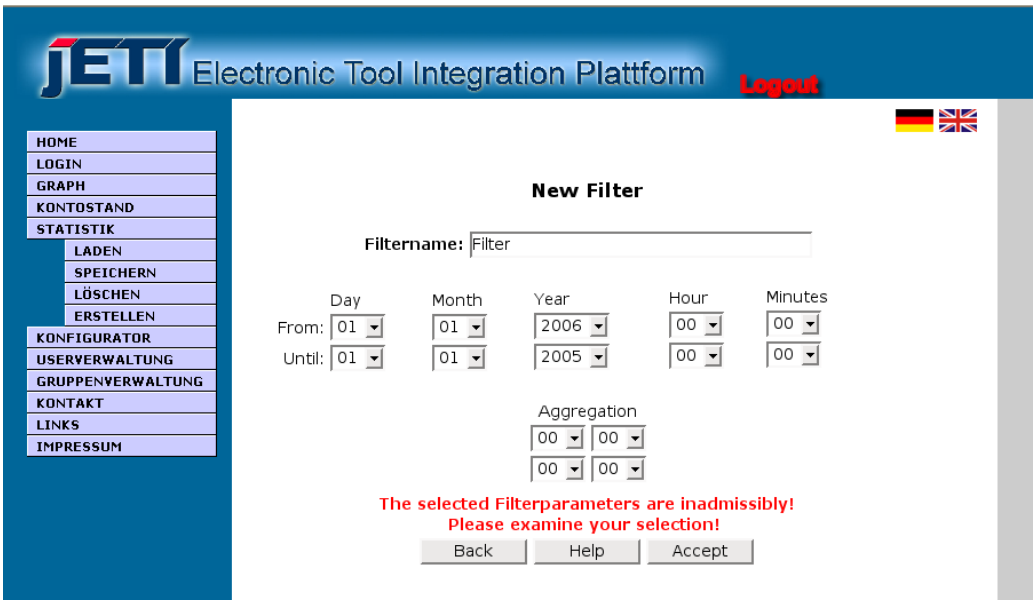

Abbildung 54: Die Umsetzung der `statisticFilter.jsp` Seite

aus den Strings der Auswahlfelder ein Datumsobjekt zu erzeugen. Nach den Überprüfungen und dem Umwandeln, werden die Filter innerhalb der Tabelle Filter gespeichert.

Filter löschen (Button **Delete**) funktioniert mit Hilfe des `statisticFilterServlet.java`, es löscht die entsprechenden Einträge in oben genannter Tabelle wieder.

5.3.4 Statistik speichern, laden & löschen

Wie vorher schon kurz beschrieben, kann die Statistik mit allen Parametern auch komplett abgespeichert werden. Dies geschieht über den Button `Save`. Durch das **statisticServlet.java** wird man zur `statisticSave.jsp` geleitet, wo man die Statistik abspeichern kann. Für das Speichern zuständig ist das `statisticSaveServlet.java` bzw. bei Kreis Diagrammen `StatisticCircleSaveServlet.java`. Dazu muss zuerst wieder ein Name eingetragen werden, der wie üblich 40 Zeichen lang sein darf und mit Hilfe der eben genannten Servlets überprüft wird. Es wird überprüft ob der Name bereits existiert und ob er nur aus ASCII Code besteht. Um die verschiedenen Arrays abzuspeichern wurden verschiedene `write` Methoden (z.B. writeTools) geschrieben, um die Arrays in Strings umzuwandeln, die dann in der Datenbank abgespeichert werden können. Dabei werden die verschiedenen Elemente zwar zu einem einzigen String zusammengefügt, aber nach jedem Element wird das Trennzeichen “;“ verwendet, damit man die einzelnen Elemente nachher wieder auseinander bekommt. Wenn der Speichervorgang erfolgreich war, bekommt der Benutzer eine Bestätigungsmeldung. Alle Attribute werden in der Datenbanktabelle StatistikSaveDB (siehe Seite 104 Abbildung 78) gespeichert.

Die gespeicherten Statistiken kann man mit dem Button **Load** innerhalb des Statistik Teils des Menüs wieder laden. Man gelangt auf die `statisticLoad.jsp` Seite (siehe Abbildung 56). Diese Seite liest mit Hilfe des CustomTags `showStatistics` alle für den User gespeicherten Daten aus der Datenbank aus und zeigt diese auf der Seite aufgelistet an. Diese werden mit dem gespeicherten Namen, dem Diagrammtyp und der gespeicherten Uhrzeit angezeigt. Für das Laden der Statistik ist das `statisticLoadServlet.java` Servlet zuständig. Es lädt die gespeicherten Datensätze aus der Datenbank und schreibt diese in die Session. Danach wird

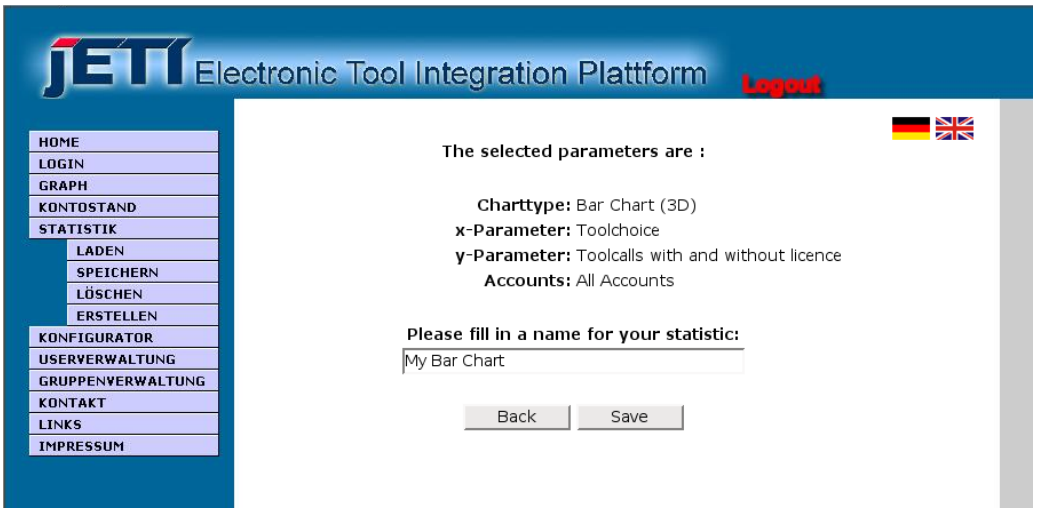

Abbildung 55: Eine Beispiel `statisticSave.jsp` Seite

festgestellt ob die geladenen Datensätze zu einem 2Achsen- oder einem Kreis-Diagramm gehören. Dann wird mit Hilfe des zugehörigen Servlets (`StatisticCircleGraphServlet.java` oder `StatisticAxisGraphServlet.java`) der Graph zu dieser Statistik neu erstellt und die `statisticGraph.jsp` Seite angezeigt.

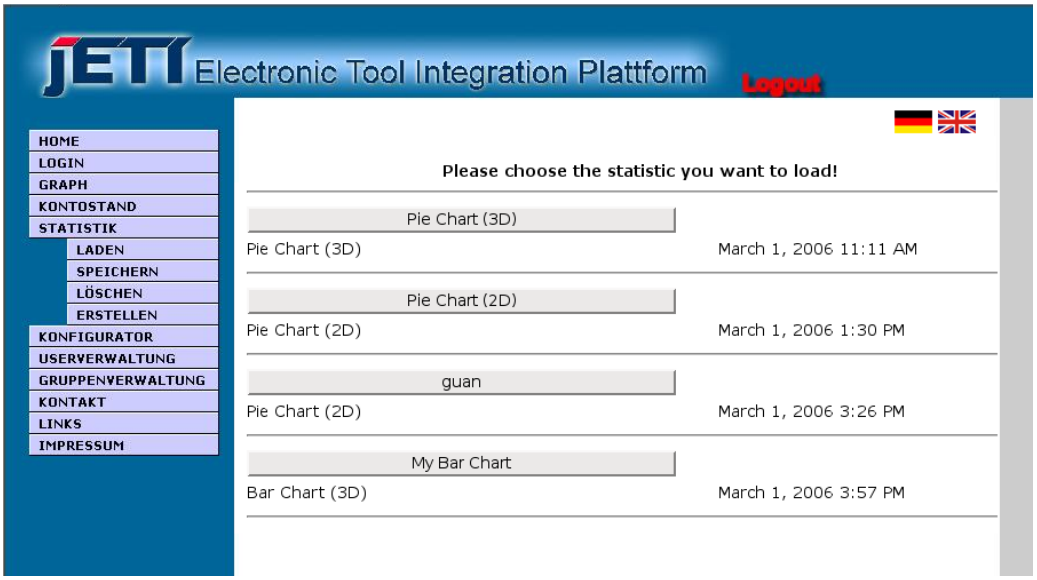

Abbildung 56: Eine Beispiel `statisticLoad.jsp` Seite

Gespeicherte Statistiken können natürlich auch gelöscht werden. Dies funktioniert über den Button **Delete** innerhalb des Statistik Teils des Menüs. Der Benutzer bekommt nun die `statisticDelete.jsp` Seite (siehe Abbildung 57) angezeigt. Diese ist wie die `statisticLoad.jsp` Seite aufgebaut. Das eigentliche Löschen geschieht mit dem Servlet `statisticDeleteServlet.java`. Es bekommt die Auswahl übergeben und sucht den entsprechenden Datensatz dann mit einer Query, wobei die genaue Auswahl nach dem WHERE spezifiziert ist, innerhalb der Datenbank und löscht diesen. Danach ist diese Statistik gelöscht und verschwindet auch aus der Auflistung die angezeigt wird.

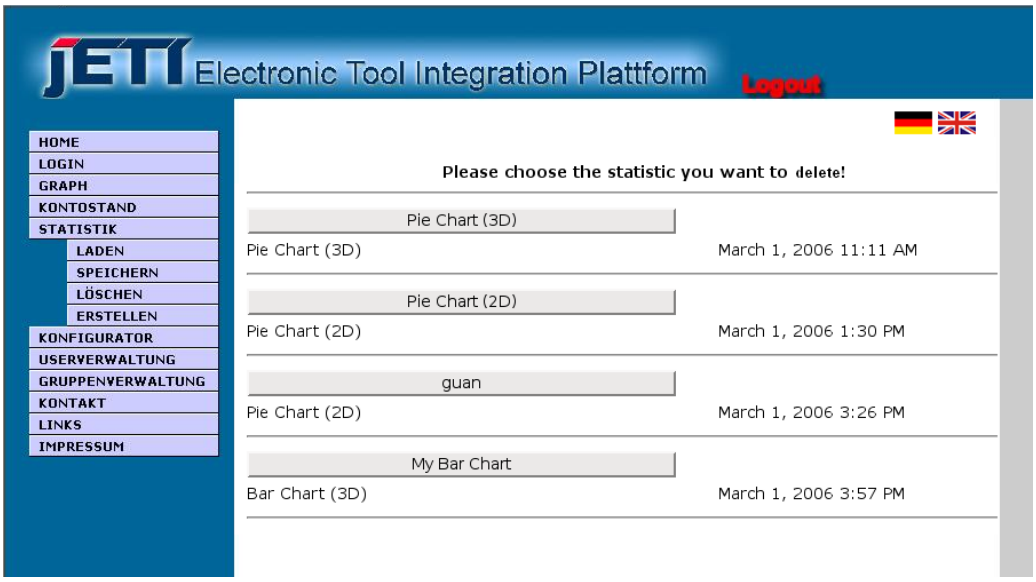

Abbildung 57: Eine Beispiel `statisticDelete.jsp` Seite

5.3.5 Zwangsfilter

Wie bereits in der Designphase beschrieben, dürfen Toolprovider in der Statistik Web Applikation nur Statistiken zu Ihren eigenen Tools sehen. Die Umsetzung des Designvorschlags kann man in Abbildung 58 sehen. Jeder Benutzer muss sich, damit er den vollen Umfang des jETI Systems benutzen kann, mit einem Login anmelden. Dieses Login und alle damit verbundenen Features werden auf jeder Seite mit den Methoden `CheckLogin` und `CheckFeatures` abgefragt.

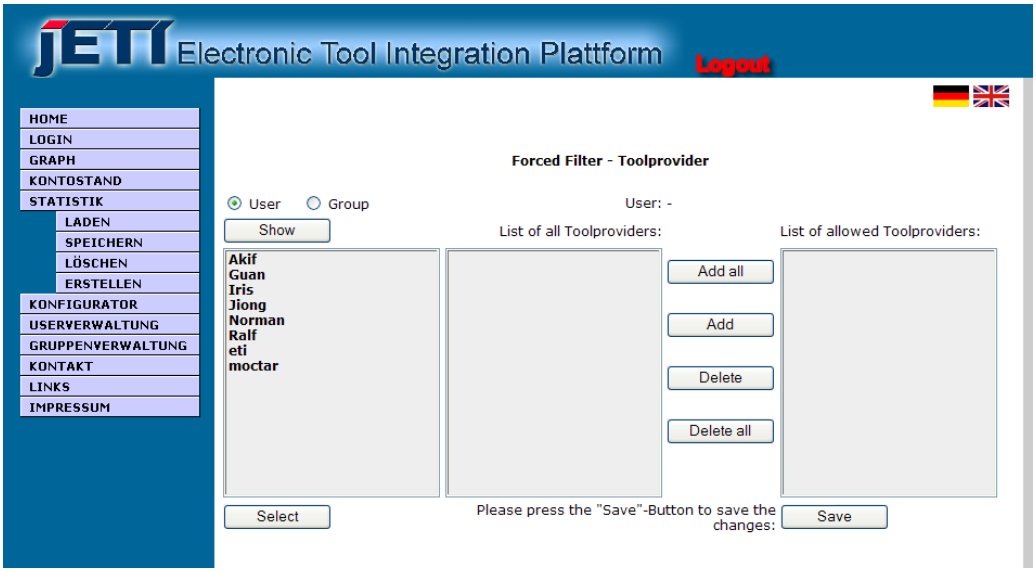

Abbildung 58: Beispiel für eine `statisticForcedFilterToolprovider.jsp` Seite

Auf der linken Seite kann man das Auswahlfeld für die Gruppen bzw. die User sehen. Hier kann der Administrator Gruppen bzw. User auswählen um bei Ihnen die Rechte zu ändern. Diese Liste wird mit dem CustomTag `ForcedFilterUserGroupList.java` dynamisch erzeugt. Das

CustomTag liest dafür alle User und Gruppen aus der Tabelle `Usrgroup` der Datenbank aus und zeigt diese an. Nach einer Auswahl (geschieht über **Select**) werden nun mit Hilfe des CustomTags `ForcedFilterUserGroupList.java` in der mittleren Spalte alle Toolprovider des Systems angezeigt. In der Spalte rechts wird mit dem CustomTag `ForcedFilterAllowedToolproviderList.java` die für den ausgewählten User bzw. der ausgewählten Gruppe eine Liste der erlaubten Toolprovider angezeigt. Mit Hilfe der Buttons zwischen der rechten und der mittleren Spalte, kann der Administrator die Auswahl ändern. Die Funktionalität der gesamten Seite wird innerhalb des `StatisticForcedFilterToolproviderServlet.java` Servlets geregelt. Es sorgt für sämtliche Funktionen und auch für das schreiben der Änderungen in die Datenbank. Auch hier wird die Auswahl zuerst in einen Vector geschrieben und erst nach der Bestätigung in ein Array umgewandelt.

5.4 Tests

Zum automatischen Testen der Statistik JSP Seiten und der Servlets wurden jUnit und Cactus benutzt. Da beides im Abschnitt 4.5.4 schon ausführlich beschrieben wurde, wird hier auf eine erneute Beschreibung verzichtet. Zum Testen der HTML Statistik wurden exemplarisch 17 verschiedene Test Klassen geschrieben. Mit Hilfe dieser 17 Testfälle decken wir den Großteil der geforderten Funktionalität ab. Dies sind im einzelnen:

1. TestSetLanguageServlet
   Hier wird getestet, ob der Pfad und die locale für die verschiedenen Sprachen richtig gesetzt werden.

2. TestStatisticAxisGraphServlet (Wenn page ungleich 3)
   In diesem Test werden 2 Methoden getestet, `doPost` und `saveChartintoSVGFile`. In der `saveChartintoSVGFile` wird getestet ob eine temporäre Datei für ein Diagramm erzeugt wird. In `doPost` wird getestet, ob *.svg und *.png Dateien erzeugt wurden und ob der Pfad richtig gesetzt wurde. Dabei ist die Vorgänger Seite nicht die `statisticLayout.jsp` Seite.

3. TestStatisticAxisGraphServlet1 (Wenn page gleich 3)
   In diesem Test werden wieder die 2 Methoden getestet. Allerdings ist die Vorgänger Seite diesmal die `statisticLayout.jsp` Seite. Zusätzlich wird noch überprüft ob die `testASCII` funktioniert.

4. TestStatisticCreateFilterServlet
   Dieser Test überprüft, ob die Filter überprüft wurden, der Filtername bereits existiert und ob der Name aus ASCII Code besteht. Nachdem der Filter erzeugt wurde, wird weiterhin überprüft ob die Parameter **startFilter**, **endFilter**, **startAgr**, **endAgr** und **filtername** nicht leer sind.

5. TestStatisticDeleteServlet
   Hier wird wieder sicher gestellt, dass der Pfad richtig gesetzt wurde.

6. TestStatisticFilterServlet
   In diesem Test wird überprüft, ob der Pfad richtig gesetzt wurde und ob Filter richtig miteinender verknüpft werden.

7. TestStatisticLoadServlet
   Überprüfung ob der Pfad richtig gesetzt wurde und das Attribut **filter** leer ist, da Filter nicht mit abgespeichert werden.

8. TestStatisticSaveServlet
   Dieser Test überprüft, ob der Pfad richtig gesetzt wurde, ob der Name der Statistik nur aus ASCIICode besteht, der Name bereits existiert, sowie ob die Statistik gespeichert wurde.

9. TestStatisticServlet0 - 3; 6 (Wenn page = 0 - 3; 6)
   Hier wird getestet ob alle Parameter die zur Weiterleitung benötigt werden erzeugt wurden.

10. TestStatisticToolChoiceServletAdd(Button = Add)
    Dieser Test überprüft den Pfad und ob die Tools richtig hinzugefügt werden.

11. TestStatisticToolChoiceServletDel(Button = Delete)
    Dieser Test überprüft den Pfad und ob die Tools richtig gelöscht werden.

12. TestStatisticToolChoiceServletBack(Button = Back)
    Überprüft ob der Pfad zur Vorgängerseite richtig gesetzt wurde.

13. TestStatisticForcedFilterToolproviderServletAdd(Button = Add)
    Hier wird überprüft ob der richtige Toolprovider hinzugefügt wird.

14. TestStatisticForcedFilterToolproviderServletSelect(Button = Select)
    Testet ob der Pfad richtig gesetzt wurde und ob die richtigen Toolprovider aufgelistet werden.

15. TestStatisticForcedFilterToolproviderServletShow(Button = Show)
    Überprüft ob die richtige Gruppen bzw. Userliste angezeigt wird.

16. TestStatisticCircleGraphServlet page = 4)
    In diesem Test werden 2 Methoden getestet, `doPost` und `saveChartintoSVGFile`. In der `saveChartintoSVGFile` wird getestet ob eine temporäre Datei für ein Diagramm erzeugt wird. In `doPost` wird getestet, ob *.svg und *.png Dateien erzeugt wurden und ob der Pfad richtig gesetzt wurde. Die Vorgänger Seite ist `statisticCircleStep1.jsp` Seite.

17. TestStatisticCircleGraphServlet1 (page = 5)
    Dieser Test prüft die gleichen Methoden wie im Testfall vorher, allerdings ist die Vorgängerseite `statisticCircleLayout.jsp`.

5.5 Ausblick

Die Applikation funktioniert, so wie es anfangs geplant war. Nichts desto trotz gibt es natürlich Verbesserungen, die man noch ins Programm einbauen könnte.

Die Statistik Web Applikation kann mit den 7 verschiedenen genannten Diagrammen arbeiten. Auch wenn diese Diagramme natürlich ausreichen um Statistiken für den Benutzer gut optisch aufzubereiten, wäre ein Ansatz für Verbesserungen, noch mehr Diagramme anzubieten. Dies würde die Auswahlmöglichkeiten des Benutzers noch weiter vergrößern.

Im Moment ist es so, dass Filter ausgewählt werden können und diese dann auch von der Applikation angewandt werden. Welche Filter angewandt werden steht immer unter dem Graphen. Wenn man das Bild nun allerdings im Bildformat *.png abspeichert, so fehlt auf dem Bild die Information welche Filter angewandt wurden. Dies sollte noch verbessert werden, da der Benutzer nach einer gewissen Zeit nicht mehr weiß mit welchen Filtern er die entsprechende Graphik erzeugt hatte.

6 HTML User Management/ABC Login

Um eine bessere Kontrolle des jETI Systems erzielen zu können, werden alle Benutzer/Gruppen von einem Administrator verwaltet.

6.1 Modellierungsphase

6.1.1 jABC Login - Modellierungsphase

von Ekoto, Armelle; Barzani, Hunervan

Um die Kommunikation zwischen Server und Client einzelner Benutzer herzustellen, wurde RMI (Remote Method Invocation) angewandt. RMI erlaubt den Zugriff auf entfernte Objekte. Für die Funktionalität von RMI meldet der Server das Objekt der Registry an. Darauf holt sich der Client die benötigten Referenzen und arbeitet quasi lokal auf dem Objekt.

6.1.2 Webseiten Flow bei der Benutzerverwaltung

von Paternoster, Iris; Kenzi, Hatim; Yu, Jiong

Um eine webbasierte Benutzerverwaltung zu erstellen, wurden zuerst die möglichen Administrationsaufgaben in einem Workflow dargestellt (Abbildung 59).

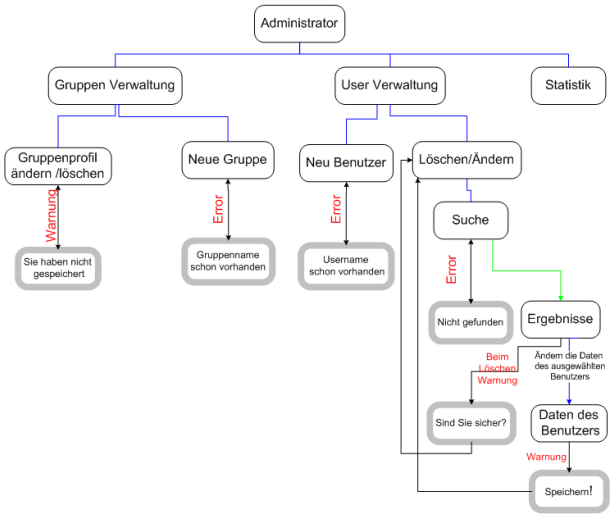

Abbildung 59: User- und Gruppen Verwaltung Flow

Der Administrator kann neue Gruppen erstellen und dazugehörige Gruppenrechte vergeben. Er kann auch neue Benutzer in das System einfügen. Die Rechte für die Benutzer werden nicht direkt zugeordnet. Sie erhalten die Rechte der jeweiligen Gruppe, zu der sie gehören. Es gibt auch die Möglichkeit, dass der Administrator Gruppen zu anderen Gruppen zuordnet.

Falls bereits eine Gruppe unter dem gewünschten Namen existiert, wird eine Fehlermeldung angezeigt und der Administrator auf die Seite “Neue Gruppe erstellen“ geleitet.

Falls er ohne die Seite zu speichern, “Gruppen Verwaltung“ aufruft, erhält er eine Warnmeldung. Es existiert aber auch die Möglichkeit, ohne Speicherung zur Gruppen Verwaltung zurückzukehren.

Der Administrator muss die Daten des Benutzers suchen, wenn er die Daten des Benutzers ändern oder den Benutzer im System löschen will. Bleibt diese Suche ergebnislos, wird eine Meldung angezeigt. Wenn mehrere Benutzer gefunden werden, werden sie als Ergebnisliste dargestellt. Der Administrator kann aus dieser Liste Benutzer auswählen, und die Befehle Löschen oder Ändern aufrufen. Beim Löschen wird er gewarnt, und beim Ändern werden die Daten des Benutzers angezeigt, wobei die Daten in änderbaren Textfeldern stehen.

Beim Login wird es einen Kontrollmechanismus geben. Dadurch wird die Anzahl der fehlgeschlagenen Logins gezählt. Wenn eine bestimmte Zahl erreicht ist, wird der Account für eine bestimmte Zeit blockiert (Abbildung 60).

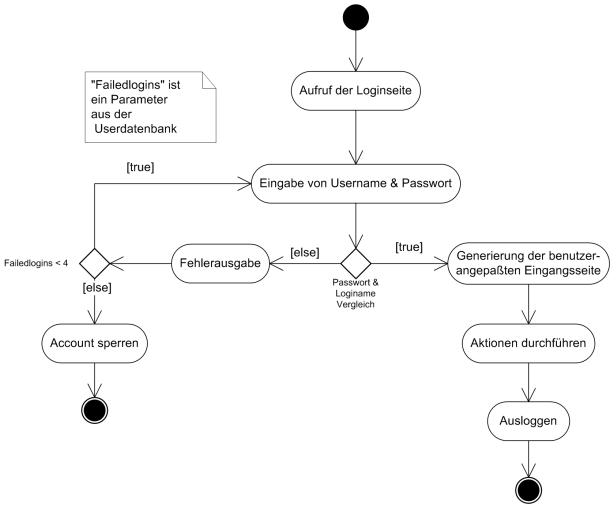

Abbildung 60: Login Workflow

6.2 Implementierungsphase

6.2.1 jABC Login - Implementierungsphase

von Ekoto, Armelle; Barzani, Hunervan
Das jETI System verfügt über eine Benutzerverwaltungdatenbank 6.2.2, wo alle Benutzer des Systems mit einem Benutzernamen und einem Kennwort identifiziert werden können. Da der Tool-Provider 7 wegen dem Lizenzkontrollsystem nur jETI-Benutzer akzeptiert, wurde eine Benutzerabfrage in den jABC Client eingebaut. Diese Benutzerabfrage wird durch RMI (Rmote Methode Invocation) ermöglicht. Es wurden ein RMI-Client und ein RMI-Server implementiert.

Der RMI-Client wird in das jETI Plugin eingebunden, während der RMI-Server auf dem Rechner gestartet wird, wo die Informationen über die Benutzerdatenbank zu finden sind. Vor dem Starten des Tracers 8 muss sich nun der Benutzer mit dem ETI-Plugin Login-Fenster erfolgreich anmelden, wenn das auszuführende Tool auf dem Executor vorhanden ist, denn der Executor erfordert nun von dem Client zusätzlich zu den Parametern des Tools den Anmelde-Namen des Benutzers. Ist das Tool auf dem Executor und der Benutzer des jABC Client erfolglos über das Login-Fenster angemeldet, wird der Tracer bei der Ausführung durch dem jETI Plugin gestoppt.

6.2.2 Userdatenbank

Paternoster, Iris; Kenzi, Hatim; Yu, Jiong

Beim erstmaligen Einrichten eines Users wird in der `UsrGroup`-Tabelle sein Loginname unter das Feld `NAME` gespeichert. Das Feld `ISUSR` wird auf true gesetzt. Durch dieses Feld kann man unterscheiden, ob es sich um einen User oder eine Gruppe handelt. Hier bekommt der User seine ID (PK). Mit dieser ID als Foreign Key werden seine komplette Daten in der Tabelle `usr` gespeichert. Zum Speichern dieser Daten stehen die in Abbildung 61 zu sehenden Felder zur Verfügung. Die Gruppen liegen ebenfalls in der Tabelle `usrgroup`. Hier wird der Gruppenname unter `NAME` gespeichert und `ISUSR` wird auf false gesetzt. Die Tabelle `GroupGrouprel` dient der Zuordnung eines Users zu einer Gruppe oder einer Gruppe zu einer anderen Gruppe. Die Tabelle `Feature` ist für das Speichern der Rechte gedacht. Die Tabelle `FeatureGrouprel` assoziiert Rechte zu Gruppen und damit zu User (Abbildung 61).

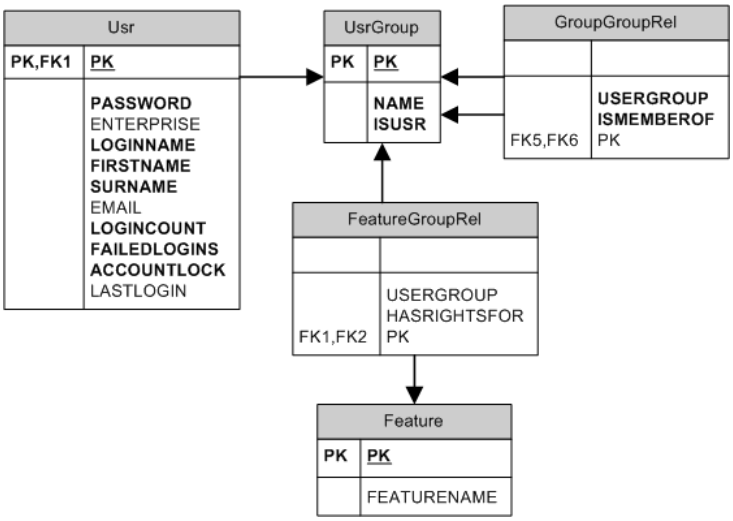

Abbildung 61: Datenbankschema der Benutzerverwaltung

7 Executor und Konfigurator

von Köse, Akif; Zaidane, Moctar; Metozounve, Fifame Pascale Judith

Dieses Kapitel behandelt den Tool-Executor und den Tool-Konfigurator, die Bestandteile des Tool-Providers sind. Mit dem Tool-Konfigurator kann der Tool-Anbieter des jETI-Systems ein Tool anmelden, die Konfiguration eines Tools bearbeiten oder das Tool komplett aus dem System löschen. Auf dem Tool-Executor werden die durch den Tool-Konfigurator bearbeiteten

Tools installiert. Diese Komponenten des Tool-Providers wurden von uns um einige Funktionalitäten erweitert.

Im folgenden wird der gesamte jETI Tool-Provider zuerst beschrieben, danach werden die durchgeführten Erweiterungen am Tool-Executor und Tool-Konfigurator vorgestellt.

7.1 Der jETI Tool-Provider

Der jETI Tool-Provider besteht aus einem Web Services basierten Adapter, der direkt mit jETI Plugin [Njoku(2005)] über SOAP [w3schools.com(), Skonnard()] kommuniziert. Dieser Adapter basiert auf dem Tomcat Servlet Container. Der Vorteil dabei ist, dass Tomcat ein kostenfreier Web-Server, der Servlets und JSP unterstützt [Tomcat(), Roßbach and Holubek(2002)]. Servlets und JSP (teilweise) werden in der plattformunabhängigen Sprache Java geschriebenen und können eine Web Applikation bilden. Das jETI System stellt dem Tomcat Servlet Container zwei verschiedene Web Applikationen, die Subsysteme des Tool-Providers bilden, zur Verfügung:

7.1.1 Tool-Konfigurator

Da das wesentliche Konzept des jETI Systems in der Vereinfachung der Integration der Tools besteht, stellt das jETI System dem Tool-Anbieter den Tool-Konfigurator zur Verfügung. Der Tool-Konfigurator bildet die erste Web-Anwendung, die in Tomcat Servlet Container läuft. Aus Sicht des Tool-Anbieter ist der Tool-Konfigurator ein Frontend, mit dessen Hilfe er seine Tools auf einfacher Weise in die jETI-Plattform integrieren kann. Er bietet dem Tool-Anbieter zur Anmeldung seines Tools ein Formular an. Die Felder des Formulars umfassen momentan nur einige Tool-Eigenschaften. Nachdem der Tool-Anbieter die vollständige Beschreibung des integrierenden Tools (erforderliche Eigenschaften), eingegeben hat, wird automatisch eine XML-basierte Konfigurationsdatei (Steuer-Datei), die auf dem Tool-Executor vorhanden ist, aktualisiert. Diese Steuerdatei enthält die eingegebenen zugelassenen Toolparameter. Falls die Eingabe eines Toolparameters mit dem Tool-Konfigurator nicht zugelassen ist, wird der Tool Anbieter aufgefordert, eine selbst geschriebene Steuerdatei aufzustellen. In diesem Zusammenhang ist der Tool Konfigurator als eine benutzerfreundliche Oberfläche zu sehen, mit dessen Hilfe Tools mit einfacher Weise beschrieben und verwaltet werden können. Eine der Aufgaben des Tool-Anbieters in dem jETI-System besteht also darin, für das integrierende Tool eine XML-basierte Beschreibung aufzustellen. Diese Beschreibung kann je Tool unterschiedlich sein und daher ist es notwendig den Tool-Konfigurator so anzupassen, dass viele Tools einfacher angemeldet werden können. In diesem Sinne wurde er erweitert, um z.B. die Lizenz-Optionen eines Tools zu ermöglichen. Diese Erweiterung wird in Abschnitt 7.3 genauer beschrieben.

7.1.2 Tool-Executor

Der Tool-Executor ist die zweite Web-Applikation, die im Tomcat Servlet Container lauffähig ist. Er stellt dem jETI-Client die jETI-Tools durch eine Steuer-Datei, die mittels des Tool-Konfigurators erstellt wurde, zum Ausführen zur Verfügung. Die Tool-Executor-Applikation kann lokal auf einem mit Internet verbundenen Rechner des Tool-Providers aufgerufen werden. Nachdem der Aufruf eines jETI-Clients erkannt wurde, kann der Executor aus den erhaltenen Parametern, die statisch oder dynamisch sein können, entweder durch eine Kommandozeile, die von dem Betriebssystem unterstützt wird oder durch eine Java-Klasse das gewünschte

jETI-Tool ausführen. Das Ziel der Gruppe war einige Erweiterungen auf dem Tool-Executor durchzuführen. Im Laufe der Projektgruppe wurde ein CORBA-Client implementiert, der bestehende Tools, die bereits über eine CORBA-Interface im jETI verfügen, ausführt. Die Aufrufe des jETI-Clients können nun sowohl erkannt werden als auch auf der Datenbank gespeichert werden. Diese Daten dienen der Statistik-Applikation (siehe Kapitel 5) als Grundlage für einfache Auswertungen. Der Tool-Executor verfügt jetzt auch über ein Lizenz-Kontrollsystem, der die Zugriffe auf die jETI-Tools anhand einer Lizenz-Datenbank überwacht und validiert. Eine Historie der Ausführungen wird in der Logging-Datenbank geschrieben. Die neuen Teile des Tool-Executors im Abschnitt 7.2 im einzelnen vorgestellt.

7.2 Erweiterungen des Tool-Executors

In den folgenden Unterabschnitten werden die an dem Tool-Executor durchgeführten Erweiterungen behandelt (siehe Abbildung 62).

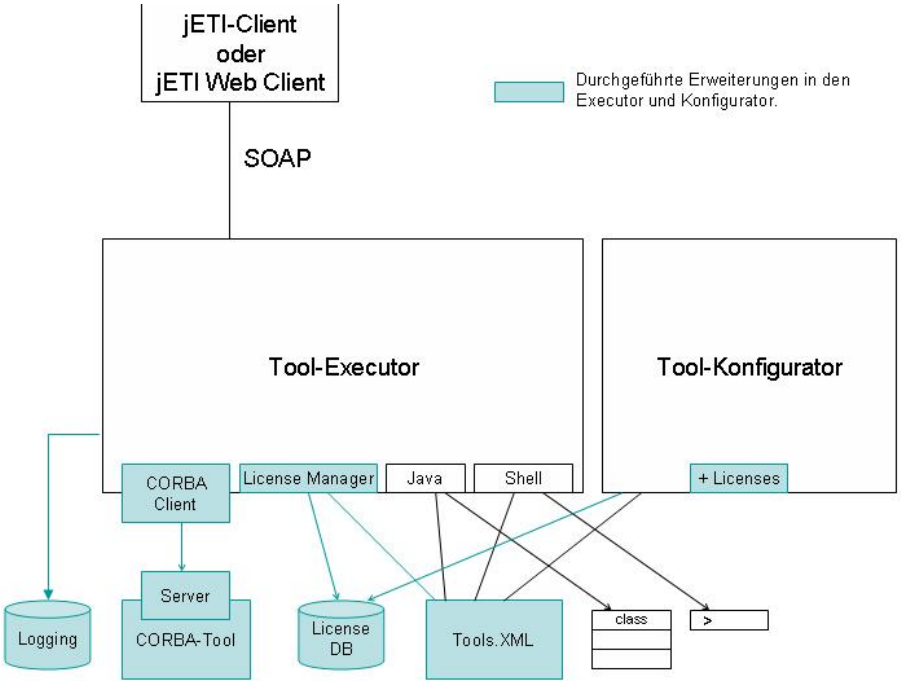

Abbildung 62: Aktueller Tool-Provider

7.2.1 Lizenz-Kontrollsystem

Die Grundidee des jETI Systems ist es, eine große Anzahl von verschiedenen Softwaretools einer breiten Masse von Benutzern mit geringem Aufwand für die einzelnen Benutzer verfügbar zu machen. Die Beschreibung aller Tools ist auf dem Tool-Executor verfügbar. Von den verfügbaren Tools gibt es Einige, für deren Ausführung eine Lizenz notwendig ist. Daher entsteht im jETI System die Notwendigkeit eines Lizenz-Kontrollsystems, das die Ausführung eines Tools überwacht. Durch das Lizenz-Kontrollsystem sieht der Tool-Anbieter einen Schutz für seine Tools vor unbefugten Benutzern. Damit wird auch das Vertrauen in das jETI System gestärkt.

7.2.1.1 Ablauf der Verifikation

In dem Lizenz-Kontrollsystem wird die Hauptaufgabe durch einen Lizenz-Manager übernommen. Der Lizenz-Manager hat die Aufgabe, die jETI-Tools, für die eine Lizenz notwendig ist, zu überwachen und die Aufrufe des jETI-Client zu validieren. Er steht zwischen dem jETI-Client und den jETI-Tools. Bei einer Verbindung zu dem Tool-Executor des jETI-Providers sendet der jETI-Client mit Hilfe des jETI Plugins den Namen des benötigten Tools und der SessionId, eine Nummer, mit der einen Verlauf einer Sitzung identifiziert wird. Zusätzlich zu diesen Paramatern wird den Anmelde-Name (userId) des Benutzers auch übergeben. Die Verbindung zum Tool-Executor wird über eine auf Webservices basierte Schnittstelle hergestellt. Nachdem der Executor die erforderlichen Parameter erhalten hat, wird geprüft, ob das Tool vorhanden ist. Falls ja, wird der Lizenz-Manager, der als Eingabe den Namen des Tools mit dem benötigten Namen des Lizenz-Moduls und dem Anmelde-Namen des Benutzers erhält, eingeschaltet. Der Name des Lizenz-Moduls ist durch die Steuer-Datei erhältlich. Der Lizenz-Manager sucht und validiert mit Hilfe der Lizenz-Datenbank diese Parameter. Falls er diese nicht für gültig erklären konnte, wird eine Fehlermeldung an den Client übermittelt. Im erfolgreichen Fall wird das Tool ausgeführt und das Ergebnis an dem jETI-Client geschickt und die Lizenz-Informationen werden dann aktualisiert. Dieser Ablauf ist in der Abbildung 63 zu sehen. Bei jedem Aufruf eines Tools wird eine Historie der Ausführung in die Logging-Datenbank gespeichert. Durch diese Historie wird ermöglicht, welcher Benutzer welches Tool mit welchem Lizenz-Modul wann und wo das Tool ausgeführt wurde, zu wissen. Im Falle, dass das Tool nicht vorhanden ist, wird der Lizenz-Manager überhaupt nicht eingeschaltet und dem Benutzer wird mitgeteilt, dass das Tool nicht aufrufbar ist.

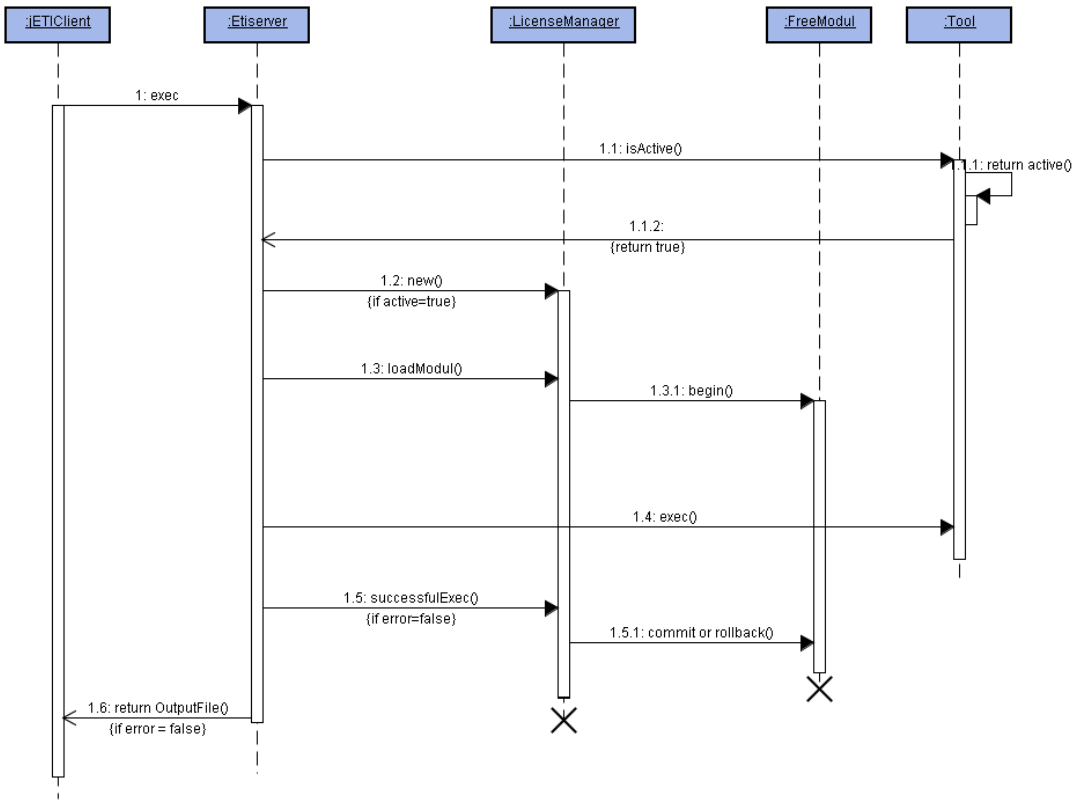

Abbildung 63: Sequenzdiagramm des Lizenz-Managers

7.2.1.2 Aufbau des Lizenz-Kontrollsystems

Das Lizenz-Kontrollsystem besteht aus einem Lizenz-Manager, einer Lizenz-Modul-Schnittstelle und Lizenz-Modulen. Der Lizenz-Manager verwaltet durch die Lizenz-Modul-Schnittstelle alle Lizenz-Module, denn sie wird von jedem Lizenz-Modul implementiert. Die Lizenz-Modul-Schnittstelle stellt drei Methoden, die Erste (`begin()`) validiert das Tool mit dem Benutzer, die Zweite (`commit()`) überschreibt die Informationen in der Lizenz-Datenbank nach erfolgreichem Aufruf eines Tools und die Dritte (`rollback()`) setzt die Veränderungen der Lizenz-Datenbank zurück, falls einen Aufruf fehlschlägt. Jedes Lizenz-Modul stellt eine bestimmte Anforderung, die der Benutzer erfüllen muss, um ein Tool auszuführen. Bevor der Lizenz-Manager einen Aufruf des jETI-Client zulässt oder ablehnt, wird das Lizenz-Modul gesucht, und das vom Lizenz-Manager gewählte Lizenz-Modul wird davon eine Instanz erzeugt. Das erzeugte Instanz des Lizenz-Moduls überprüft die Anforderung des Tools für den Benutzer anhand der von der Lizenz-Datenbank zur Verfügung gestellten Informationen. Der Lizenz-Manager lässt den Aufruf nur zu, wenn die durch das instanzierte Lizenz-Modul überprüfte Anforderung des Tools gegenüber dem Benutzer erfüllt ist. Wird die Anforderung erfüllt, so wird das Tool ausgeführt und nach erfolgreicher Ausführung wird durch das erzeugte Instanz des Lizenz-Moduls die Lizenz-Datenbank aktualisiert. Im Falle einer Ablehnung teilt der Lizenz-Manager dem jETI-Client mit, dass der Benutzer das Tool nicht ausführen darf. Dieser Aufbau besitzt viele Vorteile, z.B. die Multifunktionsfähigkeit des Lizenz-Managers und die Erweiterbarkeit des Systems um neue Lizenz-Module. In folgendem lassen sich die einzelnen Lizenz-Module beschreiben:

- LicenseModul:
  Dies ist ein Interface, welches die oben genannten Methoden, die für alle Lizenz-Module gleich sind, bereitstellt. Es wird von allen Lizenz-Modulen implementiert.
- Lizenz-Module:
  Es gibt zur Zeit folgende Lizenz-Module:
  - FreeModul: Dieses Modul macht keine Einschränkung der Aufrufe. Es kann benutzt werden, um ein Tool zu testen.
  - CallModul30: Es zählt die Aufrufe eines Tools. Die Anzahl der Aufrufe ist durch 30 beschränkt.
  - CallModul100: Es hat die gleiche Aufgabe wie CallModul30, der Unterschied ist, dass die Anzahl der Aufrufe eines Tools nicht 100 überschreiten darf.
  - TimeModul: Dieses Modul bestimmt in welcher Zeit und in welchen Tagen ein Tool ausgeführt werden kann.
  - DateModul: Es bestimmt die Anzahl der Tage und Monate für ein Tool
  - MoneyModul: Bei diesem Modul geht es um Geld, wie der Name schon andeutet. Bei jedem Aufruf wird von dem Guthaben des Benutzers die entsprechende Summe abgebucht.
  - DenyModul: Mit diesem Modul kann der Tool-Anbieter das Aufrufen seines Tools verweigert.

Diesen Lizenz-Modulen steht eine Lizenz-Tabelle zur Verfügung. Man hat sich während der Projektgruppe für eine Tabelle, die den Anmelde-Namen (`userId`) des Benutzers, den Namen des Tools, den Namen des Lizenz-Moduls und die Information für das Lizenz-Modul enthält, entschieden. Jeder Eintrag in der Tabelle ist durch einen automatischen generierten Primär-Schlüssel identifizierbar. Diese Tabelle wird durch die Abbildung 64 dargestellt.

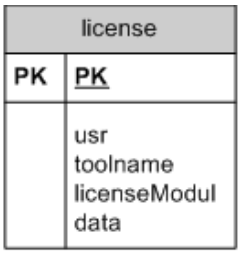

Abbildung 64: Lizenz-Tabelle

7.2.2 Logging der Tool-Ausführung

Die jETI Tools können auf verschiedenen Endpunkten verfügbar sein und von da aus ausgeführt werden. Um dem jETI-Admin einen besseren Überblick über die Ausführung der Tools zu verschaffen, wurde während der Projektarbeit das Tool-Logging im Tool-Executor eingebaut. Das Tool-Logging stellt eine Historie der Ausführungen der jETI-Tools in der Logging-Datenbank dar. Da der Tool-Executor die Beschreibung aller jETI-Tools besitzt und die vom Tool-Executer ausgeführt werden können, können alle benötigten Informationen durch das Tool-Logging in die Logging-Datenbank eingehen. In der aktuellen Logging-Datenbank werden Informationen über den Benutzer, über das aufgerufene Tool und über die Tool-Ausführungen gespeichert. Diese Informationen werden von der Statistik-Applikation ausgewertet und visuell als Reports dargestellt. Der in das jETI System eingeloggte Benutzer bekommt eine `userId`. Beim Ausführen eines Tools wird eine eindeutige SessionId generiert, zusammen mit der Ausführungszeit ist der jeweilige Benutzer eindeutig identifizierbar. Diese beiden Attribute bilden die primären Schlüssel der Tool-Logging Datenbank. Zusätzlich zu diesen beiden Attributen wird die `userId`, der Toolname, der Tool-Provider, Lizenz-Modul und die ausgeführte Aktion in die Datenbank eingespeichert (siehe Abbildung 65). Die ausgeführte Aktion kann eine Tool-Anmeldung, eine erfolgreiche Tool-Ausführung, eine nicht erfolgreiche Ausführung oder ein nicht vorhandenes Tool sein.

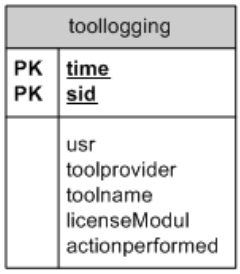

Abbildung 65: Tool-Logging Tabelle

7.2.3 Corba-Erweiterung

7.2.3.1 Einleitung

CORBA steht für “Common Object Request Broker Architecture“ und wurde im 1990 als Object Management Architecture (OMA) vorgestellt. Die CORBA-Architektur hat vier Komponenten, unter anderem der Object Request Broker, die Common Object Services, die Common

Horizontal and Vertical Facilities und die eigentlichen Application Objects. Die Zentrale Komponente von CORBA ist der Object Request Broker (ORB), welcher die Kommunikation zwischen Objekten zum Ziel hat. Die Interoperabilität basiert auf dem Internet Inter-ORB Protocol (IIOP), das einen Datenaustausch zwischen zwei Object Request Broker gewährleistet. Unter Interoperabilität wird die Fähigkeit verstanden, unterschiedliche Hard- und Softwarekomponenten miteinander kooperieren zu können.

- **Object Request Broker:** Die Hauptaufgabe vom Object Request Broker besteht darin, verteilte Objekte miteinander kommunizieren zu lassen. Auf Anwendungsebene soll kein unterschied zwischen lokalen und entfernten Methodenaufrufen feststellbar sein. Der ORB nimmt von Clients Aufträge zur Ausführung einer Operation entgegen, leitet sie zum Server zwecks Ausführung einer Operation weiter und veranlasst dort ihre Ausführung. Danach liefert er die Ergebnisse an den Client zurück. Durch diese Arbeitsweise des ORB's können sich Client und Server auf verschiedenen Rechnern mit verschiedenen Betriebssystemen befinden. Der ORB besitzt Mechanismen, die erforderlich sind, um den Server zu finden, den Client auf das Erhalten der Antwort vorzubereiten und die erforderliche Kommunikation auszuführen. Die Objekte können ferner in verschiedenen Programmiersprachen implementiert sein. Wie ein Operationsaufruf dem ORB zugeleitet wird, regelt das sogenannte "Language Mapping" der verwendeten Programmiersprache.

- **Die Interface Definition Language (IDL):** Es handelt sich um eine abstrakte, Programmiersprachen unabhängige Beschreibungssprache, die die an den Schnittstellen von Objekten sichtbare Funktionalität beschreibt und dabei Implementierungsdetails der eigentlichen Objekte verbirgt. Als eines der wichtigsten Ziele der CORBA-Architektur ist die Implementierung und die Zusammenarbeit der Objekte verschiedener Hersteller, welche in verschiedenen Programmiersprachen geschrieben werden können. Aus diesem Grund ist CORBA Sprachenunabhängig gehalten, nichts in der Architektur deutet auf eine bestimmte Implementierungssprache hin. Allerdings muss es eine Sprache geben, mit denen das Aussehen von Objekten beschrieben werden kann. Da CORBA ein von der Programmiersprache unabhängiger Standard ist, muss hier die Abbildung von IDL-Datentypen in Java spezifiziert werden.

- **IDL-Compiler:** Der IDL-Compiler ist ein wichtiger Bestandteil der CORBA-Spezifikation. Die Aufgabe dieses Compilers ist die Übersetzung der IDL-Schnittstellenbeschreibung in eine Programmiersprache.

7.2.3.2 Aufgabenbeschreibung

Ein Ziel der Projektgruppe war eine CORBA-Erweiterung im Tool Executor. Bis jetzt wurden die jETI Tools per Kommandozeile oder per Java Klasse aufgerufen. Die CORBA-Erweiterung ermöglicht auf einfachem Wege die Ausführung bestehender Tools, die bereits über eine CORBA-Schnittstelle verfügen, im ETI-Executor und damit in jETI. Es wurde dafür ein CORBA-Interface implementiert, das die Ausführung ermöglicht. Die Schnittstelle zwischen Tool-Executor und jETI Tools wird durch eine Client-Server CORBA-Applikation dargestellt. Nach der Ausführung werden die Rückgabe-Werte vom Tool-Executor interpretiert und dann an den jETI-Client weitergeleitet, wie bereits in Abbildung 62 zu sehen ist. Um das CORBA-Interface, das `ToolExecutorClient` genannt wurde, zu implementieren wurden die notwendigen Komponenten für eine CORBA-Applikation zuerst implementiert. Als Beispiel für diese Applikation wurde das Tool Povray benutzt. Aus dessen Namen entstanden `povray.Idl` für die IDL-Klasse

und `PovRayServer` für den Server. Povray ist das bekannteste Programm zum Erzeugen realistischer Grafiken und Animationen mit dem Computer. Das `ToolExecutorClient` kann mit jedem Tool funktionieren, die eine CORBA-Schnittstelle besitzen. Eine CORBA-Erweiterung ermöglicht Tools durch CORBA anzusteuern.

7.2.3.3 Implementierungsphase

IDL-Klasse:
Die IDL-Klasse dient nur zur Deklaration von Modulen, Interfaces und deren abstrakten Methoden. Sie hat ein Interface `povray`, das beim Kompilieren, die Hilfe-Klassen zur Implementierung der entsprechenden CORBA-Server erzeugt. Dieses Interface hat vier Methoden: `sayHello`, `greetPerson`, `setValue` um CORBA zu testen und `executePovRay`, die implementiert wurde um das Tool Povray zu starten, also um die CORBA-Applikation zu testen.

CORBA-Client:
`ToolExecutorClient` ist eine Java-Client-Klasse, die mit Hilfe eines CORBA-Servers beliebige enthaltende CORBA-Schnittstellen Tools aufrufen kann. Aus diesem Grund hat man ein `Dynamic Invocation Interface` (DII) benutzt. DII ist eine dynamische Schnittstelle zum ORB, die von CORBA als Alternative zu Stubs angeboten wird. DII ermöglicht eine Anwendung erst zur Laufzeit zu spezifizieren: von welchem Objekt eine Operation aufgerufen werden soll, welche Operation ausgeführt werden soll und welche Argumente die Operation hat. Mit Hilfe von sogenannten Request-Pseudo-Objekten kann eine beliebige Operation eines bis zur Laufzeit unbekannten Objekts aufgerufen werden. Der Aufruf einer beliebigen Operation mittels DII geschieht in folgenden Schritten:

- Erzeugen der Objekt-Referenz für das aufzurufende Objekt (z.B. über den Naming Service).
- Erzeugen eines Request-Pseudo-Objekts. Dabei ist mindestens das aufzurufende Objekt, der Name der auszuführenden Operation und die Liste der Argumente (Typ und Wert) anzugeben.
- Übergabe des Request-Pseudo-Objekts an den ORB (Anfang der Operation).
- Warten auf das Ergebnis der Operation (optional).
- Zerstören oder Wiederbenutzen der Request-Pseudo-Objekts. Für den Server ist es dabei nicht wichtig und auch nicht feststellbar, ob der Client den Operationsaufruf über einen Stub oder das DII veranlasst hat. Beide Sichten führen zur selben Repräsentation des Aufrufs im ORB.

Aufbau vom `ToolExecutorClient`

Für einen CORBA Aufruf braucht man Eine Object-Referenz, Den Methodenamen und die Parameter- Typen und Werte. Diese trägt man in der Konfigurations/Steuerdatei `Tool.xml` ein. Anstelle der Objekt Referenz kann auch ein CORBA-NameService Name Eingetragen werden.

Ein Server kann durch den Namendienst entfernte Objekte mit einen Namen anmelden und Clients können die entfernte Objekte unter einem Namen finden.

Ablauf: Der Corba-Client Verwendet die Objekt Referenz oder holt sich eine per NameService. Die Parameter werden in eine NVList gepackt (Liste der unterstützten Datentypen). Ein Platzhalter für das Ergebnis wird definiert. Aus Object Referenz, Methodename, Parameterliste wird ein Dynamischer Aufruf zusammengebaut und aufgerufen. CORBA serialisiert den Aufruf und schickt in an der Server. In der remote aufzurufende Methode, haben die Argumenten in der Liste von Argumenten die gleichen Eigenschaften wie die Eingabeparameter derselbe Methode in der IDL-Klasse, denn die Methode die remote aufgerufen wird ist die Methode vom Server. Der Request wird dann zum Aufruf der gewählten Methode erzeugt und mit `thisReq.invoke` ausgeführt; so können auch beliebige Methoden mit Hilfe vom `ToolExecutorClient` aufgerufen werden.

CORBA-Server

Zu Testzwecken und als Proof of Concept wurde auch eine CORBA-Server entwickelt. Neben ein paar einfachen Methoden lässt sich auch der ray Tracer Povray remote über CORBA aufrufen. Povray ist zwar ein open source produkt und auch auf den meisten Plattformen verfügbar aber in der Ideologie von ETI möchte man auch diese Software nicht unbedingt installieren müssen. Zudem verbrät Povray einen häufen Rechenzeit, so dass ein kostenpflichtiges Lizenzmodell für diesen Dienst sinnvoll erscheint. (Die Berechnung des Untergangs der Titanic war bestimmt nicht billig (anderer raytracer)). Es gibt in CORBA eine allgemeine Implementierung die dazu dient, ein Server zu initialisieren und zu starten. Diese wurde auch für den CORBA-Server gemacht. Die allgemeine CORBA-Server Implementierung besteht aus folgendem:

- Die Erzeugung und die Initialisierung des ORB's
- Die Erzeugung und Aktivierung einer Referenz zu den POA Manager
- Die Erzeugung und Registrierung eines "servant" (der "servant" oder Diener von dem durch den Server angebotenen Service) zum ORBs
- Die Erzeugung einer CORBA Object Referenz von dem "servant"
- Die Erzeugung einer CORBA Object Referenz von dem "naming context" und Anbindung der Referenz mit dem jetzigen Applikation.
- Das Starten des ORB's bzw. des Server.

Eine ausführliche Beschreibung der allgemeinen Implementierung eines CORBA-Server ist im Buch [Farley et al.(2002)Farley, Crawford, and Flanagan] gegeben. Ein CORBA-Server implementiert alle Methode, die in der IDL-Klasse deklariert wurden.

7.3 Erweiterung des Tool-Konfigurators

Der Konfigurator ist ein webbasiertes Tool zur Erstellung einer XML-Datei, der sogenannten `tools.xml`. Die tools.xml enthält Informationen über die einzelnen Tools wie z.B. Toolname, Eingabe- und Ausgabedatei usw. Nach der Generierung der `tools.xml` werden mit Hilfe des

SIB-Creators, ein Programm zur Erstellung von SIB's aus der `tools.xml`, die einzelnen Tools erstellt. Eine Teilaufgabe der PG war es den Konfigurator im Hinblick auf den Lizenz-Manager zu erweitern. Die Lizenzen, die durch den Lizenz-Manager überprüft werden, sollen bei der Eingabe der Toolinformationen im Konfigurator mit berücksichtigt werden. Die Lizenzen werden also von dem jeweiligen Toolprovider ausgewählt und mit in die `tools.xml` gespeichert.

Der Anfangs zur Verfügung stehende Konfigurator hatte keine Einstellungen bzgl. der Lizenzen. Die Auswahl der Lizenz-Module wird mit Hilfe einer Combobox realisiert, welche die zur Verfügung stehenden Lizenz-Module beinhaltet. Ferner mussten die Klassen, welche die Tool-Beschreibungen enthalten, um die zusätzlichen Attribute für die Lizenzen ergänzt werden. Diese Klassen werden bei der Generierung der XML-Datei benötigt. In der XML-Datei erscheint dann ein `<licence>` Tag, in welchem das Lizenzmodul für das jeweilige Tool angeführt wird. Zum Schluss wurde das Lesen aus der XML Datei implementiert, wodurch die Informationen aus der XML-Datei in die JSP Seiten übertragen werden können.

Bei der Erstellung eines Tools werden die SessionId, die Zeit, das erstellte Tool und der Toolprovider in die Datenbank eingeloggt, gleiches passiert beim Löschen des Tools. Die beiden Einträge unterscheiden sich nur in dem Eintrag beim `actionperformed`.

7.4 Ausblick

Nach der Durchführungen der oben beschriebenen Erweiterungen an dem Executor und Konfigurator sind diese beiden Bestandteile des Tool-Providers wieder mit mehr Funktionalitäten lauffähig. Trotz dieser Funktionalitäten können noch Verbesserungen durchgeführt werden. In dem Executor könnten noch Sicherheitsfunktionen eingebaut werden, so dass nur signierte Parameter von dem jETI-Client akzeptiert werden. Da das jETI System nun über eine Benutzerverwaltungsystem verfügt, könnte man der Tool-Konfigurator so zusammenfügen, dass der Tool-Anbieter nur über eine Anmeldung (Login) an dem jETI System den Tool-Konfigurator erreichen und somit nur seine Tools ansehen kann.

8 Parallel Tracer

von Ekoto, Armelle; Barzani, Hunervan

8.1 Motivation

Das jABC Client Framework ist für die Modellierung von Softwaresystemen zuständig. Graphen werden dabei anhand von SIB's erstellt. Dabei erfordert diese Modellierung keine Programmierkenntnisse. Um eine Ausführungsschicht für Modelle zu realisieren, wird das Tracer Plugin eingesetzt. Nach ihrer Realisierung werden die Modelle in den jABC Client hinzugefügt. Das Tracer Plugin interpretiert Graphmodelle und bestimmt das nächste auszuführende SIB. Zur Ausführung eines Graphen wird eine Tracer Konsole eingesetzt. Die Konsole besteht aus Buttons, die die Ausführungsschritte des Tracer Plugins bestimmen.

Bis jetzt war die Ausführung der Knoten und Kanten des Graphen sequenziell. Die Aufgabe der Gruppe bestand darin, den Tracer für Threads zu erweitern, um die einzelnen Threads kontrollieren zu können. Um die Parallelität des Tracers zu realisieren, sollten zwei spezielle Kompo-

nenten entwickelt werden, welche die parallele Ausführung erst ermöglichen. Die Fork Komponente soll dazu dienen, mehrere Graphsequenzen, die voneinander unabhängig sind, parallel abzuarbeiten. Die durch Fork initialisierten Threads werden schließlich durch die Join Komponente synchronisiert, also es wird gewartet, bis alle vom Fork ausgehenden Graphsequenzen beendet werden.

Zusätzlich zur parallelen Ausführung, wurde ein Mechanismus entwickelt, der das Erkennen von parallelisierbaren Sequenzen in Graphen ermöglicht.

Um dies zu erreichen, musste ein Algorithmus implementiert werden, der in einem Graphen SIB's auf Datenabhängigkeiten mit Hilfe von Parametervergleichen untersucht. SIB's, zwischen denen keine Abhängigkeiten existieren, können parallelisiert werden. Dazu werden die Graphen mit den entsprechenden Fork und Join Komponenten ergänzt und umgebaut.

Des weiteren wurde eine Funktion bzw. ein Algorithmus implementiert, der den Graphen auf Fehler bzw. undefinierte Konstruktionen untersucht und entsprechende Hinweise liefert. Diese Fehler wurden zuvor festgelegt und vor der Ausführung wird überprüft, ob ein Graph diese Fehler besitzt oder nicht. Es werden nicht beliebige Fehler untersucht, sondern vielmehr Fehler, die zu einer Komplexität von Graphenmodellen führen.

8.2 Modellierungsphase

In diesem Abschnitt werden einige Überlegungen, die den jABC Client erweitern, beschrieben:

- Die Implementierung von speziellen Thread-SIB's Fork und Join, um die parallele Ausführung durchführen zu können. Die Fork Komponente sollte die Parallelausführung und die Join Komponente die Synchronisation der Threads ermöglichen.
- Eine Tracer Erweiterung für die verschiedenen Threads, d.h. die Threads werden ab dem Fork SIB parallel und dynamisch ausgeführt. Dazu sollte die GUI auch entsprechend erweitert werden, d.h. die Buttons der einzelnen Threads sollten in einer JTable im Tracer-Fenster dynamisch verwaltet werden. Dabei soll das Fenster in etwa wie in der Abbildung 66 dargestellt, aussehen
- Die Erkennung von undefinierten Konstruktionen in Graphen. Ungünstige Testfälle mit Fork und Join SIB's sollten festgelegt werden. Bei der Graphenausführung müssen diese Testfälle gesucht und behandelt werden.
- Die Erkennung der parallelen Situationen. Um die parallelen Stellen zu erkennen, musste ein Algorithmus benutzt bzw. implementiert werden, der dies ermöglicht und zudem noch die parallelen Stellen entsprechend mit den entwickelten SIB's Fork und Join, automatisch umbaut.

8.2.1 Algorithmen Entwicklung

8.2.1.1 Parallele Verarbeitung

Ein Betriebssystem übernimmt die Vermittlungsrolle zwischen den unterschiedlichen Programmen und Prozessen, die sich ihrerseits gegenseitig beeinflussen. Es gibt zwei Arten von Abläufen:

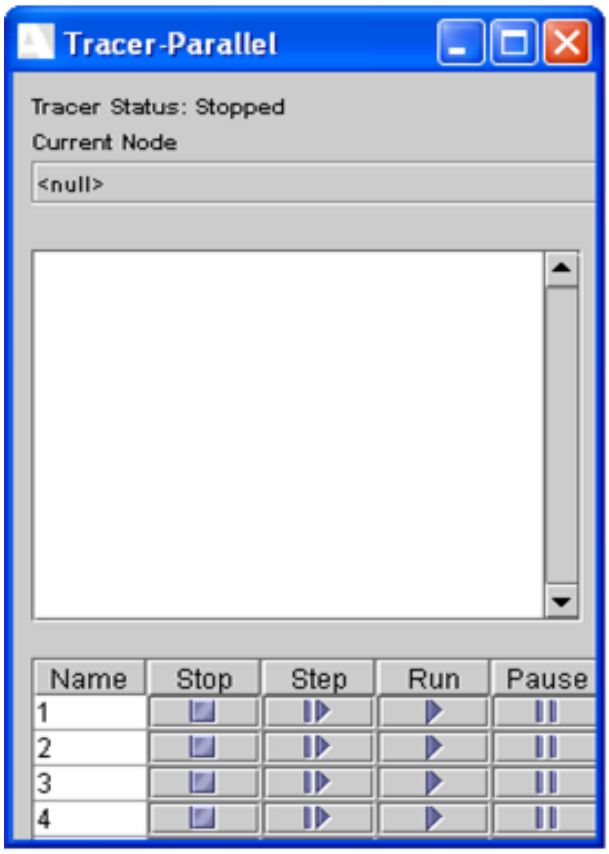

Abbildung 66: Das Tracer-Fenster

Unabhängige Abläufe: Prozesse, die parallel ausgeführt werden und sich gegenseitig nicht beeinflussen, d.h. die für ihre Ausführung unterschiedliche Systemvorrausetzungen nutzen.

Abhängige Abläufe: Prozesse, die sich gegenseitig beeinflussen und voneinander abhängig sind, weil sie miteinander was gemeinsam haben (z.B. gemeinsame Nutzung eines Betriebsmittels oder Speicherbereichs). Parallele Prozesse sind auch in verteilten Systemen, z.B. Netzwerken möglich, sie können sowohl in einem Rechner mit einem Prozessor entstehen, als auch in mehreren Rechnern mit mehreren Prozessoren.

Abhängig vom System gibt es folgende Formen von Parallelität:

- Quasi-Parallelität: Parallele Abläufe sind quasi-parallel, wenn ein Prozessor mehrere Prozesse gleichzeitig, aber in kleinen Stücken bearbeitet.
- Parallelität: Parallele Abläufe sind (echt) parallel, wenn mehrere Prozessoren gleichzeitig ein oder mehrere Prozesse bearbeiten.
- Nebenläufigkeit: Mehrere Abläufe, die parallel oder quasi-parallel ausgeführt werden, heißen nebenläufig, wenn sie sich gegenseitig beeinflussen und voneinander abhängig sind, d.h.: Nebenläufigkeit = Parallelität + Abhängigkeit.

Um die Verwaltung jedes einzelnen parallelen Prozesses muss man sich erst dann kümmern, wenn es sich um voneinander abhängige, d.h. nebenläufige Prozesse handelt. Dann muss man sich um die Koordinierung von solchen Prozessen kümmern. Für die Koordination von Prozessen gibt es zwei Möglichkeiten:

- Synchrones Ablaufverhalten: Die Abläufe sind getaktet, jeder Prozess kommt zu einem bestimmten Zeitpunkt an einer bestimmte Stelle seiner Ausführung an.

- Asynchrones Ablaufverhalten: Die Abläufe sind ungeregelt, die Koordination wird gewährleistet, wenn ein bestimmtes Ereignis erzielt wird (z.B. kommt ein Signal(Piepton) - wird eine Variable initialisiert).

8.2.1.2 Parallele Ausführung

Der Parallel Tracer realisiert eine Parallelverarbeitung von mehreren Threads durch Fork SIB's. Das Fork SIB wird in einer Klasse Fork implementiert, die von der Klasse SIB erbt. Sie besitzt eine Methode `GetOutGoingBranches()`, die die Anzahl der Kanten liefert, die von einem SIB ausgehen.

Die Ausführung von Threads führt aber zu einem Laufzeitproblem in der Parallelität. Da verschiedene Threads unterschiedliche Laufzeiten haben können, kommt es zu Verzögerungen. Durch diese Verzögerung ist die Synchronität wiederhergestellt. Join SIB's ermöglichen deren Synchronisation, indem ein Thread auf die anderen Threads wartet, bis alle abgearbeitet sind. Das Join SIB kann mehrere eingehende Kanten aber nur eine ausgehende Kante, wobei jede Kante für einen bestimmten Thread steht. Die ausgehende Kante wird durchgelaufen, wenn alle eingehende Kanten durchgelaufen sind. Dafür musste erkannt werden, wie viele Prozesse ausgehend von dem Fork SIB gestartet werden. Dafür wurde eine Datenverwaltung entwickelt. Die Kante, die zum Fork SIB führt, sollte gelöscht bzw. aus der Verwaltung entfernt werden und die ausgehende Kanten aus dem Fork SIB hinzugefügt werden(siehe Abbildung 67)

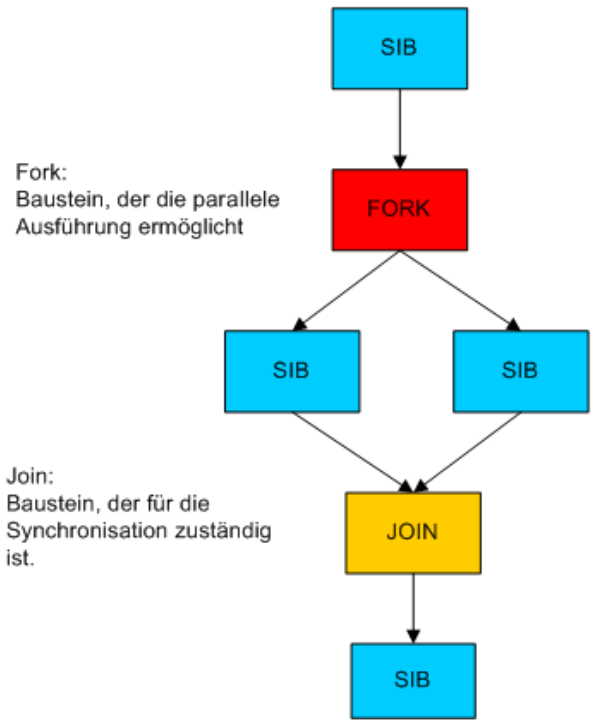

Abbildung 67: Ein Beispiel für parallele SIB's

Das Join SIB wartet, bis alle Prozesse angekommen sind. Soweit dies erfolgt ist, werden die vorherigen Prozesse aus der Verwaltung entfernt und ein neuer Thread gestartet, dessen Prozess

in der Verwaltung gespeichert wird. In der Abbildung 67 sieht man eine typische Positionierung im Graphen für Fork und Join.

Nach der parallelen Verarbeitung musste ein Algorithmus entworfen werden, der die Erkennung von geeigneten Graphsituationen für die Parallelisierung ermöglicht.

8.2.1.3 Parallelisierungsalgorithmus

Zunächst musste man sich überlegen, welche Informationen man braucht, um zu einer parallelen Ausführung zu gelangen. Als erstes stellt sich die Frage, wann zwei SIB's parallel ausführbar sind. Dazu sind drei Fälle zu betrachten:

- wenn keine Datenabhängigkeiten existieren
- wenn sie schon in parallelen Strängen ablaufen
- oder wenn sie in einanderausschließende Fälle ablaufen, wie bei IF-SIB's

Um dann herauszufinden, wie man zwei SIB's parallelisieren kann, sucht man nach Datenabhängigkeiten.

Man spricht von einer Datenabhängigkeit oder (read after write)-Abhängigkeit, wenn ein SIB Daten bereitstellt, die von einem anderen SIB gebraucht werden. Das heißt, dass ein SIB auf ein anderes SIB warten muss, das die benötigten Daten zur Verfügung stellt. Antidatenabhängigkeit oder (Write after read)-Abhängigkeit, liegt vor, wenn der Output von einem SIB gleich dem Input von einem anderen SIB ist. Die Datei muss noch vor dem geändertem Inhalt gelesen werden. Das ist der sequentielle Aspekt und darum muss die Reihenfolge eingehalten werden. D.h. an dieser Stelle kann nicht parallelisiert werden. Ausgabeabhängigkeit oder (write after write)-Abhängigkeit besteht, falls zwei SIB's in die selbe output-Datei schreiben. Da die Parallelisierung auch eine Änderung der Reihenfolge nach sich ziehen kann und dass dann am Ende durch die falsche Reihenfolge der SIB's auch die falsche Information in der Ausgabedatei stehen könnte, wird hier auch die Ausgabeabhängigkeit berücksichtigt. Zur Erklärung: Durch die Ausgabeabhängigkeit wird verhindert, dass zwei SIB's gleichzeitig in dieselbe Datei schreiben. D.h. die SIB's werden nicht parallelisiert, bleiben in der Sequenz und werden nacheinander ausgeführt.

Zunächst betrachtet man eine einfache Sequenz. Man baut einen Hilfsgraphen, der für jedes SIB einen Knoten enthält. Und jetzt vergleicht man die Ein- und Ausgaben jedes SIB's mit Ein- und Ausgaben jedes anderen SIB's, um die Datenabhängigkeiten, Antidatenabhängigkeiten und Ausgabeabhängigkeiten herauszufinden.

Die Ergebnisse der Abhängigkeitsbeziehungen zwischen den SIB's werden im Hilfsgraphen durch eine Kante gezeigt. Diese wird dann zwischen den zwei Knoten gezogen, die die jeweiligen SIB's repräsentieren. Danach bilden die Knoten jetzt eine Zusammenhangskomponente, die nicht ohne weiteres parallelisiert wird. Eine Zusammenhangskomponente ist ein Graph bestehend aus einer Menge von Knoten und ungerichteten Katnten. Jeder Knoten repräsentiert jeweils ein SIB und zwischen zwei Knoten wird eine Kante gezogen, wenn es eine Datenabhängigkeit zwischen diesen beiden SIB's gibt.

Um mit diesem Hilfsgraphen die maximale Parallelisierung dieser Sequenz ermitteln zu können, müssen folgende Schritte unternommen werden:

- Ermittlung aller Zusammenhangskomponenten, durch eine Tiefensuche auf den Hilfsgraphen. Nach jeder Tiefensuche ergibt die Anzahl der gefundenen Zusammenhangskomponenten die Anzahl der Kanten vom Fork SIB, die im umgebauten Graphen dann benutzt werden.
- Falls es mehr als eine Zusammenhangskomponente gibt, dann wird ein Fork und ein Join SIB in den Zielgraphen hinzugefügt. Der Zielgraph ist der modifizierte Originalgraph (d.h. der Originalgraph wird mit den entsprechenden SIB's wie Fork und Join ergänzt, die die Parallelisierung ermöglichen). Für jede dieser Zusammenhangskomponenten wird eine Kante vom Fork zum Knoten mit der größten Erreichbarkeitsanzahl gezogen, der zuvor aus der Zusammenhangskomponente gelöscht und in den Zielgraphen hinzugefügt wurde. Jetzt wird rekursiv für jede zerschnittene Zusammenhangskomponente dieser Algorithmus aufgerufen. Der Algorithmus liefert das letzte Element der umgebauten Subsequenzen zurück, das dann entsprechend mit dem Join SIB verbunden wird, auf dieser Ebene im Zielgraphen.
- Falls es nur eine Zusammenhangskomponente gibt, dann wird der Knoten mit der größten Anzahl der Knoten, die von ihm aus in der Originalsequenz erreichbar sind, gelöscht. Dies wiederum wurde mit der Tiefensuche im Originalgraphen bestimmt. Dieser gelöschte Knoten wird in den Zielgraphen eingefügt. Dann wird der Algorithmus rekursiv für den Rest der gerade zerschnittenen Zusammenhangskomponente aufgerufen, d.h. auf die Komponente ohne den gelöschten Knoten.
- Abbruchbedingung der Rekursion: Wenn der übergebene Teilgraph einelementig ist, dann wird dieses Element in den Zielgraphen eingefügt und an die aufrufende Rekursionsebene zurückgegeben.

In der Abbildung 68 ist ein sequentieller Graph mit dem entsprechenden Hilfsgraph zu sehen und dazu sind auch die einzelnen Schritte der Zusammenhangskomponentenbildung eingezeichnet. Zum Schluss hat man nur noch eine einelementige Zusammenhangskomponente und das ist die Abbruchbedingung für die Rekursion, wie oben schon erklärt wurde.

Als nächstes soll die Abbildung 69 einen sequentiellen Graphen darstellen mit den Parametern vom ersten SIB.

In der Abbildung 70 sind die Parameter der einzelnen SIB's zu sehen.

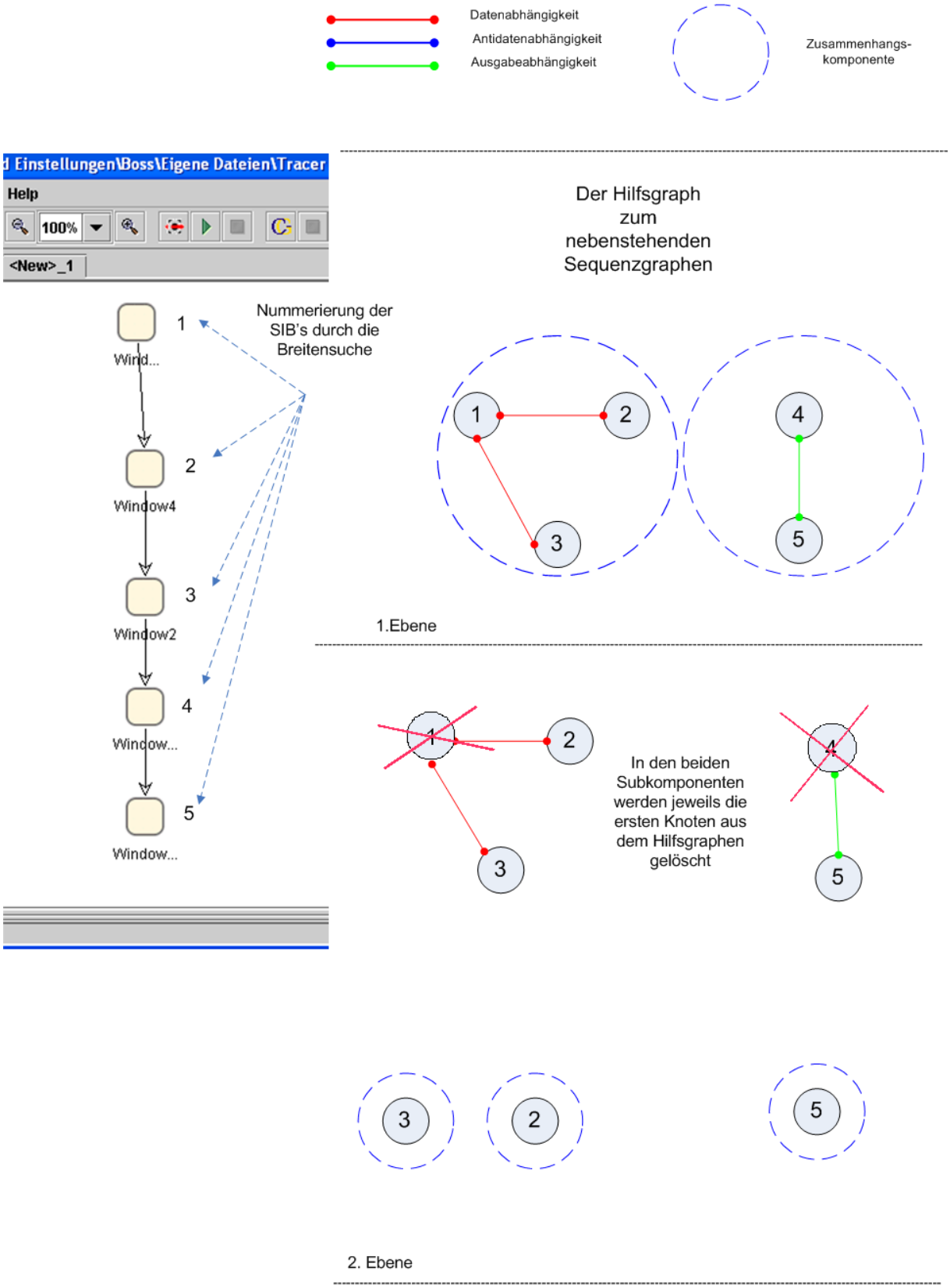

Abbildung 68: Hilfsgraph zu dem gegebenen Sequenzgraphen

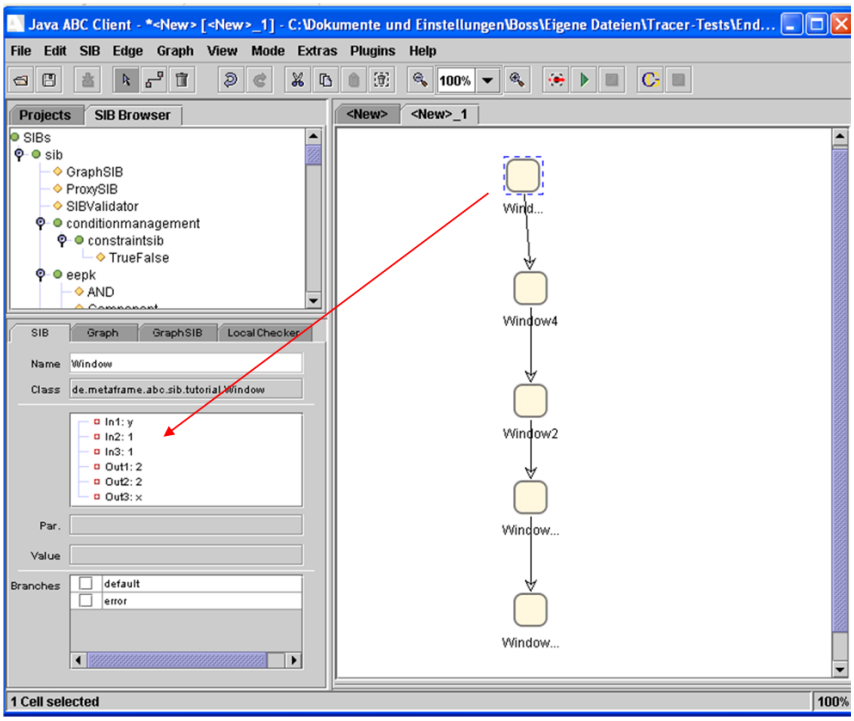

Abbildung 69: Ein sequentieller Graph

In der Abbildung 71 ist ein sequenzieller Graph vor und nach dem Umbau zu sehen.

Nun, nachdem der Algorithmus für Sequentielle Graphen erläutert wurde, wird als nächstes ein Graph betrachtet, der nicht sequenziell ist. D.h. es wurde überlegt, was man macht, wenn man nicht sequentielle Graphen hat, sondern schon alternative Sequenzen. Als erstes dürfen nur die Abhängigkeiten zwischen SIB's überprüft werden, die auch voneinander erreichbar sind. Dafür muss schon am Anfang eine Liste erstellt werden, in der gespeichert wird, welcher Knoten welche Knoten im gerichteten Originalgraphen erreicht, damit man nur SIB's auf Datenabhängigkeiten untersucht, die auch wirklich im Bezug zueinander stehen.

Auf der anderen Seite müssen Kontrollflussabhängigkeiten berücksichtigt werden, so dass direkte Nachfolger der verzweigenden Knoten auf jeden Fall auf die Entscheidung in den verzweigenden Knoten warten und damit nicht von den Parametern, sondern vom Kontrollfluss abhängen.

Außer dieser Änderung muss der rekursive Algorithmus auf diese Situation mit den verzweigenden SIB's angepasst werden. Das bedeutet, wenn der Algorithmus einen verzweigenden Knoten aus einer Zusammenhangskomponente entfernt, müssen folgende Schritte unternommen werden:

- maximale unabhängige Sequenzen finden
- ausgehende Kanten der letzten Elemente der Sequenz sichern
- rekursiven Aufruf des Algorithmus mit jeder maximalen Sequenz als Subkomponente durchführen
- die Restsequenz mit den gemeinsamen Elementen werden nach dem ursprünglichen Algorithmus weiter betrachtet

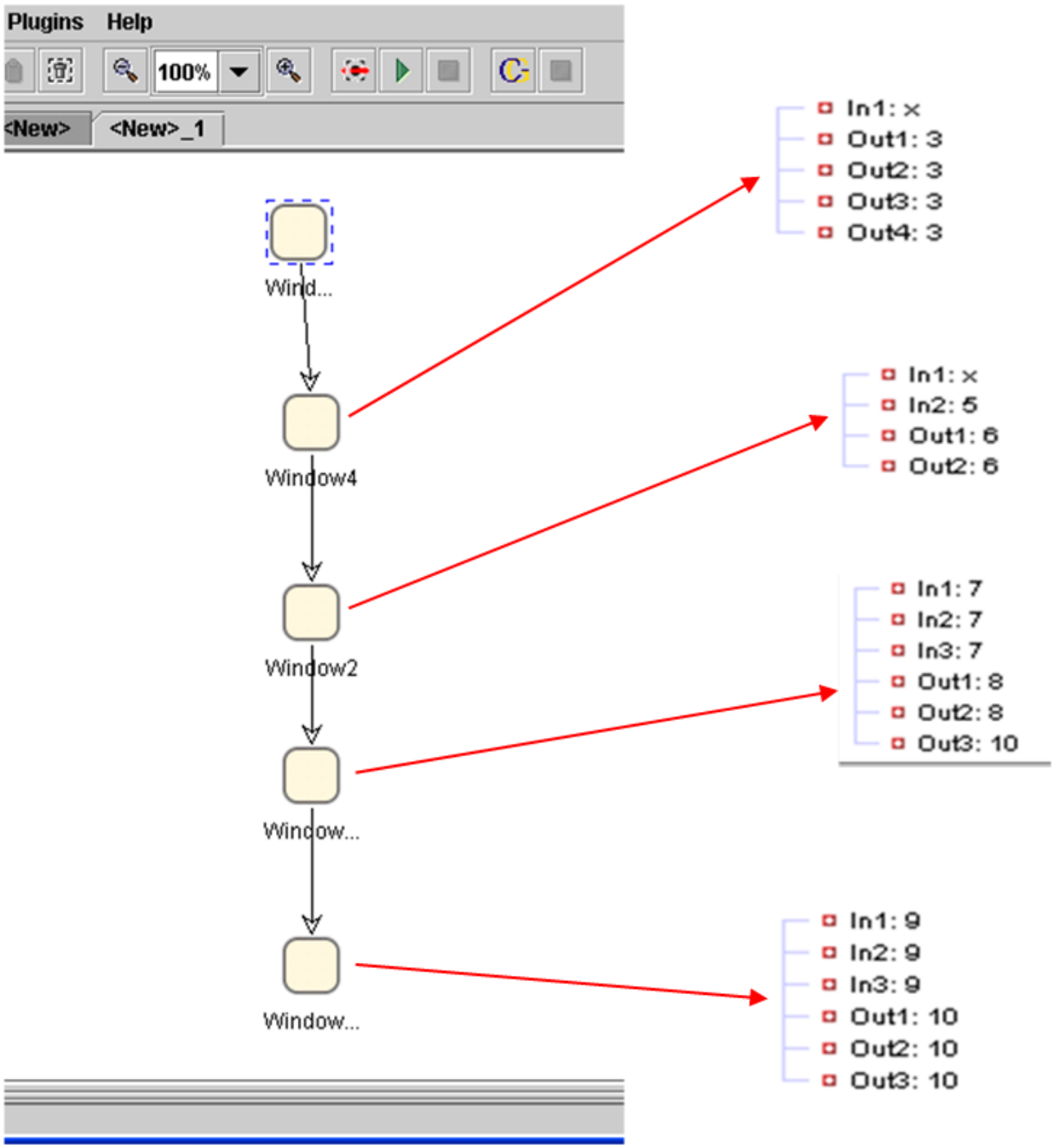

Abbildung 70: Ein sequentieller Graph mit verschiedener Anzahl von Parmatern

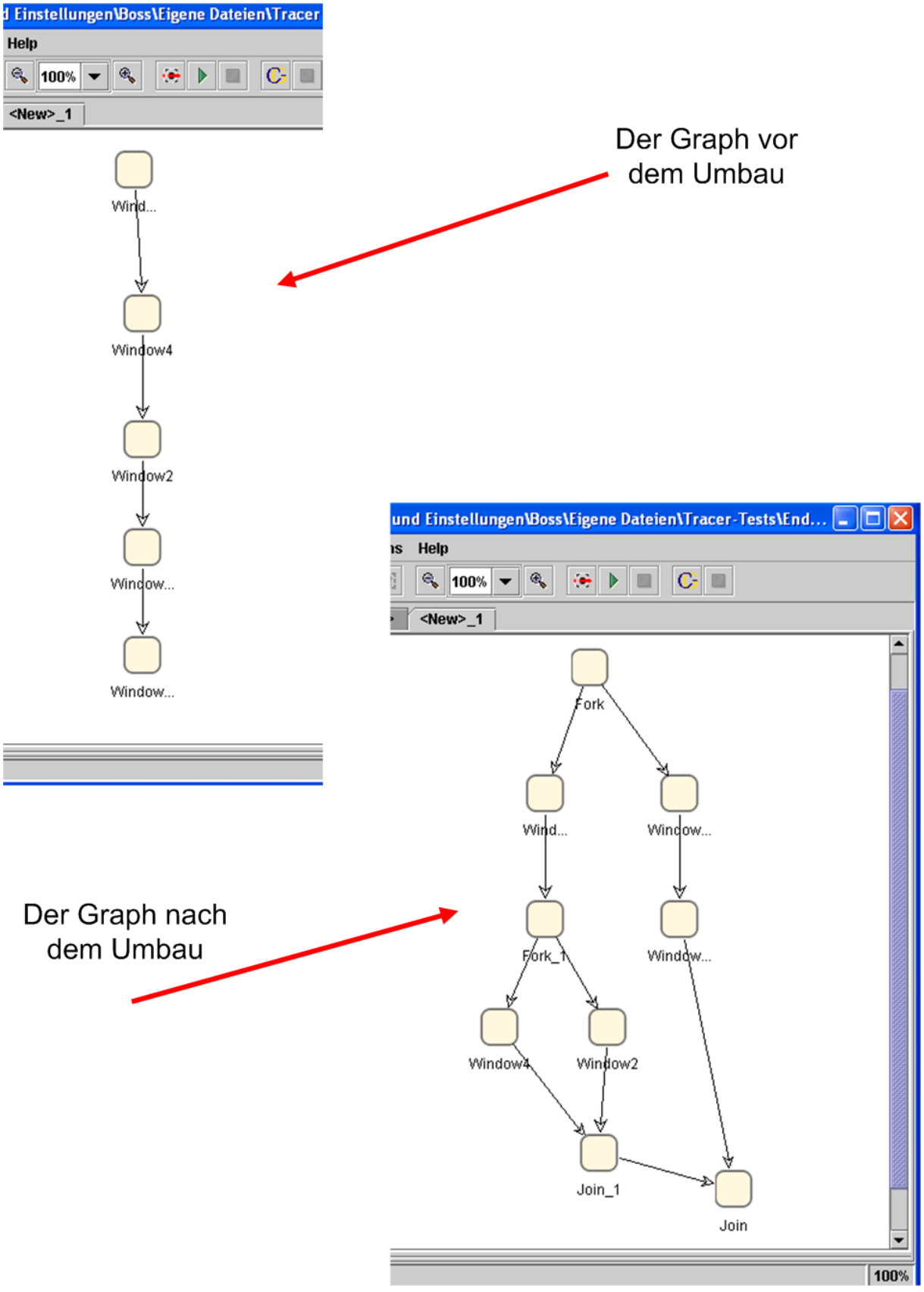

Abbildung 71: Der Sequentielle Graph nach dem Umbau

Um die maximalen Sequenzen nach einem IF bzw. Fork SIB zu finden, betrachtet man zunächst alle Knoten bzw. SIB's, die direkt vom IF bzw. Fork SIB aus erreichbar sind. Dies sind in der Abbildung 72 die Knoten bzw. SIB's 1,2 und 3.

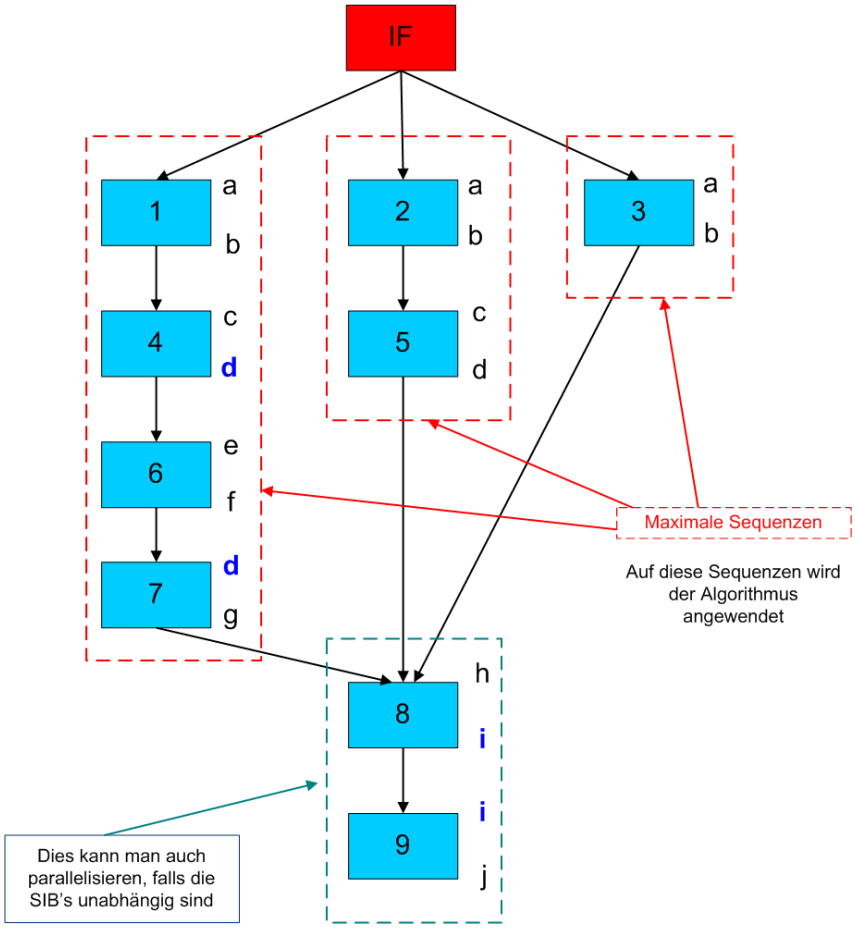

Abbildung 72: Graph mit Verzweigung

Jedes dieser SIB's ist jeweils das erste Element in der jeweiligen Liste. Für jeden Strang nach dem IF SIB wird eine Liste erzeugt, in die, die einzelnen Knoten hinzugefügt werden. Die Elemente dieser Listen werden dann auf Abhängigkeiten untersucht, so dass eventuelle Fork SIB's eingebaut werden können. Für jede Liste muss dann ein neuer Hilfsgraph erzeugt werden, da sonst der Originalgraph verfälscht werden würde.

Nach Anwendung des Algorithmus auf die einzelnen Sequenzen erhält man einen neuen Graphen, wie in der (Abbildgung 73) auf der rechten Seite zu erkennen ist.

8.2.1.4 Erkennung zu vermeidender Graph-Konstruktionen

Der nächste Schritt war, Graphen auf zu vermeidende Konstruktionen mit Fork und Join SIB's zu untersuchen. Zunächst musste man sich überlegen, welche Fälle mit Fork und Join SIB's zum Testen sinnvoll sind. Wir hatten überlegt, dass die Platzierung von zwei Fork oder Join SIB's hintereinander nicht erlaubt ist, da dies die Synchronisation nicht vereinfacht. Ein anderer Fall war, dass nur Fork oder Join SIB's in dem Graph vorhanden waren, d.h ein Fork SIB hätte zum Beispiel nur ein anderes Fork SIB als Nachfolger. Außerdem wollten wir festlegen, dass

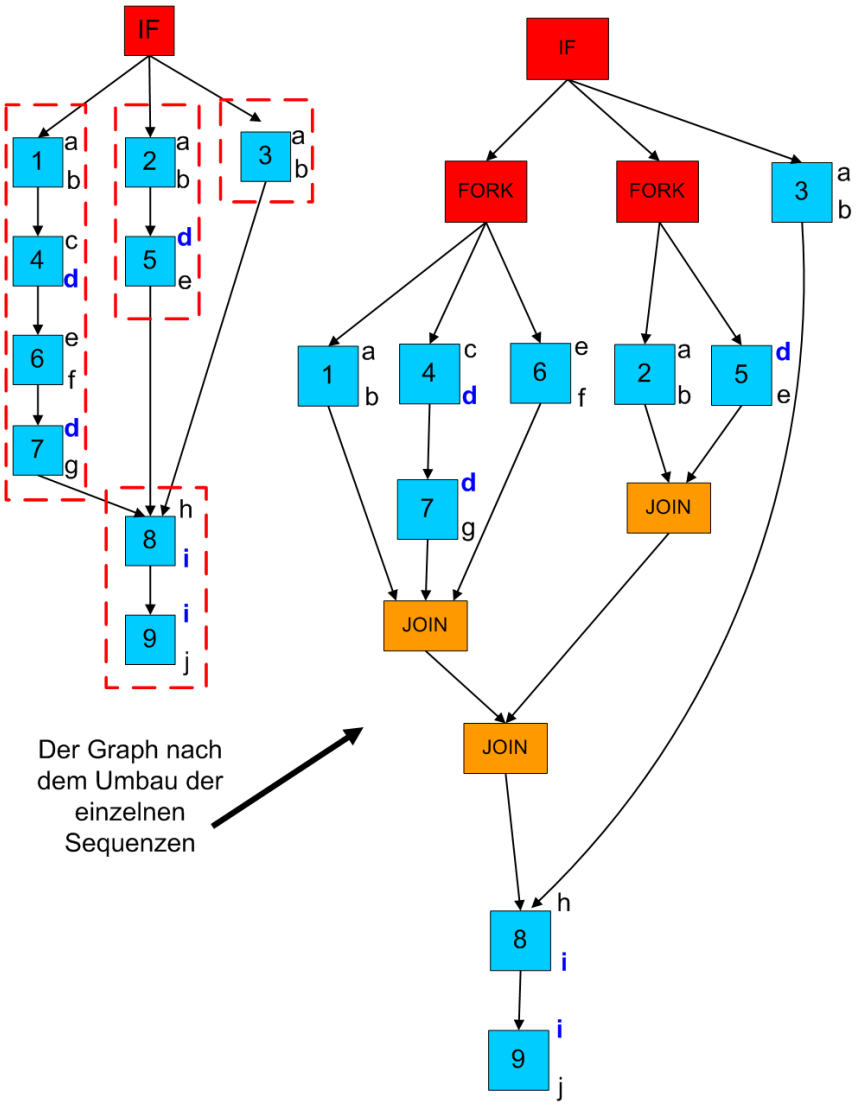

Abbildung 73: Graph mit Verzweigung vor und nach dem Umbau

nach dem Join SIB ausschließlich nur eine Kante erlaubt werden sollte. Leider könnte man diese Fälle nicht als ungünstig betrachtet, da sie alle eigentlich problemlos funktionieren dürften. Für das Join SIB war von vorne rein vorgesehen, dass nur eine Kante nach dem SIB ausgehen sollte. Unsere Überlegungen sollten sich auf die Probleme, die die Prozesse mit Fork und Join SIB's komplizieren könnten basieren. Es wurden vier Fälle festgelegt, die die sinnvollsten Fehler darstellen sollten:

- Zyklen mit Fork oder Join SIB's, d.h Pfade von dem SIB zu sich selbst sind nicht erlaubt. Dieser Fall wurde ausgewählt um die mehrmalige Durchführung von Prozessen zu vermeiden, die zu endlose Schleifen führen (siehe Abbildung 74).

Abbildung 74: Zyklen mit FORK-und JOIN-SIB's

- Ein Prozess, der mit einem Fork oder Join SIB anfängt darf nicht zwei Mal durchgeführt werden. Dieser Fall ähnelt dem ersten Fall, nur hier werden nach Fork SIB's andere SIB's gesetzt und durchgelaufen (siehe Abbildung 75).

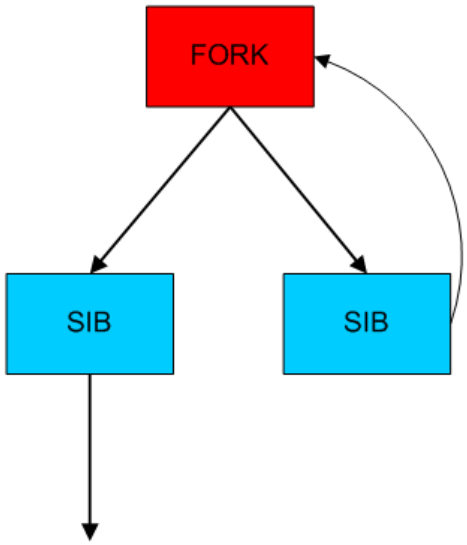

Abbildung 75: Zyklen in einem Prozess mit FORK-SIB

- Ein Fork oder ein Join SIB darf nicht in einem Zyklus vorhanden sein. Zum Entscheiden werden Zyklen ohne Fork und Join SIB's erlaubt. Aber sobald ein Fork zum Beispiel in dem Zyklus erkannt wird, wird der Prozess als ungültig erklärt (siehe dazu Abbildung 76).

- Es darf keine Crosskanten zwischen zwei Prozesse existieren. Eine Crosskante geht von rechts nach links zu einem Knoten, der weder ein ancestor noch ein descendant von seinem Vorgänger ist. Mit diesem Fall, werden zwei verschiedene Threads und ganz besonders die Kanten zwischen die SIB's betrachtet. Crosskanten im selben Prozess sind

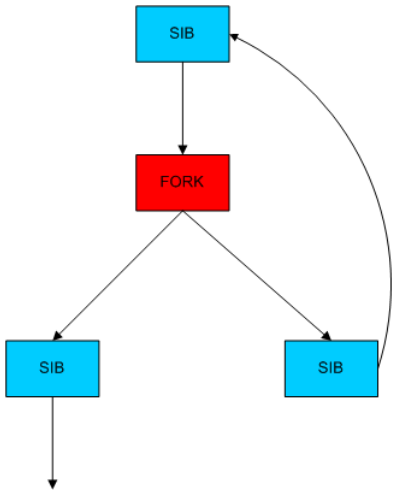

Abbildung 76: Zyklen mit FORK-SIB

allerdings erlaubt. Schwierigkeiten tauchen dann auf, wenn zwei verschiedene Prozesse miteinander verbunden werden. Dann werden sich zwei Prozesse treffen. Dies wird in der (Abbildung 77) gezeigt.

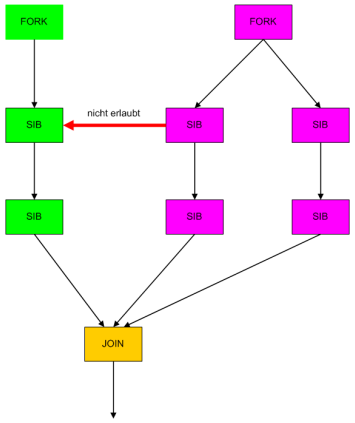

Abbildung 77: Crosskante zwischen zwei Prozesse

8.3 Implementierungsphase

8.3.1 Parallelisierung des Tracers

Um Fork und Join SIB's zu erstellen, wurden neue Klassen implementiert und bestehende Strukturen erweitert. Eine Klasse Tracer Thread Manager wurde für die Datenverwaltung entwickelt. Diese Klasse dient zum Speichern und Verwalten von Threads. Wenn ein Thread an einem Fork SIB angekommen ist, werden die Threads entsprechend der gekennzeichneten Branches gestartet. Die Threads werden in einem Zwischenspeicher abgelegt. Zuerst muss erkannt werden wie viele Prozesse ausgehend von Fork gestartet werden. Die Kante, die zum Fork SIB führt, wird gelöscht bzw. aus dem Tracer Thread Manager entfernt und die ausgehenden Kanten aus dem Fork SIB dem Tracer Thread Manager hinzugefügt. Die Threads werden danach gestartet und laufen parallel ab. Für die Synchronisation ist das Join SIB zuständig. Sobald alle Prozesse beim

Join SIB angekommen sind, werden die vorherigen Prozesse aus dem Tracer Thread Manager entfernt. Ein neuer Thread wird gestartet, dessen Prozess in Tracer Thread Manager gespeichert wird.

Bei der Erstellung von Graphen, ist dabei zu beachten, dass sich das Fork SIB unmittelbar vor der parallelen Stelle im Graphen befinden soll. Das Join SIB muss am Ende der parallelen Ausführung stehen. Jetzt betrachtet man die entsprechende Ausführung mit Hilfe des Tracers.

8.3.2 Erkennung von paralleliserbaren Graph-Sequenzen

In diesem Abschnitt wird erläutert, wie der Algorithmus zu Erkennung der parallelisierbaren Stellen im Graphen aufgebaut ist und es werden einige wichtige Methoden etwas ausführlicher beschrieben.

Um den Algorithmus aufbauen zu können, waren zwei Klassen notwendig:

1. eine Hilfsklasse Node und
2. eine Hauptklasse ParallelReconstruction

Als nächstes wird die Hilfsklasse Node beschrieben. Die Klasse Node hat unter anderem die folgenden Methoden:

- `dfs()`, sie führt die Tiefensuche ausgehend von diesem Knoten als Wurzelknoten aus, für den sie aufgerufen wurde und liefert alle erreichbaren Knoten zurück.
- `connectTo()`, sie erstellt eine ungerichtete Kante zwischen zwei Knoten.
- `connectLocal()`, sie erstellt eine gerichtete Kante zwischen zwei Knoten.
- `disconnect()`, löscht alle ein- und ausgehenden Kanten für den aufrufenden Knoten.
- `disconnectLocal()`, löscht nur die ausgehenden Kante für den aufrufenden Knoten.

Nun eine Beschreibung der Hauptklasse ParallelReconstruction:
Die wichtigsten Methoden der Hauptklasse ParallelReconstruction sind folgende:

- `bfs()`, erstellt einen Hilfsgraphen mit der entsprechenden Breitensuchennummerierung, die auf dem Origianlgraphen durchgeführt wird.
- `compare()`, prüft die Menge der SIB's auf Datenabhängigkeiten
- `supcomp()`, ist der rekursive Algorithmus zur Bestimmung der Parallelisierung. Als Eingabe bekommt sie den Hilfsgraphen, aus dem sie den Zielgraphen erstellt. An dieser Stelle werden alle Fallunterscheidungen gemacht nämlich für normale Sequenzen und Verzweigungen.
- `maxSeq()`, ist für die Bestimmung maximaler unabhängiger Sequenzen nach dem verzweigenden Knoten zuständig. Als Eingabe bekommt die Methode den verzweigenden Knoten.

- `posBFS()`, ist für die Positionierung des Algorithmus zuständig, so dass die SIB's im Zielgraphen einigermaßen gut im jABC-Framework angeordnet werden.

Im Konstruktor `ParallelReconstruction()` wird zunächst der Originalgraph durch die Nodes nachgebaut, um die Erreichbarkeit der einzelnen Knoten zu ermitteln, bzw. die Nachbarschaftsrelationen (siehe Methode `dfs()` in der Klasse Node). Gleichzeitig erhält der Graph durch die Methode `bfs()` eine Nummerierung und die Nodes werden im Array helpgraph gespeichert.

Danach werden alle Knoten in dieser Nachbildung gelöscht und durch `compare()` die Datenabhängigkeiten und Kontrollflussabhängigkeiten berechnet. Entsprechend den Abhängigkeiten werden neue ungerichtete Kanten in den helpgraph eingefügt. Daraufhin wird die Methode `supcomp()` auf alle Komponenten aus dem helpgraph aufgerufen und der Zielgraph helpgraph2, also der parallelisierte Graph erstellt.

Nach der Erstellung des Zielgraphen wird der Originalgraph nach dieser Vorlage modifiziert, (d.h. Fork und Join SIB's werden hinzugefügt und die Kanten entsprechend umgebogen). Zuletzt wird die Position der SIB's durch die Methode `posBFS()` optimiert.

8.3.3 Untersuchung der Graphen auf zu vermeidende Konstruktionen

Für die Durchsuchung von Graphen wurde entschieden, die Tiefensuche zu verwenden. Der Tiefendurchlauf geht von einem Knoten in die Tiefe, indem einer der Söhne besucht wird, dann einer von dessen Söhnen usw. Es lassen sich hier drei Varianten Preorder-, Inorder- und Postorder-Durchlauf formulieren. Bei der Preorder Strategie, wird zunächst die Wurzel, dann der linke und anschließend der rechte Unterbaum besucht. Bei der Inorder Strategie wird die Wurzel nach dem linken und rechten Unterbaum besucht. Für den Postorder Durchlauf wird zunächst Postorder für den linken, dann Postorder für den rechten Unterbaum aufgerufen, bevor die Wurzel besucht wird. In unserem Fall wird die Preorder Strategie verwendet. Eine Klasse SIBGraph_DFS wurde dafür implementiert. Die wichtigsten Methoden dieser Klasse sind: `visit(), dfs()`. SIB's und ihre Nachfolger werden untersucht und markiert. Die besuchten SIB's werden dann der rekursiven Methode `visit()` als Parameter übergeben. Jeder besuchte SIB wird durch diese Methode als erste Aktion entsprechend markiert.

Für die Suche nach Zyklen wurde eine Hashmap- Datenstruktur benutzt. Diese Datenstruktur wurde ausgewählt, weil sie effizient ist und exakte Ergebnisse zurückliefert. Die Hashmap ist durch die Graphenstruktur vorgegeben. Alle SIB's und deren UIDs werden in die Hashmap eingefügt. Um Zyklen mit Fork und Join SIB's zu verhindern, wurden zuerst diese SIB's einzeln betrachtet. Wenn wir das Fork SIB z.B. betrachten, sollte erstmal überprüft werden, ob dieses SIB sowie seine entsprechende UID in der Hashmap vorhanden sind. Wenn ja, muss verhindert werden, dass ein zweites Fork SIB in die Hashmap hinzugefügt wird, d.h diese SIB's dürfen nur einmal in die Hashmap eingefügt werden.

Durch diesen Mechanismus werden auch Prozesse mit Fork und Join SIB's nicht erlaubt. Allerdings gab es am Anfang Probleme mit der Implementierung. Das Problem lag darin, Fork und Join SIB's individuell zu betrachten. Bei der Überprüfung der Hashmap wurden alle SIB's betrachtet, sodass Zyklen für diese SIB's auch nicht erlaubt wurden. Um das Problem zu lösen, musste die Hashmap zuerst Fork und Join SIB's erkennen. Danach konnten Zyklen ohne Fork und Join SIB's problemlos erstellt werden.

Cross Kanten zwischen Prozesse werden nicht erlaubt. Dafür wird für jeden Prozess eine bestimmte Farbe für die SIB's festgelegt. Eigentlich sollten die SIB's nur Nummern als Markierung haben, um den Graphen nicht farbig darzustellen. Um eine Verbindung zwischen den Prozessen zu verhindern, dürften zwei SIB's, die unterschiedliche Nummern haben, sich nicht verfolgen. Um das zu ermöglichen wurde ebenfalls eine Hashmap-Datenstruktur benutzt. In der Hashmap wurden nur die markierten SIB's von dem zweiten Teilgraph eingefügt. Damit unser Fall funktioniert, sollte das Einfügen von einem Teilgraph, der eine andere Markierung hat, in die Hashmap nicht erlaubt sein. Natürlich kann so ein Vorgehen beim ersten Versuch nicht automatisch laufen. Ein Problem entstand, weil die markierten SIB's von dem ersten Teilgraph auch in die Hashmap eingefügt wurden, d.h nicht das Einfügen von einem SIB mit einer anderen Markierung, sondern das Einfügen von dem gesamten Prozess wurde in Anspruch genommen. Das führte zu einem zweiten Problem, weil es sollte bedeuten, dass sobald versucht wird, die SIB's vom ersten Teilgraph in die Map einzufügen, der Prozess vom Anfang an als ungültig erklärt wird. Nach mehreren Überlegungen musste nur die Hashmap darauf hingewiesen werden, dass nur das zweite Einfügen der ersten Markierungsnummer nicht erlaubt ist. Allerdings ist eine Crosskante zwischen zwei internen Prozessen erlaubt.

8.4 Ausblick

Während der PG wurde der Tracer um einige Funktionen bezüglich der Parallelität erweitert. Der Tracer kann jetzt dynamisch neue Threads erzeugen und beenden. Parallelisierbare Sequenzen im Graphen werden erkannt, und die Graphen können entsprechend umgebaut werden. Dieser Algorithmus könnte man so erweitern, dass er auch in der Lage ist, Graphen die schon Parallelisirungskomponenten enthalten nach weiteren möglichen Parallelisierungsstellen durchsucht und dementsprechend umbaut. Des weiteren könnte man die Erkennung zu vermeidender Graph Konstruktonen so erweitern, das sie in der lage ist, einen Teil der verbotenen Konstruktionen automatisch aufzulösen. Dies schein und insbesondere im Falle der Cross-Kanten Problematik möglich.

9 Rückblick

Im Januar startete die PG470 - Java Electronic Tool Integration Platform für alle Teilnehmer mit der ersten Sitzung. Auf dieser Sitzung wurden allen Teilnehmern der Projektgruppe (PG) einzelne Themen zur Ausarbeitung gegeben. Dieses Thema sollte von den Teilnehmern in einem Seminar vor den anderen Teilnehmern in einem 45 minütigen Vortrag vorgestellt werden. Die Themenverteilung sah wie folgt aus:

1.	Eclipse CVS	Markus
2.	Corba	Pascale
3.	Parallele Alg.	Sergej
4.	FreeChart	Norman
5.	UML	Hunervan
6.	SOAP	Zhongyue
7.	Java ABC	Iris
8.	jETI	Moctar
9.	Hibernate	Dino ersetzt durch Akif
10.	JSP/Servlet	Jiong
11.	Tomcat	Hatim
12.	jUnit	Armelle

Diese sogenannte Seminarphase dient dazu, allen Teilnehmern der PG einen genauen Einblick in die entsprechenden Themen, die für die PG benötigt werden, zu geben. Leider fiel unser erstes Teammitglied (Dino) noch vor dem Vorstellen der Themen aus und wurde durch Akif ersetzt. Dies hatte zur Folge das die Ausarbeitung von Hibernate erst mit einer großen Verspätung vorgestellt wurde um Akif auch die benötigte Zeit dafür einzuräumen. Anders als normalerweise üblich ist die PG für die Präsentation der Ausarbeitungen nicht übers Wochenende verreist, sondern hat diese im April in einem Seminarraum des Lehrstuhls 5 über 2 Tage abgehalten.

Danach wurden die Teilnehmer in 4 Untergruppen zu je 3 Mitgliedern aufgeteilt und diese Untergruppen begannen danach mit Ihrer Arbeit. Der Fortschritt der Untergruppen wurde in wöchentlichen Sitzungen mit allen PG Teilnehmern von den Betreuern überprüft. In einer PG sollen vor allen Dingen Teamarbeit und selbstständige Zeiteinteilung vermittelt werden und so war die erste Aufgabe aller Untergruppen eine Erstellung eines Pflichtenheftes für die Untergruppe. In diesem Pflichtenheft sollte auch eine ungefähre Zeiteinteilung für die verschiedenen Aufgaben enthalten sein. Danach begann für alle Untergruppen die Modellierungsphase. In dieser Phase haben sich die Teilnehmer mit den verschiedenen Programmen und Techniken vertraut gemacht und damit begonnen, die ersten Entwürfe für ihre Aufgaben anzufertigen. Fast alle Untergruppen haben ihre Ziele für das erste Semester ohne größere Probleme erreichen können.

In den Semesterferien zwischen dem ersten und dem zweiten Semester der PG stand eine Zwischenpräsentation an. Diese fand am PG Tag des Lehrstuhls 5 stand. Auf diesem PG Tag wurden 2 Endpräsentationen und 2 Zwischenpräsentationen präsentiert. Für die Zwischenpräsentation wurde eine Präsentation mit einigen Folien angefertigt. Außerdem haben wir den anwesenden Personen eine Live Demonstration unseres bisherigen Fortschritts gezeigt. Die Betreuer waren mit dem bisherigen Stand und der Präsentation sehr zufrieden.

Das zweite Semester begann ähnlich wie das erste. Es musste zuerst ein Pflichtenheft für das Semester angefertigt werden. Leider verlief das zweite Semester nicht ganz so gut wie das erste, so dass fast alle Untergruppen am Ende doch noch in Zeitnot gerieten. Im Dezember verließ uns dann Sergej auf eigenen Wunsch, so dass die PG danach nur noch mit 11 Teilnehmern weiter arbeiten konnte. Nichts desto trotz hat die PG ihre an sie gestellten Aufgaben dennoch erfüllt.

Die verschiedenen Untergruppen sind in Ihren Aufgaben auf einige Probleme gestoßen, die aber letztendlich alle gelöst werden konnten. Die Untergruppe für den HTML Client stieß beispielsweise auf das Problem, das Sie Probleme hatten auf die Eigenschaften der einzelnen SIB's, das heißt auf die verschiedenen Attribute, Methoden und Strukturen zuzugreifen. Die Lösung wurde hier mit Hilfe von Reflection gefunden.

Die Untergruppe des Statistik und Monitoring Teils hatte anfangs Probleme mit Hibernate und der entsprechenden Datenbankanbindung. Außerdem gab es Probleme beim erstellen der HQL Abfragen und beim Erstellen der verschiedenen Datasets. Dieses Problem wurde durch ein genaues Case Sensitives arbeiten gelöst, da HQL in unserer Web Applikation anscheinend doch nicht incase-sensitive arbeitet.

Die Untergruppe für die Parallel Ausführung hatte vor allen Dingen Probleme mit dem jABC, da es dafür scheinbar keine gute API Dokumentation gibt. Mit einer detailierten Dokumentation hätte man sich deutlich schneller in das jABC rinarbeiten können. Auch wäre es sicherlich von Vorteil gewesen, wenn benötigte Programme von vornherein zur Verfügung gestellt worden wären.

10 Ausblick & Fazit

10.1 Ausblick des ETI Systems

In diesem Abschnitt findet man Weiterentwicklungsmöglichkeiten für das ETI System. Zur Zeit ist es so, dass das ETI System Batch Tools ausführt. Das bedeutet das für ein Tool nur noch die Parameter mit denen es ausgeführt werden soll eingegeben werden müssen und das Tool dann seine Aufgabe mit den eingegeben Parametern ausführt. Eine Weiterentwicklung wäre bspw. wenn man “Interaktive“ Tools im System bereitstellen würde. Interaktiv bedeutet hier, das der Benutzer quasi nur einen Zusammenhang eingeben muss und der Client extrahiert daraus alle benötigten Parameter. Weiterhin wäre es schön wenn das System graphische Dialoge bzw. Graphische Tools anbieten würde. Außerdem könnten vorgefertigte allgemeine Graphen im System angeboten werden. Das heißt für Standard Schritte die in fast jedem Graphen vorkommen, könnte das System bereits die nötigen Sequenzen zusammengefasst haben. Ein weiteres großes Problem dürften Tools mit einer sehr langen Bearbeitungszeit sein. Das Problem ist dabei, wenn der Toolprovider mehrere Tage für die Bearbeitung braucht, weil z.B. mehrere aufwändige Simulationen oder etwas ähnliches durchgeführt sollte. Wie wird der Benutzer nach 3 Tagen wieder erkannt, da 1. zwischenzeitlich eine Zwangstrennung von seinem Internet Service Provider stattgefunden hat und 2. man dem Benutzer sowieso nicht zumuten möchte 3 Tage seinen Rechner an zu lassen. Hier müsste also an einer Art von Offline Kommunikation gearbeitet werden, die dafür sorgt das der Toolprovider auch wenn die Session nicht mehr da ist, der Benutzer trotzdem wieder erkannt wird.

Eine wichtige Verbesserung in der heutigen Zeit wäre das Verschlüsseln der Kommunikation. Zur Zeit findet noch keine Verschlüsselung statt. Des Weiteren wäre auch eine vorherige Kompression der Daten vor dem Versenden der Daten wünschenswert um die Übertragungsmengen minimal zu halten. Hier könnte man auch überprüfen ob z.B. eine Minimierung der Kommunikation möglich wäre. Zur Zeit laufen alle Kommunikationen über den jeweiligen Client ab. Dies bedeutet der Client schickt Daten an Toolprovider A und bekommt bearbeitete Daten zurück. Diese werden nun vom Client zur Weiterverarbeitung zu Toolprovider B geschickt und auch dieser schickt seine bearbeiteten Daten zurück zum Client. Hier wäre also ein Ansatzpunkt, das die Kommunikation nicht immer über den Client laufen müsste, sondern Toolprovider A seine Daten direkt an Toolprovider B schicken kann und erst dieser die bearbeiteten Daten zurück an den Client schickt. So würde man sich Kommunikationsschritte sparen.

Wenn das System sich noch weiter vergrößert (davon gehen wir aus), dann wäre es auch noch schön wenn eine intelligente Auswahl der Toolprovider für den Benutzer stattfinden würde.

Dies bedeutet, je mehr Toolprovider im System ihre Tools zur Verfügung stellen, desto wahrscheinlicher ist es, das Standard Tools wie z.B. `Solarize` oder `Rotate` von verschiedenen Toolprovidern angeboten werden. Hier wäre es also wünschenswert, wenn der Benutzer dann das Tool ausgewählt bekommt, dass für seine Aufgaben das beste ist, bspw. es wird das schnellste Tool oder das Tool was für den Benutzer am billigsten ist, genommen.

10.2 Fazit

In diesem Fazit wird beschrieben, was den Teilnehmern besonders gefallen hat und wie man den Ablauf der PG hätte verbessern können.

Alle Teilnehmer sind sich einig, dass die PG gut abgelaufen ist. Allerdings gab es an verschiedenen Stellen Verbesserungsmöglichkeiten. Sehr gut fanden die Teilnehmer, das man sehr viele neue Technologien innerhalb dieser 2 Semester kennen gelernt hat und mit diesen nun auch umgehen kann. Die Teilnehmer der PG haben sich während der Zeit in die Technologien CORBA, SOAP, Hibernate, JSP/Servlets, SQL und LaTeX eingearbeitet. Zusätzlich wurde das Arbeiten mit einem CVS einstudiert. Positiv ist zu erwähnen, dass das Arbeitsklima innerhalb der Untergruppen sehr positiv war und sich die Teilnehmer auch über diese Untergruppen hinaus bei Problemen so gut es ging geholfen haben. Die Atmosphäre der gesamten Gruppe war sehr gut und das obwohl wir uns nicht über das übliche Wochenende zur Seminarphase besser kennengelernt haben. Außerdem sind unsere Betreuer Harald und Ralf noch positiv hervorzuheben. Sie waren sehr nett, hatten immer Zeit für uns und gaben Hilfe wo Sie nur konnten. Vielen Dank dafür.

Die Projektgruppe hatte ein großes Ziel, Studenten sollten lernen in einem Team zu arbeiten. Dabei ist es immer wichtig, Verantwortung für bestimmte Teilgebiete zu übernehmen, für seine Aufgaben einzustehen und natürlich auch zu bestimmten Terminen mit diesen Aufgaben fertig zu sein. Dieses haben alle Teilnehmer gelernt, auch wenn es am Anfang nicht so gut klappte, da die Absprachen unter den verschiedenen Untergruppen nicht funktioniert haben. Weiterhin ist positiv zu erwähnen, dass es in dieser PG auch möglich war zuhause zu arbeiten und man nicht unbedingt im Pool des Lehrstuhls programmieren musste. Allerdings haben die wenigstens auch zuhause ein System mit Linux gehabt und es gab daher einige Probleme die Dateien, die unter Windows gecodet wurden, auf den Linux Rechnern des Pools zum laufen zu bringen. Vielleicht wäre es hier noch sinnvoll im Lehrstuhl Pool doch den ein oder anderen Rechner mit Windows zu bestücken.

Negativ zu erwähnen ist, das die PG während der Zeit insgesamt 2 Präsentationen abhalten musste. Dies ist natürlich zur Übung von Präsentationen sehr förderlich, hat der PG letzt endlich aber sehr viel Zeit gekostet.

Den Ablauf der PG könnte man verbessern, indem man bereits am Anfang alle wichtigen Aufgaben (Sitzungsleiter, Protokollführer, Zwischenberichtsbeauftragter, Endberichtsbeauftragter usw.) auf die Leute verteilt. Zusätzlich sollten direkt am Anfang genaue Termine für wichtige Ereignisse (Zwischenbericht, Endbericht, Endpräsentation usw.) festgelegt werden und diese müssen dann auch eingehalten werden. Die Betreuer (und natürlich auch die anderen Teilnehmer) hätten noch deutlicher auf das Einhalten von Deadlines achten können, als sie es schon getan haben. Es war für die meisten Teilnehmer das erste Mal, dass sie so ein großes Projekt erarbeitet haben. Das man die Zeit dabei selber einteilen konnte war zwar schön, allerdings wären mehr Tipps zur Zeiteinteilung vielleicht doch hilfreich gewesen, da man so doch einige

Sonderschichten einlegen musste. Außerdem könnte man öfter Quellcodereviews machen, damit die Teilnehmer nicht mitten im Semester feststellen, der “Quellcode“ ist nicht sauber genug geschrieben.

Alle haben gemerkt, dass ein strukturiertes herangehen an eine solch große Aufgabe deutlich sinnvoller ist als ein einfaches auf die Arbeit stürzen ohne großen Plan. Trotzdem ist die PG mit dem Erreichten sehr zufrieden und bedankt sich bei den Betreuern für die sehr gute Unterstützung.

Literatur

[Api()] JAVA Api. Resourcebundle. http://java.sun.com/j2se/1.4.2/docs/api/java/util/ ResourceBundle.html.

[Bauer and King(2005)] Christian Bauer and Gavin King. *Hibernate In Action*. Manning Publications Co., Greenwich 2005, 2005. ISBN 1932394-15-X.

[Bergsten(2001)] Hans Bergsten. *Java Server Pages (Deutsche Ausgabe)*. O'Reilly Verlag GmbH & Co. KG, Köln 2001, 3. auflage edition, 2001. ISBN 3-89721-281-1.

[charting the web()] Cewolf charting the web. Introduction. http://cewolf.sourceforge.net/new/index.html.

[Cuber(2005)] Ulrich Cuber. *Einstieg in Eclipse3*. Galileo Press GmbH, Bonn 2005, 1. auflage edition, 2005. ISBN 3-89842-552-5.

[Eberhart and Fischer(2000)] Andreas Eberhart and Stefan Fischer. *Java-Bausteine für E-Commerce - Anwendungen - Verteilte Anwendungen mit Servlets, CORBA und XML*. Hanser: München, Wien, 2000. ISBN 3446213724.

[Farley et al.(2002)Farley, Crawford, and Flanagan] Jim Farley, William Crawford, and David Flanagan. *Java Enterprise in a Nutshell*. O'Reilly Verlag GmbH & Co. KG, 2. auflage edition, 2002. ISBN 3897213346.

[Foundation()] Apache Software Foundation. Logging services. http://logging.apache.org/log4j/docs/.

[Gamma et al.(1994)Gamma, Helm, Johnson, and Vlissides] Erich Gamma, Richard Helm, Ralph Johnson, and John M. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley: Amsterdam, Bonn, Tokyo, 1994. ISBN 0201633612.

[Goll et al.(2001)Goll, Weiß, and Müller] Joachim Goll, Cornelia Weiß, and Frank Müller. *Java als erste Programmiersprache*. Teubner Verlag, Stuttgart, 3. auflage edition, 2001. ISBN 3-519-22642-1.

[Group(2000)] Object Management Group. The common object request broker: Architecture and specification, 2000. http://www.omg.org.

[Gruhn and Thiel(2000)] Volker Gruhn and Andreas Thiel. *Komponentenmodelle - DCOM, Javabeans, Entreprise JavaBeans, CORBA*. Addison-Wesley: München, 1. auflage edition, 2000. ISBN 382731724X.

[hibernate.org()] hibernate.org. Hql: The hibernate query language. http://www.hibernate.org/hib_docs/reference/en/html/queryhql.html.

[Javadoc()] Javadoc. Jfreechart. http://www.jfree.org/jfreechart/javadoc/index.html.

[Jobst et al.(2001)Jobst, Hofmann, and Schabenberger] Fritz Jobst, Johann Hofmann, and Roland Schabenberger. *Programmieren mit COM und CORBA – Einführung in die Architekturen für verteilte Anwendungen*. Hanser: München, Wien, 2001. ISBN 3-446-21479-8.

[junit.org()] junit.org. http://www.junit.org/index.htm.

[Kabir(2003)] Mohammed J. Kabir. *Die Apache Server 2 Bibel*. mitp, Bonn, 2. auflage edition, 2003. ISBN 3-8266-0896-8.

[Kaminaris and Annunziato(2001)] Stephanie Fesler Kaminaris and Jose Annunziato. *Jetzt lerne ich JavaServer Pages*. Markt und Technik Verlag, 2001. ISBN 3-8272-6009-4.

[Krüger(2003)] Guido Krüger. *Handbuch der Java-Programmierhandbuch*. Addison Wesley Verlag, München, 3. auflage edition, 2003. ISBN 3-8273-2120-4.

[Laborenz(2005)] Kai Laborenz. *CSS-Praxis*. Galileo Press GmbH, Bonn 2005, 3. auflage edition, 2005. ISBN 3-89842-577-0.

[Linnhoff-Popien(1998)] Claudia Linnhoff-Popien. *CORBA Kommunikation und Management*. Springer: Berlin, Heidelberg, 1998. ISBN 3540640134.

[Microsoft()] Microsoft. Microsoft sql server. http://www.microsoft.com/sql/.

[Middendorf(2003)] Stefan Middendorf. *JAVA-Programmierhandbuch und Referenz*. dpunkt.verlag, Heidelberg, 3. auflage edition, 2003. ISBN 3-89864-157-0.

[Moczar(2005)] Lajos Moczar. *Tomcat 5: Einsatz in Unternehemensanwendungen mit JSP und Servlet*. Addison-Wesley, München u.a., 1. auflage edition, 2005. ISBN 3-8273-2202-2.

[MySQL()] MySQL. Mysql open source datenbank. http://www.mysql.de.

[Münz and Nefzger(1999)] Stefan Münz and Wolfgang Nefzger. *HTML 4.0 Handbuch*. Franzis'Verlag GmbH, Poing 1999, 3. auflage edition, 1999. ISBN 3-7723-7514-6.

[Njoku(2005)] Marc Njoku. Entwicklung einer systemerweiterung der jabc-entwicklungsplattform zur ansterung verteilter toolsequenzen, 2005.

[ORACLE()] ORACLE. Oracle datenbanksystem. http://www.oracle.com.

[Orfali(1998)] Robert Orfali. *Führung durch die CORBA-Welt*. Addison-Wesley: Bonn, New York, Tokyo, 1998. ISBN 3-8273-1325-2.

[Pages()] Cover Pages. Simple object access protocol (soap). http://xml.coverpages.org/soap.html.

[Pekgöz(2002)] Numan Pekgöz. *JSP*. Pusula Yayincilik ve Iletisim Ltd., Istanbul 2002, 1. auflage edition, 2002. ISBN 975-7092-95-9.

[PostgreSQL()] PostgreSQL. Postgresql open source database. http://www.postgresql.org.

[Prohorenko and Prohorenko(2004)] Alexander Prohorenko and Olexiy Prohorenko. Using junit with eclipse ide, 2004. http://www.onjava.com/pub/a/onjava/2004/02/04/juie.html.

[Riedemann(1997)] Eike Hagen Riedemann. *Testmethoden für Sequenzielle und nebenläufige Software-Systemen*. Teubner Verlag, 1997. ISBN 3519022745.

[Roßbach and Holubek(2002)] Peter Roßbach and Andreas Holubek. *Java Servlet und JSP mit Tomcat 4*. Software und Support Verlag, Frankfurt, 1. auflage edition, 2002. ISBN 3-935042-27-2.

[Siegel(2000)] Jon Siegel. What's coming in corba 3.0, 2000. http://www.omg.org/technology/corba/corba3releaseinfo.htm.

[Skonnard()] Aaron Skonnard. Soap: The simple object access protocol. http://www.microsoft.com/mind/0100/soap/soap.asp.

[SOAP:Workshop()] SOAP:Workshop. What is soap?

http://www.topxml.com/soapworkshop/articles/intro/default.asp.

[Spielman(2001)] Sue Spielman. Designing jsp custom tag libraries, 2001. http://www.onjava.com/pub/a/onjava/2000/12/15/jsp_custom_tags.html.

[Stitz()] Lars Stitz. Soap - simple object access protocol. http://www.fh-wedel.de/ši/seminare/ws00/Ausarbeitung/6.soap/soap00.htm.

[Tomcat()] Apache Tomcat. http://jakarta.apache.org/tomcat.

[Ullenboom(2005)] Christian Ullenboom. *Java ist auch eine Insel*. Galileo Press Gmbh, Bonn, 4. auflage edition, 2005. ISBN 3-89842-326-6.

[Vonhoegen(2004)] Helmut Vonhoegen. *Einstieg in JavaServer Pages 2.0*. Galileo Press, Bonn, 1. auflage edition, 2004. ISBN 3-89842-360-3.

[w3schools.com()] w3schools.com. Soap tutorial. http://www.w3schools.com/soap/default.asp.

[Westphal()] Frank Westphal. Unit tests mit junit.

http://www.frankwestphal.de/ UnitTestingmitJUnit.html.

[Willms(2003)] Roland Willms. *LATEX - echt einfach*. Franzis Verlag GmbH, Poing 2003, 3. auflage edition, 2003. ISBN 3-7723-6888-3.

[Wunderlich(a)] Lars Wunderlich. Schöner, bunter, mächtiger ..., a. http://www.eclipse-magazin.de/itr/online_artikel/psecom,id,654,nodeid,230.html.

[Wunderlich(b)] Lars Wunderlich. Die dritte sonnenfinsternis, b. http://www.javamagazin.de/itr/online_artikel/psecom,id,591,nodeid,11.html.

A Anhang HTML Client

A.1 web.xml

```
<servlet>
    <servlet-name>ServletTestRunner</servlet-name>
    <servlet-class>
        org.apache.cactus.server.runner.ServletTestRunner
    </servlet-class>
</servlet>

<servlet>
    <servlet-name>ServletRedirector</servlet-name>
    <servlet-class>
        org.apache.cactus.server.ServletTestRedirector
    </servlet-class>
</servlet>

<servlet-mapping>
    <servlet-name>ServletTestRunner</servlet-name>
    <url-pattern>/ServletTestRunner</url-pattern>
</servlet-mapping>

<servlet-mapping>
    <servlet-name>ServletRedirector</servlet-name>
    <url-pattern>/ServletRedirector</url-pattern>
</servlet-mapping>
```

A.2 Ein erster Test

```
1  ExamizeGraph.java
2
3  package de.unido.ls5.jeti.htmlclient.servlet.graph;
4
5  import java.io.IOException;
6  import javax.servlet.*;
7  import javax.servlet.http.*;
8
9  public class ExamineGraph extends HttpServlet{
10
11     String pathForEdges;
12     String locale;
13     String path;
14
15     public void init(ServletConfig config) throws ServletException {
16         super.init(config);
17         pathForEdges = config.getInitParameter("pathForEdges");
18 }
19
20     public void service(HttpServletRequest request, HttpServletResponse response)
21             throws ServletException, IOException{
22
23         // get the parameter from Request "locale", for examine, into which language
24         // the web page to be indicated is
25         locale=request.getParameter("locale");
26         path = "/"+locale+"/htmlclient/ChoiceType.jsp";
27
28         // get the parameter from Request "judge", for examine,
29         // which for type of the clicked cell is
30         String judge = request.getParameter("judge");
31
32         /*************************************************************************/
33         // if "judge" equals to "null.gif", that means, the clicked cell is empty,
34         // and user wants to have a new object, then goto web page "ChoiceType.jsp",
35         // where user can choice the type of new object
36         /*************************************************************************/
37         if(judge.equals(pathForEdges+"null.gif"))
38         path = "/"+locale+"/htmlclient/ChoiceType.jsp";
39
40         /*************************************************************************/
```

```
41          // if "judge" doesn't euqal to "null.gif", that means, the clicked cell
42          // contains an object, and user wants to change the characteristics of the object,
43          // where user can choice:
44          // - Edit Graph
45          // - Edit Parameter
46          // - Delete
47          /*****************************************************************************/
48          else
49              path = "/"+locale+"/htmlclient/ChoiceEdit.jsp";
50
51              // With an RequestDispatcher object a Servlet can connect itself to another
52              // Servlet(or another Servlet).
53              RequestDispatcher dispatcher = getServletContext().getRequestDispatcher(path);
54              dispatcher.forward(request, response);
55      }
56
57      // get the Attribute "locale"
58      public String getLocale(){
59          return locale;
60      }
61
62      // get the Attribute "path"
63      public String getPath(){
64          return path;
65      }
66  }
```

A.3 Testklasse

```
1  TestExamizeGraph.java
2
3  package de.unido.ls5.jeti.htmlclient.servlet.graph;
4
5  import java.io.IOException;
6  import javax.servlet.*;
7  // In order to use Cactus, org.apache.cactus.* must be imported.
8  import org.apache.cactus.*;
9
10 /*****************************************************************************/
11 // ServletTestCase: extend this class for writing tests for unit testing code that
12 // uses Servlet API objects (HttpServletRequest, HttpServletResponse, HttpSession,
13 // ServletConfig, ServletContext, ...), like Servlets or any java classes which have
14 // methods that manipulates Servlet API objects
15 /*****************************************************************************/
16 public class TestExamineGraph extends ServletTestCase{
17
18      /*************************************************************************/
19      // For each XXX test case, a corresponding beginXXX() method(optional) can
20      // be defined and used to initialize HTTP related parameters (HTTP parameters,
21      // cookies, HTTP headers, URL to simulate, ...).
22      /*************************************************************************/
23      public void beginservice(WebRequest webreq) {
24          webreq.addParameter("locale","de");
25          webreq.addParameter("judge","pictures/graph/null.gif");
26      }
27
28      public void testservice() throws ServletException, IOException{
29
30          // instantiate the class to test
31          ExamineGraph eg = new ExamineGraph();
32
33          // initialise parameters setting in web.xml
34          config.setInitParameter("pathForEdges", "pictures/graph/");
35          eg.init(config);
36
37          // call the method to test
38          eg.service(request, response);
39
40          // Perform JUnit standard asserts (asserts(..), assertEquals(...), fail(...), ...)
41          // to verify that the test was successful
42          assertTrue("received␣wrong␣response", eg.getPath().equals
43                              ("/"+eg.getLocale()+"/htmlclient/ChoiceType.jsp"));
```

```
44      }
45
46      /*************************************************************************/
47      // For each XXX test case, a corresponding endXXX() method can be defined and used
48      // to verify the returned HTTP related parameters from test case (like the returned
49      // content of the HTTP response, any returned cookies, returned HTTP headers, ...).
50      /*************************************************************************/
51      public void endservice(WebResponse resp) {
52          //We don't do anything here but we could review the response
53      }
54  }
```

B Anhang Statistik Web Applikation

In diesem Abschnitt sind die Datenbank Tabellen die die Statistik betreffen aufgelistet. Außerdem findet man eine Übersicht über die JSP Seiten sowie deren dazugehörige sessionVariable `page`.

B.1 Statistik Datenbank

Das Layout der Statistik Tabelle der jetiDatenbank kann man in Abbildung 78. Das Layout der Tabellen filter und ForcedFilterToolprovider kann man in Abbildung 79 sehen. Das Lesen, Speichern und Löschen in den Datenbanken wird von den JSP Webseiten mit Hilfe von Hibernate und der dazugehörigen Query Language durchgeführt.

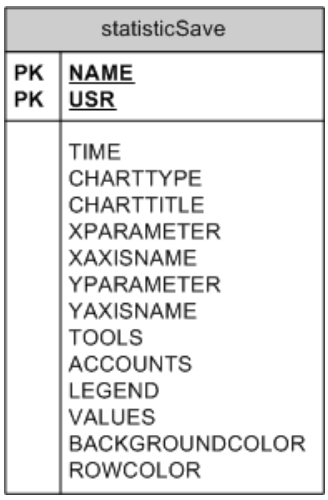

Abbildung 78: Das Layout der StatistikSave

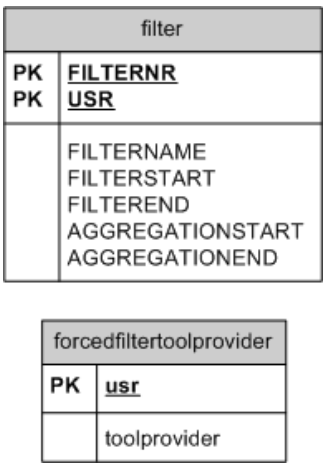

Abbildung 79: Das Layout der Datenbank Tabellen für die Filter und der forcedfiltertoolprovider

B.2 Übersicht über die Seitenzahlen

Wie im Text bereits öfters erwähnt, werden die Seiten innerhalb der Statistik Web Applikation mit Hilfe der Session-Variable `page` eindeutig identifiziert. Hier nun eine Auflistung über die verwendeten Zahlen, sowie die Zugehörigkeit der JSP-Seiten zu den verwendeten Zahlen.

Seitenzahl	Name
0	statisticNew.jsp
1	statistic2AxisStep1.jsp
2	statistic2AxisStep2.jsp
3	statistic2AxisLayout.jsp
4	statisticCircleStep1.jsp
5	statisticCircleLayout.jsp
6	statisticGraph.jsp
7	statisticLoad.jsp
8	statisticSave.jsp
9	statisticDelete.jsp
10	statisticToolChoices.jsp
11	statisticHelp.jsp
12	statisticFilterChoice.jsp
13	statisticFilter.jsp
14	statisticPieToolChoices.jsp
15	statisticError.jsp
16	statisticCircleSave.jsp
17	statisticForcedFilterToolprovider.jsp