Endbericht

Entwicklung eines Konfigurations-werkzeugs für Computersysteme

Nabil Azzam Jai, Pavel Baranov, Christoph Beerens,
Yunchan Cho, Said El Bouzakri, Abdulatif Elwashali,
Qinghua Fan, Jan Hendrik Jellesma, Michael Krücken,
Dirk Löfler, Alexander Nolte, Jianping Xie

PG 469
am Fachbereich Informaik
Universität Dortmund
Lehrstuhl V

22. März 2006

Veranstalter:

Prof. Dr. Bernhard Steffen
Dipl. Inform. Markus Bajohr
Dipl. Ing. Martin Karusseit

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung

Im Rahmen der Projektgruppe 469 an der Universität Dortmund wurde das von der Projektgruppe 451 entwickelte Werkzeug MaTRICS weiter entwickelt. MaTRICS steht für "**Ma**nagement **T**ool for **R**emote **I**ntelligent **C**onfiguration of **S**ystems" und stellt ein ausgeklügeltes und verteilt arbeitendes Konfigurationswerkzeug für Rechner- und Softwaresysteme dar. Zum besseren Verständnis des MaTRICS-Systems möchten wir an dieser Stelle auf den Endbericht der Projektgruppe 451 verweisen, in dem das Konzept, die Motivation und die Zusammenarbeit verschiedener MaTRICS-Komponenten ausführlich beschrieben werden. Des Weiteren setzen wir die Kenntnis der Basis-Komponenten des MaTRICS-Systems (ConfigManager, ConfigClient, ConfigAgent) voraus und gehen auf diese nicht näher ein.

Von der Projektgruppe 451 sind bereits, in Form von ServiceLogic-Graphen (SLG), folgende Basisfunktionen entwickelt worden:

- Adressierung der ConfigClients
- Bereitstellung der Konfigurationsdienste
- Weiterleitung von Aktionen an die ConfigClients
- Verwaltung von Konfigurationsdaten
- Bereitstellung der Kommunikationsprotokolle

Das Ziel unserer Projektgruppe war zum einen die Entwicklung von Komponenten für das MaTRICS-System, die dieses um zusätzliche Funktionen erweitern, zum anderen aber die der besseren Performance dienende Modifikation der vorhandenen Basis-Komponenten.

Dieser Bericht soll dem Leser einen Überblick über die Arbeit der Projektgruppe geben. Im Kapitel 2 wird über das vor dem Beginn der Projektgruppe gehaltene Seminar

berichtet. Kapitel 3 befasst sich mit den ersten Aufgaben, den so genannten Kurzthemen. In Kapitel 4 werden diverse Tools sowie die Entwicklungsumgebung vorgestellt, die in unserer Arbeit zum Einsatz gekommen sind. Kapitel 5 behandelt die Zwischenthemen, die zur Erweiterung der Basiskomponenten des MaTRICS-Systems gestellt wurden. Im Kapitel 6 handelt es sich um Konfigurationsdienste, die innerhalb der MaTRICS entwickelt wurden. Abschließend gibt Kapitel 7 ein kurzes Fazit.

Kapitel 2

Seminararbeiten

Vor Beginn der eigentlichen Projektgruppenarbeit stand ein Seminar auf dem Programm, welches in den ersten beiden Sitzungen geplant wurde. Dabei ging es Hauptsächlich um die Themenverteilung und die entsprechenden " Deadlines ". Der Folienmaster für die Vorträge wurde von Alexander Nolte und Dirk Löffler erstellt. Der zeitliche Rahmen des Seminars wurde wie folgt festgelegt: Deadlines:
13.02.2005 Abgabe der ersten Gliederung mit ungefährer Abschätzung des Umfangs
13.03.2005 Abgabe der vorläufigen Ausarbeitung inklusive der Folien
23.03.2005 Abgabe der Endversion der Ausarbeitung und der Folien

2.1 Ablauf des Seminars

Das Seminar wurde vom 30.03.05 bis 31.03.05, im Haus Bommerholz, durchgeführt und diente der Einführung in unser zukünftige Arbeit während der Projektgruppenzeit. Mittels der vorgegebenen Themen wurden den Teilnehmern neue Tools und Programme vorgestellt, die für die weiteren Arbeiten erforderlich waren. Desweiteren wurden Techniken vorgestellt, die für die Arbeit mit der MaTRICS benötigt werden, wie z.B. J2EE und RMI. Das Seminar verteilte sich auf zwei Tage mit jeweils sechs Vorträgen pro Tag (Tagesordnung siehe 2.2.1 und 2.2.2).

2.1.1 Mittwoch

- Thema 1:
  Objektorientiertes Design auf Basis von UML Vorgetragen von: Qinghua Fan
- Thema 2:
  Grundlagen von J2EE und Enterprise Services Vorgetragen von: Pavel Baranov

- Thema 3:
  Tools für die Software-Entwicklung Vorgetragen von: Christoph Beerens
- Thema 4:
  Kommunikation mit RMI und CORBA Vorgetragen von: Jan Hendrik Jellesma
- Thema 5:
  Testen und loggen mit JUnit und log4j Vorgetragen von: Michael Krücken
- Thema 6:
  XML Verarbeitung in Java Vorgetragen von: Jianping Xie

Der Abend gehörte allein den Teilnehmern und diente dem gegenseitigen Kennenlernen.

2.1.2 Donnerstag

- Thema 7:
  Webdesign mit xHTML, WML und Velocity Vorgetragen von: Abdullatif Elwashali
- Thema 8:
  Die Applikationsserver JBOSS und Tomcat Vorgetragen von: Said El Bouzakri
- Thema 9:
  Die Entwicklungsumgebung ABC mit EWIS Vorgetragen von: Yunchan Cho
- Thema 10:
  Grundlagen von SNMP Vorgetragen von: Nabil Azzam Jai
- Thema 11:
  Konfiguration und Dienste unter Linux Vorgetragen von: Alexander Nolte
- Thema 12:
  Konfiguration des Apache 1.3.x Webservers Vorgetragen von: Dirk Löffler

2.2 Ausarbeitungen

Die Ausarbeitung sollten ungefähr 15 bis 20 Seiten stark sein. Diese wurden mit Latex erstellt. Die Folien hingegen sollten mit OpenOffice erstellt werden. Eine PDF-Version der Ausarbeitung wurde anschliessend auf dem internen Webserver für alle Projektgruppenmitglieder zur Verfügung gestellt.
Der Umfang des Vortrages sollte eine halben Stunde nicht überschreiten. Anschliessend standen zu jedem Thema etwa zehn Minuten für eine Diskussion zur Verfügung.

Ein detailliertes Programm wurde von Said El Bouzakri an alle Teilnehmer verteilt und ist am Ende des Kapitels zu sehen.

Zu den Themen TogetherJ, ABC, Eclise und Apache wurde eine 5- bis 10-minutige Tool-Demo präsentiert. Diese Tool-Demos sollte den Teilnehmer die verwendeten Programme näher bringen, um das spätere Arbeiten zu vereinfachen.

Alles in allem kann man die Fahrt ins Haus Bommerholz in jeglicher Hinsicht als gelungenen Einstand bezeichnen. Vom Informationsgehalt bis hin zu den persönlichen und sozialen Aspekten wurde jede Minute der zwei Tage voll ausgeschöpft und mit sinnvollen Inhalten gefüllt. Die Gruppe wuchs während der Tage deutlich mehr zusammen und konnte nach Beendigung des Seminars mit breitgefächertem Wissen in die eigentlichen Arbeiten der Projektgruppe einsteigen.

Kapitel 3

Kurzthemen

Nachdem sich die Teilnehmer der Projektgruppe in einer ersten Phase mit der praktischen Anwendung des ABC[1] vertraut gemacht haben, bestand die nächste Aufgabe nun darin, dieses Wissen bei der Weiterentwicklung der MaTRICS[6] anzuwenden.

Da die MaTRICS[6] ein mittlerweile hochkomplexes System darstellt, wäre ein direkter Quereinstieg in das System für alle Beteiligten wohl äußerst schwierig gewesen. Aus diesem Grund wurden vier so genannte Kurzthemen eingeführt, die jeweils von drei Leuten in einem Zeitraum von etwa zwei Monaten bearbeitet werden sollten. Inhaltlich wurden die Themen von den PG-Betreuern so gewählt, dass die Ausarbeitungen dieser vier Themen einerseits zwar unabhängig von der eigentlichen MaTRICS[6] erstellt wurden, andererseits aber als Basis zur späteren Weiterentwicklung der MaTRICS[6] verwendet werden können.

Des Weiteren basierten die Themen weitestgehend auf Inhalten, die zuvor im Seminar aufbereitet wurden. Daher konnten die Ausarbeitungen aus dem Seminar als Hilfestellungen bzw. als Nachschlagewerke benutzt werden. Spätestens zu diesem Zeitpunkt sollten die Intentionen der einzelnen Seminarthemen deutlich geworden sein. Die Vorstellung der vier Themen sowie die Einteilung der Kleingruppen erfolgte in der PG-Sitzung am 21. April 2005 und verlief durchaus reibungslos.

Nabil, Pavel und Said bildeten das erste Team und bearbeiteten entsprechend Thema 1. Die Aufgabe bestand darin, eine SIB-Bibliothek[2] zur Administration und zum Monitoring via SNMP[5] zu entwickeln. Da sich viele Eingebettete Systeme, und vor allem auch Router und Switches via SNMP auslesen und konfigurieren lassen, soll diese SIB-Bibliothek[2] die Basis zur späteren Funktionserweiterung der MaTRICS[6] auf diesem Gebiet darstellen. Da die MaTRICS[6] bislang nur Dienste konfigurieren kann, die durch eine Konfigurationsdatei beschrieben werden, ist eine solche Erweiterung sehr sinnvoll.

Christoph, Jan und Michael bildeten das zweite Team und beschäftigten sich mit der Implementierung einer SIB-Bibliothek[2] zur Diagnose und zum Testen von Webdiensten. Mittels der ausgearbeiteten SIBs[2] sollte eine Art Framework geschaffen werden, mit dem das automatische Testen von Webdiensten ermöglicht werden sollte. Hierzu soll-

ten neben einfachen Verfügbarkeitstests auch komplexere Tests mittels Service-Logic-Graphen[10] beschreibbar sein.

Das dritte Team bildeten Abdulatif, Alexander und Dirk, deren Aufgabe darin bestand, eine SIB-Bibliothek[2] zur Konvertierung von Konfigurationsdateien in das allgemeine XML-Format zu erstellen. Befinden sich Konfigurationsdateien im XML-Format, gestaltet sich deren Modifikation einheitlich und kann einfacher und syntaktisch korrekt durchgeführt werden.

Das vierte Thema knüpfte unmittelbar bei der korrekten Modifikation von XML-Dateien an. Zu diesem Zweck erstellte das Team 4 eine entsprechende SIB-Bibliothek[2]. Für dieses Thema entschieden sich Jianping, Qinghua, und Yunchan.

Um Fehlentwicklungen wider den Vorstellungen der Betreuer zu vermeiden, wurden einzelne Themenbesprechungen durchgeführt in denen Lösungsideen besprochen wurden. Die Struktur der vier Themen wurde einheitlich geregelt. Grundsätzlich sollten verschiedene SIB-Bibliotheken mit Hilfe der bekannten Entwicklungswerkzeuge erstellt werden. Das Vorgehen bei der Entwicklung sowie ein zeitlicher Rahmen mit einzuhaltenden Deadlines wurde in der PG-Sitzung am 28. April 2005 festgelegt. Die Deadlines fielen jeweils mit einer PG-Sitzung zusammen, so dass die Ergebnisse der jeweiligen Entwicklungsphase unverzüglich vorgestellt werden konnten.

Der erste Schritt bestand in der UML-Modellierung. Für diese Phase einigte sich die PG auf einen zeitlichen Rahmen von drei Wochen. Als Werkzeug diente Together[7] von Borland[8]. In den darauffolgenden vier Wochen sollte die Implementierung erfolgen. Mit Hilfe des ABC[1] wurden SIB-Strukturen[2] erzeugt, deren eigentliche Funktionalitäten mit Eclipse[9] implementiert wurden. Eine weitere Woche wurde zum Testen der korrekten Funktionalität vereinbart. Um eine spätere Nutzung der erstellten SIB-Bibliotheken[2] zu ermöglichen, fertigten die Teams abschließend eine englische Dokumentation an.

Im Folgenden werden wir nun genauer auf die einzelnen Kurzthemen eingehen. Hierzu werden jeweils die entsprechenden Problemstellungen sowie die zugehörigen Lösungen der Teams zu diesem Thema vorgestellt.

3.1 Administration und Monitoring via SNMP

Die Aufgabe bestand darin, eine SIB-Palette zur Administration und zum Monitoring via SNMP zu erstellen. Zur Zeit ermöglicht die MaTRICS die Konfiguration von Diensten, die mit Hilfe von Konfigurationsdateien beschrieben werden. In der Regel werden aber Konfigurationen von Netzwerkkomponenten, wie z.B. Routern, Switches oder andere eingebettete Systeme mittels SNMP administriert. Desweiteren lassen sich mit SNMP auch interne Zustände von Systemen abfragen. Dieses ermöglicht beispielsweise eine Alarmierung bei Ausfall einer Netzwerkkomponente, z.B. eines Routers. Im Rahmen dieser Aufgabe wurde eine SIB-Bibliothek entwickelt, mit deren Hilfe die Verwaltung von Netzwerkkomponenten via SNMP möglich ist.

3.1.1 Modellierung

Zur Darstellung von den bei der Kommunikation via SNMP möglichen Abläufen kamen UML-Diagramme zum Einsatz.

Anwendungsfalldiagramm Anhand des Anwendungsfalldiagramms aus Abb. 3.1 lassen sich die wichtigsten von SNMP unterstützten Anwendungsfälle zeigen. Der Manager richtet eine Anfrage an einen Agenten in Form von `Get-`, `GetNext-` bzw. `Set-Operationen`. Der Agent reagiert auf diese Anfrage mit einer Antwort, deren Inhalt etwas darüber aussagt, ob die Anfrage erfolgreich bearbeitet werden konnte.

- **get value:** Zum angegebenen Object Identifier (OID) wird der Wert aus dem zu verwalteten Gerät geliefert.
- **get next value:** Diese Operation liefert zu einem angegebenen OID den Wert seines Nachfolgers. Vielen Werten sind in den MIB-Tabellen (Management Information Base) keine speziellen Namen zugewiesen. Daher kann man sie nur mit dem `GetNext`-Befehl ansprechen.
- **set value:** Die meisten verwalteten Objekte haben einen voreingestellten Wert, der beim Start des SNMP-Agenten initialisiert wird. Um den Inhalt dieses Objektes nach dem Start des Agenten auf einen bestimmten Wert setzen zu können, wendet man den `Set`-Befehl an.
- **get trap:** Diese Operation wird genutzt, um einem oder mehreren Managementsystemen unaufgefordert mitzuteilen, dass ein speziell definierter Fall (eine Ausnahmesituation) eingetreten ist.
- **open MIB/get OID/get information from MIB:** Der Manager kann natürlich auch alle Informationen, sowie bestimmte OIDs von einer MIB abfragen.

Aktivitätsdiagramm Das Aktivitätsdiagramm aus Abb. 3.2 zeigt den Kommunikationsablauf von SNMP auf einem detaillierteren Niveau, als es im Anwendungsfalldiagramm der Fall ist. Zuerst muss eine Verbindung zum SNMP-Agenten erstellt werden. Ist die Verbindung aufgebaut, kann eine der folgenden Operationen durchgeführt werden:

- **get all OIDs (values):**
  Zuerst wird der erste OID ermittelt. Dann wird, solange ein Nachfolger-OID existiert, die Operation **get next** aufgerufen und der OID-Wert in einen Vektor gespeichert, welcher dann auch zurückgeliefert wird.
- **get OID value:**
  Zum angegebenen OID wird dessen Wert zurückgeliefert.
- **get next OID (value):**
  Zum angegebenen OID wird der Wert dessen Nachfolgers zurückgeliefert.

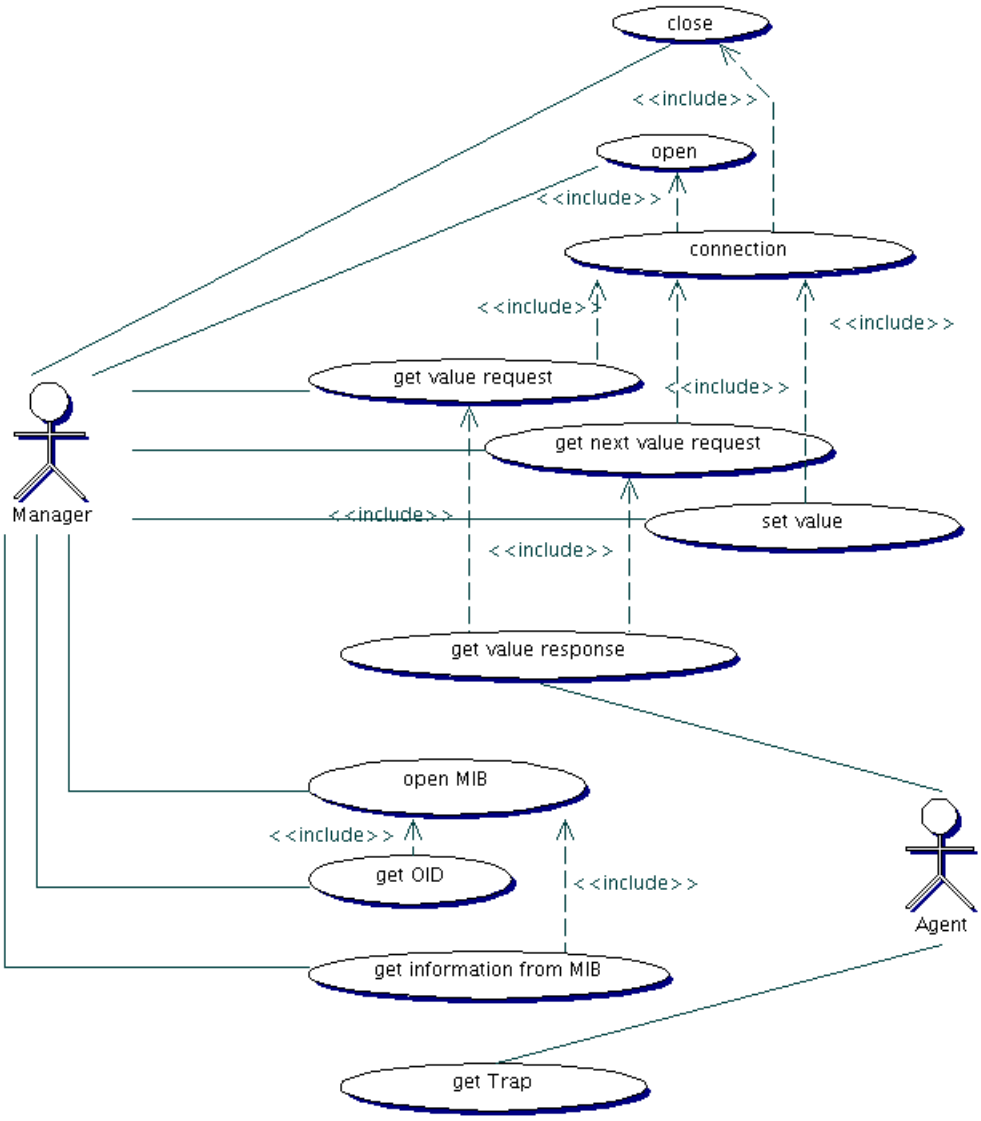

Abbildung 3.1: SNMP - Anwendungsfalldiagramm

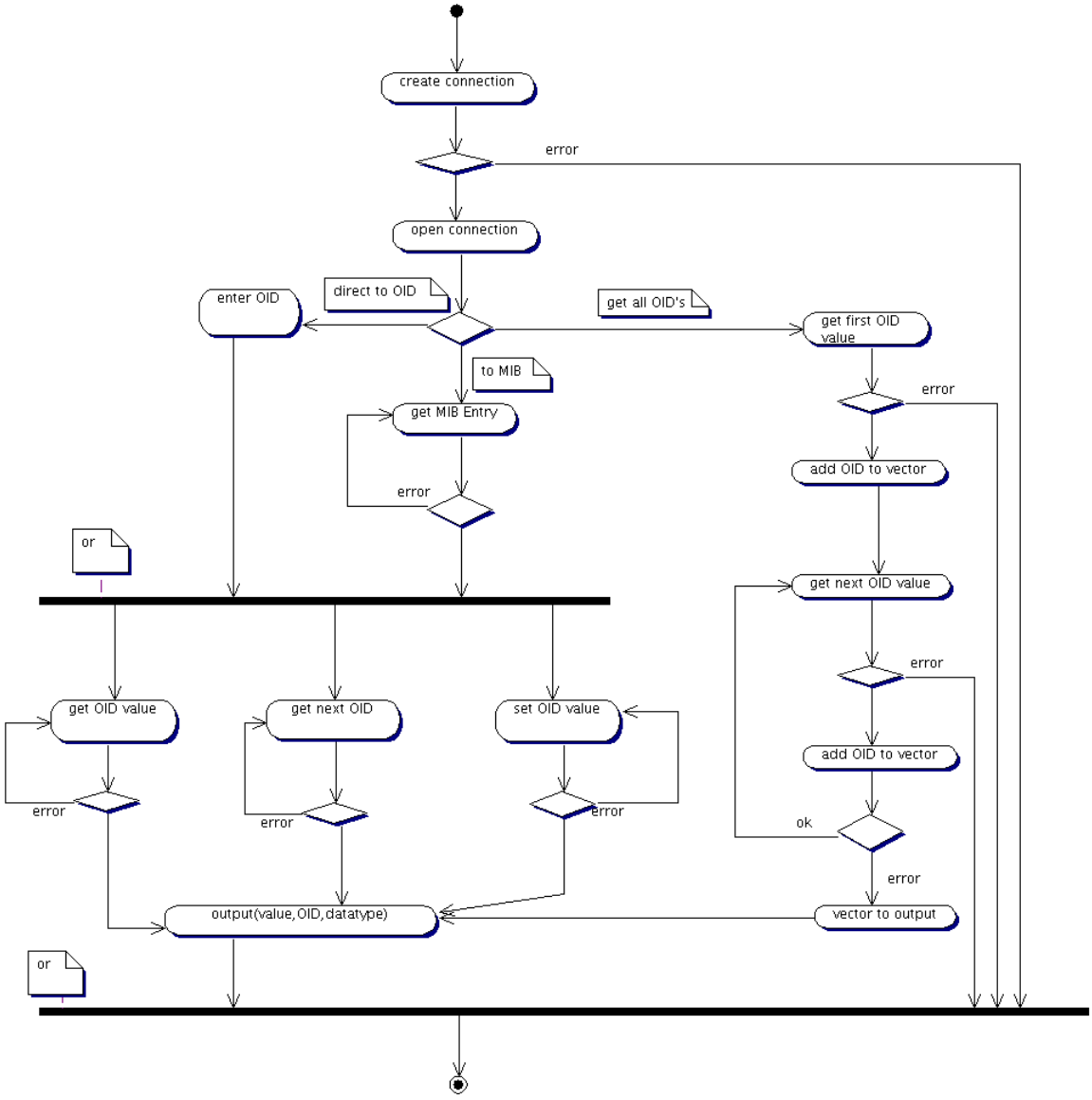

Abbildung 3.2: SNMP - Aktivitätsdiagramm

- **set OID value:**
  Dem angegebenen OID wird der angegebene Wert zugewiesen.

Sequenzdiagramm Das Sequenzdiagramm (siehe Abb. 3.3) zeigt beispielhaft den

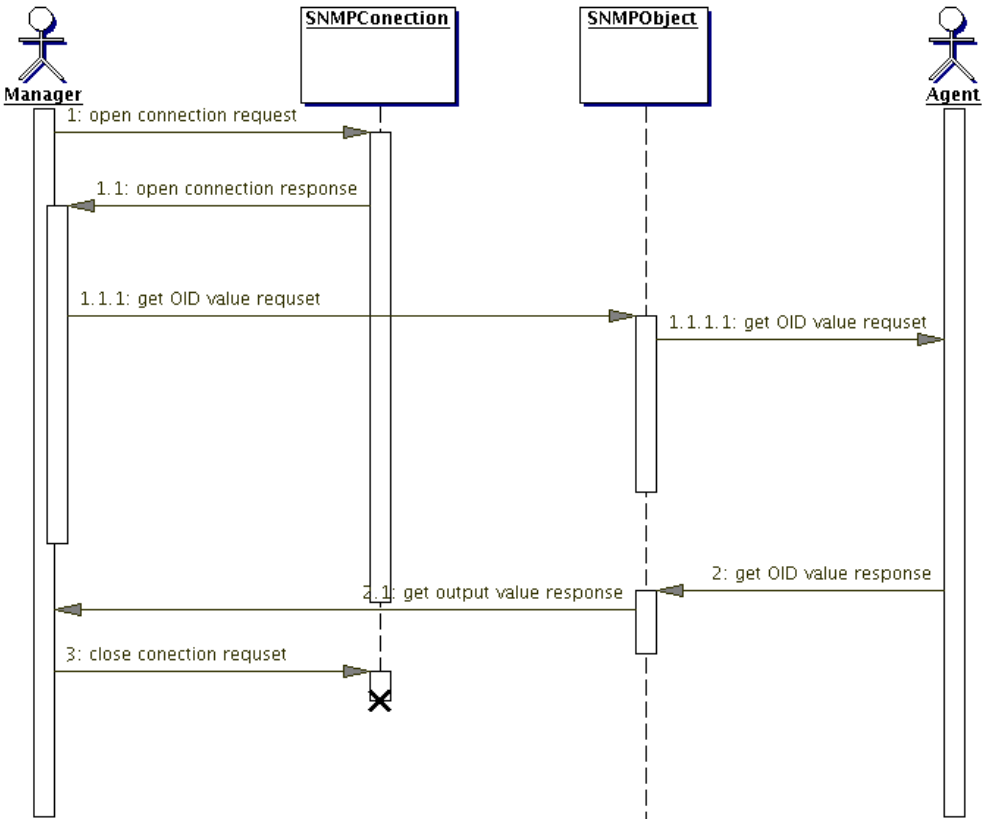

Abbildung 3.3: SNMP - Sequenzdiagramm

Ablauf der Get-OID-Value-Funktion. Zuerst muss eine Verbindung zum Agent aufgebaut werden. Dann wird die Get-OID-Value-Anfrage an den Agent geschickt. Anschließend bekommt der Manager eine Antwort zurück und die Verbindung kann geschlossen werden.

3.1.2 Implementierung

Unsere Implementierung basiert auf dem JAVA-Package für SNMP sowie der zugehörigen API, die von der GICL-Forschungsgruppe der Drexel University (Philadelphia, USA) entwickelt wurden.

Die von uns entwickelten SIBs unterstützen SNMP in der Version 1 bzw. RFC 1157. Die einzelnen Befehle des Protokolls sind auf einzelne SIBs abgebildet worden. So ist sicher gestellt, dass die SNMP-Befehle beliebig miteinander kombiniert werden können. Dabei konzentrierten wir uns auf die Unterstützung von grundlegenden SNMP-Funktionen:

- Aufbau einer Verbindung
- Auslesen von allen vorhandenen OID-Werten
- Auslesen eines OID-Wertes zu einem gegebenen OID
- Auslesen eines OID-Wertes zu einem gegebenen Vorgänger-OID (Funktion ”get next”)
- Setzen eines OID-Wertes zu einem gegebenen OID
- Abbau der Verbindung

Darüberhinaus ist ein SIB entwickelt worden, welcher den packagespezifischen Datentyp, bestehend aus einem Paar aus OID und zugehörigem OID-Wert, in zwei einzelne Objekte aufteilt - also den OID und den Wert.

Alle SIBs sind in einer SIB-Palette zusammen gefasst worden und stehen somit für den Einsatz in der Entwicklungsumgebung ABC zur Verfügung.

3.1.3 Tests

Die folgenden Teilgraphen zeigen einen möglichen Einsatz der entwickelten SIBs.

SNMP_openConnection:
Der Graph SNMP_openConnection (Abb. 3.4) beginnt mit den Initialisierungs-SIBs `Start` und `ShowFile`. Zur Herstellung der Verbindung sind die Adresse des verwalteten Gerätes sowie der Community-String anzugeben. Die SNMP-Version läßt sich direkt im SIB `SNMP_openConnection` einstellen. Diese Daten werden gespeichert.
Als nächstes wird der SIB `SNMP_openConnection-SNMP` ausgeführt. Dieser SIB stellt eine Verbindung zwischen dem SNMP-Manager und dem SNMP-Agenten her. Diese Verbindung wird in einer Session gespeichert.

SNMP_getValue:
In dem Teilgraphen SNMP_getValue (Abb. 3.5) wird der SIB `SNMP_getValue` vorgestellt. Zum angegebenen OID soll der dazugehörige Wert zurückgegeben werden. Zuerst wird die Verbindung aus der Session ausgelesen und zusammen mit den im SIB `ShowFile` eingegebenen Daten in den CallContext geschrieben. Danach wird der SIB `SNMP_getValue` ausgeführt. Mit dem SIB `ObjectToString` wird eine String-Repräsentation der Objekte zurückgeliefert.

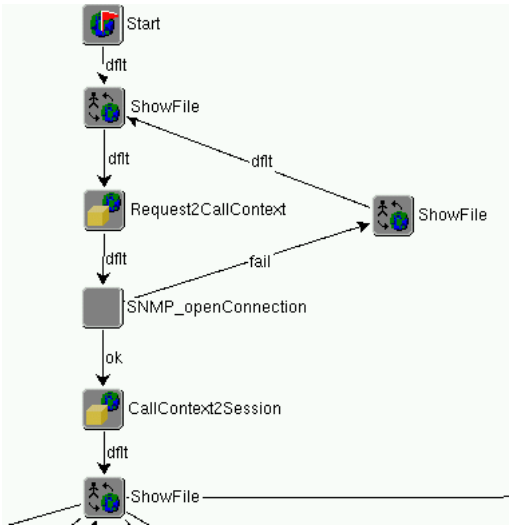

Abbildung 3.4: SNMP-openConnection

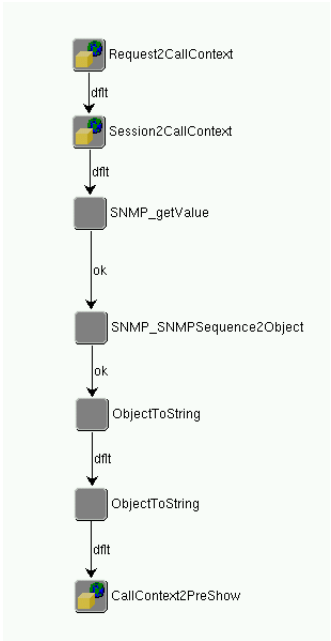

Abbildung 3.5: SNMP-getValue

SNMP_getNextValue:
Der SIB `SNMP_getNextValue` (Abb. 3.6) ist dem SIB `SNMP_getValue` ähnlich. Allerdings wird hier nicht der Wert des angegebenen OIDs, sondern seines Nachfolgers, zurückgegeben.

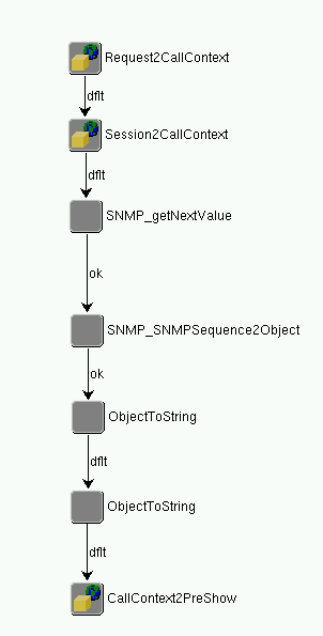

Abbildung 3.6: SNMP-getNextValue

SNMP_setValue:
Mittels des SIBs `SNMP_setValue` (Abb. 3.7) kann ein neues OID-Wert gesetzt werden. Zum Ändern eines OID-Wertes müssen an den SIB entsprechende CallContext-Schlüssel für den OID selbst, den neuen Wert sowie dessen Datentyp übergeben werden.

SNMP_getAllValues:
Mit dem SIB `SNMP_getAllValues` (Abb. 3.8) werden die Werte aller in dem verwalteten Gerät vorhandenen OIDs samt ihrer Werte ausgelesen und ausgegeben.

SNMP_closeConnection: Nachdem man die gewünschten Operationen durchgeführt hat, muss die Verbindung zu dem verwalteten Gerät geschlossen werden. Der SIB `SNMP_closeConnection` (Abb. 3.9) bekommt die zu schließende Verbindung übergeben und baut diese ab.

3.2 Automatisches Testing von Webservices

Unsere Aufgabe war es, eine SIB-Bibliothek für das Entwicklungstool ABC zu entwickeln die es möglich machen sollte, Webservices automatisch auf Erreichbarkeit und Funktio-

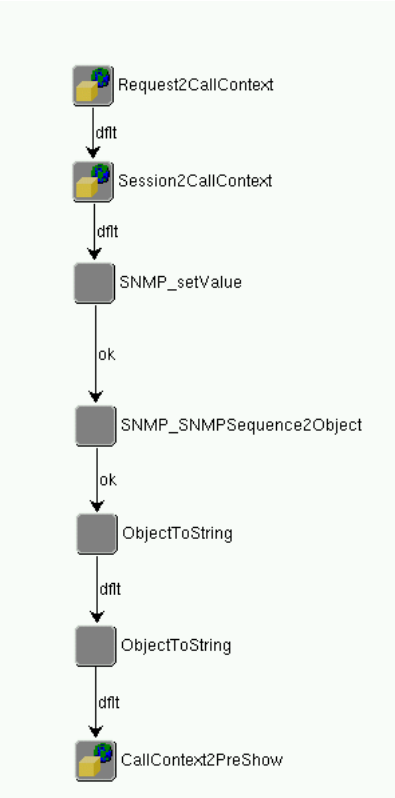

Abbildung 3.7: SNMP-setValue

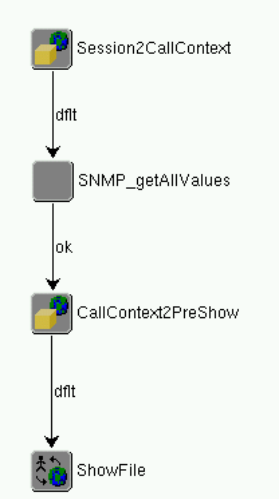

Abbildung 3.8: SNMP-getAllValues

nalität zu überprüfen. Es galt dabei, geeignete Kernfunktionalitäten eines solchen Tests in modularer Form als einzelne SIBs zu implementieren. Damit sollte es möglich sein, möglichst intuitiv einen Service Logic Graphen in ABC zu erstellen, welcher den gewünschten Webservice testet. Testen beinhaltet hier einerseits das Testen auf Erreichbarkeit und/oder auf die Korrektheit spezieller Funktionen. Erreichbarkeit kann durch einen einfachen http-request getestet werden, Tests der Funktionalität, zum Beispiel eines Logins, dagegen sind etwas komplexer. Fehler sollten dem Entwickler wenn möglich per email mitgeteilt werden und so ein schnelles Handeln erleichtern.

3.2.1 Modellierung

Die Modellierung unserer Aufgabe erfolgte in UML. Daher erstellten wir, von den von uns gesammelten Anwendungsfällen (siehe Abb.3.10) ausgehend Aktivitäts- und Sequenzdiagramme (siehe Abb.3.11 und Abb.3.12). Diese Diagramme verwendeten wir als Basis für die Implementierung.

Anwendungsfalldiagramm
Möchte der Entwickler eines Webservices diesen mit Hilfe unserer SIB-Bibliothek testen, so muss er als erstes eine Verbindung zu diesem Webservice aufbauen. Dies geschieht in der Regel durch das absetzen eines initialen http-requests. Hierzu muss ein URL-Objekt angelegt werden, welches Parameter wie URL, Benutzername, Passwort, http-Request-Methode etc. des Webservices enthält. Die Kapselung in dieses URL-Objekt ermöglicht die spätere Wiederverwendung ohne Neudefinition.
Das Ergebnis dieser Anfrage wird dann mit vordefinierten Werten, welche zur Entwicklungszeit des Testservices unter ABC festgelegt werden, verglichen. Dies geschieht hauptsächlich durch String-Vergleiche. Im Fehlerfall wie auch im Erfolgsfall besteht schon hier die Möglichkeit, den Entwickler über den aktuellen, evtl. fehlerhaften Zustand seines Systems zu informieren. Aufgrund der modularen Entwicklung mit ABC kann diese Benachrichtigung über beliebige Kommunikationswege erfolgen, welche in Form eines SIBs implementiert wurden. Derzeit existieren unter anderem SIBs für die Benachrichtigung per email und für die Benachrichtigung per Bildschirmausgabe.
Wenn die Verbindung zum Webservice steht, hat der Entwickler vier Möglichkeiten diesen zu testen.
Die einfachste Möglichkeit besteht in einer einfachen Suche nach einem vordefinierten String (`searchString`) innerhalb der angeforderten Webseite. So kann beispielsweise verifiziert werden, dass eine bestimmte Seite innerhalb eines Workflows angezeigt wird.
Desweiteren besteht die Möglichkeit, sich die auf der Webseite vorhandenen Links ausgeben zu lassen und mit diesen weiter zu navigieren (`getLinkByName`) oder Parameter aus zurückgegebenen Links zu extrahieren (`getLinkParameter`).
Ebenso können Metaparameter oder Formularparameter ausgelesen werden, um z.B. berechnete Eingaben zu überprüfen (`getMetaParameter`, `getFormParameter`).

Aktivitätsdiagramm

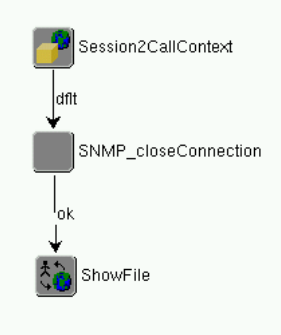

Abbildung 3.9: SNMP-closeConnection

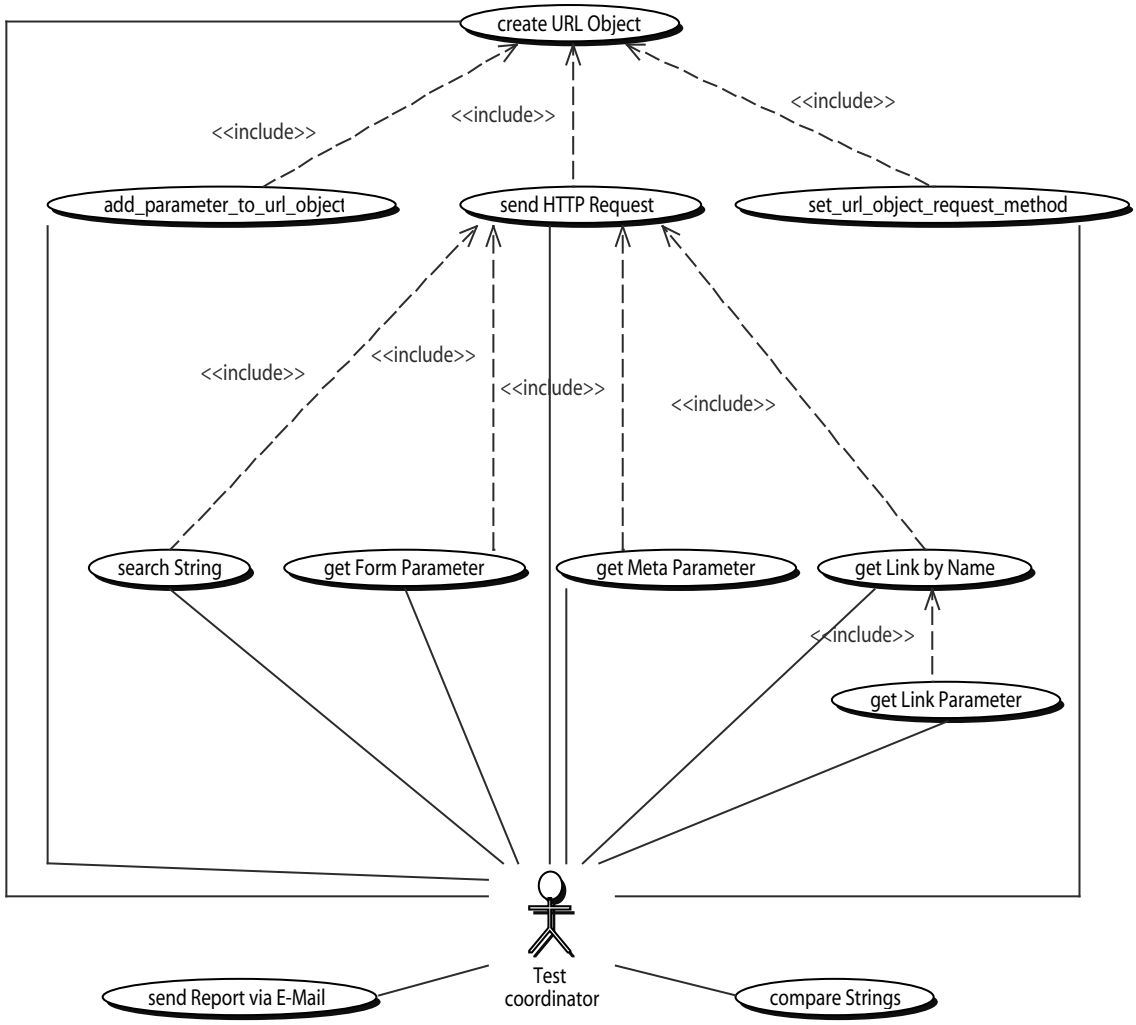

Abbildung 3.10: Webtesting - Anwendungsfalldiagramm Diagramm

Um den Ablauf eines Tests anschaulich zu machen, erstellten wir ein Aktivitätsdiagramm, in dem die Verwendung unserer SIBs und ihr Zusammenspiel inklusive der Terminierungszeitpunkte des Tests dargestellt sind.
Mit dem in Abb.3.11 dargestellten Aktivitätsdiagramm wäre das folgende Beispiel realisierbar.
Als erstes wird ein URL Objekt für den initialen http-request erzeugt. Hierbei kann es sich um eine triviale URL der Form http://www.meineServices.de/ServiceXYZ handeln. Nachdem dieser http-request abgefeuert wurde, kann bereits eine erste Aussage des Zustands des Webservices getroffen werden. Fehler können an dieser Stelle bereits durch einen Timeout des Requests oder durch einen, einem Fehler zugeordneten, http-response code erkannt werden. Wurde kein Fehler erkannt, kann mit den Tests fortgefahren werden. Es steht nun also fest, dass der Dienst erreichbar ist und eine gültige http-response empfangen wurde. Nicht bekannt ist bislang, ob es sich hierbei auch um die erwartete Webseite handelt. Da der Testentwickler sich mit dem Dienst auseinnandergesetzt hat ist ihm bekannt, dass als erstes eine Aufforderung zum Login zu sehen sein müsste. Dies wird nun mit einem `searchString` mit dem Parameter `"Bitte␣melden␣Sie␣sich␣an"` überprüft.
Nach diesem Test steht nun fest dass der Dienst erreichbar ist und die erwartete Aufforderung zum Login gesendet wird. Nun folgt der Test dem intuitiven Benutzerverhalten und simuliert einen Login. Hierzu wird ein neues URL Objekt erzeugt. Diesesmal handelt es sich bei der Zieladresse um das Login Skript, weswegen der Name und das Passwort als Parameter an das URL Objekt angehängt werden.
Nach dem erfolgreichen Login ist eine Seite mit einem Formular zu sehen. Das Formular enthält die Adressdaten der angemeldeten Person. Außerdem existiert ein Link, mit welchem ein Stadtplan aufgerufen wird welcher die Adresse der angemeldeten Person anzeigt. Ein denkbarer Test besteht nun darin, die Straßenangabe des Formulares mit der des Routenplaners zu vergleichen.
Aus dem Formular kann der Straßenname recht einfach mir einem `getFormParameter` gewonnen werden. Um an den Parameter des Links zu gelangen muss zunächst der Link beschafft werden. Dies geschieht mit einem `getLinkByName`. Anschließend kann der entsprechende Parameter mit `getLinkParameter` ausgelesen werden.
Waren diese Schritte erfolgreich können nun die beiden Parameter verglichen werden.
Dies ist nur ein kleines Beispiel für die Möglichkeiten, die die SIB Bibliothek den Entwicklern von Testservices bietet, die Webservices testen und überwachen müssen.

Anmerkung zu Abbildung 3.11: Der Übersicht halber wurde auf eine UML-konforme Verwendung des Endzustandes verzichtet und an den entsprechenden Stellen ein seperater Endzustand eingefügt.

Sequenzdiagramm
Das von uns erstellte Sequenzdiagramm zeigt nun den Ablauf eines Tests in einem temporalen und kausalen Zusammenhang. Hierbei sind die Verschachtelung innerhalb einzelner Funktionsaufrufe und die Lebenszeiten der einzelnen Objekte zu erkennen. Außerdem

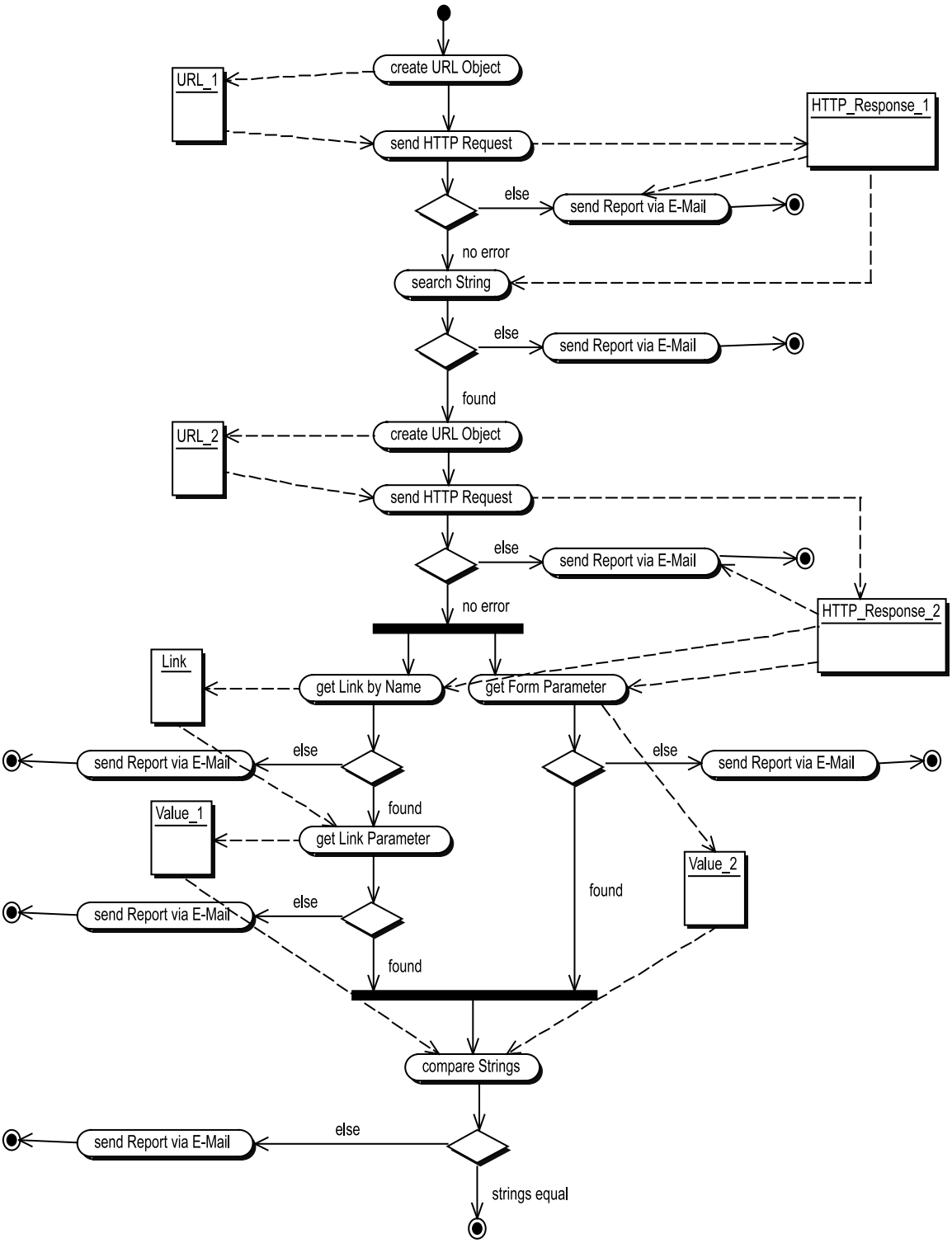

Abbildung 3.11: Webtesting - Aktivitätsdiagramm

wird deutlich, welche SIBs auf welche Funktionen und Objekte zugreifen.

Für tiefergehende Informationen zur Modellierung sei hier auf die Dokumentation ([11]) zu diesem Thema verwiesen.

3.2.2 Implementierung

Nachdem die Modellierung abgeschlossen und vorgestellt wurde, bestand der nächste Schritt darin, die dort beschriebene Funktionalität umzusetzen. Hierzu entwickelten wir, gemäß dem Design, eine Reihe von SIBs[2]. Da die gewünschte Funktionalität bereits in der Designphase festgelegt wurde, ging es in erster Linie darum, die 'Blackboxes' bzw. die Klassen und Methoden des Designs mit Programmcode zu füllen, um ein Verhalten zu erreichen, wie es im Design beschrieben ist.
Die Implementierung erfolgte in zwei Schritten.
Zunächst wurden die Gerüste der SIBs[2] mit deren Beschreibungsdateien erstellt. Hierbei war es erforderlich, alle benötigten Schnittstellen korrekt zu beschreiben. Die so erzeugten SIB-Strukturen[2] konnten schließlich ins ABC eingebunden und zum Bau eines Service-Logic-Graphen[10] verwendet werden, ohne dass deren eigentliche Funktionalität bis dahin implementiert wurde. Somit konnten Design- und Syntaxfehler bereits vor der Implementierung vorzeitig entdeckt und behoben werden. Die eigentliche Funktionalität wurde schließlich mit Hilfe des Eclipse[9] implementiert.
Zusätzlich zu den SIBs[2] war die Implementierung zweier weiterer Klassen für die korrekte Funktionalität erforderlich. Die Klasse `TestWWW_url` dient dabei als simpler Container für eine URL und ihre Parameter, während die Klasse `TestWWW_parser` ein HTML Dokument nach Links oder Formularen durchsuchen und diese zur weiteren Verwendung zurückgeben kann.
Um ein hohes Maß an Wiederverwendbarkeit gewährleisten zu können, entschieden wir uns für kleine SIBs[2] mit jeweils einfacher Funktionalität. Große SIBs[2] können zwar mehrere Funktionen übernehmen, wirken dann aber unübersichtlich, unhandlich und werden nicht gerne wiederverwendet.
Desweiteren wurden die SIBs[2] relativ unspezifisch gestaltet. In Folge dessen haben wir vollständig auf hard-codierte Variablenwerte verzichtet. Stattdessen werden Variablenwerte so gesetzt, dass man die Werte unter einem spezifischen Schlüsselnamen im Call-Context schreibt und den entsprechenden Schlüsselnamen an den SIB[2] übergibt.
Dieses Vorgehen macht zwar den Einsatz von zusätzlichen SIBs[2] erforderlich, um Werte in den Call-Context zu speichern, sorgt aber für die oben genannten Vorteile bei der Wiederverwendung.
Insgesamt entstanden auf diesem Weg neun SIBs[2], von denen vier SIBs[2] das Herstellen einer Verbindung ermöglichen sollen. Die übrigen fünf SIBs[2] dienen dann der eigentlichen Analyse der resultierenden Webseite bzw. dem eigentlichen Testen des Webservices.
Für den Verbindungsaufbau ist in erster Linie der Gebrauch des SIB `TestWWW_create_url_object` erforderlich. Wie der Name bereits verrät, wird auf

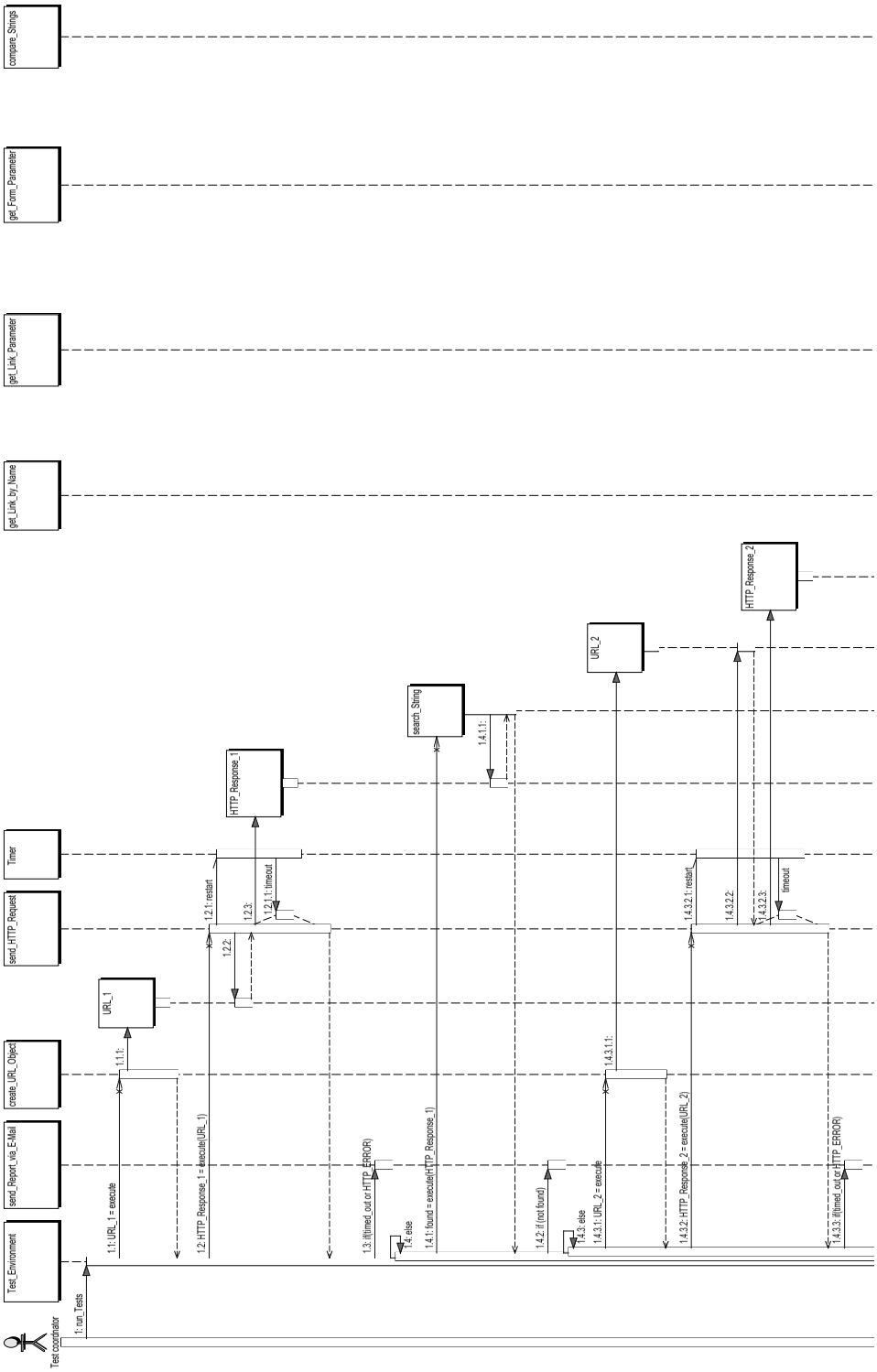

Abbildung 3.12: Webtesting - Sequenzdiagramm

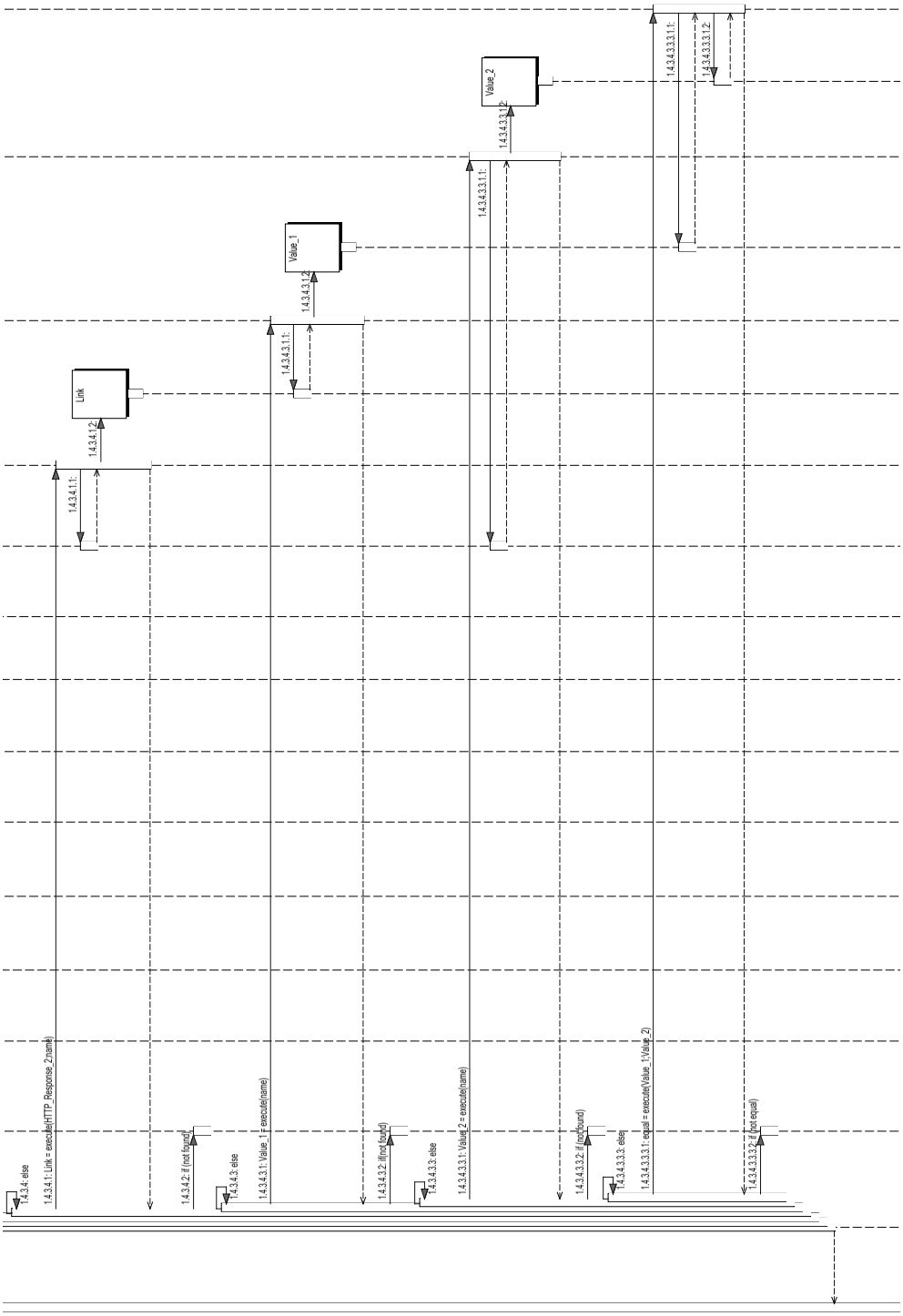

Abbildung 3.13: Webtesting - Sequenzdiagramm Teil2

diesem Weg ein URL-Objekt erzeugt, indem alle nötigen Informationen, zunächst einmal gesammelt werden, bevor der eigentliche Verbindungsaufbau beginnt. Name, Passwort und die URL-Adresse können direkt mit diesem SIB[2] gesetzt werden, wobei die Werte, wie oben beschrieben, aus dem Call-Context geholt werden. Wurde der SIB[2] fehlerfrei abgearbeitet, wird er über eine OK-Kante verlassen, andernfalls über eine fail-Kante.
Um dem entstandenen URL-Objekt Parameterpaare in Form von Name und Wert hinzuzufügen, wurde das SIB `TestWWW_add_parameter_to_url_object` implememtiert. Die Parameterpaare werden dann zusätzlich im URL-Objekt gespeichert und bei der späteren Webservice-Anfrage je nach Art der Anfrage-Methode mit übertragen.
Standardmäßig wird die Get-Methode verwendet. Um auch das Testen von Webdiensten zu ermöglichen, die lediglich auf Post-Methoden arbeiten, lässt sich die Methode mit dem SIB `TestWWW_set_url_object_request_method` zwischen 'Get' und 'Post' umschalten. Die entsprechende Übertragungsart der Parameter regelt das SIB[2] dann eigenständig. Wie zuvor werden auch diese beiden SIBs[2] nach erfolgreicher Abarbeitung über die OK-Kante und ansonsten über die fail-Kante verlassen.
Das eigentliche Kernstück des Verbindungsaufbaus wurde in dem SIB `TestWWW_send_http_request` implementiert. Es dient dazu, die Verbindung zu einem URL-Objekt herzustellen, dass, wie oben beschrieben, vollständig konfiguriert ist. Je nach Art der Antwort des getesteten Servers, legt das SIB[2] die Weichen zur Weiterbehandlung, indem es über entsprechende Ausgangskanten verlassen wird. Tritt beim Verbindungsaufbau beispielsweise ein Fehler auf Client-Seite auf, wird das SIB[2] über die error4xx-Kante verlassen. Entsprechendes gilt für einen Serverfehler (error5xx-Kante), einen Timeout (timeout-Kante) und einem allgemeinen Fehler (fail-Kante). Zusätzlich wird in jedem Fehlerfall auch die zugehörige detaillierte Fehlermeldung (wenn vorhanden) in dem Call-Context abgelegt, so dass sie bei Bedarf ausgelesen und verwendet werden kann. Im Falle einer fehlerfreien Abarbeitung des SIBs[2], wird dieses über die OK-Kante verlassen. Der vom getesteten Server erhaltene HTML-Code, also die resultierende Webseite, wird dann im Call-Context gespeichert und dient als Basis zur Inhaltsüberprüfung.
Der Inhalt der resultierenden Webseite kann je nach Bedarf überprüft werden. Wie bereits zu Beginn dieses Kapitels erwähnt, war es hierzu erforderlich, einen eigenen Parser zu entwickeln, um Teile des HTML-Codes ausfindig zu machen.
Fünf SIBs[2] stehen zur weiterführenden Inhaltsprüfung auf Service-Logic-Ebene zur Verfügung.
Um Form- oder Meta-Parameter aus dem HTML-Code herauslesen zu können, wurden die SIBs[2] `TestWWW_get_form_parameter` und `TestWWW_get_meta_parameter` erstellt. In beiden Fällen wird mittels des genannten Parsers nach den Parametern gesucht. Schließlich werden diese in den Call-Context geschrieben, und die SIBs[2] jeweils über die found-Kante verlassen. Werden die Parameter nicht gefunden, werden die SIBs[2] entsprechend über die not_found-Kante verlassen, im Fehlerfall über eine fail-Kante.
Um nach einem kompletten 'Link-Tag' innerhalb des HTML-Codes zu suchen und diesen zu extrahieren, entstand das SIB `TestWWW_get_link_by_name`. Hier sucht der Parser nach einem spezifischen 'Link-Tag' und speichert diesen im Call-Context. Je nach Erfolg

bei der Ausführung des SIBs[2], wird dieses in der Service-Logic-Ebene über die Kanten found, not_found oder fail verlassen.
Der extrahierte 'Link-Tag' kann im Anschluss erneut weiter überprüft werden. Dies ist mit dem SIB[2] `TestWWW_get_link_parameter` möglich, wobei nach einem Parameter innerhalb des extrahierten Codes gesucht wird. Entsprechend den vorherigen SIBs[2], werden die drei Ausgangskanten found, not_found und fail zur Verfügung gestellt.
Abschließend stellen wir mit dem SIB[2] `TestWWW_search_string` die Möglichkeit zur Verfügung, eine Zeichenfolge innerhalb eines Textes zu suchen - also einen String innerhalb eines anderen Strings. Dieses SIB[2] sollte die einfachste Art der Inhaltsüberprüfung abdecken.
Wie wir im folgenden Abschnitt sehen werden, lassen sich mittels dieser SIB-Bibliotheken[2] zahlreiche Testverfahren auf der Service-Logic-Ebene einfach beschreiben.

3.2.3 Tests

Die Testphase diente der Überprüfung und der Fehlerbehebung der implementierten Bibliotheken. An dieser Stelle möchten wir anhand eines Beispiels zeigen, wie sich unsere TestWWW-SIBs[2] auf Service-Logic-Ebene zu einem vollständigen Webservice-Test kombinieren lassen.
Abbildung 3.14 zeigt einen typischen ersten Testgraphen, der recht simpel und daher einfach zu verstehen ist. Der Graph baut eine Verbindung zu einer Suchmaschine auf, und übergibt dabei ein beliebiges Wort, wonach gesucht werden soll. Die resultierende Webseite, also diejenige, die die Suchergebnisse beinhaltet, wird dabei abgefangen und der Inhalt überprüft.
Beispielsweise könnte man der Suchmaschine das Wort 'Informatik Uni' übergeben und in der erhaltenen Webseite nach 'Uni-Dortmund' suchen, um sicherzustellen, dass 'Uni-Dortmund' der Suchmaschine bekannt und ein Teil der Antwort bezüglich der o.g. Anfrage ist.
Um das korrekte Arbeiten der SIBs[2] kontrollieren zu können, wurden drei einfache Webseiten erzeugt. Eine Seite zum Starten des Tests, eine weitere für die Ausgabe nach einem erfolgreichen Test, sowie eine Seite zur Ausgabe der Fehlerinformation, falls der Test gescheitert ist. Das besondere eines solchen Testgraphen besteht u.a. darin, dass man auch im Fehlerfall eigenständig festlegen kann, was als nächstes getan werden soll. So kann man beispielsweise auch bestimmen, ob und welche Fehlermeldung ausgegeben wird.
In dem Graphen beginnen wir typischerweise mit dem Start-SIB[2], gefolgt von einem ShowFile-SIB[2], dass die Start-Seite unseres Tests anzeigen soll. Wählt man auf dieser Startseite den entsprechenden Button aus, beginnt der Test.
Wie bereits im vorherigen Abschnitt erwähnt, arbeiten unsere SIBs[2] mit Werten aus dem Call-Context. Aus diesem Grund müssen die Adresse, der Username sowie das Passwort im nächsten Schritt in den Call-Context geschrieben werden, damit das URL-Objekt im

übernächsten Schritt generiert werden kann. In der Service-Logic-Ebene ist es hierzu wichtig, identische Schlüsselnamen für den Call-Context zu verwenden. Tritt beim Erstellen des URL-Objektes ein Fehler auf, so sieht man, dass ein String festgelegt und im Call-Context gespeichert wird, der als Fehlermeldung ausgegeben werden soll. Letzteres erfolgt erneut mit dem ShowFile-SIB[2], wobei in diesem Fall die Error-Seite angezeigt wird. Da die Fehlermeldung innerhalb der Error-Seite dynamisch ist, ist es erforderlich, die vorher festgelegte Fehlermeldung in den PreShow zu transportieren.
Entsprechend wird beim Festlegen der Anfrage-Methode, beim Hinzufügen von Parametern sowie bei der eigentlichen Anfrage selbst vorgegangen. Im Falle eines 4xx- oder 5xx-Fehlers ist es nicht zwingend erforderlich, selbst eine Fehlermeldung festzulegen, da das SIB `TestWWW_send_http_request` eigenständig Fehlermeldungen im Call-Context ablegt, auf die hier direkt zugegriffen werden kann.
Wurde die Anfrage erfolgreich abgefeuert, kann nun in dem resultierenden HTML-Code gesucht werden. Hierzu ist es erneut erforderlich, den zu suchenden String zunächst in den Call-Context zu speichern und auf identische Schlüsselnamen zu achten. Wurde der String gefunden, war der Test erfolgreich und die entsprechende Webseite kann angezeigt werden.

3.3 Konvertierung von Konfigurationsdateien nach XML

Konfigurationsdateien werden in der Regel in einem nicht einheitlichen Format gespeichert. Dabei werden für die Analyse bzw. Modifikation solcher Dateien jeweils spezielle Parser benötigt.
Eine Idee der MaTRICS ist es, beliebige Konfigurationsdateien zu verarbeiten. Dazu müssen zuerst Konfigurationsdateien in einem einheitlichen Format (XML) übersetzt werden. In diesem Format lassen sich dann alle nötigen Modifikationen syntaktisch korrekt durchführen. Vor der eigentlichen Übertragung einer Konfigurationsdatei zum ConfigClient muss die XML-Beschreibung mittels XSLT in das ursprüngliche Format der Konfigurationsdatei zurück konvertiert werden.

Die Aufgabe bestand zum einem im Design eines allgemeinen Parsers zur Erstellung der XML-Beschreibung aus einer Konfigurationsdatei, zum anderen war ein Mechanismus mittels XSLT zur Rück-Transformation dieser XML-Beschreibung zu modellieren.
Dabei waren die eigentlichen Funktionalitäten in Form von SIBs zu implementieren. Die Verwendung der einzelnen SIBs für die verschiedenen Workflows sollte an Hand eines kleinen Beispiel SLGs demonstriert werden.
Das Design war unter Zuhilfenahme von UML zu erstellen. Der eigentliche Programmablauf war dabei in Form von Aktivitätsdiagrammen zu designen. Die Implementierung sollte mit Hilfe eines Klassendiagramms unterstüzt werden.
Als Abschluss war eine englische Dokumentation zu erstellen, die das eigentliche Design der SIB Bibliothek und eine Art Howto enthält.

Abbildung 3.14: Webtesting - Testgraph

3.3.1 Modellierung

Anwendungsfalldiagramm:

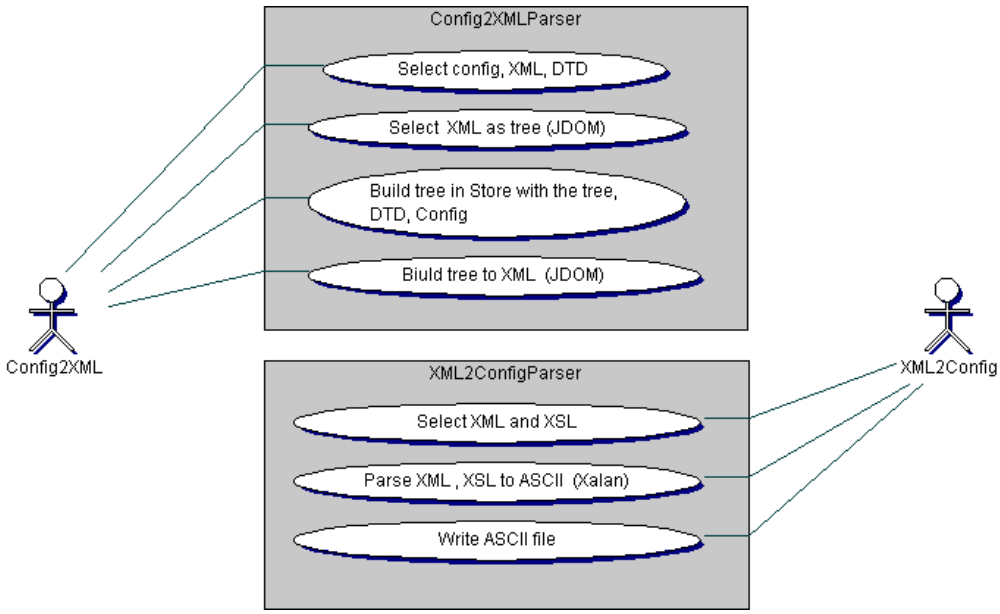

Abbildung 3.15: Konvertierung von Konfigurationsdateien nach XML - Anwendungsfalldiagramm

Das in Abbildung 3.15 zu sehende Anwendungsfalldiagramm zeigt einen groben Ablauf unserer Aufgabe.
Zu diesem Zeitpunkt der Designphase wurden erste Ansätze für die spätere Gestaltung unser Programmstruktur getroffen. Dabei kann das gesamte Projekt in zwei Fälle unterteilt werden, welche sich auch deutlich im Anwendungsfalldiagramm zeigen. In der ersten Phase (Config2XMLParser) werden die benötigten Daten (XML (Blockdefinitionen), Configdatei, DTD) eingelesen. Die aufgelisteten Dateien haben folgende Bedeutung für die Konvertierung:

- Die Configdatei ist die zu konvertierende Datei.
- Die DTD-Datei gibt die notwendigen Informationen über den Aufbau der zu erstellenden XML-Konfigurationsdatei und beschreibt, wie die zu erzeugende XML-Datei aussehen wird. Sie definiert somit den Namensraum der späteren XML-Datei.
- Die mitgelieferte XML-Datei (Blockdefinitionen) gibt Informationen über die Characters, Seperators wie Blockstart und Blockend, die in der Configdatei vorkommen. Sie dient somit als eine Art Syntaxbeschreibung der Configdatei.

Im Kapitel 3.4.4 haben wir eine genaue Auflistung und Beschreibung aller benötigten Dateien für Sie bereitgestellt.

Nun kann mit Hilfe der Blockdefinitionen die Configdatei in Ihre logischen Einheiten zerlegt werden. Anschließend dienen die Namenskonventionen in der DTD Datei zur Erzeugung des XML Dokument Baumes, welcher mit Hilfe der API JDOM in eine XML-Datei geschrieben wird.
Der zweite Fall (XML2ConfigParser) befasst sich mit der Rücktransformation der erstellten XML-Configdatei. Dazu werden die XML-Configdatei sowie ein XSL-Stylesheet (diese Datei gibt Informationen über die Vorlage und die Struktur der originalen Configdatei) ausgelesen. Anschliessend wird die XML-Configdatei mit Hilfe der API XALAN in ihr ursprüngliches Format zurück transformiert.

Zum Abschluss dieser Phase wird die rücktransformierte XML-Configdatei zurückgeschrieben und ersetzt die ehemalige Configdatei. Die genauen Abläufe und technischen Hintergründe entnehmen Sie bitte dem Bericht der Implementierungsphase aus Kapitel 3.4.3.

Aktivitätsdiagramm:
Das Aktivitätsdiagramm (Abb. 3.16) zeigt den Ablauf der Konvertierung einer Konfigurationsdatei nach XML. Die Rückkonvertierung ist mittels der API XALAN sehr einfach zu bewerkstelligen und wird somit hier nicht dargestellt.
Bei der Konvertierung einer Konfigurationsdatei nach XML werden zunächst die benötigten Hilfsdateien für die Blockdefinitionen und Namenskonventionen eingelesen. Anschliessend wird die zu konvertierende Konfigurationsdatei zeilenweise eingelesen und verarbeitet. An dieser Stelle können nun mehrere Fälle auftreten, die im folgenden kurz erläutert werden sollen:

- Wird ein Kommentarzeichen gefunden, so wird diese Zeile direkt als Kommentar markiert, in den XML Baum geschrieben und nicht weiter verarbeitet.
- Wird ein Blockstart gefunden, so wird eine passende ConfigGroup erzeugt und die Zeile wird weiter verarbeitet.
- Wird ein weiterer Blockstart gefunden, so wird der aktuelle Block auf einem Stack abgelegt, der neu gefundene Block wird zum aktuellen Block und die Zeile wird weiter verarbeitet.
- Wird ein Blockende gefunden, so wird der aktuelle Block geschlossen und ggf. der letzte noch auf dem Stack liegende Block zum aktuellen Block erklärt.
- Wird ein Element gefunden, so wird ein passendes ConfigElement erzeugt und die Zeile wird weiter verarbeitet.
- Wird ein Separator gefunden, so wird ein zu dem vor dem Separator liegenden Element passendes ConfigElement erzeugt und die Zeile wird weiter verarbeitet.

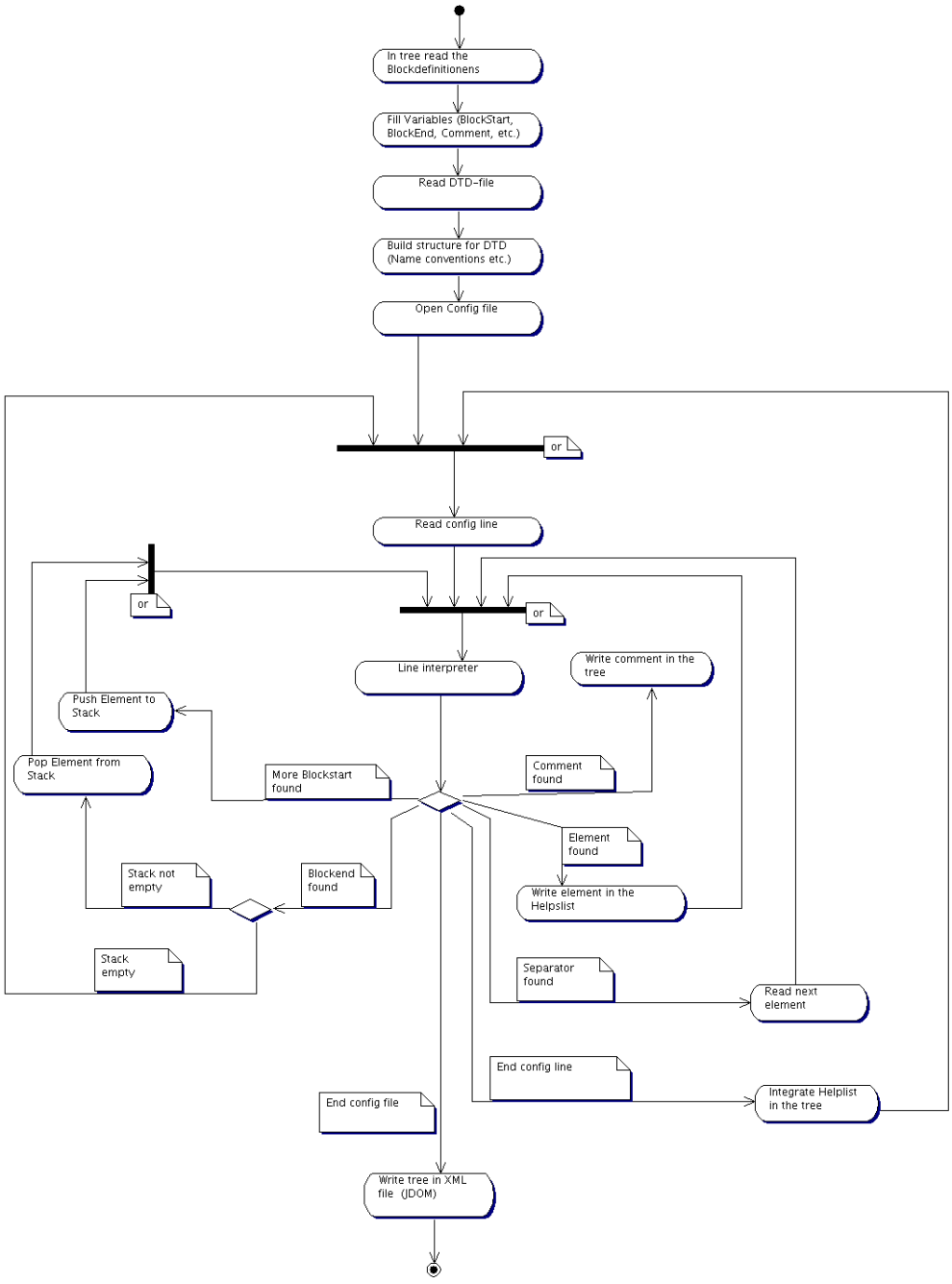

Abbildung 3.16: Konvertierung von Konfigurationsdateien nach XML - Aktivitätsdiagramm

- Ist die Zeile fertig verarbeitet, so wird die nächste Zeile eingelesen.

Der hier beschriebene Ablauf findet so lange statt, bis die Konfigurationsdatei komplett verarbeitet wurde. Abschließend (am Ende einer Configdatei) wird der erzeugte XML-Baum mittels JDOM in eine XML-Datei auf das Filesystem geschrieben. Eine detailliertere Beschreibung entnehmen Sie bitte dem Kapitel Implementierung (3.3.2).

Klassendiagramm:

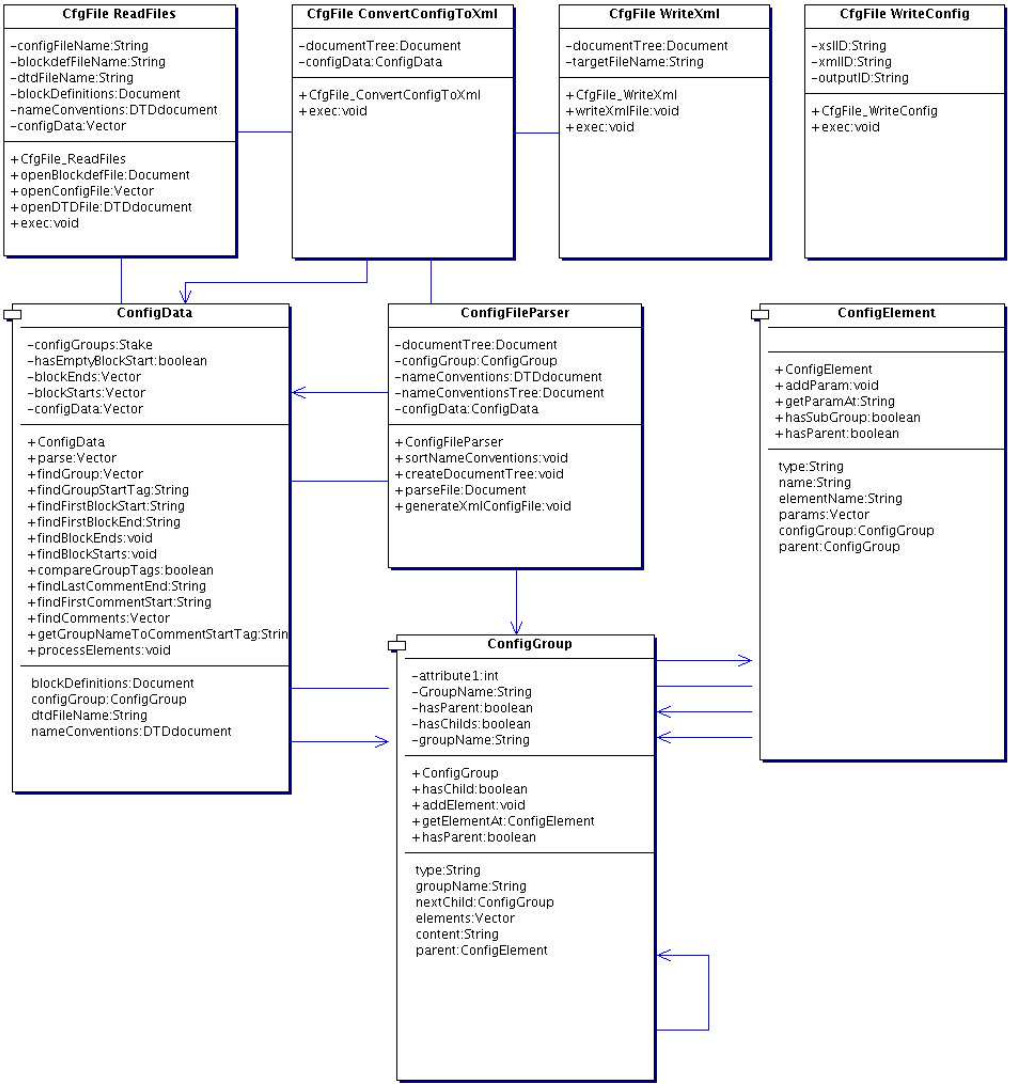

Abbildung 3.17: Konvertierung von Konfigurationsdateien nach XML - Klassendiagramm

Das Klassendiagramm, bzw. die Beschreibung der einzelnen Klassen und Methoden wird

in der Implementierungsphase vorgenommen. Das Klassendiagramm selber ist in der Abbildung 3.17 zu sehen.

3.3.2 Implementierung

Die Implementierung und der Ablauf des Moduls lassen sich in 3 Schritte gliedern:

1. Einlesen der erhaltenen Daten
2. Übertrag der Inhalte der Configdatei mit Hilfe der Blockdefinitionen in ihre Java-Container (ConfigElement, ConfigGroup)
3. Aufbau des XML-Baumes zusammen mit den Namenskonventionen

Zunächst müssen die XML-Blockdefinitionen und die DTD-Namenskonventionen eingelesen und nutzbar gemacht werden. Dies gelingt bei den Blockdefinitionen sehr leicht mit der API JDOM, da diese aus den erhaltenen Daten direkt einen Baum mit den gegebenen Inhalten erstellt. Das Einlesen der Namenskonventionen gestaltet sich jedoch als schwieriger, da hier statt JDOM die API Castor eingesetzt wird. Aus den dort erhaltenen Daten wird dann ein JDOM Document Tree aufgebaut, der später zusammen mit den Daten aus der Configdatei das Endergebnis bilden wird. Diese Aufgaben übernimmt dabei der ConfigFileParser.

Nachdem dem Modul nun alle benötigten Daten vorliegen, kann der Konvertierungsprozess beginnen. Dazu liest der Algorithmus die Configdatei nun zunächst zeilenweise ein. Jede gelesene Zeile wird anschließend auf Trennzeichen untersucht, die in den XML-Blockdefinitionen hinterlegt sind. Die einzelnen Teile werden schließlich in die passenden Container eingebettet (ConfigElement, ConfigGroup). Dabei werden die einzelnen ConfigElements in einem Vector verarbeitet, der zu der jeweiligen ConfigGroup zugeordnet ist. Sollten Untergruppen vorhanden sein, so wird die entsprechenden Untergruppe als Kind zu ihrem jeweiligen ConfigElement abgelegt. Dadurch entsteht ein genaues Abbild der Struktur (Verschachtelungen etc.), die in der Config Datei vorgegeben ist. Dieser Ablauf passiert in der Klasse ConfigData.

Zum Schluss muss die ausgelesene Struktur mit Hilfe der Namenskonventionen, die in der DTD-Datei vorgegeben sind, in den gewünschten XML Baum geparst werden. Dazu durchläuft der Algorithmus die erzeugte Baumstruktur (ConfigElement, ConfigGroup) und füllt mit diesen anhand der Namenskonventionen den finalen XML Document Tree. Es werden dabei zunächst die ConfigGroups zusammen mit ihren benötigten Unterstrukturen erzeugt, um anschließend darunter die ConfigElements mit den Inhalten aus der Configdatei zu füllen. Diese Aufgabe übernimmt dabei der ConfigFileParser.

Zu der Struktur der ConfigGroups und ConfigElements sei dabei noch folgendes gesagt: Sie dienen einerseits als Container für die gefundenen Elemente, bilden aber auch durch die Verknüpfung untereinander die semantische Struktur der gelesenen Configdatei ab.

Dazu beinhaltet jede ConfigGroup einen Zeiger auf Ihren Nachfolger (selbe Baumebene), der selbst wieder eine ConfigGroup ist, sowie einen Vector, der die ConfigElements dieser Gruppe enthält. Jedes ConfigElement kann dabei wiederum eine ConfigGroup als Unterelement haben. Dies stellt die Möglichkeit von verschachtelten Gruppen sicher, welches in der Abbildung 3.18 verdeutlicht wird.

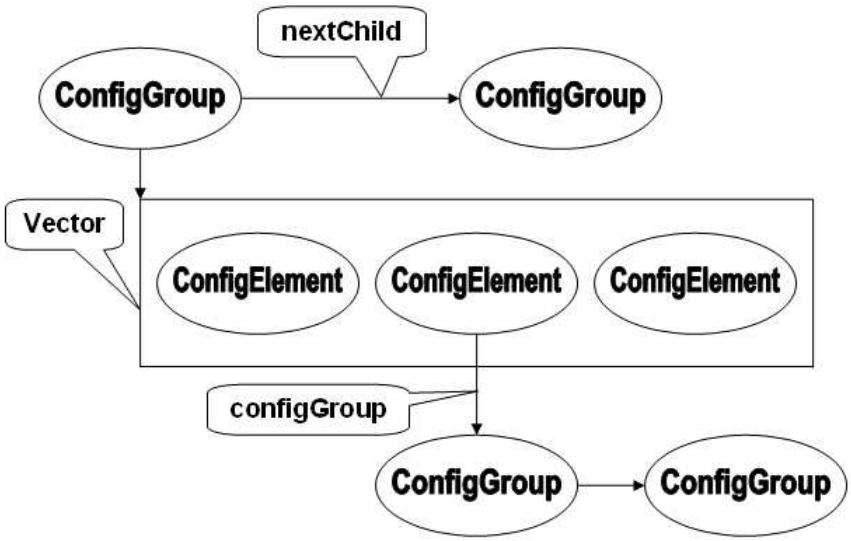

Abbildung 3.18: Konvertierung von Konfigurationsdateien nach XML - Datenstrukturschema

Etwaige Kommentare werden hierbei durch das type-Feld der beiden o.g. Klassen (ConfigElement, ConfigGroup) definiert. Die SIB-Klassen dienen dabei nur zur Ablaufsteuerung. Der eigentliche Algorithmus findet in den genannten Klassen statt. Dabei wird ein Objekt vom Typ ConfigData zwischen den SIB-Klassen weitergereicht, welches alle benötigten Daten enthält und so auch als Container dient.

3.3.3 Tests

Die Testphase zeigt die Korrektheit der Konvertierung von Configdateien nach XML-Dateien. Während der Implementierung wurde mit Hilfe von Testumgebungen der Quellcode ausgiebig getestet.
Um die Implementierung im ABC zu testen, wurde ein Workflow mit SIBs im ABC modelliert. In dem folgenden Workflow ist die Funktionsweise der Applikation in Zusammenhang mit der Implementierung dargestellt worden (Abb. 3.19).

Der Testgraph beinhaltet zwei Phasen. In der ersten Phase wird eine Konfigurationsdatei nach XML konvertiert. Die zweite Phase konvertiert die eben erzeugte XML-Datei zurück in das Ausgangsformat.

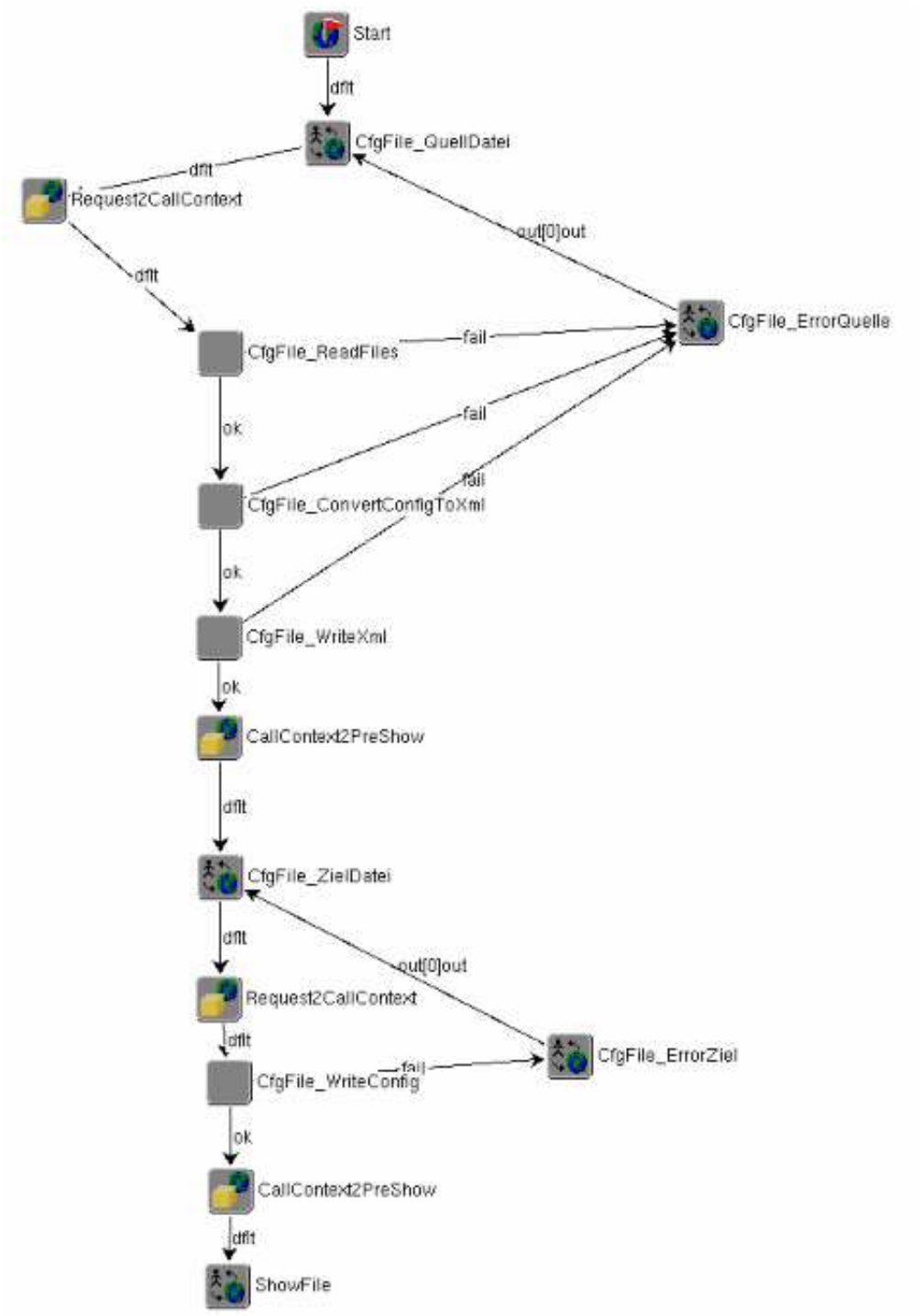

Abbildung 3.19: Konvertierung von Konfigurationsdateien nach XML - Testgraph

Der Testgraph startet mit der Angabe der Quelle der zu konvertierenden Konfigurationsdatei, der benötigten Hilfsdateien (Blockdefinitionen, Namenskonventionen) und des späteren Zieles für die konvertierte (XML) Datei. Die Eingabe geschieht am SIB `CfgFile_QuellDatei`. Sollten die angegebenen Dateien vorhanden sein, so werden diese eingelesen. Diese Aufgabe übernimmt der SIB `CfgFile_ReadFiles`. Anschliessend wird die angegebene Datei konvertiert (`CfgFile_ConvertConfigToXml`) und an das ausgewählte Ziel geschrieben (`CfgFile_WriteXml`).

Im Anschluss an die Konvertierung kann der Nutzer nun das Ziel der zurückzukonvertierenden Datei und des dazu nötigen XSL-Stylesheets auswählen. Die geschieht am SIB `CfgFile_ZielDatei`. Anschliessend wird die Datei in ihr ursprüngliches Format zurückkonvertiert und auf das angegebene Ziel geschrieben (`CfgFile_WriteConfig`).

Um die Implementierung zu testen, wurde der Testgraph aus Abb. 3.19 verwendet.

Folgende Datein werden benutzt:

1. Config-Datei passwd.conf:
   Ein einfaches Beispiel einer Config-Passwd-Datei

```
root:X:0:0:admin:/home/root:/bin/bash
name:password:user_id:group_id:info:homedir:shell
user:X:1:1:guest:/home/user:/bin/bash
#ich bin ein kommentar
```

2. DTD-Datei für die zu erzeugende passwd.xml:
   Ein Beispiel für eine DTD-Datei die beschreibt, wie die erzeugte XML-Datei aussehen wird.

```
<!- passwd.dtd ->
<!ELEMENT config (data+)>
<!ELEMENT data (group|group_comment)>
<!ELEMENT group (line+)>
<!ELEMENT line (element|line_comment)>
<!ELEMENT element (group_tag, name, passwd,
  user_id, group_id, info, homedir, shell) >
<!ELEMENT group_comment (#PCDATA)>
<!ELEMENT line_comment (#PCDATA)>
<!ELEMENT element_comment (#PCDATA)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT passwd (#PCDATA)>
<!ELEMENT user_id (#PCDATA)>
<!ELEMENT group_id (#PCDATA)>
<!ELEMENT info (#PCDATA)>
<!ELEMENT homedir (#PCDATA)>
<!ELEMENT shell (#PCDATA)>
```

3. XML-Datei blockdef.xml:
   Gibt Informationen über die Characters, Seperators wie Blockstart und Blockend, die in der Config-Datei vorkommen.

```
<?xml version="1.0" ?>
<!DOCTYPE blockdef PUBLIC "-//Sun Microsystems,
 Inc.//DTD Web Application 2.2//EN" "blockdef.dtd">
<blockdef>
  <block>
    <name>data</name>
    <start></start>
```

```
    <end></end>
    <comment>
      <comment_start></comment_start>
      <comment_end></comment_end>
    </comment>
    <separators>
      <element>:</element>
    </separators>
    <param>
      <param_start></param_start>
      <param_end></param_end>
      <param_separators>
        <param_element></param_element>
      </param_separators>
    </param>
  </block>
</blockdef>
```

4. DTD-Datei für blockdef.xml:
   Gibt ausführliche Informationen über die Struktur der Blockdef.xml und die Rheinfolge der Elemente.

```
<!- blockdef.dtd ->
<!ELEMENT blockdef (block+)>
<!ELEMENT block (name, start, end, comment+, separators, param)>
<!ELEMENT comment (comment_start, comment_end)>
<!ELEMENT separators (element+)>
<!ELEMENT param (param_start, param_end, param_separators)>
<!ELEMENT param_separators (param_element+)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT start (#PCDATA)>
<!ELEMENT end (#PCDATA)>
<!ELEMENT element (#PCDATA)>
<!ELEMENT comment_start (#PCDATA)>
<!ELEMENT comment_end (#PCDATA)>
<!ELEMENT param_start (#PCDATA)>
<!ELEMENT param_end (#PCDATA)>
<!ELEMENT param_element (#PCDATA)>
```

Die Konvertierung der XML-Datei zu einer Config-Datei
Folgend Dateien werden bei der Konvertierung der XML-Datei zu einer Config-Datei benutzt:

- XML-Datei(passwd.xml), die wieder nach Config-Datei konvertiert werden soll

```
<?xml version="1.0" ?>
<!DOCTYPE config PUBLIC
  "-//Sun Microsystems, Inc.//DTD Web Application 2.2//EN"
  "passwdd.dtd">
<config>
<data>
    <group_comment>
      <line>
        <line_comment>
              #name:password:user_id:group_id:info:homedir:shell
        </line_comment>
      </line>
    </group_comment>
  </data>
  <data>
    <group>
      <line>
        <element>
          <name>root</name>
          <passwd>x</passwd>
          <user_id>0</user_id>
          <group_id>0</group_id>
          <info>admin</info>
          <homedir>/home/root</homedir>
          <shell>/bin/bash</shell>
        </element>
      </line>
    </group>
  </data>
  <data>
    <group>
      <line>
        <element>
          <name>user</name>
          <passwd>x</passwd>
          <user_id>1</user_id>
          <group_id>1</group_id>
          <info>guest</info>
          <homedir>/home/root</homedir>
          <shell>/bin/bash</shell>
        </element>
```

```
      </line>
    </group>
  </data>

  <data>
    <group>
      <line>
        <line_comment>
  #ich bin ein kommentar
        </line_comment>
      </line>
    </group>
  </data>
</config>
```

- XSL-Datei(passwd.xsl), die Informationen über die Vorlage und die Struktur der Config-Datei gibt.

```
<?xml version="1.0"?>
<xsl:stylesheet
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="text"/>
<xsl:template match="config">
<xsl:apply-templates/>
</xsl:template>
<xsl:strip-space elements="data"/>
<xsl:strip-space elements="group"/>
<xsl:strip-space elements="line"/>
<xsl:template match="element">
<xsl:value-of select="name"/>
<xsl:element name="sep">:</xsl:element>
<xsl:value-of select="passwd"/>
<xsl:element name="sep">:</xsl:element>
<xsl:value-of select="user_id"/>
<xsl:element name="sep">:</xsl:element>
<xsl:value-of select="group_id"/>
<xsl:element name="sep">:</xsl:element>
<xsl:value-of select="info"/>
<xsl:element name="sep">:</xsl:element>
<xsl:value-of select="homedir"/>
<xsl:element name="sep">:</xsl:element>
<xsl:value-of select="shell"/>
</xsl:template>
</xsl:stylesheet>
```

3.4 Manipulation von XML-Dateien

Ein Bestandteil der MaTRICS ist es, alle Konfigurationsdateien in einem einheitlichem XML-Format zu verarbeiten. Dabei werden für die Modifikation dieser XML-Dateien Methoden benötigt, welche die in XML-Format vorliegenden Dateien, modifizieren können.

Die Aufgabe besteht darin, eine allgemeine SIB-Bibliothek zur Verarbeitung von XML-Dateien zu erstellen. Die Funktionalität besteht aus den gängigen Manipulationsoperationen, wie beispielweise das Ändern eines Tags oder das Löschen bzw. Einfügen eines Blocks. Das Design ist unter Zuhilfenahme von UML zu erstellen. Die eigentlichen Funktionalitäten sind in Form von SIBs zu implementieren. Die Verwendung der einzelnen SIBs für die verschiedenen Workflows sind im Kapitel 3.4.3 Tests anhand eines Testgraphen (Abb.3.23) dargestellt.

3.4.1 Modellierung

Anwendungsfalldiagramm
Bei diesem Thema geht es um die Manipulation von XML-Dateien. Die Aufgabe besteht

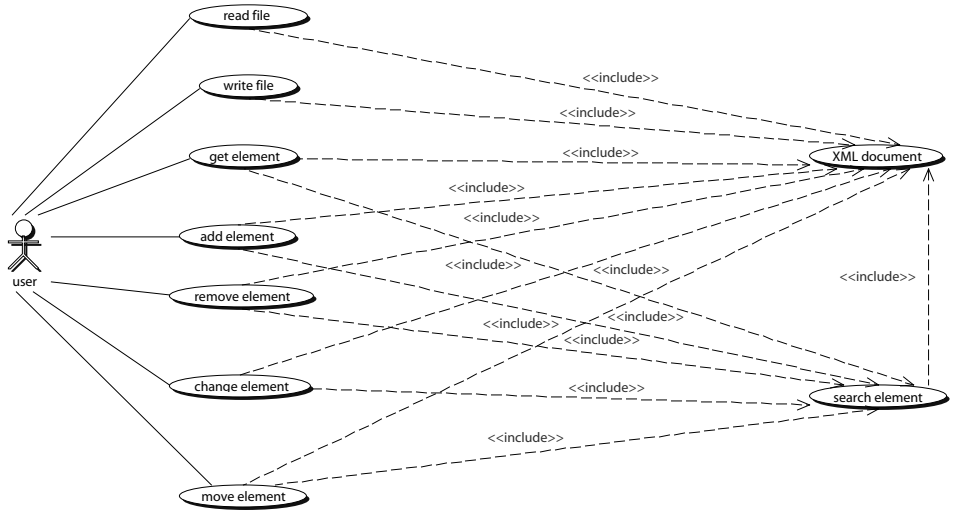

Abbildung 3.20: XML_Manipulation: Anwendungsfalldiagramm der XML-Manipulation

im Erstellen einer SIB-Bibliothek zur Verarbeitung von XML-Dateien. Zum Beispiel kann ein Benutzer durch unsere Web-Applikation ein Element innerhalb einer XML-Datei löschen, änderen, verschieben oder ein neues Element einfügen. So besteht unsere

Web-Applikation aus folgenden Anwendugsfälle, `read file`, `write file`, `get element`, `remove element`, `change element`, `move element` und `add element`. Für die Implementierung der SIBs, die die einzelnen Funktionalitäten zur Manipulation von einer XML-Datei bereitstellen, benutzen wir die API JDOM. Innerhalb unserer Web-Applikation wird zuerst eine XML-Datei gelesen und geparst, was dem Anwendungsfall, `read file` entspricht. In diesem Schritt wird zusätzlich aus der gegebenen Datei ein JDOM-Document erzeugt. Dann kann ein Benutzer durch die anderen einzelnen Anwendungsfälle dieses Document-Objekt manipulieren. Nach der Manipulation kann man durch den Anwendungsfall, `write file` die alte XML-Datei überschreiben.

Aktivitätsdiagramm

Im jeden einzelnen Workflow zur Manipulation eines Document-Objektes, kann ein XML-Element gefunden werden, indem man den XPath von diesem XML-Element angibt. Der XPath ist eine Sprache, die den Weg beschreibt, um eine bestimmte Komponente in einem JDOM-Document zu finden. Wenn es kein solches Element gibt, wird eine Fehlermeldung in einer HTML-Seite angezeigt und der Workflow wird beendet. Danach hat man die Möchlichkeit die Bearbeitung erneut auszuführen.

1. read file: In diesem Workflow gibt ein Benutzer zuerst einen Dateiname oder eine URL für eine XML-Datei in ein HTML-Formular ein. Dann wird diese XML-Datei geparst, und mittels JDOM ein Objekt vom Typ Document erzeugt. Danach kann dieses Document manipuliert werden.

2. get element, remove element, change element, add element:

   Unsere Web-Applikation bietet dem Benutzer eine Schnittstelle an, mit der er eine von verschiedenen Aktionen zur Manipulation einer XML-Datei auswählen kann. In jeder einzelnen Aktion wird zuerst das gewünschte Element durch den XPath gefunden, welches man dann löschen, ändern, verschieben kann oder unter dem ein neues Element einfügen werden soll. Wenn dieses Element existiert, wird zuerst dieses Element in einer HTML-Seite angezeigt. Dann kann man eine der verfügbaren Aktionen ausführen. Zum Beispiel kann man dieses Element mit einem neuen Name, einem neuen Attribute oder einem neuem Text versehen.
   Oder man gibt unter diesem Element in Form von Indizes die gewüschte Position ein, in die ein neues Element eingefügt werden soll, und man kann danach auch die Attribute und den Text für dieses neue Element einfügen.
   Im Folgenden wird das Verschieben eines Elementes beschrieben.

3. move element: Durch diesen Workflow kann ein Benutzer ein XML-Element in einem JDOM-Document an eine andere Position verschieben. Zuerst muss das erste Element, das der Benutzer verschieben möchte, gefunden werden. Danach muss das zweite Element nämelich das Ziel Element auch gefunden werden, unter das der Benutzer das erste Element verschieben möchte. Wenn diese beiden Elemente existieren, werden zunächt diese Elemente gezeigt. Dann gibt der Benutzer in Form

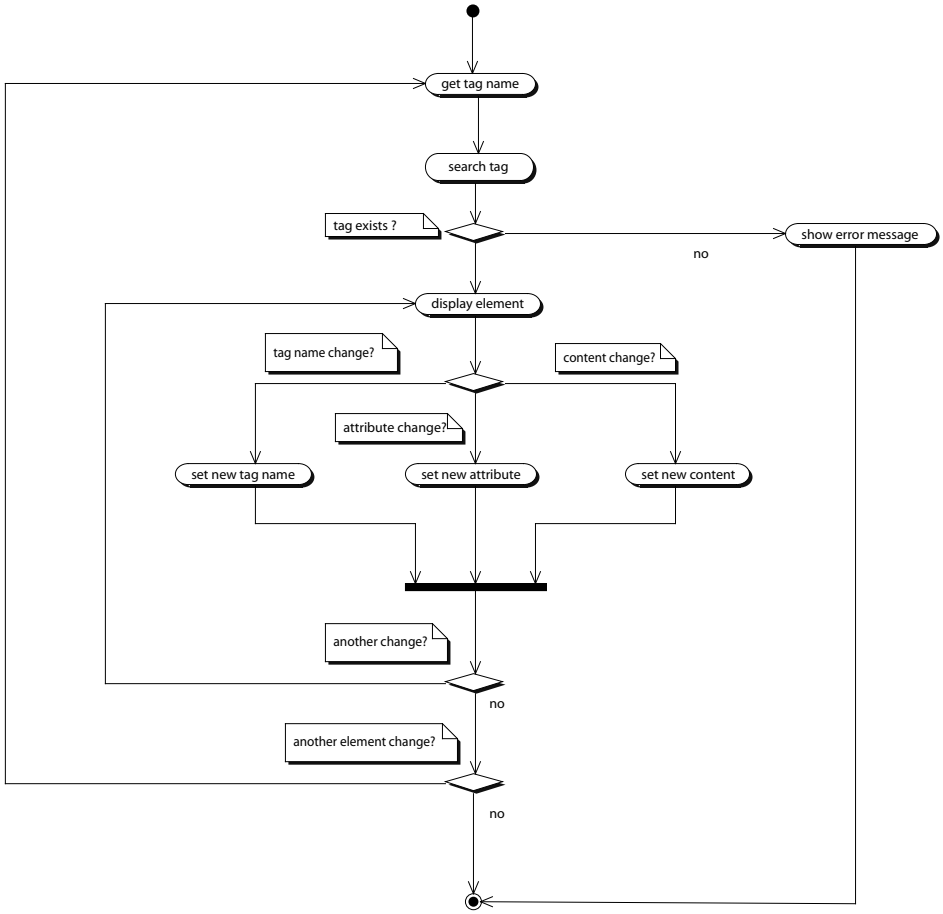

Abbildung 3.21: XML_Manipulation: Aktivitätsdiagramm des Ändern von Elementen

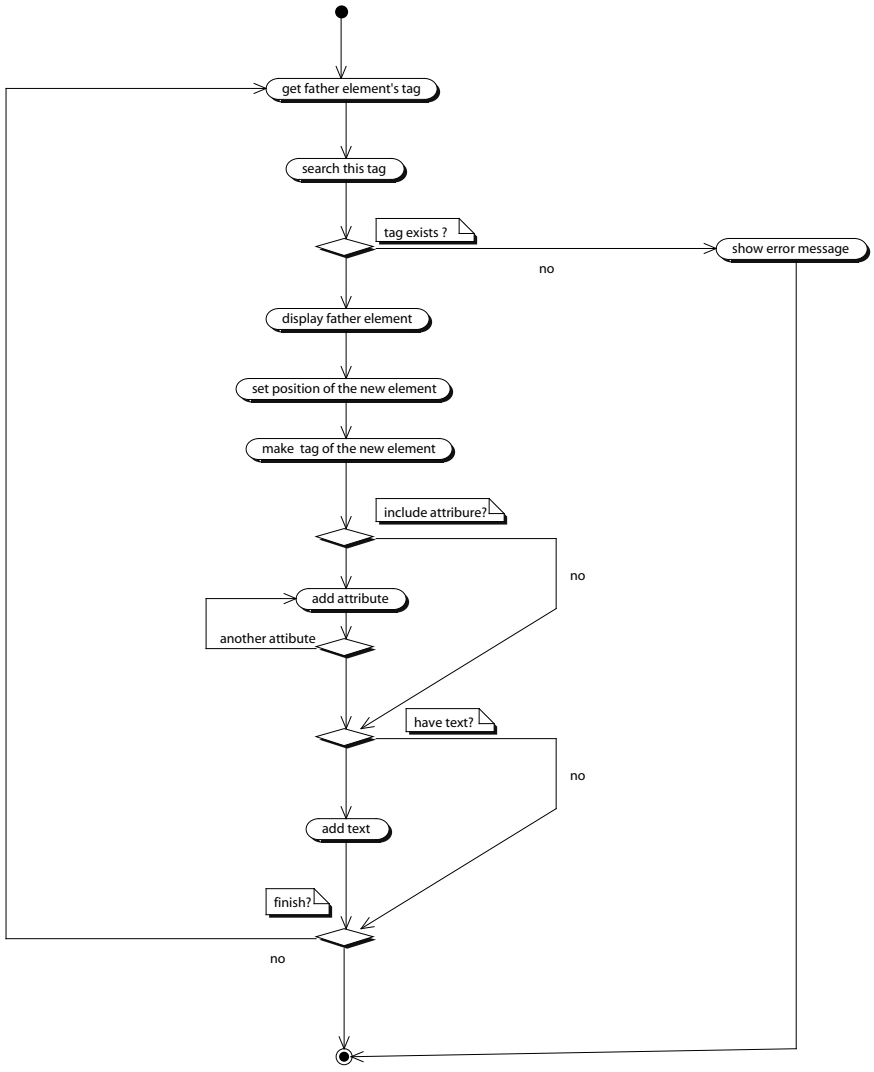

Abbildung 3.22: XML_Manipulation: Aktivitätsdiagramm des Einfügen von Elementen

von Indizes die gewüschte Position unter dem zweiten Element, wohin der Benutzer das erste Element verschieben möchte, ein. Danach wird das erste Element in diese gewünschte Position unter dem Ziel-Element verschoben.

4. write file: Nach der Manipulation von einem JDOM-Document kann durch diesen Workflow die alte XML-Datei überschrieben werden.

3.4.2 Implementierung

Die hier implementierte SIB-Bibliothek besteht aus insgesamt siebenundzwanzig SIBs. Die SIBs realisieren die elementaren Funktionen für die Manipulation von XML-Dateien und basieren auf der API JDOM. An dieser Stelle wurde eine kleine Auswahl von SIBs getroffen, um eine Aufgabe der Verarbeitung einer XML-Datei zu realisieren, z.B. ein Element löschen, ein neues Element einfügen oder ein Element verändern.

Die Benutzung der SIBs ist für den Benutzer einheitlich gestaltet. Zu Beginn wird die zu verarbeitende XML-Datei vom Benutzer in ein Formular eingegeben. Dabei kann der Benutzer sich zwischen zwei Optionen entscheiden. Entweder gibt er eine URL oder einen Pfad inklusive Dateinamen ein. Der Dateiname oder die URL wird nach der Eingabe als String in den CallContext und anschliessend in den SessionContext geschrieben. Dieser Schritt gewährleistet, dass wir später nach der Verarbeitung durch den SIB `XML_outputXMLFile` die alte Datei überschreiben können. Nachdem der Benutzer den Name der XML-Datei eingegeben hat, können wir den SIB `XML_parseXML` nutzen, um die XML-Datei zu parsen. Nach dem Parsen bekommen wir ein JDOM-Dokument zurück. Das JDOM-Dokument wird auch zuerst in den CallContext und danach in den SessionContext geschrieben, da man es während der Manipulation einer XML-Datei mehrfach benötigt. Das sogenannte Root-Element stellt ein zentrales Element dar. Durch das Root-Element und den XPath kann man ein beliebiges Element des JDOM-Dokumentes finden. Das Root-Element ist die Wurzel des JDOM-Dokumentes und wird mittels `XML_getRootElement` geholt und für spätere Verwendung in den CallContext und den SessionContext geschrieben. Ein XPath gibt einen Pfad für jedes Element in einen JDOM-Dokument ausgehend vom Root-Element an. Anhand des folgenden XML-Datei-Beispieles wird die bedeutung des XPath verdeutlicht.

```
<?xml version="1.0" ?>
<!DOCTYPE blockdef PUBLIC "-//Sun Microsystems,
 Inc.//DTD Web Application 2.2//EN" "blockdef.dtd">
<blockdef>
  <block>
    <name>data</name>
    <start></start>
    <end></end>
    <comment>
```

```
      <comment_start></comment_start>
      <comment_end></comment_end>
    </comment>
    <separators>
      <element>:</element>
    </separators>
    <param>
      <param_start></param_start>
      <param_end></param_end>
      <param_separators>
        <param_element></param_element>
      </param_separators>
    </param>
  </block>
</blockdef>
```

Das alles umgebende Element (hier : blockdef) wird das ”Root-Element” genannt. Der XPath für das Element ”comment_start” ist ”`/blockdef/block/comment/comment_start`”.

Ein Benutzer kann nun zwischen den verschiedenen Manipulationsaktionen wählen. Es stehen dabei Aktionen wie z.B. ein bestimmtes Element suchen, ein bestimmtes Element löschen, ein neues Element einfügen, ein bestimmtes Element verändern, oder ein bestimmtes Element verschieben zur Verfügung.

Durch die SIBs aus der SIB-Bibliothek können wir den Workflow für die Manipulationen einer XML-Datei aufbauen. Nach jeder Aktion wird nur das JDOM-Dokument verändert und nicht die XML-Datei selbst. Deswegen nutzen wir am Ende den SIB `XML_outputXMLFile`, um die alte XML-Datei zu überschreiben.

Als Beispiel wird hier der SIB `XML_addElement` vorgestellt.

Mit Hilfe dieses SIBs kann man ein Kind-Element in eine gegebene Position unter einem Vater-Element einfügen. Die Position für das Kind-Element wird als Index eingegeben, z.B. wenn das Vater-Element am Anfang drei Kind-Elemente hat, gibt es demnach vier mögliche Positionen für das neue Kind-Element. Diese vier Positionen können mittels der Indizes 0, 1, 2 und 3 bestimmt werden. Dabei positioniert der Index ”0” das neue Kind-Element vor dem vorhandenen ersten Kind-Element und der Index ”3” entsprechend nach dem letzten. Der Typ des Indizes aus dem CallContext ist ein String. Im Java Code des SIBs wird der Typ des Indizes in den Typ Integer geändert. Der SIB hat folgende Parameter: `element_to_add_key(String)`, `father_element_key(String)`, `index_key(String)` und Verzweigungen: *ok*, *fail*.

Die andere sechsundzwanzig SIBs in der SIB-Bibliothek sind:

XML_changeElementName	**XML_getIndexFromChild**
XML_convertObject2string	**XML_getRootElement**
XML_createAttribute	**XML_getText**
XML_createElement	**XML_moveElement**
XML_findElementByPath	**XML_newIndex4Choice**
XML_getAttribute	**XML_outputXMLFile**
XML_getAttributeName	**XML_parseXML**
XML_Attributes	**XML_removeAttribute**
XML_getAttributeValue	**XML_removeElement**
XML_getChildrenList	**XML_setAttribute**
XML_getElementAtIndex	**XML_setAttributeName**
XML_getElementFromList	**XML_setAttributeValue**
XML_getFatherElement	**XML_setText**

3.4.3 Tests

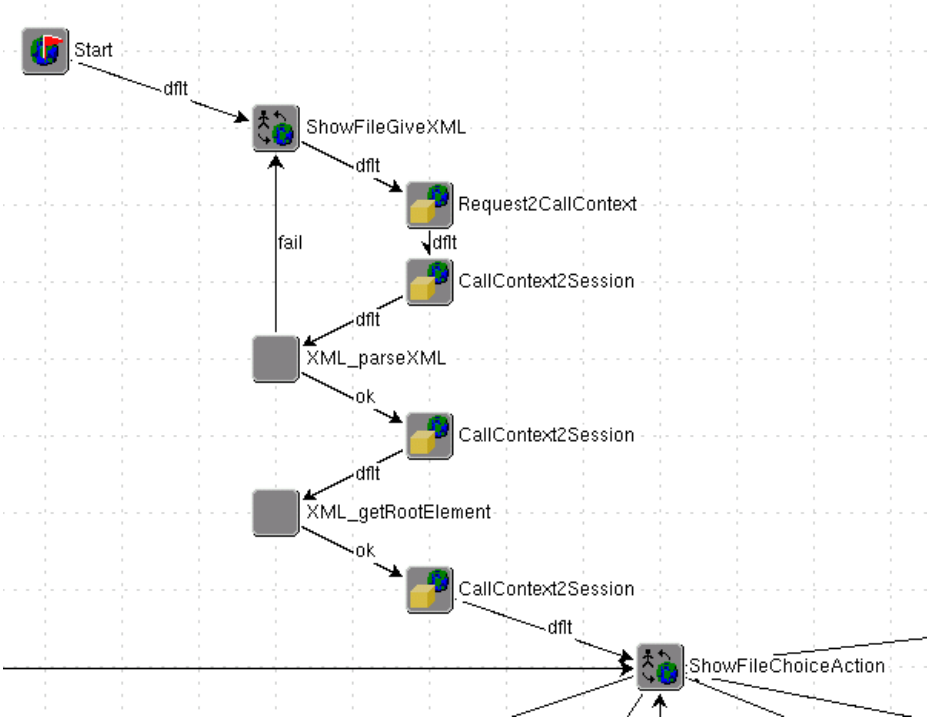

Abbildung 3.23: XML_Manipulation: Erstellen eines JDOM-Dokument

Die Testphase überprüft die Korrektheit der Manipulation von XML-Datein. An dieser Stelle möchten wir anhand eines Beispiels zeigen, wie sich unser Testgraph auf Service-Logic-Ebene bei einem vollständigen durchlauf verhält.

Der Graph XML_parsetXML

Der Aufbau des Graph (Abb.3.23) beginnt mit den Initialisierungs-SIBs Start und Show-File. Um die XML-Datei zu bekommen, wird der Name von der XML-Datei, sowie die Quelle, vom Benutzer eingegeben. Der SIB `XML_parseXML` parst die XML-Datei und baut ein JDOM-Dokument auf. Als nächstes wird der SIB `XML_getRootElement` ausgeführt. Dieser SIB bekommt das RootElement vom JDOM-Dokument übergeben. Der Name der XML-Datei, das JDOM-Dokument sowie das RootElement werden in Sessions gespeichert. Der SIB `ShowFile_ChoiceAction` zeigt anschliessend ein Formular im Browser an, auf dem der Benutzer eine von sechs Aktionen wählen kann. In unserem Testbeispiel wird die Remove-Element-Aktion gewählt.

Der Graph XML_removeElement

In diesem Teilgraph (Abb.3.24) wird die Funktion XML_removeElement ausgeführt. Zuerst wrid das Dokument Objekt aus der Session geholt. Mit dem SIB `XML_convertObject2String` wird eine String-Repräsentation des Objektes zurückgeliefert. Das ganzen Dokument Objekt wird als String vom CallContext im Browser angezeigt. Dort wird dann der XPath von Benutzer angegeben. Der im CallContext geschriebene XPath und das von der Session ausgelesene RootElement werden als Parameter in dem SIB `XML_findElementByPath` benutzt. Der SIB `XML_findElementByPath` liefert ein List von Element. Die Liste wird in die Session gespeichert. Danach wird das Listen Objekt durch den SIB `XML_convertObjekt2String` als String mit einem Index-Formular im Browser angezeigt. Der vom Benutzer angegebene Index wird in den CallContext geschrieben. Als nächstes wird der SIB `XML_getElementFromList` ausgeführt. Mit den Parametern (Liste von Element und dem Index) wird das gewünschte Element gefunden, worauf hin das gefundene Element durch den SIB `XML_removeElement` gelöscht wird. Das geänderte Dokument wird aus der Session geholt und im Browser angezeigt.

Durchführen des Testbeispiel

Zuerst wird der Name und die Quelle für eine XML-Datei vom Benutzer angegeben (Abb.3.25). Danach wird ein Aktion-Wählen-Fenster gezeigt (Abb.3.26). Hier wählen wir die Remove-Element-Aktion. Als nächstes wird das ganze Dokument angezeigt und der XPath vom Benutzer eingegeben (Abb.3.27). In der Abb.3.28 wird die Liste von Element angezeigt und der Index vom Benutzer angegeben. Abschliessend (Abb.3.29) wird das neue Dokument angezeigt.

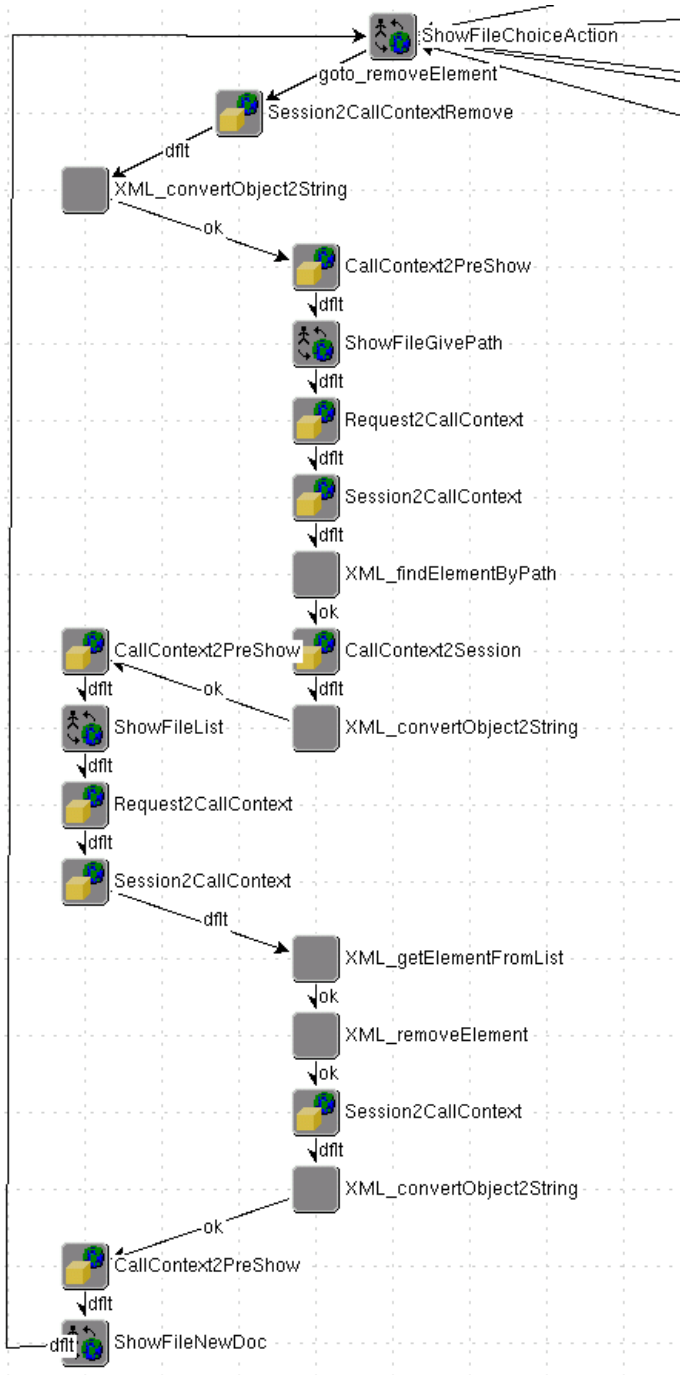

Abbildung 3.24: XML_Manipulation: Löschen eines Elementes

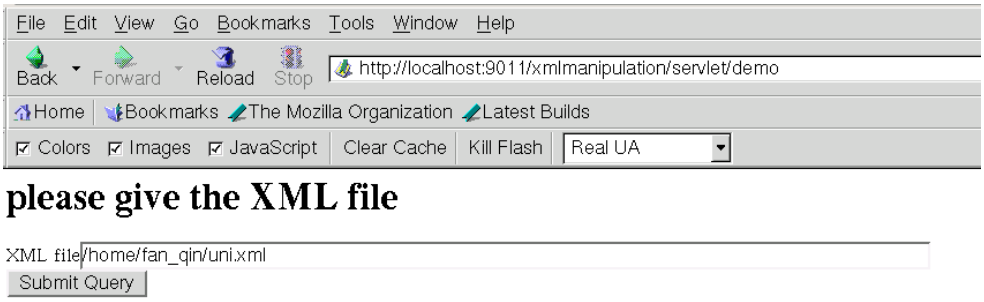

Abbildung 3.25: XML_Manipulation: Ausgabe einer XML-Datei

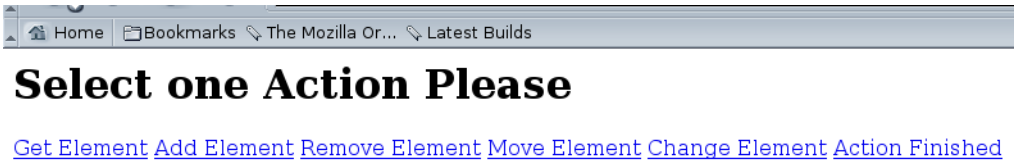

Abbildung 3.26: XML_Manipulation: wählen einer Aktion

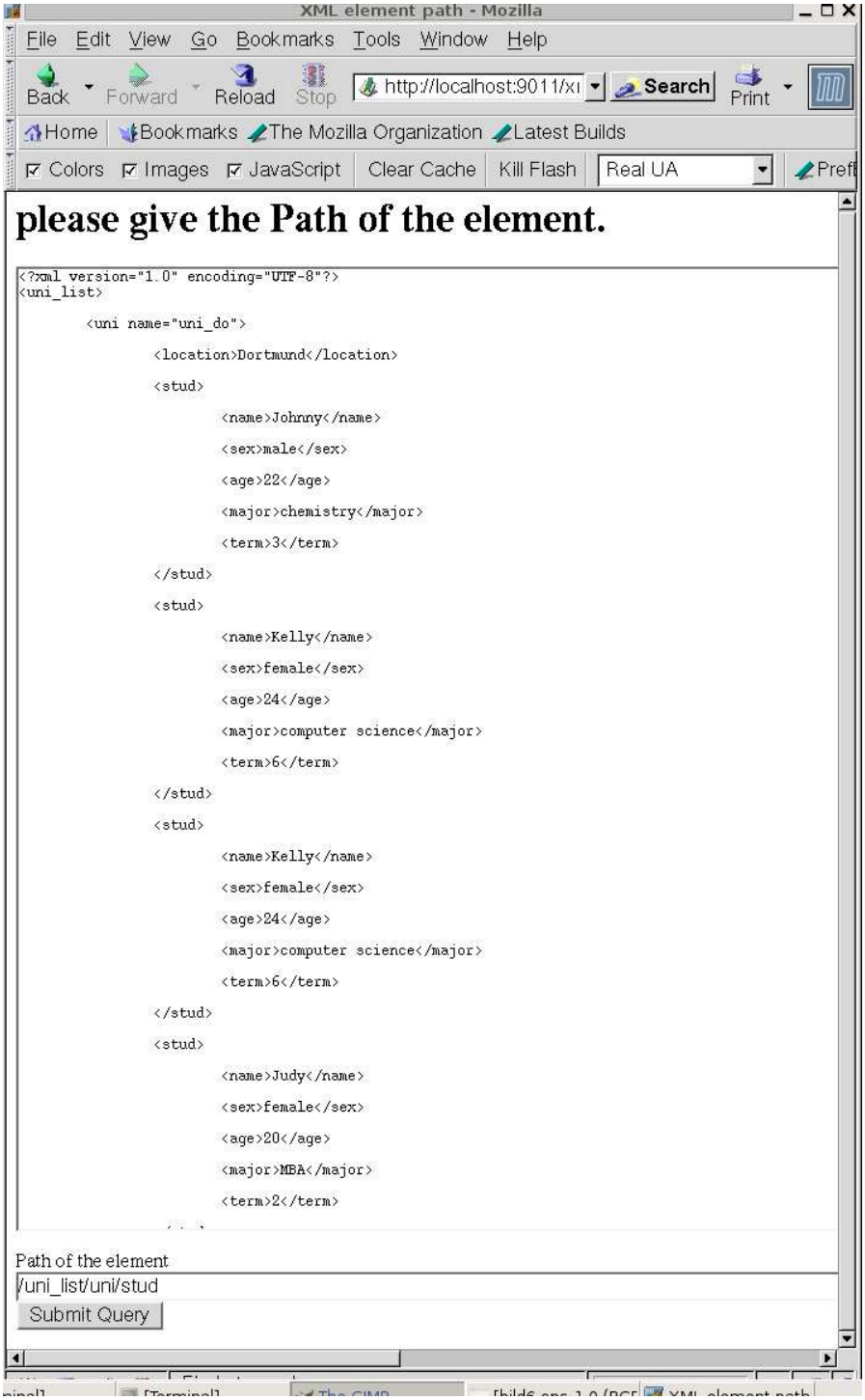

Abbildung 3.27: XML_Manipulation: anzeigen des gesamten Dokumentes

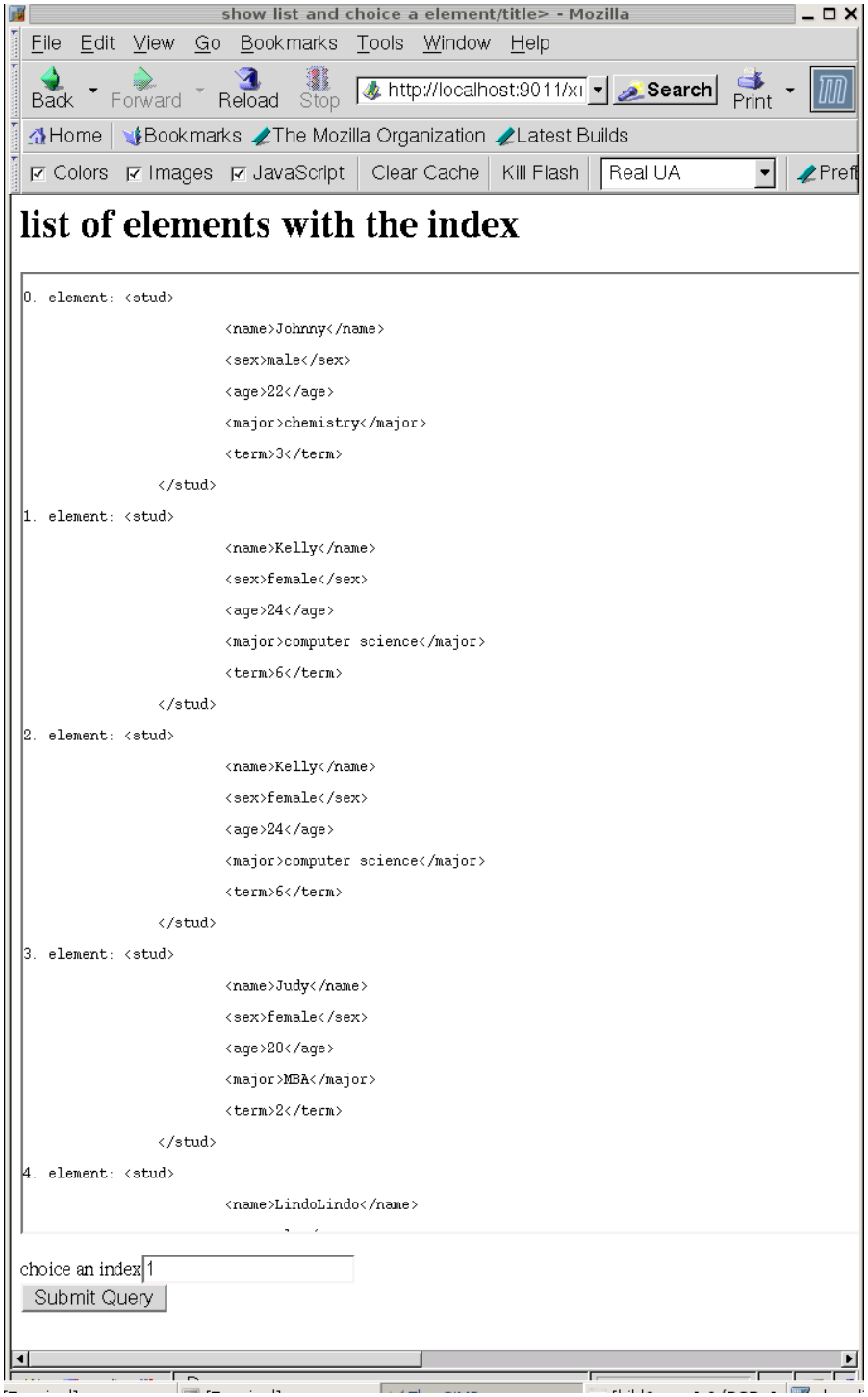

Abbildung 3.28: XML_Manipulation: anzeigen der Liste von Elementen

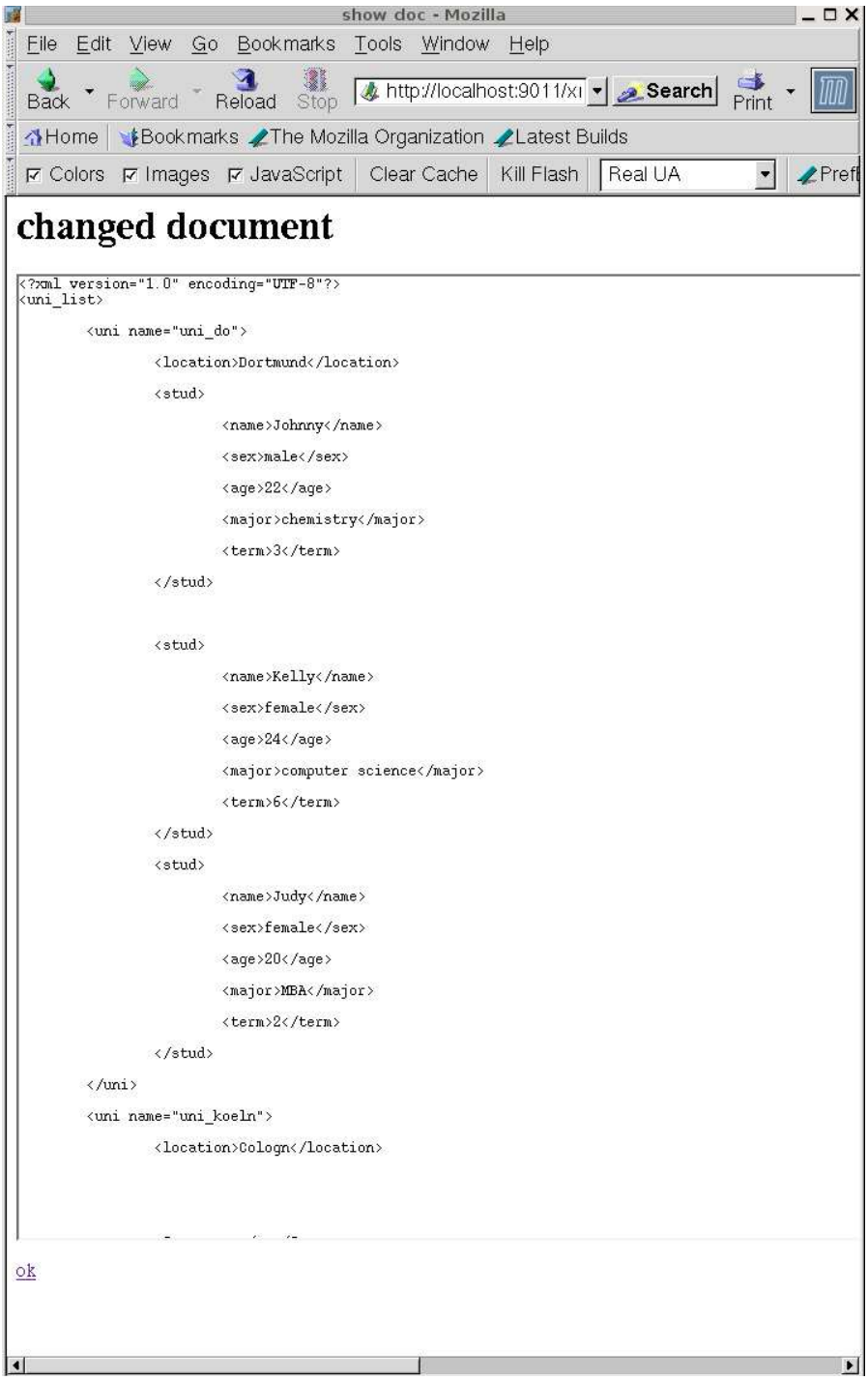

Abbildung 3.29: XML_Manipulation: anzeigen des manipulierten Dokumentes

Kapitel 4

Entwicklungsumgebung und Tools

4.1 ABC

Das Agent Building Center (ABC) ist eine Entwicklungsumgebung, welche die programmierungsfreie Entwicklung von Applikationen in Form von Graphen ermöglicht.

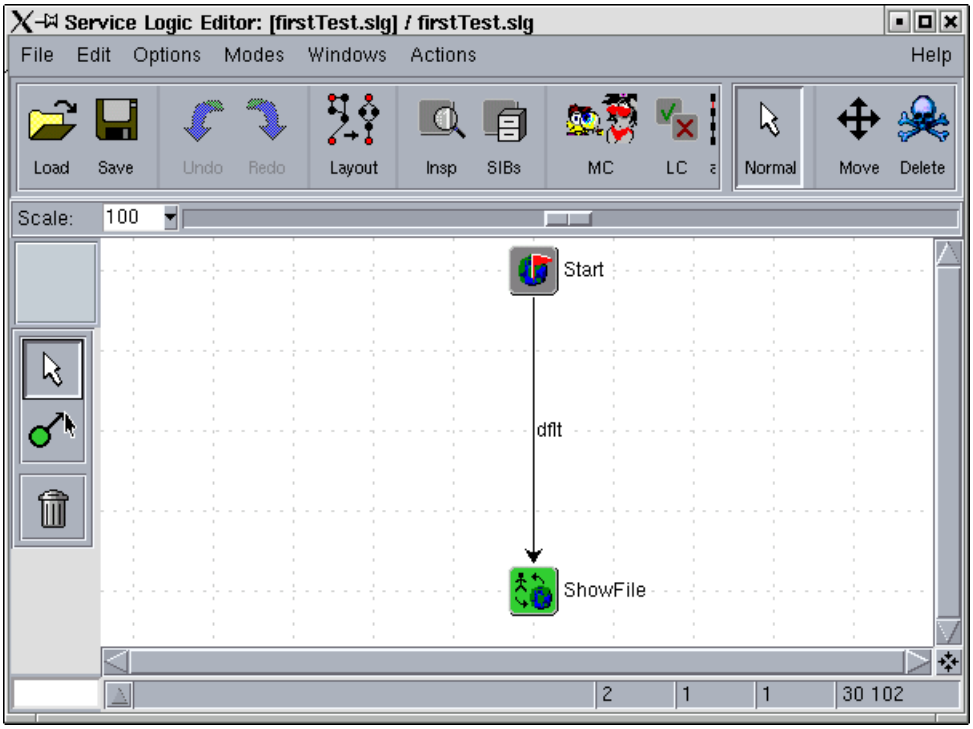

Abbildung 4.1: Der Service-Logic-Editor

Service-Independant-Building-Blocks (SIBs) stellen hierbei die Kernkomponenten eines Services dar. Sie repräsentieren einzelne Funktionalitäten innerhalb der zu entwickelnden

Applikation. Die Applikation selbst wird durch einen Graphen (Service-Logic-Graph) dargestellt, in dem die SIBs als Knoten dargestellt werden. Jeder SIB besitzt zwei Ebenen, die logische und die funktionale. Die logische Ebene beschreibt die Schnittstellen des SIB. Sie besteht aus Parametern und Verzweigungen. Es wird hier also der Workflow designed. Die funktionale Ebene hingegen wird durch den Quellcode dargestellt, der die SIBs implementiert. Dieser ist bei der EWIS-Umgebung in Java oder C++ geschrieben.

SIBs mit ähnlichen Funktionen oder SIBs, die auf ähnlichen Daten operieren, sind zu Klassen zusammengefasst, so dass ein schnelleres Auffinden des gewünschten SIBs bei der Auswahl in ABC/EWIS möglich ist.

Durch die Verknüpfung der SIBs entstehen schliesslich grössere Webservices. Diese Verknüpfung entsteht durch gerichtete Kanten, die den Kontrollfluss auf dem Graphen darstellen. Den Branches eines SIBs können dann Kanten auf das logisch folgende SIB zugewiesen werden.

Innerhalb der MaTRICS werden Webservices angeboten, welche im EWIS/ABC unkompliziert entwickelt werden können. Daher ist es auch Nicht-Programmierern möglich, eigene Services zu erstellen und dem Gesamtsystem zur Verfügung zu stellen.

In Abb.4.1 ist der Service-Logic-Editor mit einem kleinen Beispielgraphen dargestellt.

4.2 Tomcat

Ein herkömmlicher Webserver, wie zum Beispiel der Apache (ein OpenSource Projekt der Apache Foundation [3]), bietet im Groben die Funktionalität, HTTP-Requests über ein Netzwerk - beispielsweise das Internet - entgegenzunehmen und mit der angeforderten Webseite zu antworten. Diese Antwort wird dann auf Benutzerseite im Browser angezeigt. Diese Webseiten können sowohl statisch als auch dynamisch generiert sein (unter Benutzung von PHP, D-HTML, etc.). Allerdings ist auf Serverseite keine komplexe Programmierung möglich. Eine solche Funktionalität muss daher auf der Clientseite zum Beispiel durch JavaScript gelöst werden.

Im Falle der MaTRICS hätte man es dabei mit einer Komplexität zu tun, die man unmöglich den verschiedenen Client-Architekturen der Benutzer zumuten kann. Die Geschwindigkeit und Stabilität der Ausführung würde zu stark durch diese beinflusst werden.

Aus diesem Grund wurde auf eine Kombination von Webserver und JSP/Servlet Container zurückgegriffen, welche eine serverseitige Laufzeitumgebung für die MaTRICS auf speziellen, leistungsstarken Applicationservern bietet. Programme und Berechnungen werden nun transparent für den Benutzer ausgeführt und nur das Ergebnis wird zum Client übertragen. Diese Laufzeitumgebung ist ebenfalls ein OpenSource Projekt der Apache Foundation und nennt sich Tomcat [4]. Innerhalb des Tomcat werden die einzelnen Module der MaTRICS als sogenannte Servlets abgearbeitet, welche alle auf der Programmiersprache Java basieren. Dadurch wird zudem eine einfache plattformübergreifende Programmie-

rung der MaTRICS möglich. Seine hohe Modulariät gestattet es zudem, Erweiterungen der MaTRICS relativ einfach hinzuzufügen.

Kapitel 5

Zwischenthemen

Nun waren die einzelnen Module der verschiedenen Gruppen fertiggestellt und einsatzbereit. Aber es ging bei unserem Projekt nicht darum einzelne voneinander unabhängige Programme zu entwickeln, vielmehr war es das Ziel in einer relativ großen Gruppe an verschiedenen Modulen zu arbeiten, die später nicht nur zusammengefügt, sondern auch in das große System der MaTRICS eingebettet werden sollten. Die Einarbeitung in ein so großes bereits bestehendes System stellte dabei eine große Herausforderung dar.
Um sich möglichst schnell und reibungslos einarbeiten zu können, wollten wir uns im Folgendem mit der Erweiterung der Basiskomponenten der MaTRICS beschäftigen, welche in Abbildung 5.1 zu sehen sind. Dies half uns einerseits einen praktischen Einstieg in die Materie zu bekommen und andererseits die bereits in den Kurzthemen entwickelten Module in die MaTRICS zu integrieren.

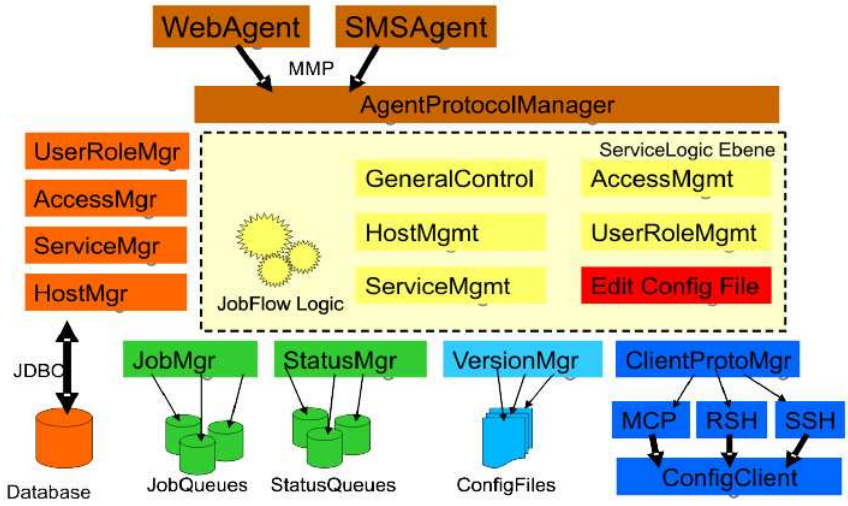

Abbildung 5.1: Schematische Darstellung des ConfigManagers

Jede der Gruppen, die bereits die Module der Kurzthemen entwickelt hatten, sollte nun an einer Stelle der MaTRICS deren Funktionalität erweitern und die neu entwickelten Module in den vorhandenen Code einarbeiten. Die einzelnen Themen seien an dieser Stelle kurz beschrieben:

- Gruppe 1 sollte Anpassungen an der JobFlow Engine vornehmen, die u.A. den Taskmanager so erweitern, dass bereits abgearbeitete Tasks neu ausgeführt werden können.
- Gruppe 2 hatte drei Aufgaben zu bewältigen. Zunächst sollte der ClassLoader der MaTRICS erweitert und das MatricsManagementProtocol (MMP) verbessert werden. Anschließend sollte von der Gruppe noch eine SIB Bibliothek entwickelt werden, welche es ermöglicht innerhalb eines Dienstes vor und zurück zu springen ohne dass hierbei die gemachten Eingaben verloren gehen.
- Gruppe 3 sollte das bereits entwickelte Modul zur Umwandlung von Config Dateien in das XML Format in den VersionManager integrieren.
- Schließlich ist es Aufgabe von Gruppe 4 gewesen, den ServiceManager so zu erweitern, dass er in der Lage ist für jeden neuen Dienst eine Menge von Dateien zu verwalten, die zur Benutzung des Dienstes benötigt werden.

Die genauen Beschreibungen der einzelnen Aufgaben finden Sie in den folgenden Kapiteln.

5.1 Anpassungen an der JobFlow-Engine

Die JobFlow-Engine ist ein zentraler Bestandteil der MaTRICS. Sie basiert auf Java Message Service JMS, wobei für jeden **ConfigClient** separate Job- und Status-Queues angelegt werden. Jeweils eine Queue für die Task-Objekte: newTasks (neue Tasks), oldTasks (abgearbeitete Tasks), parkedTasks (fehlgeschlagene Tasks), holdingTrack (mehrmals fehlgeschlagene Tasks).
Eine Persistenzschicht (PostgreSQL-Datenbank) ist für das Speichern alter abgearbeiteter Task- und Status-Objekte vorhanden. Die Verwaltung erfolgt über die folgenden Manager-Komponenten:

- **TaskManager:** Ist für die Verarbeitung von Konfigurationsaufgaben zuständig, die aus mehreren Jobs bestehen, die auf verschiedenen ConfigClients ausgeführt werden.
- **JobManager:** Seine Aufgabe ist die Verwaltung der Jobs für einen ConfigClient.

- **StatusManager:** Ist für die Verwaltung bereits verarbeiteter Jobs zuständig.

Im Folgenden werden das Design, die Implementierung und ein Anwendungsbeispiel der MaTRICS JobFlow-Engine dargestellt.
Die JobFlow-Engine beschäftigt sich damit, Aufgaben auf ConfigClients in Form von Tasks auszuführen.
Dabei stellt ein Task eine Menge von zusammengehörigen, unteilbaren Jobs dar, deren Ausführungsreihenfolge durch Abhängigkeiten festgelegt werden kann. Ist ein Job eines Tasks fehlgeschlagen, so wird ein Rollback des gesamten Tasks durchgeführt, es werden also alle zu diesem Task gehörenden, bereits abgearbeiteten Jobs rückgängig gemacht.

5.1.1 Modellierung

Innerhalb der JobFlow-Engine sind drei Manager implementiert: ein JobManager, ein TaskManager sowie ein StatusManager. Diese Klassen ermöglichen den Zugriff auf die Queues und auf die Datenbank. Passend zum EWIS Database Konzept gibt es noch die Basisklassen Job, Task und Status.

Anwendungsfalldiagramm
Die Abbildung 5.2 bietet eine Übersicht über die Anwendungsfälle der JobFlow-Engine.
Der **TaskManager** war so zu erweitern, dass bereits abgearbeitete Tasks erneut aus-

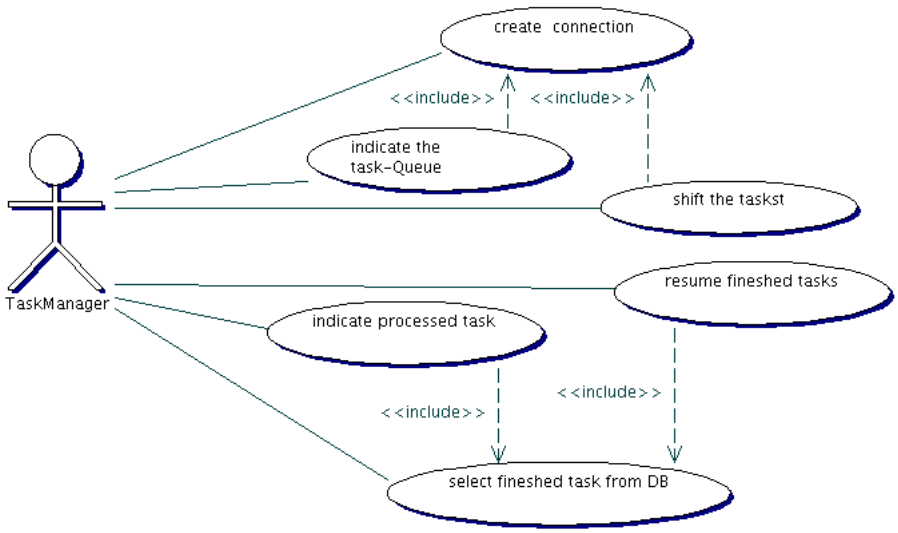

Abbildung 5.2: JobFlow-Engine - Anwendungsfalldiagramm

geführt werden können. Zur Verwaltung der Tasks sollte ein Dienst mit den folgenden

Funktionalitäten erstellt werden.

- Anzeigen der Task-Queue mit den dementsprechenden Jobs
- Verschieben von Tasks in eine andere Queue
- Anzeige der verarbeiteten Tasks
- Wiederholung bereits verarbeiteter Tasks

Aktivitätsdiagramm
Abbildung 5.3 bietet eine Übersicht über die Funktionalität der Jobflow-Engine. Die verschiedenen Operationen auf den vier Task-Queues newTasks, oldTasks, parkedTasks und holdingTrack, in welchen Task-Objekte je nach Ausführungszustand eingequeuet werden, werden im Detail gezeigt. Dazu siehe auch Abb 5.4

5.1.2 Implementierung

Innerhalb der JobFlow-Engine wurde eine neue Funktionalität implementiert, die es ermöglicht, Tasks von einer Queue in eine andere zu verschieben. Die Methode heißt *MoveTaskToAnotheQueue*. Zunächst wird eine Verbindung zu JMS erstellt. Anschließend können alle Queues eingesehen und Tasks gegebenenfalls verschoben werden.

Mögliche Verschiebungen:

- move old task to the newTaskqueue
- move new task to the holdedTaskqueue
- move holding task to the oldTaskqueue

Früher wurden die Job Status in einer eigenen Queue gespeichert. Nun ist es möglich sie direkt in die Datenbank zu schreiben. Das Diagramm 5.4 beschreibt die neuen Funktionalitäten, die zu der JobFlow-Engine hinzugefügt wurden.
Zu der entsprechenden ABC SIB Bibliothek wurde folgendes neue SIB hinzugefügt:
Das SIB **TaskMGR_MoveTaskToAnotherQueue** ermöglicht die Übergänge zwischen den Queues. Wenn man einen Task von der OldTaskQueue in die newTaskQueue verschiebt, werden alle Jobs die zu diesem Task gehören mitverschoben. Die Abbildung 5.5 zeigt den Ausschnitt des Graphen *MonitorMMP* mit dem neuen SIB.

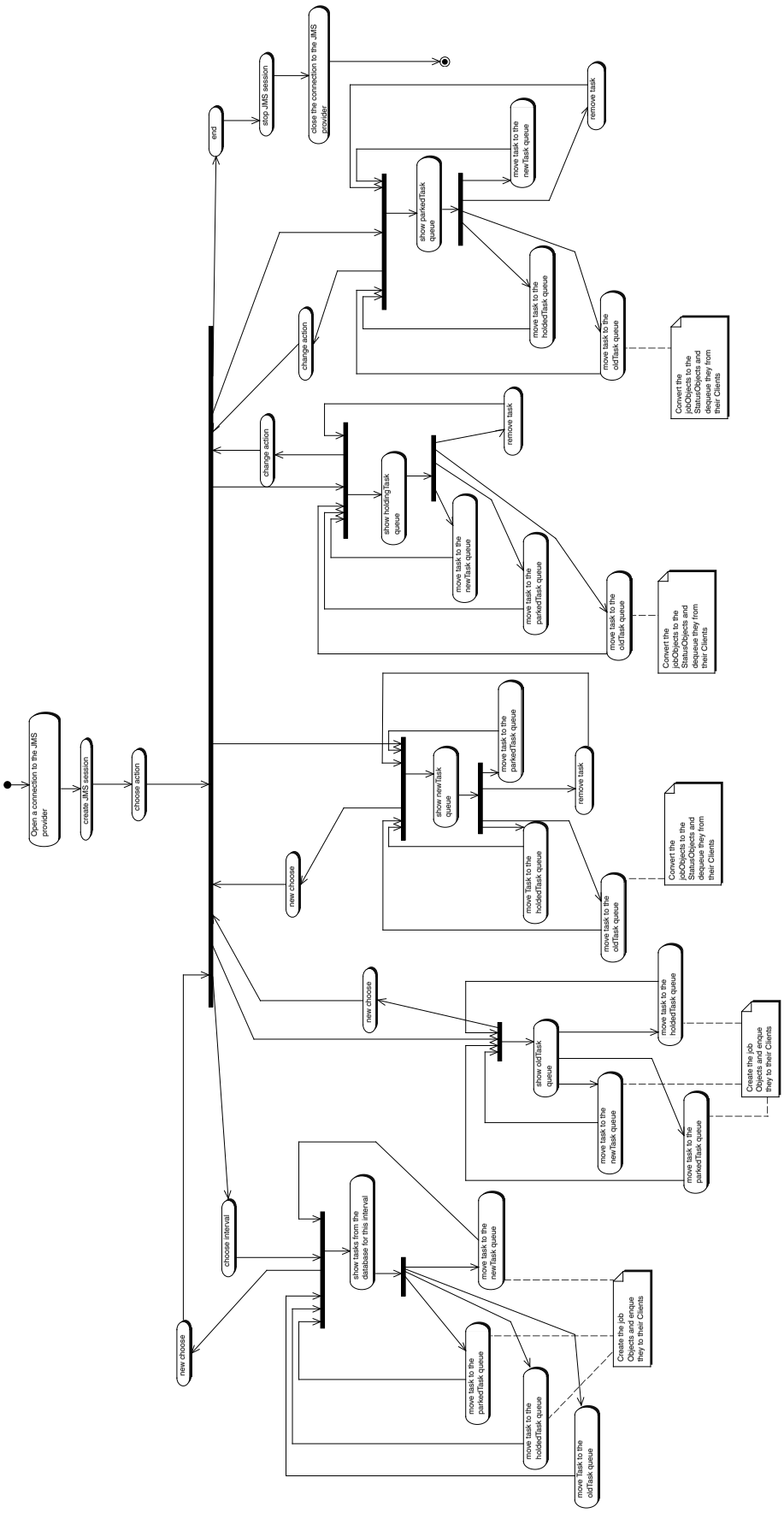

Abbildung 5.3: JobFlow-Engine - Aktivitätsdiagramm

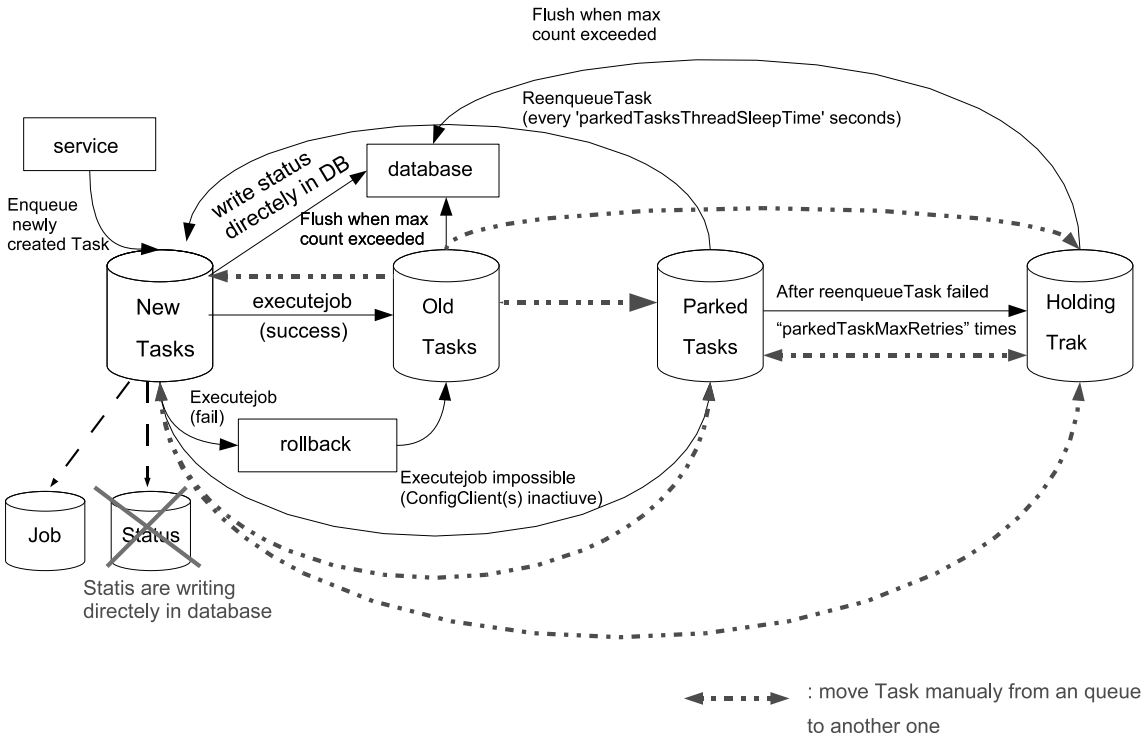

Abbildung 5.4: JobFlow-Engine - Die möglichen Übergänge von Task Objekten zwischen den Queues

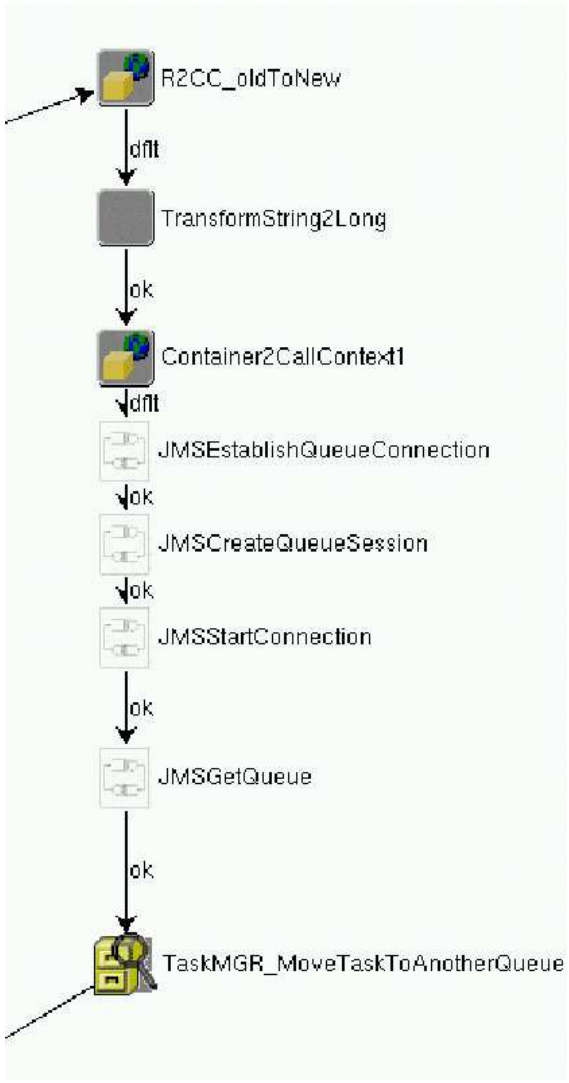

Abbildung 5.5: JobFlow-Engine - SIB zum Verschieben von Tasks

5.1.3 Tests

Zum Testen der entwickelten Funktionalitäten des Queuemanagements und Queuemonitorings wurden, mit Hilfe des vorhandenen Dienstes EditConfigFile, einige Tasks erzeugt. Mit dem neuen Monitor-Dienst konnte man nun den Inhalt der jeweiligen Queues (inklusive der Jobs-Zuordnung zu den einzelnen Tasks) anzeigen lassen.

Die Tasks konnte man beliebig zwischen den vorhandenen Queues verschieben. Einzelne Einschränkung: es wurde bewußt auf die Möglichkeit verzichtet, Tasks nach OldTaskQueue verschieben zu können. Der Grund war, dass sonst das "korrekte" Ausführen der Jobs und Tasks simuliert werden musste, ohne dass sie in Wirklichkeit ausgeführt wurden. Dies würde unmöglich machen, eine in Realität ausgeführter Task bzw. Job von der einen simuliert ausgeführten zu unterscheiden.

Parallel zu diesem Test wurde auch das direkte Schreiben von JobStatus in die Datenbank getestet. Dafür wurde in der web.xml der entsprechende Parameter, welcher die Länge der StatusQueue definiert, auf Null gesetzt. Das korrekte Schreiben vom Status wurde überprüft, indem man kurz in die Datenbank reinschaute.

5.2 Redesign der MaTRICS Dienste

5.2.1 Das MaTRICS Management Protocol (MMP)

Wie wir bereits in den vorherigen Kapiteln gesehen haben, handelt es sich bei der MaTRICS um ein verteiltes System, wobei der Config-Manager den eigentlichen Kern des Systems darstellt. Hierzu besteht eine Datenbankanbindung, in der die zu konfigurierenden ConfigClients sowie die zur Verfügung stehenden Services, samt Informationen und Abhängigkeiten zueinander gespeichert sind.
Um die Informationen der Datenbanken auf höherer Ebene aufzubereiten, zu strukturieren, und von außen leichter zugänglich zu machen, wurden sogenannte Business-Objekte ins Leben gerufen. Ein Business-Objekt stellt eine für sich logische Einheit dar, die es erlaubt, zugehörige Informationen schnell und einfach auszulesen und zu ändern, ohne dabei direkt auf die Datenbankschnittstelle zugreifen zu müssen.
Services wie auch verwaltungstechnische Aufgaben innerhalb der MaTRICS werden in der Regel über einen sogenannten Agent angesprochen. In unserem Fall beschränken wir uns zunächst auf einen WebAgent, was besagt, dass wir mittels eines gewöhnlichen Browsers die Funktionalität der MaTRICS nutzen können. Hierzu ist es lediglich erforderlich, einen einfachen Web-Dienst auf dem Agent zu betreiben, der über eine Verbindung mit Config-Manager kommuniziert. Auf diese Art und Weise ist der gesamte Ablauf im Kern der MaTRICS unabhängig von der Art des eingesetzten Agents. In Zukunft wäre also auch ein PDA-Agent, ein SMS-Agent oder ein sprachgesteuerter Agent möglich. Die Verbindung zwischen dem Agent und dem Config-Manager selbst muss aus zwei Perspektiven betrachtet werden. Zum einen ist es erforderlich, dass der Agent auf Funktionalitäten

der MaTRICS zugreifen kann. Dies geschieht mittels Remote-Method-Invocations (RMI). Hierbei werden die vom Config-Manager angebotenen Objekte und deren Methoden erfasst und in sogenannten Stubs beim Agent nachgebildet, so dass der Agent diese aufrufen kann. Wird eine solche Methode aufgerufen, sorgt der Stub dafür, dass die Anfrage eingepackt und an einen entsprechenden Skeleton des Config-Managers weitergeleitet wird. Der Skeleton packt die Methoden-Anfrage dann lokal am Server aus und ruft sie schließlich auf dem eigentlichen Zielobjekt auf. In der Gegenrichtung ist es erforderlich, dass der Agent Informationen darüber erhält, was dieser dem Anwender anzeigen bzw. ausgeben soll. Dies geschieht mit einem paketorientierten MMP-Protokoll. Hierbei enthalten die übertragenden MMP-Pakete alle Informationen, die für die Darstellung erforderlich sind, wie das zugehörige Template sowie zugehörige Parameter, mit denen dessen dynamische Werte gefüllt werden.
Da wir beim WebAgent eine Velocity-Instanz einsetzen, um die Parameter der Webseiten dynamisch zu füllen, ist es außerdem möglich, innerhalb der Webseite auf einfache Java-Objekte bzw. auf deren Methoden zuzugreifen. Und genau an dieser Stelle kommen die zu Beginn genannten Business-Objekte wieder ins Spiel. Die Idee ist die folgende: Anstatt alle einzelnen Informationen, die für die Anzeige benötigt werden, in Variablen zu packen, um sie dann auf der Webseite anzeigen zu lassen, werden einfach die benötigten Business-Objekte übertragen, die im Kern der MaTRICS sowieso benutzt werden, wie bereits zu Beginn erklährt. Der Vorteil liegt jetzt darin, dass die Web-Seite selbst die benötigten Daten aus den Objekten auslesen kann.

Eine erste Implementierung

Die oben beschriebenen Ideen wurden von der Projektgruppe 451 soweit umgesetzt, allerdings stieß man kurzfristig auf das Problem, dass Business-Objekte nicht zum WebAgent übertragen werden konnten. Der Fehler beruhte auf der Tatsache, dass Business-Objekte, wie oben beschrieben, auf Datenbanken zugreifen und entsprechende Klassen importieren müssen, was zur Folge hat, dass diese Objekte nicht serialisierbar und somit nicht sendefähig sind. Die wirklich unschöne Lösung bestand nun darin, für jedes Business-Objekt jeweils eine sendefähige Kopie zu implementieren, in die dann zum Sendezeitpunkt konvertiert wurde, und die dann anstelle des originalen Business-Objekts gesendet wurde. Diese Kopien waren befreit von Datenbankfunktionen und enthielten nur noch Getter- und Setter-Methoden einfacher Variablen sowie Verweise auf andere Business-Objekte. Bei der Konvertierung wurden solche Verweise ebenfalls durch Verweise auf zugehörige konvertierte Business-Objekte ersetzt, um das Geflecht der Business-Objekte aufrechtzuerhalten.

Das verbesserte MMP-Protokoll

Die Idee zur Verbesserung bestand nun darin, eine Art Tunnelmechanismus für den Transfer der nicht-sendefähigen Business-Objekte zu generieren, um so auf die zusätzlichen Implementierungen von sendefähigen Kopien verzichten zu können. Mit anderen Worten: Die Objekte, die sich nicht übertragen lassen, weden analysiert, charakterisiert und in Bauteile zerlegt, die gesendet werden können. Auf der Empfängerseite muss mithilfe dieser Konstruktionsanleitung das ursprüngliche Objekt wieder nachkonstruiert werden. Die Umsetzung dieser Idee bestand aus drei Schritten, die hier möglichst verständlich und

einfach erklärt werden sollen.
Der erste Schritt bestand darin, die sendefähigen MMP-Einheiten (MMPEntity) so zu erweitern, dass sie auch als Kapseln für den Transfer von Bauanleitung genutzt werden konnten. Hierzu muss man sich zunächst darüber im klaren sein, was für eine Rekonstruktion eines Objektes erforderlich ist. Einsichtigerweise werden neben dem zugehörigen Klassennamen auch die aktuellen Variablen-Werte jedes Business-Objekts benötigt. Entsprechend wurden die MMP-Einheiten um einen String für die Aufnahme des Klassennamens sowie um eine HashMap für die Aufnahme aller Variablen-Wert-Paare erweitert. Schließlich wurde noch eine boolische Variable eingebaut, um die gekapselten MMP-Einheiten von anderen MMP-Einheiten zu unterscheiden. Der Idee einer Konvertierung vor dem Senden eines Business-Objekts konnte aus der ersten Implementierung übernommen werden. Allerdings wurden die Business-Objekte bei dem neuen Konvertiervorgang jetzt wie beschrieben direkt in ein MMPEntity-Objekt eingekapselt. Um gegen Mehrfachkonvertierung, die sich durch das angesprochene Geflecht der Business-Objekte ergibt, geschützt zu sein, wurde eine Loop-Detection eingebaut, so dass jedes Business-Objekt je Konvertierungsaufruf nur einmal konvertiert werden kann.
Der zweite Schritt bestand nun darin, die übertragenden Kapseln auf Agent-Seite zu identifizieren und wieder zu entkapseln. Folglich wurde ein neuer DecapsulateBusinessObject-SIB in den Service-Logik-Graphen des WebAgent eingebaut, der nach solchen eingekapselten Business-Objekten sucht, ihre Bauanleitung extrahiert und sie nachkonstruiert. Hierzu wird mithilfe des Klassennamens zunächst ein neues Objekt instanziiert, welches dann über die gespeicherten Werte neu gefüllt wird. Da die übergeordneten MMP-Pakete auch Arrays oder Treesets von Business-Objekten enthalten können, mussten die neu instanziierten Objekte ein identische Strukturen zurückgeführt werden.
Der dritte und abschließende Schritt bestand aus der Anpassung der einzelnen Business-Objekte. Da wir die Business-Objekte gemäß den Original-Objekten neu instanziiert haben, entstanden so auch Methoden, die auf die beim WebAgent fehlende Datenbank zugreifen würden. Ensprechend mussten solche Methoden daran unterscheiden, wo sie instanziert wurden. Zu diesem Zweck wurde ein boolische Identifikator LittleBrother eingeführt, an dem die Methoden ausmachen können, ob sie sich beim WebAgent oder beim Config-Manager befinden. Um Datenbankzugreifende Operationen denoch auch auf Seite des WebAgents zu ermöglichen, wurden Puffer-Speicher in die Business-Objekte eingebaut, welche auf Config-Manager-Seite gefüllt wurden, sobald eine Datenbankabfrage stattgefunden hat. Auf WebAgent-Seite wurde schließlich aus dem Puffer gelesen, wenn festgestellt wurde, dass es sich um eine Little-Brother-Instanz handelte.

5.2.2 Der Classloader

Ein Problem des Designs der MaTRICS als verteiltes System besteht darin, dass die Agents Zugriff auf Komponenten des Basissystems benötigen. So bedient sich zum Beispiel das MaTRICS Management Protocol des Java RMI Mechanismus um eine Verbindung zwischen den Agents und dem AgentProtocolManager herzustellen. Damit

der Agent die Methoden des AgentProtocolManagers verwenden kann muss er dessen Stub kennen.
Da die Agents und der AgentProtocolManager typischerweise nicht auf derselben Maschine ausgeführt werden, steht ihnen auch keine gemeinsame Codebase zur Verfügung.
Damit nun nicht bei jeder Änderung am AgentProtocolManager alle Agents aktualisiert werden müssen hat die Projektgruppe PG451 eine auf http basierende Codebase eingeführt über welche der aktuelle Stub des AgentProtocolManagers von den Agents nachgeladen werden kann.

Die Aufgabe
Informationen werden zwischen den Agents und MaTRICS Diensten in Form von Objekten ausgetauscht. Hier besteht nun wieder das Problem, dass die Klassen der Objekte welche an die Agents übertragen werden diesen auch bekannt sein müssen.
Die Aufgabe bestand darin die bestehende http CodeBase zu erweitern, so dass beliebige Klassen von den Agents nachgeladen werden können.

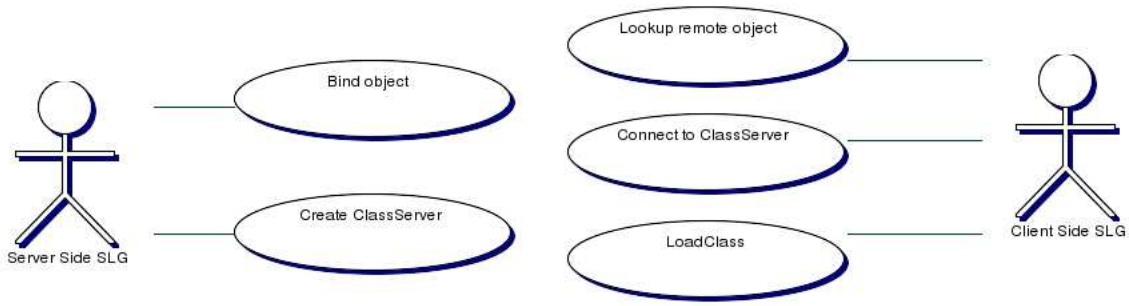

Abbildung 5.6: RMI - Anwendungsfalldiagramm

Die Lösung
Als wir begannen uns mit der Aufgabe zu beschäftigen kamen wir schnell zu der Überzeugung, dass die http basierte Codebase keine sehr gute Lösung darstellt. Der negativste Aspekt an dieser Codebase besteht darin, dass nur zu diesem Zweck ein WebServer verwaltet werden muss.
Da die MaTRICS bereits Aspekte von RMI benutzt, haben wir uns entschlossen die http basierte Codebase durch einen RMI basierten ClassServer zu ersetzen.
Während diese Idee bei den Betreuern auf große Begeisterung stieß löste sie bei uns die nächsten Probleme aus. Die Vorgänger PG hatte SIBs zur Verfügung gestellt um eine Methode eines RMI Serverobjektes aufzurufen. Hierzu wurde ein SIB Namens RMIInvokeMethodWithParameters erstellt. Mit Hilfe dieses SIBs ist es möglich die Methode eines RMI Server Objektes aufzurufen und den Rückgabewert zu verarbeiten. Hierzu müssen die Klasse, die Methode und die Parameter auf Graph Ebene bekannt sein und als Parameter an den SIB übergeben werden.

Die eigentliche Idee von RMI besteht darin, Referenzen auf entfernte Objekte zu beziehen und von da an mit den entfernten Objekten zu arbeiten als wären es lokale Objekte. Die Verteilung soll weitestgehend verborgen werden.

Das Design

Wie in Abbildung 5.6 zu sehen ist, sind zwei Akteure an RMI beteiligt. Akteure sind hier die Service Logic Graphen auf Client und auf Server Seite.

Der Server ist in der Lage

- Objekte an eine RMI Registry zu binden
- einen ClassServer bereitzustellen

Der Client ist in der Lage

- eine Referenz auf ein entferntes Objet von einer a RMI Registry zu beziehen (lookup)
- sich zu einem ClassServer zu verbinden
- Klassendefinitionen über den ClassServer nachzuladen

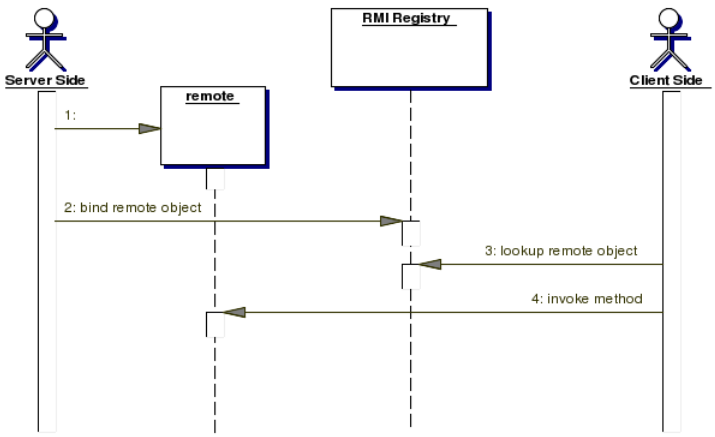

Abbildung 5.7: RMI - Sequenzdiagramm

Das in Abbildung 5.7 gezeigte Sequenzdiagramm beschreibt den Ablauf einer Remote Method Invokation.

Zunächst muss auf der Serverseite ein Remote Objekt erzeugt und an die Registrierung gebunden werden.
Anschließend kann auf der Clientseite eine Referenz auf dieses Objekt von der Registry bezogen und eine Methode auf dem entfernten Objekt aufgerufen werden.

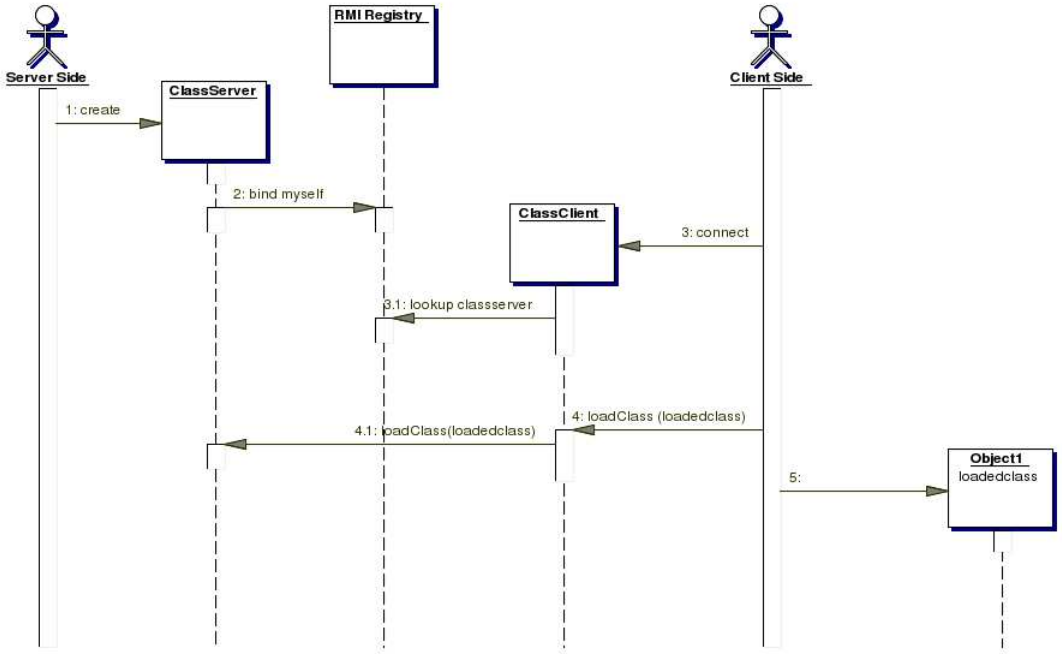

Abbildung 5.8: RMI - ClassServer Sequenzdiagramm

Das Sequenzdiagramm in Abbildung 5.8 verdeutlicht die Funktionsweise des RMI ClassServers.

Um Klassendefinitionen an andere Systeme abgeben zu können kreiert die Serverseite einen ClassServer. Dieser ClassServer bindet sich selbst an eine RMI Registry.
Anschließend kann sich die Clientseite mit Hilfe eines ClassClient zu diesem ClassServer verbinden. Hierzu bezieht der ClassClient eine Referenz auf den ClassServer von der Registry.
Von nun an kann die Clientseite beliebige Klassendefinitionen von der Serverseite nachladen und mit diesen arbeiten als hätten sie schon immer zur Verfügung gestanden.

Die Realisierung
Um die zuvor beschriebenen Anwendungsfälle zu realisieren haben wir die SIB Bibliothek RMI um entsprechende SIBs erweitert. Mit diesen SIBs sind die Abläufe welche in den Sequenzdiagrammen beschrieben sind in Graphform zu realisieren.
Anschließend haben wir diese SIBs genutzt und die MaTRICS mit einem ClassServer ausgestattet.

Der ClassServer wird gemeinsam mit dem AgentProtocolManager gestartet.
Der ClassClient wird von dem WebAgent benutzt um die benötigten Klassen nachzuladen. Welche Klassen nachgeladen werden sollen wird über einen entsprechenden Parameter in der Konfigurationsdatei (web.xml) des WebAgent festgelegt.
Dank dieser Erweiterung kommt die MaTRICS nun ohne http Codebase aus und die MMP Objekte müssen nicht länger per Hand in den Classpath des WebAgent kopiert werden. Bei jedem Start läd der WebAgent automatisch die aktuellen Klassendefinitionen über den RMI ClassServer nach.

5.2.3 Entwicklung von Navigations-SIBs

In der SIB Bibliothek des WebAgents befinden sich SIBs, mit denen ein Serviceentwickler ohne große Probleme seinen Webseiten ein Navigationsmenü hinzufügen kann. Dieses hat den großen Vorteil, dass es eingebene Daten (z.B. auf Formularen) speichert und bei Rückkehr auf die entsprechende Webseite wiederherstellt. Der Benutzer muss so also nicht alles wieder erneut eingeben, was besonders bei grossen Formularen extrem lästig ist und normalerweise bei Navigation über die ”Forward” und ”Back” Funktionen üblicher Webbrowser auftritt.

Die SIB-Bibliothek teilt sich in nur 4 SIBs auf. Zu Beginn eines Services muss ein Initialisierungs-SIB (`WorkflowMenu_InitSessionHash`) Platz im Hauptspeicher für die zu speichernden Eingaben reservieren. Dieser muss für jede Webseite angelegt werden.

Danach sollten die verschiedenen Untermenüs initialisert werden (`WorkflowMenu_Generate`). Es können mit diesem SIB beliebig viele Menüs anhand von Vorlagen erstellt werden, welche dann über eine Variable (über Velocity) auf der Webseite eingefügt werden. Bei den Vorlagen handelt es sich um HTML-Fragmente, die zur Ausführungszeit ebenfalls über die Velocity-Engine geparst werden, bevor die Webseite angezeigt wird.

Bevor eine nun Webseite überhaupt angezeigt wird, müssen die Parameter wiederhergestellt werden und dieser mitgeteilt werden. Hierfür ist ebenfalls ein entsprechender SIB (`WorkflowMenu_RestoreParameters`) zuständig. Er holt anhand einer Kennung, welche die Webseite eindeutig indentifiziert, die passenden Daten aus dem Speicher und stellt sie der Velocity-Engine zur Verfügung.

Zur Auswertung der in den Menüs vorhandenen Buttons ist schliesslich der `WorkflowMenu_Evaluate`-SIB zuständig. Er entscheidet, wie der Workflow (auf SLG-Ebene) weitergeht und ”rettet” die auf der Webseite gemachten Eingaben.

Abb. 5.9 zeigt eine Webseite mit den Navigations-Buttons, welche unabhängig von der Webseite über ABC definiert werden können.

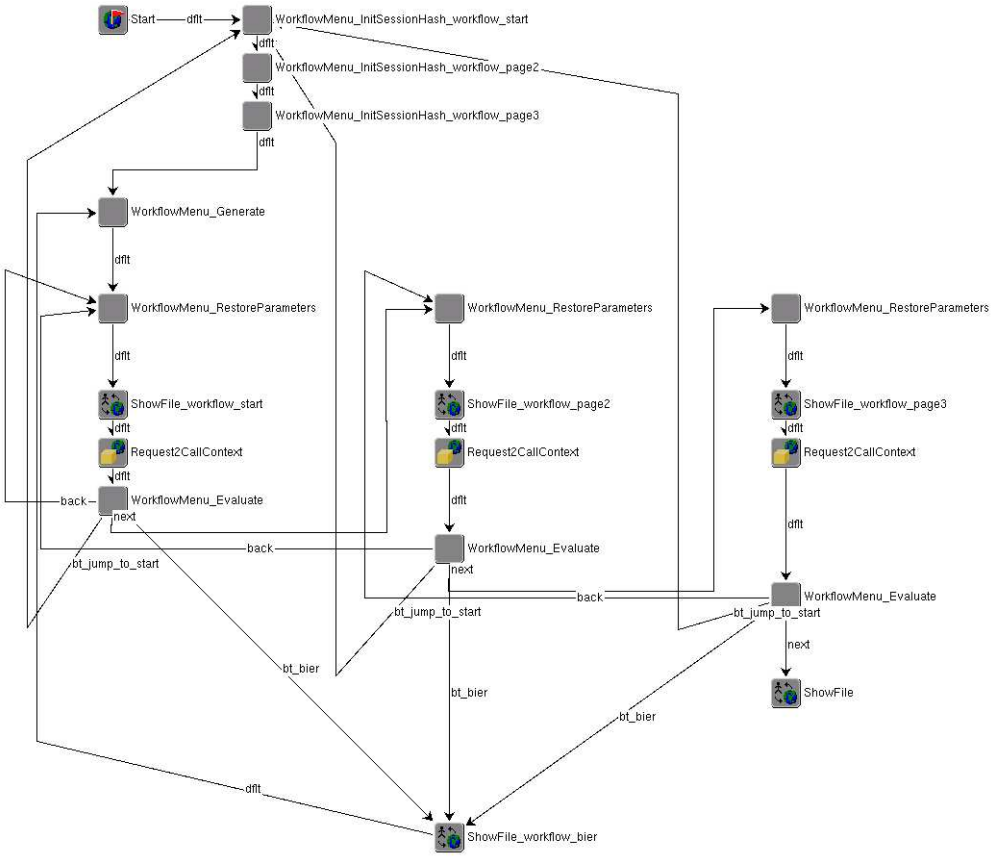

Abbildung 5.9: Workflow Navigationsmenü - Beispiel ABC-Graph

5.3 Erweiterung des VersionManagers

Das Zwischenthema "Erweiterung des VersionManagers" sollte von der Gruppe 3, bestehend aus Abdulatif Elwaschali, Dirk Löffler und Alexander Nolte, in einem Zeitrahmen von zwei Monaten bearbeitet werden. Der VersionManager hat folgende Aufgaben in der MaTRICS:

1. Realisierung der Versionskontrolle beliebiger Konfigurationsdateien der ConfigClients
2. Übersicht aller Versionen einer Datei
3. Differenzen bzw. Änderungen zwischen einzelnen Versionen
4. Anbindung an einen CVS Server zur Speicherung der Dateien
5. Mechanismus zur Sicherstellung der Aktualität der eingecheckten Dateien mit denen auf dem ConfigClient.
6. Anbindung an eine Datenbank zur Speicherung von Informationen, die mit CVS nicht abgebildet werden können

Ziele der Gruppe 3: Der Versionsmanager sollte mit Hilfe der von der Gruppe 3 im Rahmen des Kurzthemas entwickelten Methode beliebige Konfigurationsdateien nach XML konvertieren können. Diese sollten anschließend anstelle der originalen Konfigurationsdateien im CVS Repository verwaltet werden und damit innerhalb der MaTRICS diversen Modifikationen zur Verfügung stehen.
Vor der Übertragung zum ConfigClient mittels des ClientProtocolManagers sollten die Dateien wieder in ihr ursprüngliches Format zurückkonvertiert werden.
Die zu entwickelnden Komponenten sollten eine Erweiterung darstellen so dass auch weiterhin Konfigurationsdateien verwaltet werden können, die nicht nach XML konvertiert werden sollen oder können.

Das Design war in UML in Form von Aktivitätsdiagrammen, Anwendungsfalldiagrammen und Klassendiagrammen vorzunehmen. Dabei sollte das Design der vorherigen Projektgruppe als Grundlage dienen und um oben beschriebene Funktionalitäten erweitert werden. Die bereits vorhandene englische Dokumentation war ebenfalls zu aktualisieren bzw. zu überarbeiten.

Ein weiteres Ziel des Zwischenthemas war das Kennenlernen der MaTRICS in ihrem gesamten Umfang. Die einzelnen Komponenten sollten jedem Projektgruppenmitglied nahe gebracht werden, sodass jeder ein gutes Verständnis für die Zusammenarbeit der einzelnen Module der MaTRICS bekam, weitere Änderungen durchführen sowie Probleme erkennen und ggf. beheben konnte. Im Verlauf des Kurzthemas setzte sich die Gruppe 3 zwangsläufig eng mit dem Versionsmanager und der Datenbank auseinander.

5.3.1 Modellierung

Neue Aktivitäten

Um den VersionManager um die beschriebene Funktionalität erweitern zu können, mussten die folgenden 2 Aktivitäten angepasst werden:

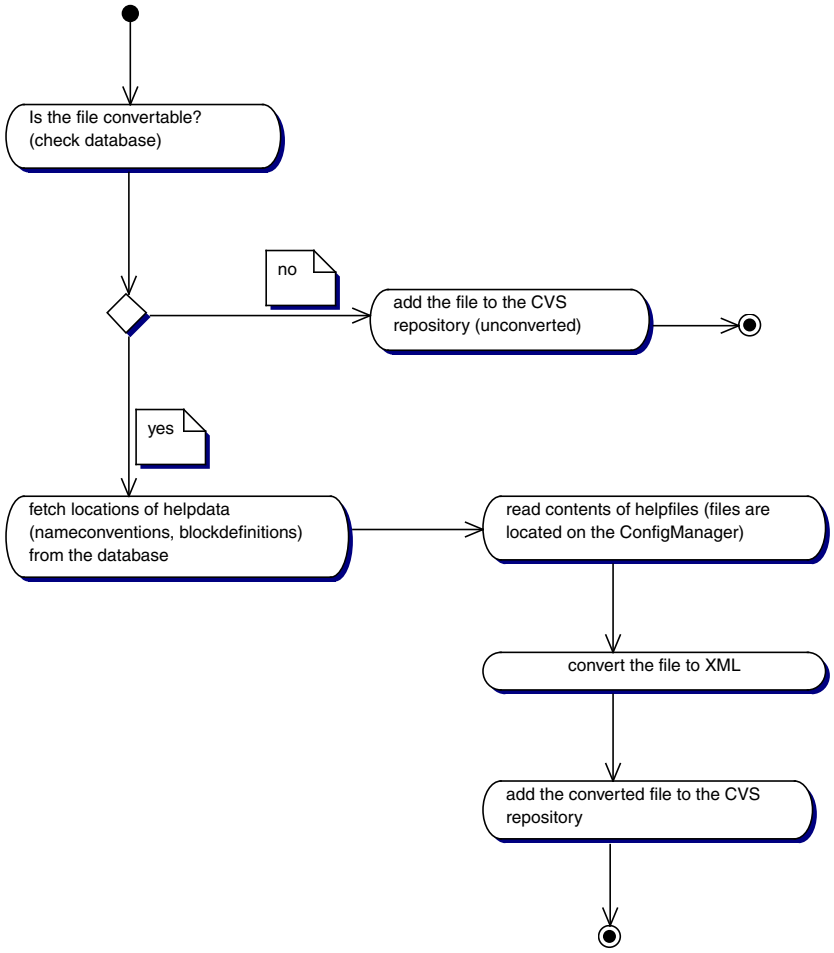

Abbildung 5.10: Erweiterung des VersionManagers - Aktivität für das Hinzufügen einer (konvertierten) Datei

- **Datei zum CVS Repository hinzufügen**
  Bevor eine Konfigurationsdatei neu zum CVS Repository hinzugefügt wird, wird anhand einer Datenbank überprüft, ob diese nach XML konvertierbar ist.
  Sollte die Konfigurationsdatei nicht konvertierbar sein, so wir sie zum CVS Repository hinzugefügt. Dies stellt sicher, dass auch nicht konvertierbare Dateien weiterhin

von der MaTRICS verarbeitet werden können.
Sollte die Konfigurationsdatei konvertierbar sein, so werden aus der Datenbank die Pfade für die benötigten Hilfsdateien (Blockdefinitionen, Namenskonventionen) aus der Datenbank geholt und die entsprechenden Dateien auf dem ConfigManager eingelesen. Anschließend wird die Konfigurationsdatei nach XML konvertiert und zum CVS Repository hinzugefügt.
Die beschriebene Aktivität wird in Abb. 5.10 dargestellt.

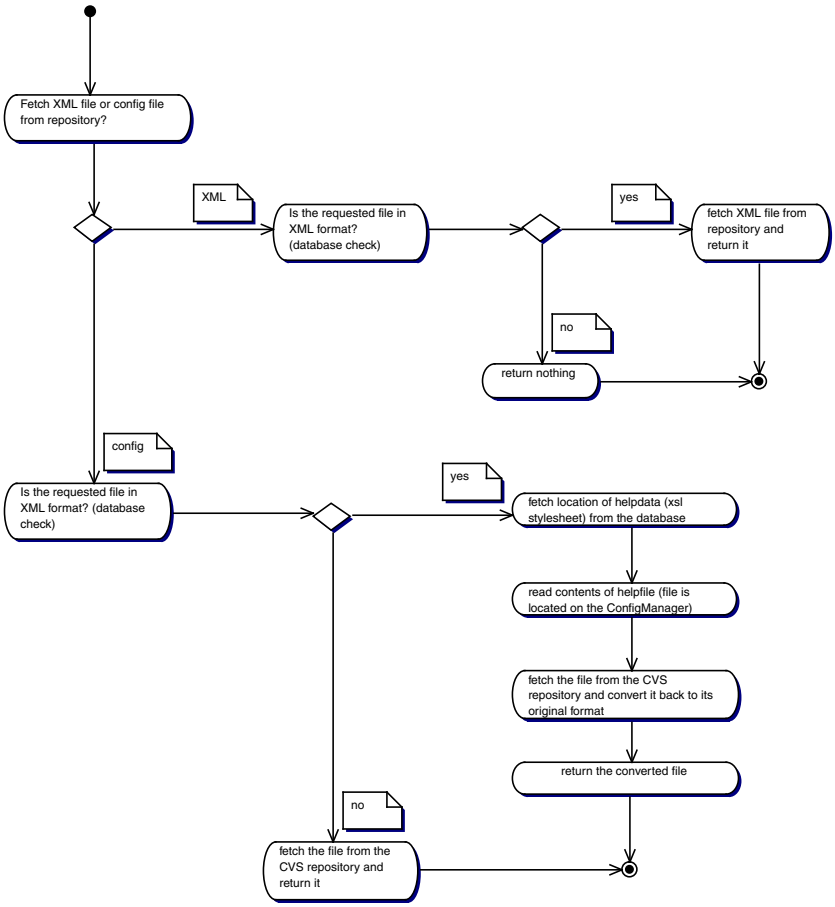

Abbildung 5.11: Erweiterung des VersionManagers - Aktivität für das Auschecken einer (konvertierten) Datei

- **Datei aus dem CVS Repository auschecken**
  Wenn eine Konfigurationsdatei aus dem CVS Repository ausgecheckt werden soll, muss zunächst entschieden werden, ob diese später als XML Datei oder in ihrem ursprünglichen Format weiterverarbeitet werden soll.
  Soll die angefragte Konfigurationsdatei im XML Format ausgecheckt werden, so

wird zunächst die Datenbank angefragt, ob die Konfigurationsdatei nach XML konvertierbar ist. Wenn sie konvertierbar ist, so liegt sie im CVS auch im XML Format vor und wird ausgecheckt. Ist sie nicht konvertierbar, so liegt sie im CVS nicht im XML Format vor und es wird nichts zurückgegeben.
Soll die angefragte Konfigurationsdatei in ihrem ursprünglichen Format aus dem CVS Repository ausgecheckt werden, so wird zunächst die Datenbank angefragt, ob die Konfigurationsdatei konvertierbar ist. Sollte dies der Fall sein, so wird der Pfad des zur Rückkonvertierung benötigten XSL Stylesheets aus der Datenbank ausgelesen und das Stylesheet auf dem ConfigManager eingelesen. Anschließend wird die Konfigurationsdatei in ihr ursprüngliches Format zuückkonvertiert und zurückgegeben. Sollte die Datei nicht konvertierbar sein, so wird sie nur aus dem CVS Repository ausgecheckt und zurückgegeben.

Erweiterung des Klassendiagramms

Wie in Abb. 5.12 zu sehen ist, wurden diverse Methoden zum Versionmanager hinzugefügt, um die Konvertierung in den bestehenden Workflow der MaTRICS anzubinden. Zur Anbindung an die in den Kurzthemen etwickelte Bibliothek zur Konvertierung von Konfigurationsdateien nach XML (vgl. Kap. 3.3) werden dabei die Methoden `collectDataForConversionToXml`, `startConversionToXml` and `startConversionToConfig` benötigt. Diese stellen sicher, dass der Bibliothek die benötigten Informationen zur Verfügung gestellt werden.
Des Weiteren wird die Methode `getXMLConfFileAdministrator` für die Anbindung an die Datenbank benötigt, die u. A. die Informationen über die Pfade der Hilfsdateien (Blockdefinitionen, Namenskonventionen) auf dem ConfigManager beinhaltet.
Schließlich wird die Aktivität **Datei zum CVS Repository hinzufügen** von der Methode `addConfigFileToRepositoryWithServiceFile` repräsentiert, während die Aktivität **Datei aus dem CVS Repository auschecken** von den Methoden `fetchConfigFileFromRepositoryWithServiceFile`) (zum Auschecken einer Konfigurationsdatei in ihrem ursprünglichen Format), `fetchXmlFileFromRepositoryWithServiceFile` (zum Auschecken einer Konfigurationsdatei im XML Format) und ihren weiteren Ausprägungen (active, latest) repräsentiert wird.

5.3.2 Implementierung

Um die Anbindung der Klassen zum Konvertieren von Konfigurationsdateien nach XML zu ermöglichen, musste zunächst eine neue Datenbank erstellt werden. Diese enthält einerseits die Pfade zu den benötigten Hilfsdateien (Blockdefinitionen und Namenskonventionen) und andererseits Informationen, die eine eindeutige Identifizierung der zu konvertierenden Datei über den ServiceManager zulassen. Dieser liefert zusätzlich die Information, ob die Datei nach XML konvertierbar ist oder nicht.

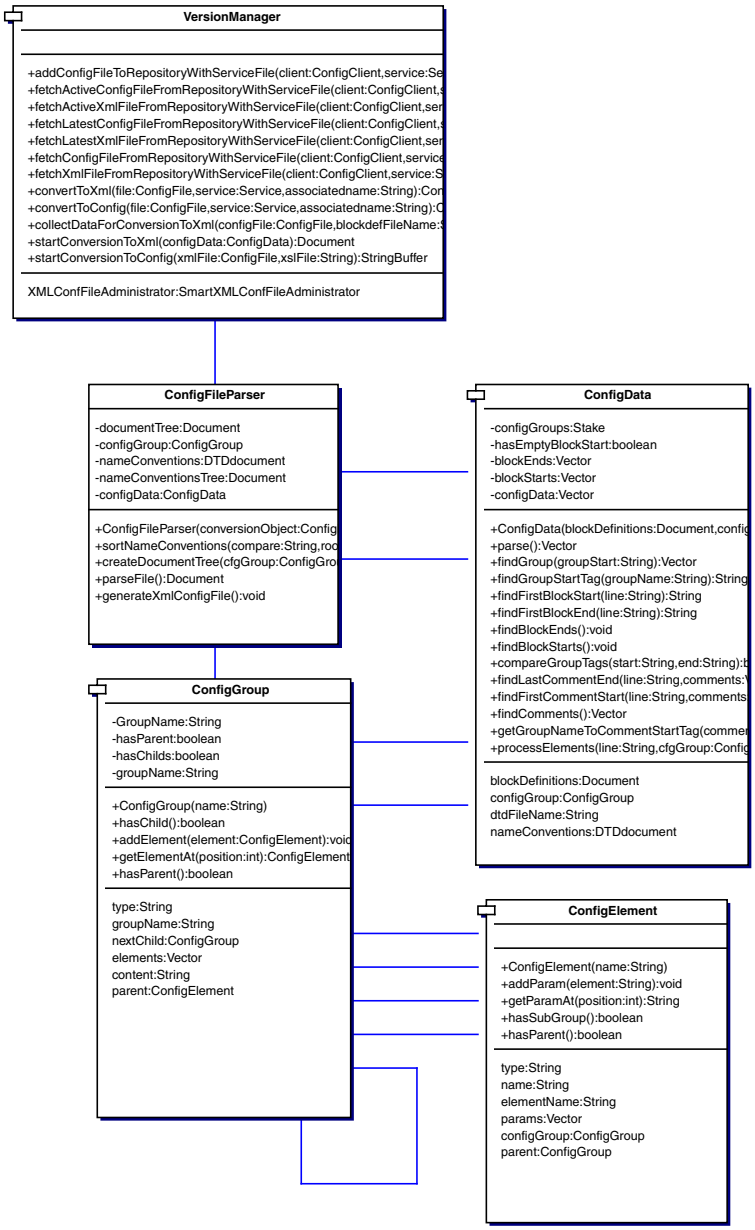

Abbildung 5.12: Erweiterung des VersionManagers - Erweiterungen im Klassendiagramm

Konvertierung nach XML
Die Konvertierung einer Konfigurationsdatei nach XML wird durchgeführt, wenn diese zum ersten Mal bearbeitet und danach zum CVS hinzugefügt wird. Dementsprechend wird die Datei zunächst vom ConfigClient auf den ConfigManager übertragen. Anschließend wird mit Hilfe des ServiceManagers überprüft, ob die Datei konvertierbar ist. Sollte dies der Fall sein, so werden aus der oben beschriebenen Tabelle die benötigten Informationen geholt, die Datei wird nach XML konvertiert und anschließend ins CVS eingecheckt. Sollte die Datei nicht konvertierbar sein, oder sollten die benötigten Hilfsdateien fehlen, so wird sie unkonvertiert ins CVS eingecheckt. Die beschriebene Funktion kann über den SIB `VersionMGR_AddConfigFileToRepositoryWithServiceFile` erreicht werden.

Konvertierung zum ursprünglichen Format
Die Datei wird zum ursprünglichen Format zurückkonvertiert, wenn sie wieder aus dem CVS ausgecheckt und z.B. auf den ConfigClient übertragen wird. Hierzu wird zunächst wieder überprüft, ob sie konvertierbar ist und ob die benötigten Hilfsdateien vorhanden sind, denn sollte dies der Fall sein, so wurde die Datei auch konvertiert ins CVS eingecheckt. Ist die Datei kovertierbar, so wird das benötigte XSL Stylesheet eingelesen und die Datei wird mit dessen Hilfe in ihr ursprüngliches Format gebracht. Sollte die Datei nicht konvertierbar sein, so ist sie auch unkonvertiert eingecheckt worden. Deswegen wird sie nun auch unkonvertiert ausgecheckt. Diese Funktion kann über die SIBs `VersionMGR_FetchConfigFileFromRepositoryWithServiceFile` (incl. Latest und Active) erreicht werden.

Auschecken ohne Konvertierung
Sollte man intern mit der eingecheckten XML-Datei weiter arbeiten wollen, ohne sie zurück zu konvertieren (wie etwa im Fall des UnixUserManagements), so können hierzu die SIBs `VersionMGR_FetchXmlFileFromRepositoryWithServiceFile` (incl. Latest und Active) genutzt werden. Diese geben die unkonvertierte XML-Datei zurück. Liegt die Datei nicht im XML-Format vor, so wird nichts zurückgegeben.

5.3.3 Tests

Die beschriebenen Funktionen wurden anhand des bereits vorhandenen Dienstes Edit ConfigFile getestet. Hierzu wurden zunächst die Tabellen manuell mit den benötigten Werten gefüllt (z.B.: Pfad zu den Namenskonventionen). In dem Test wurde als Beispiel die Konvertierung der Datei `passwd` nach XML durchgeführt. Bereits in dem Fenster zum Editieren des Dateiinhaltes lag sie nun in XML-Form vor und konnte angepasst werden. Nach erneutem Ausführen des Dienstes, konnte die Datei aus dem CVS ausgecheckt und anschließend in ihrem ursprünglichen Format bearbeitet werden.

Des Weiteren werden fast alle SIBs bereits im UnixUserManagement verwendet.

5.4 Erweiterung des ServiceManagers

Die drei Hauptkomponenten des Managements in der MaTRICS sind:

- HostManagement
- ServiceManagement
- GeneralControl und Navigation

ServiceManagement:

- Verwaltung der ”installierten” Konfigurations-Dienste innerhalb des ConfigManages
- Anbindung an eine Datenbank
- SLG koordiniert die Verwaltung der Dienste (Dienst deaktivieren, etc.)

Der ServiceManager war so zu erweitern, dass - bei der Registrierung eines neuen Konfigurations-Dienstes - auch Service-Dateien, die für diesen Dienst erforderlich sind, verwaltet werden konnten. Dabei kann es sich um Dateien beliebigen Typs handeln. Wir beschränkten uns jedoch auf die Unterscheidung von *Binär-* und *Konfigurationsdateien*. Da der VersionManager der MaTRICS darauf ausgelegt ist, solche Dateien mit in die Versionsverwaltung einzubeziehen und insbesondere bei Konfigurationdateien eine Konvertierung der Dateien nach XML bevorzugt, um diese später einheitlich bearbeiten zu können, entschieden wir uns für die Attribute *Binärdatei oder Konfigurationsdatei* sowie *konvertierbar in XML*.

Der Zugriff auf die Persistenzschicht, welche mit Hilfe einer PostgreSQL-Datenbank realisiert wurde, erfolgt via JDBC bzw. Smart-Konzept mit bereits vorgefertigten Basisklassen, welche vom Modul *EWIS-User* bereitgestellt werden. Dazu waren Administrator-Klassen zu implementieren, über die die Business-Objekte verwaltet werden.

Da Programme auf einzelnen ConfigClients an unterschiedlichen Positionen im Dateisystem installiert werden können, müssen - bei der Registrierung eines Services im ServiceManager - für jeden ausgewählten ConfigClient alle Konfigurationsdateien relativ zum Wurzelverzeichnis eines solchen Programms angegeben werden. Das Wurzelverzeichnis selbst wird dabei vom Administrator *ConfigClientServiceAssocAdministrator* verwaltet und wird im ServiceManagement mit ”`RootPath`” bezeichnet.

Diese neue Funktionalität sollte in den bereits existierenden Dienst des ServiceManagements integiert werden.

5.4.1 Modellierung

Innerhalb der Modellierungsphase wurde das bisherige Klassendiagramm verändert, um die Erweiterung des ServiceManagers anzupassen. Außerdem wurden bestehende Aktivitätsdiagramme geändert und neue Aktivitätsdiagramme hinzugefügt.

Änderungen im Klassendiagramm vom ServiceManager

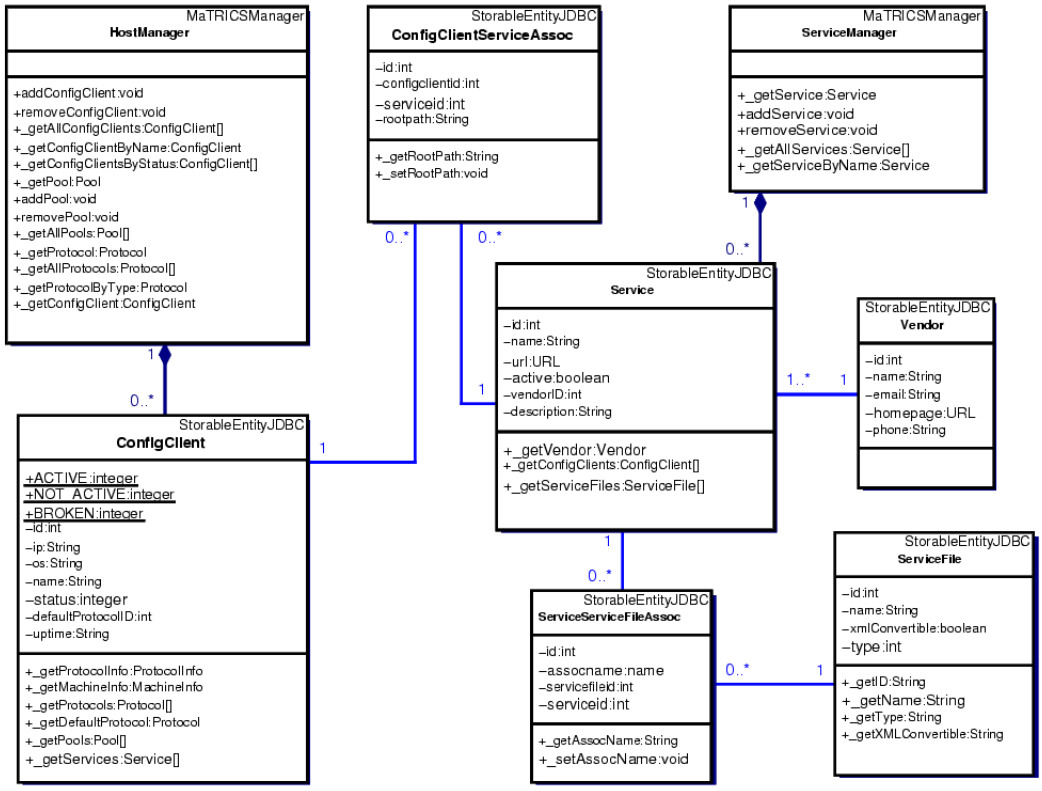

Abbildung 5.13: Erweiterung des Servicemanagers - Veränderung des Klassendiagramms

Der ServiceManager ist die zentrale Klasse des ServiceManagements und stellt mit Hilfe eigener SIBs ein Interface dar, von dem externe Klassen innerhalb eines ServiceLogicGraphen gebrauch machen können. Im *neuen* ServiceManager der MaTRICS können Konfigurationsdienste erstmalig sogenannte `ServiceFiles` erfordern. Hierzu wurde die neue Klasse `ServiceFile` dem Klassendiagramm hinzugefügt, die die Attribute `id`, `name`, `xmlConvertible` und `type` enthält. Die Verbindung zwischen den `Services` und den `ServiceFiles` stellt die Klasse `ServiceServiceFileAssoc` her. Diese hat ein zusätzliches Attribut `name`. Das Attribut `rootpath`, das für die Verwaltung des Wurzelverzeichnisses eines Programms auf den unterschiedlichen ConfigClients erstellt wurde, wurde in

die Klasse `ConfigClientServiceAssoc` platziert. Dieser Teil der Veränderung des Klassendiagramms ist von Abb. 5.13 gezeigt.

Ein neues Aktivitätsdiagramm im ServiceManager

Es gibt mehrere neue Aktivitätsdiagramme für die Erweiterung des ServiceManagers der MaTRICS. Um ein Beispiel zu zeigen, wird hier das Aktivitätsdiagramm für das Hinzufügen eines neuen *ServiceFiles* vorgestellt (Siehe Abb. 5.14).

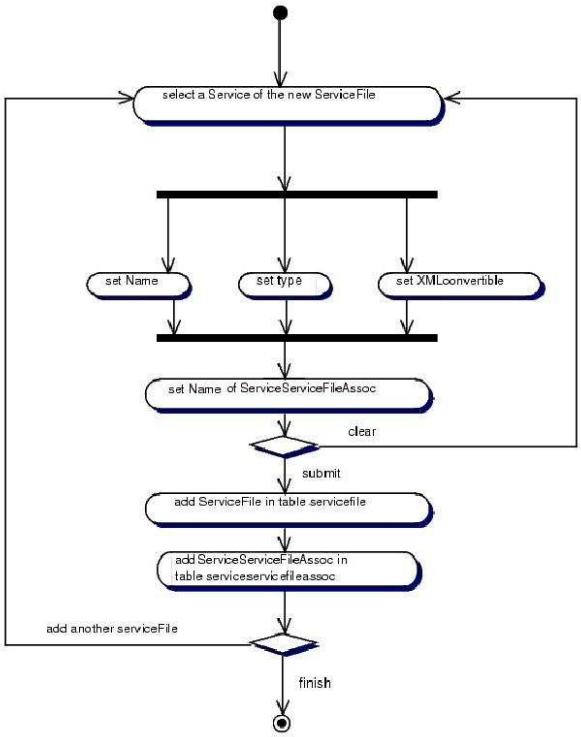

Abbildung 5.14: Erweiterung des Servicemanagers - Aktivitätsdiagramm für das Hinzufügen eines *ServiceFiles*

Für das Hinzufügen eines neuen *ServiceFiles* muss man zunächst einen Dienst in der MaTRICS hinzufügen oder bearbeiten. Danach werden die Attribute `id`, `name`, `xmlConvertible` und `type` festgelegt. Abschließend wird noch ein assoziierter Name für die Einträge in die Tabelle `ServiceServiceFileAssoc` gesetzt. Schließlich werden die neuen Dateinamen und -pfade in die Datenbank der MaTRICS gespeichert.

5.4.2 Implementierung

Die Implementierung in ABC wurde durch die Erweiterung des ServiceLogicGraphen ”ServiceManagement” der PG-451 umgesetzt.

Der *ServiceManagement*-Graph startet mit einem ShowFile-SIB, gefolgt von fünf Verzweigungen, für die Auswahl eines Teildienstes. Die drei Teildienste `listAllServices` zum Anzeigen und Modifizieren bereits konfigurierter Dienste, `addService`-Dienst zum Hinzufügen neuer Dienste sowie `listAllVendors` zum Anzeigen aller bislang definierten Service-Erzeuger wurden bereits von der PG-451 erstellt.

Zwei weitere Teildienste `fileSystem` und `rootPath` wurden von uns neu hinzugefügt. Alle *ServiceFiles* samt deren Beziehungen zu den Diensten werden unter `fileSystem` verwaltet. Unter `rootPath` werden Wurzelverzeichnisse der Programme auf den unterschiedlichen ConfigClients verwaltet.

Für die neuen Funktionalitäten des ServiceManagements wurden folgende SIBs geschieben:

- SIB ServiceMGR_AddServiceFileAndAssoc: eine neue Datei und den entsprechenden ServiceServiceFileAssoc-Eintrag hinzufügen.
- SIB ServiceMGR_AddServiceServiceFileAssoc: einen neuen ServiceServiceFileAssoc-Eintrag hinzufügen
- SIB ServiceMGR_EditRootPath: den Wert eines Wurzelverzeichnisses editieren
- SIB ServiceMGR_EditServiceFileAndAssoc: eine Datei und ihren ServiceServiceFileAssoc-Eintrag editieren
- SIB ServiceMGR_GetAllFreeServiceFiles: alle Dateien ausgeben, die mit keinem Dienst durch einen ServiceServiceFileAssoc verknüpft sind (diese Dateien werden ”Free File” genannt).
- SIB ServiceMGR_GetAllServiceServiceFileAssocsByServiceID: alle ServiceServiceFileAssoc-Objekte eines Diensts holen
- SIB ServiceMGR_GetConfigCientServiceAssocByServiceID: alle ConfigClientServiceAssoc-Objekte eines Diensts holen
- SIB ServiceMGR_GetOtherServiceFilesByServiceName: liefert alle *ServiceFiles* aus der Datenbank, die nicht mit dem Dienst verbunden sind.
- SIB ServiceMGR_GetServiceByName: liefert das Service-Objekt mit dem angegebenen Namen.
- SIB ServiceMGR_GetSerivceFileByChoice: eine ausgewählte Datei aus einer Liste in der Form ”#id: name” holen
- SIB ServiceMGR_GetServiceSerivceFileAssocByID: einen bestimmten ServiceSerivceFileAssoc aus der Datenbank holen

- SIB ServiceMGR_RemoveServiceFile: einen bestimmten Datei aus der Datenbank löschen
- SIB ServiceMGR_RemoveServiceServiceFileAssoc: ein bestimmtes ServiceServiceFileAsso-Objekt bzw. den Datenbankeintrag löschen

Abb. 5.15 und Abb. 5.16 zeigen zwei Teile des ServiceLogicGraphs ”ServiceManagement”.

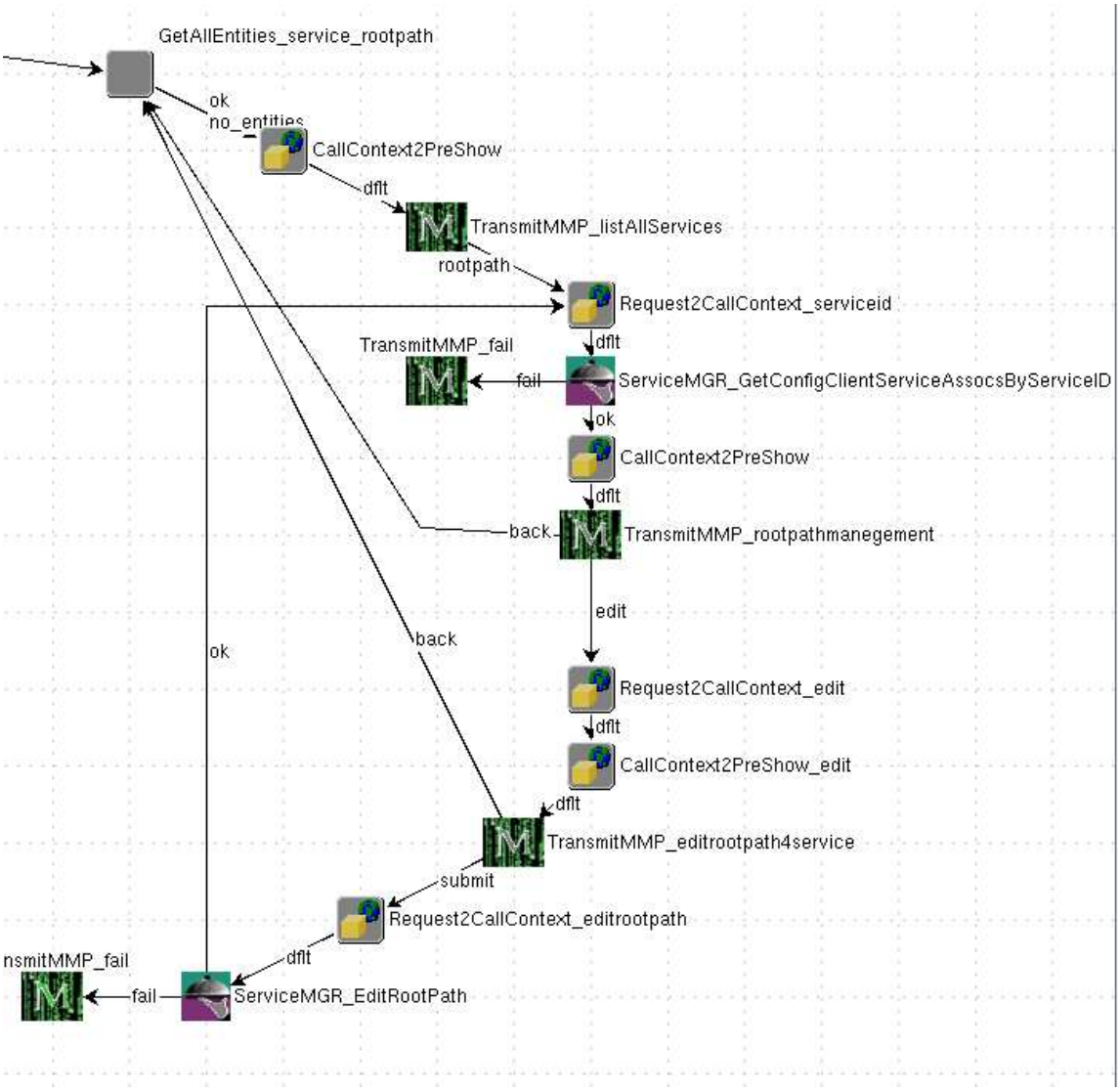

Abbildung 5.15: Erweiterung des Servicemanagers - Teil des SLG: Wurzelverzeichnis Editieren

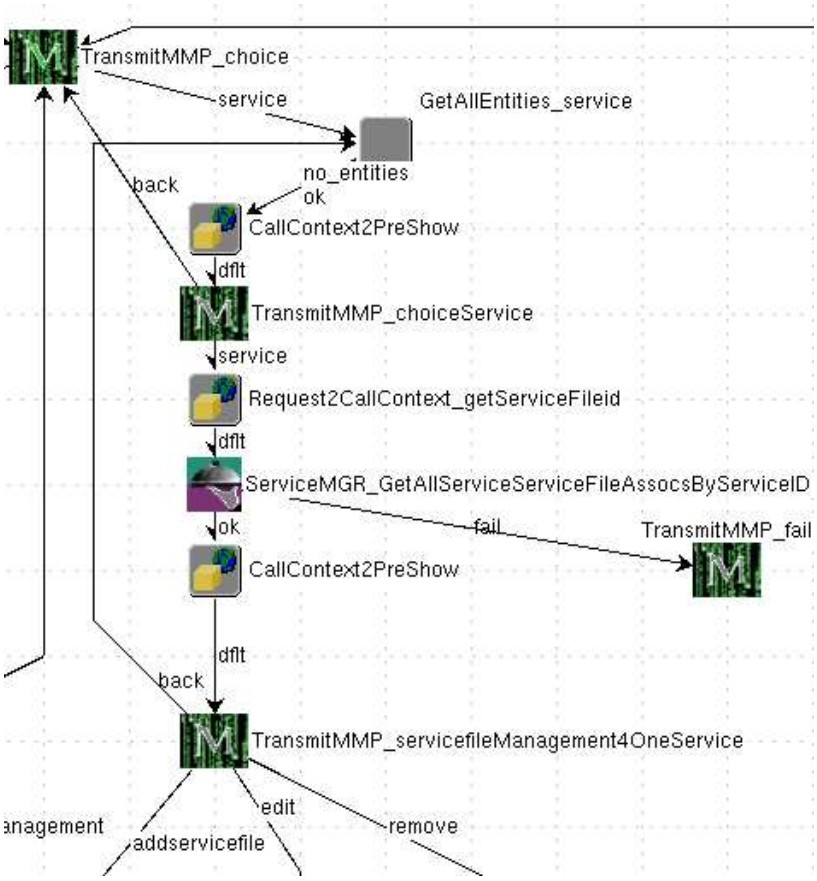

Abbildung 5.16: Erweiterung des Servicemanagers - Teil des SLG: Alle Datein für einen Dienst Anzeigen

5.4.3 Tests

Die Startseite des ServiceManagements sieht nun wie Abb. 5.17 aus.

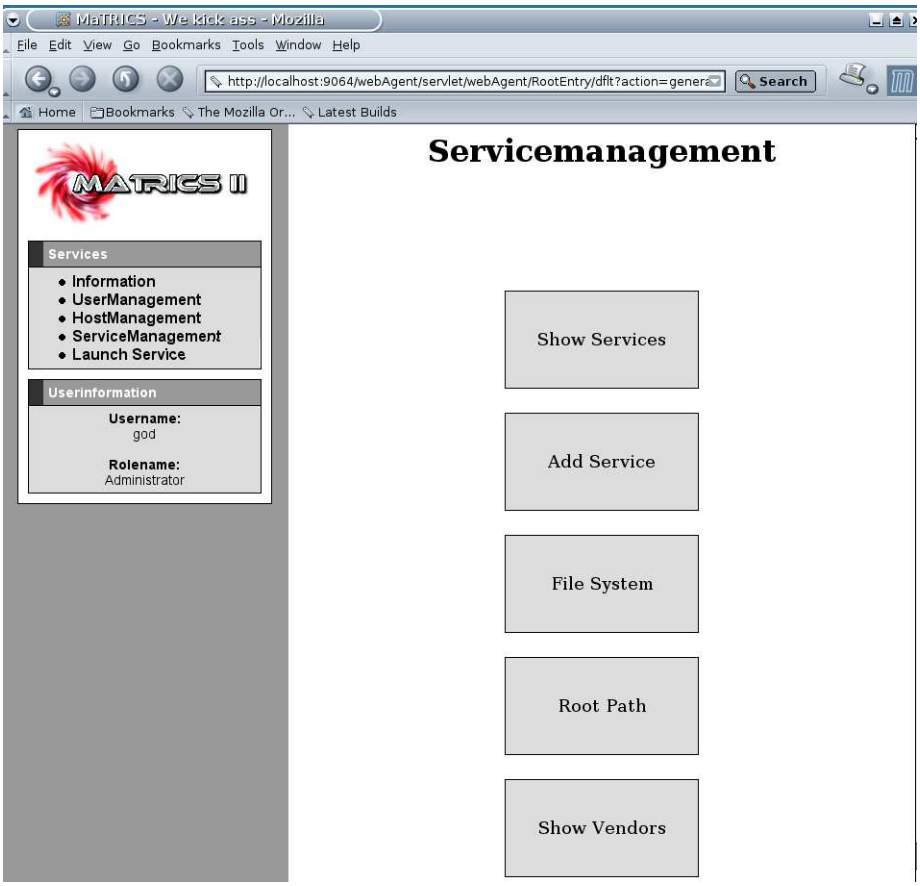

Abbildung 5.17: Erweiterung des Servicemanagements - neue Startseite des ServiceManagements

Beispiel 1: Anzeige aller Dateien

Abb. 5.18 zeigt eine Webseite, die alle verfügbaren *ServiceFiles* anzeigt. Die obere Tabelle zeigt alle ServiceServiceFileAssoc-Einträge. Für jeden dieser Einträge werden auch die Informationen der entsprechenden Datei angezeigt. Der Name des Diensts wird auch gezeigt. Durch entsprechende Auswahl kann man jeden dieser Einträge entfernen oder bearbeiten. Unter der Tabelle befindet sich eine entsprechende Verknüpfung, um neue *ServiceFile*-Einträge hinzuzufügen.

Die untere Tabelle zeigt alle diejenigen Dateien, die noch keinem Dienst zugeordnet sind (`Free Files`). Jede dieser Dateien kann gelöscht werden.

Beispiel 2: Editierung des Wurzelverzeichnisses

Abb. 5.19 zeigt die grafische Schnittstelle zum Editieren des Wurzelverzeichnisses.

Service File Management:

ServiceServiceFile AssocName	Assoc ID	Service File	File ID	Type	XML convertible	Service	Actions	
pw	1	passwd	1	plain text	true	**UnixUserManagement**	edit	remove
gr	2	group	2	plain text	true	**UnixUserManagement**	edit	remove
ng	3	netgroup	3	plain text	true	**UnixUserManagement**	edit	remove
sh	4	shadow	4	plain text	true	**UnixUserManagement**	edit	remove
testassoc2	6	testsf2	7	binary	true	**test**	edit	remove
testassoc3	7	testservicefile4	6	plain text	true	**test**	edit	remove
test2assoc	8	testservicefile4	6	plain text	true	**test2**	edit	remove

add service file and serviceServiceFileAssoc

Free service file(s) (service files that are not associated with a serviceServiceFileAssoc) :

Service File ID	Service File	Service File Type	XML convertible	Action
5	passwd.dtd	plain text	false	remove
8	testz1	plain text	true	remove
9	testz2	plain text	true	remove

back

Abbildung 5.18: Erweiterung des Servicemanagements - Startseite des ServiceManagements

root path in every config client of service " example " :

ConfigClientID	ConfigClientIP	ConfigClientName	Root Path for Service	Action
4	100.100.100.104	auge.cs.uni-dortmund.de	/home/etc/	edit root path
7	129.217.27.130	werdohl.ls05.cs.uni-dortmund.de	/etc/	edit root path
6	129.217.27.140	brilon.ls05.cs.uni-dortmund.de	/home/user2/example/	edit root path
5	129.217.27.135	olsberg.ls05.cs.uni-dortmund.de	/etc/	edit root path

back

Abbildung 5.19: Erweiterung des Servicemanagements - Startseite des ServiceManagements

Kapitel 6

Hauptthemen

Nachdem wir nun also einige der Komponenten des ConfigManagers verbessert und erweitert hatten, konnten wir uns daran begeben Dienste für die MaTRICS zu entwickeln. Dieses Mal wurden zwei Gruppen mit jeweils 6 Teilnehmern gebildet um die Aufgaben zu bearbeiten.

- Gruppe 1 sollte einen Dienst für die MaTRICS entwickeln mit welchem es möglich sein sollte die Pakete auf einem Debian System mit APT zu verwalten.
- Gruppe 2 sollte einen Dienst entwickeln mit welchem eine Benutzerverwaltung auf Linux Systemen möglich ist.

Außerdem wurde dieses Mal die Organisation gänzlich in die Hände der Teilnehmer gelegt. Jeder Teilnehmer bekam eine Aufgabe zugeteilt für die er verantwortlich war. Mit Verantwortlich ist hier gemeint, dass die Aufgabe zwar von allen Teilnehmern bearbeitet wird, der Verantwortliche jedoch für den Erfolg und die Qualität der Arbeiten verantwortlich ist.
Die Aufteilung wurde von den Teilnehmern wie folgt festgelegt:

- **Gruppe 1**:
  Gruppensprecher: Jan Hendrik Jellesma
  Software Design: Said El Bouzakri
  Workflow Design: Yunchan Cho
  Implementierung: Alexander Nolte
  Testen: Abdulatif Elwashali
  Dokumentation: Dirk Löffler
- **Gruppe 2**:
  Gruppensprecher: Michael Krücken

Software Design: Nabil Azzam Jai
Workflow Design: Quingha Fan
Implementierung: Pavel Baranov
Testen: Jianping Xie
Dokumentation: Christoph Beerens

6.1 APT Service

Innerhalb der MaTRICS war ein Konfigurationsdienst zu entwickeln mit dem sich Installationsaufgaben mit Hilfe des APT-Tools (Advanced Package Tool) ausführen lassen. Bei APT handelt es sich um ein (Software-)Paket Management System welches im Umfeld der Linux Distribution Debian entwickelt wurde. Unser Dienst sollte das Zusammenspiel der folgenden Dienste verwirklichen:

- Installieren von Paketen
- Entfernen von Paketen
- Aktualisieren von Paketen
- Verwaltungsdienste

Die Grundlage des APT sind die Repositories aus welchen die Pakete bezogen werden. Eine Liste dieser Repositories wird in der Datei sources.list verwaltet. Zu den Verwaltungsdiensten gehört es diese Liste bearbeiten zu können.
Die MaTRICS sollte den Umgang mit APT erleichtern und robuster gegen Fehler machen. Außerdem sollte es möglich sein, Installationsaufgaben auf Rechnerpools durchführen zu können ohne jeden Rechner einzeln bedienen zu müssen.
Die nächsten Kapitel werden zeigen, wie wir die Lösung dieser Aufgabe angegangen sind und welche Vorteile der Benutzer durch die MaTRICS erhält.

6.1.1 Modellierung

Die Aufgabe von APT-Service
Die MaTRICS sollte ein APT-Service als Installationssoftware, mit Hilfe von Debian-APT-Tools entwickelt, erhalten. Das Ziel ist es Software-Pakete in unterschiedlichen Versionen installieren, deinstallieren, upgraden, neu konfigurieren und das ganze System aktualisieren.

Mit dem UML-Design wird der APT-Service wie folgt modelliert:

Anwendungsfall
Der Anwendungsfall zeigt einen groben Ablauf der Aufgabe (Abb. 6.1). Der APT-Services ermöglicht dem Benutzer die Software-Änderungen auf eine Gruppe von Rechnern – ConfigClients in einem Pool – durchzuführen oder auf einen Rechner in einem beliebigen Pool zuzugreifen und Pakete in verschiedenen Versionen mit Hilfe von Debian-APT-Tools zu installieren, reinstallieren, updaten, upgraden und die Sourcen zu verwalten.

- **Installieren bzw. Reinstallieren:** In diesem Anwendungsfall können Pakete auf einem System durch den Benutzer installiert bzw. reinstalliert werden. Bei diesem Vorgang wird automatisch die Abhängigkeit zu anderen Paketen überprüft, ggf. werden dabei neu benötigte Pakete auf der Basis der Informationen in `/etc/apt/sources.list` installiert.
- **Deinstallieren:** ein Paket aus dem System löschen bzw. deinstallieren.
- **Liste der verfügbaren Pakete aktualisieren:** In diesem Fall aktualisiert der Benutzer die Liste der verfügbaren Pakete. APT liest also die Informationen aus den Dateien `Packages.gz` des jeweiligen Systems ein, die in den Archiven `/etc/apt/sources.list` stehen.
- **Upgrade:** Um alle auf dem System installierten Pakete auf die aktuelle Version zu bringen, kann der Benutzer ein Upgrade auf einem bzw. auf mehreren Systemen (in einem Pool) durchführen. Damit werden alle bereits installierten Pakete, von denen eine neuere Version verfügbar ist, aktualisiert.
- **Sourcen verwalten:** Hier wird die Konfigurationsdatei `/etc/apt/sources.list` bearbeitet. In dieser Datei befinden sich die Informationen, von welcher Quelle die Pakete geholt werden sollen. Es gibt verschiedene Installationsquellen, wie zum Beispiel cdrom, file, http und ftp, die man ändern bzw. hinzufügen kann.
- **Liste der verfügbaren Paketen anzeigen:** Es wird eine Liste aller verfügbaren Pakete angezeigt.
- **Liste der installierten bzw. nicht installierten Paketen anzeigen:** Eine Liste der installierten bzw. nicht installierten Pakete wird angezeigt.

Aktivitätsdiagramme
apt-update-activity:
In diesem Teil des neuen Moduls (siehe Abb. 6.2) der MaTRICS soll die Paketliste eines ConfigClients aktualisiert werden. Dabei wird der Pool ausgewählt, in dem der ConfigClient steht. Anschließend wählt man den ConfigClient selbst aus und baut mittels des SSH-Protokolls eine Verbindung zu diesem ConfigClient auf. Nun kann für diesen ConfigClient die Liste der verfügbaren Pakete mit dem Befehl `apt-get update` ermittelt werden. Diese Liste (`sources.list`) beinhaltet die verfügbaren Pakete des ConfigClients, welche für weitere Anwendungen, zum Beispiel für das re-install oder das update benötigt wird. Diese Liste wird anschließend in die Datenbank geschrieben, damit sie für weitere Anwendungsfälle zur Verfügung steht.

Liste der installierten Pakete:
In Debian-Systemen gibt es die Datei `status`, wo die verfügbaren Pakete und ihr entsprechender Status steht. Die Datei wird aus dem CVS ausgecheckt. Danach wird sie geparst, gelesen und schließlich angezeigt. Siehe Abbildung 6.3

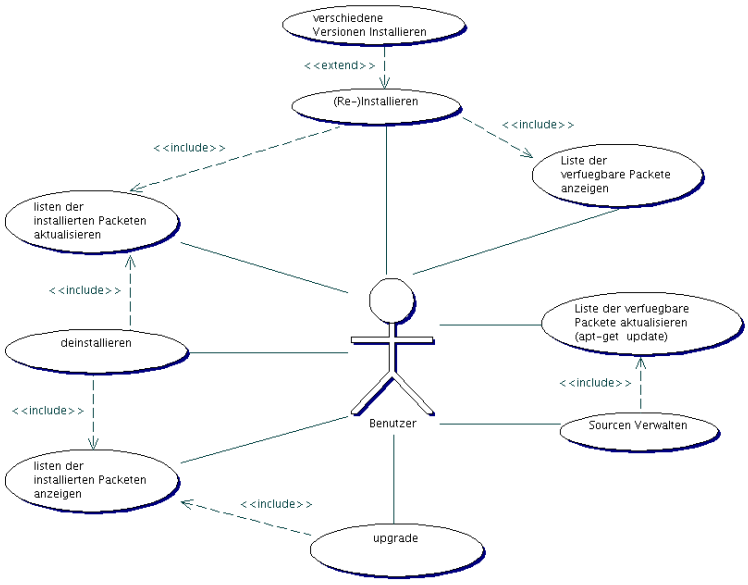

Abbildung 6.1: APT - Anwendungsfalldiagramm

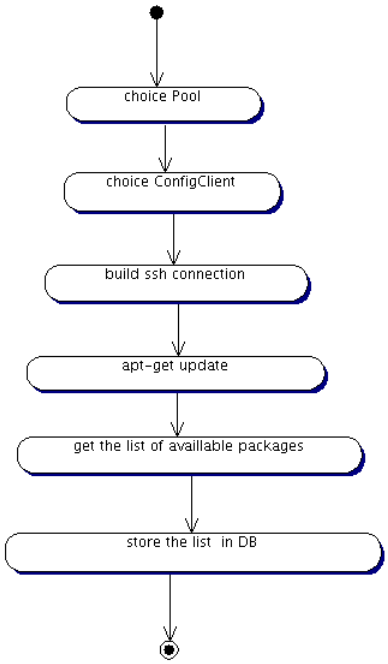

Abbildung 6.2: APT - Aktivitätsdigramm: apt update

Update: Liste der installierten Pakete:
Dieser Teil des APT-Moduls beschäftigt sich mit Informationen über installierte, nicht installierte sowie über die Installation verfügbarer Pakete. Das Programm `apt-get` benutzt eine Datenbank, um herauszufinden, wie es die vom Benuter angeforderten Pakete installieren soll (z.B. wichtig für den bereich (re-)install) und welche zusätzlichen Pakete benötigt werden, damit die ausgewählten Pakete ordentlich funktionieren. Diese Liste wird mittels des Kommandos `apt-get update`.`apt-get` aktualisiert und befindet sich üblicherweise auf dem ConfigClient (`etc/apt/sources.list`). In diesem Teil des Moduls geschieht dies auf die im Aktivitätsdiagramm gezeigte Art und Weise (siehe Abb. 6.3). Es wird ein Pool und ein ConfigClient aus diesem ausgewählt. Danach erfolgt der Aufbau einer SSH-Verbindung zum gewählten ConfigClient. Die `sources.list` wird aus dem CVS geholt, mittels `update.apt-get` aktualisiert, angezeigt und anschließend in die Datenbank geschrieben.

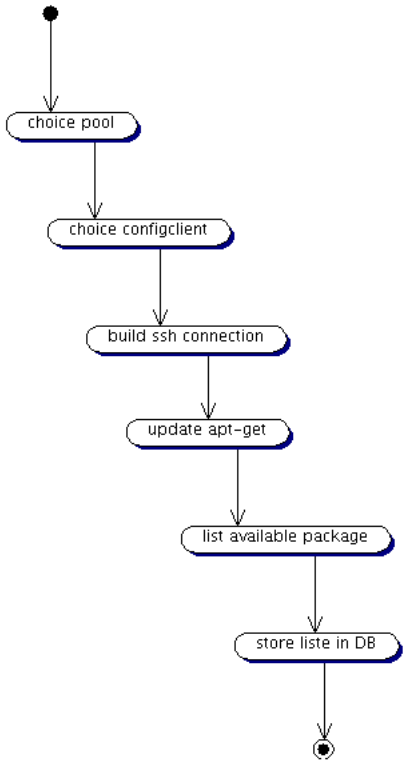

Abbildung 6.3: APT - Aktivitätsdigramm: Liste der installierten Pakete

apt-upgrade-activity:
Der Befehl `upgrade` aktualisiert die installierten Pakete auf einem ConfigClient (siehe

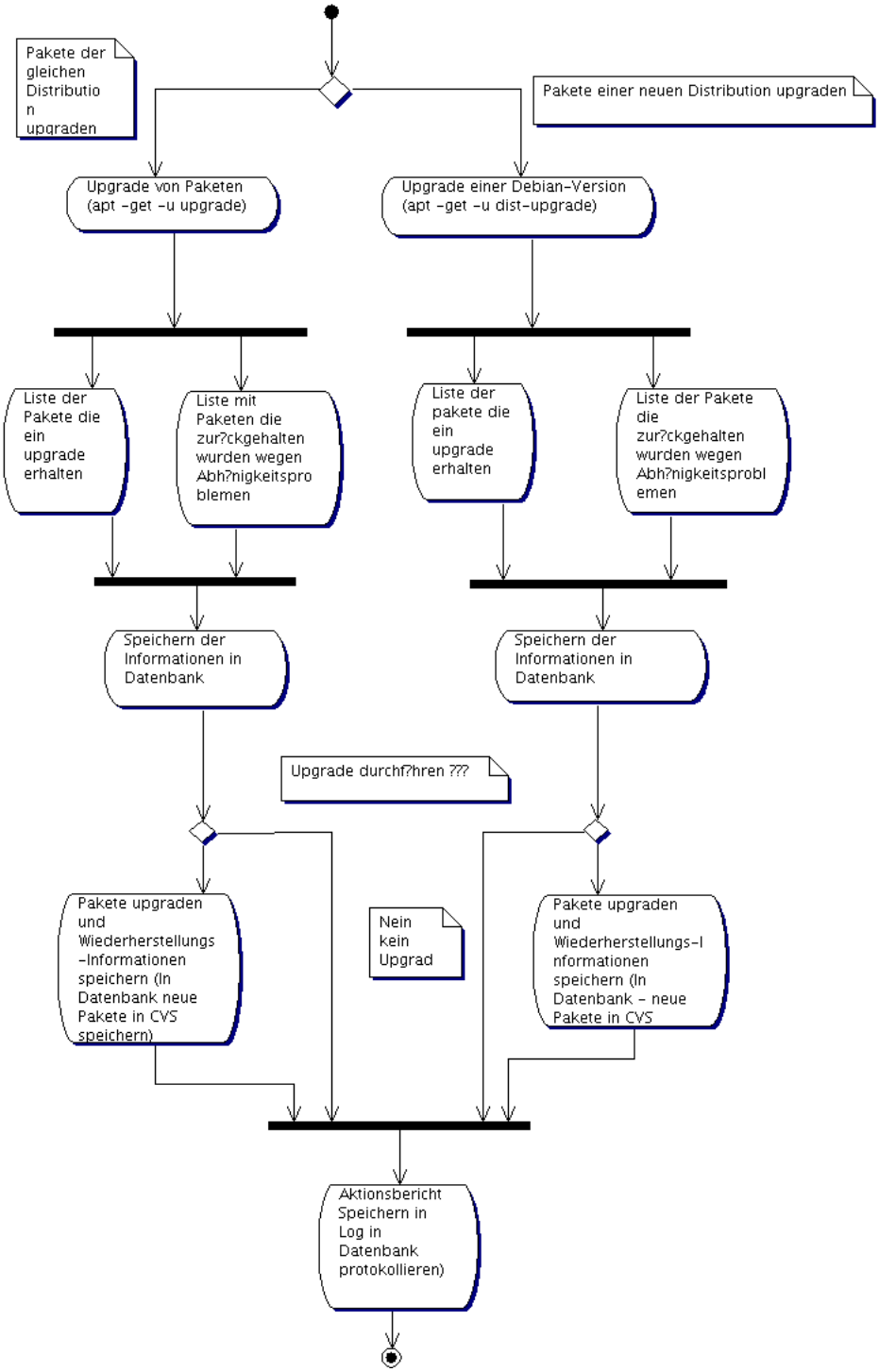

Abbildung 6.4: APT - Aktivitätsdigramm: APT-Upgrade

Abb. 6.4). Dabei wird zwischen zwei Anwendungsfällen unterschieden.
Im ersten Fall (apt -get -u upgrade) werden Pakete der gleichen Distribution aktualisiert. Während dieser Aktualisierung werden wiederum zwei Listen benötigt. Die eine Liste beinhaltet die Pakete, die eine Aktualisierung erhalten, während die andere Liste jene Pakete beinhaltet, welche zurückgehalten wurden. Zurückgehaltene Pakete sind solche, die aus irgendeinem Grund nicht installiert werden können, zum Beispiel wegen unerfüllbarer Abhängigkeiten oder ähnlichem. Sobald diese beiden Listen verfügbar sind, werden sie in die Datenbank geschrieben. Der Benutzer kann nun entscheiden, ob er das Aktualisieren durchführen möchte oder nicht. Sollte an dieser Stelle abgebrochen werden, so wird trotzdem ein Aktionsbericht in die Datenbank geschrieben. Entscheidet sich der Benutzer für eine Aktualisierung, so werden die Pakete der ersten Liste aktualisiert. Für eine spätere Wiederherstellung des alten Systems werden die Aktualisierungen in die Datenbank geschrieben. Zusätzlich werden die neuen Pakete ins CVS gespeichert. Auch hier erfolgt eine Speicherung des Aktionsberichtes in der Datenbank.
Im zweiten Fall (apt -get -u dist-upgrade) wird eine Aktualisierung einer Debian-Version durchgeführt. Hierbei ist zu bemerken, dass Pakete aktualisiert werden, aber auch neue Pakete (also neue Abhänigkeiten) zusätzlich installiert werden. Auch hierbei gibt es die zwei Listentypen des ersten Ablaufes. Der Unterschied zwischen den beiden Fällen besteht lediglich in der Installation neuer Pakete mit ggf. neuen Abhänigkeiten, wobei beim ersten Fall lediglich die Pakete selbst ein Update erhalten und keine neuen Pakete installiert werden. Der weitere Verlauf ist analog zum ersten Fall.

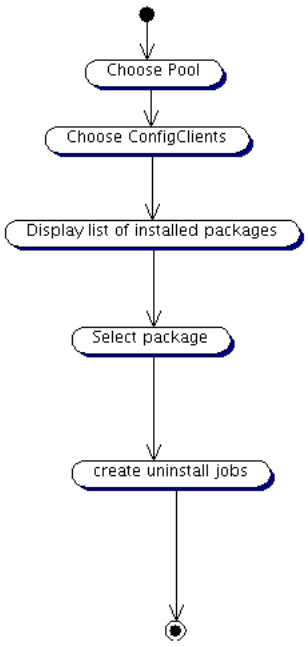

Abbildung 6.5: APT - Aktivitätsdigramm: APT-Uninstall

apt-uninstall-activity:
Diese Aktivität des APT-Moduls kann einzelne Pakete von einem ConfigClient wieder entfernen (siehe Abb. 6.5). Dazu muss wiederum der Pool mit dem ConfigClient gewählt werden. Die Auswahl des ConfigClients erfolgt direkt im Anschluss an die Poolwahl. Danach kann die Verbindung zum ConfigClient hergestellt werden bzw. eine Liste der installierten Pakete aufgerufen werden. Anhand dieser Liste wählt der Benutzer nun jene Pakete aus, die vom ConfigClient entfernt werden sollen. Sobald die Auswahl abgeschlossen ist, wird der uninstall Job erzeugt und an den Jobmanager weitergereicht. Anschließend wird intern die Liste der installierten Pakete angeglichen, sodass bei erneutem Aufrufen der Funktion die aktuelle Liste vorhanden ist.

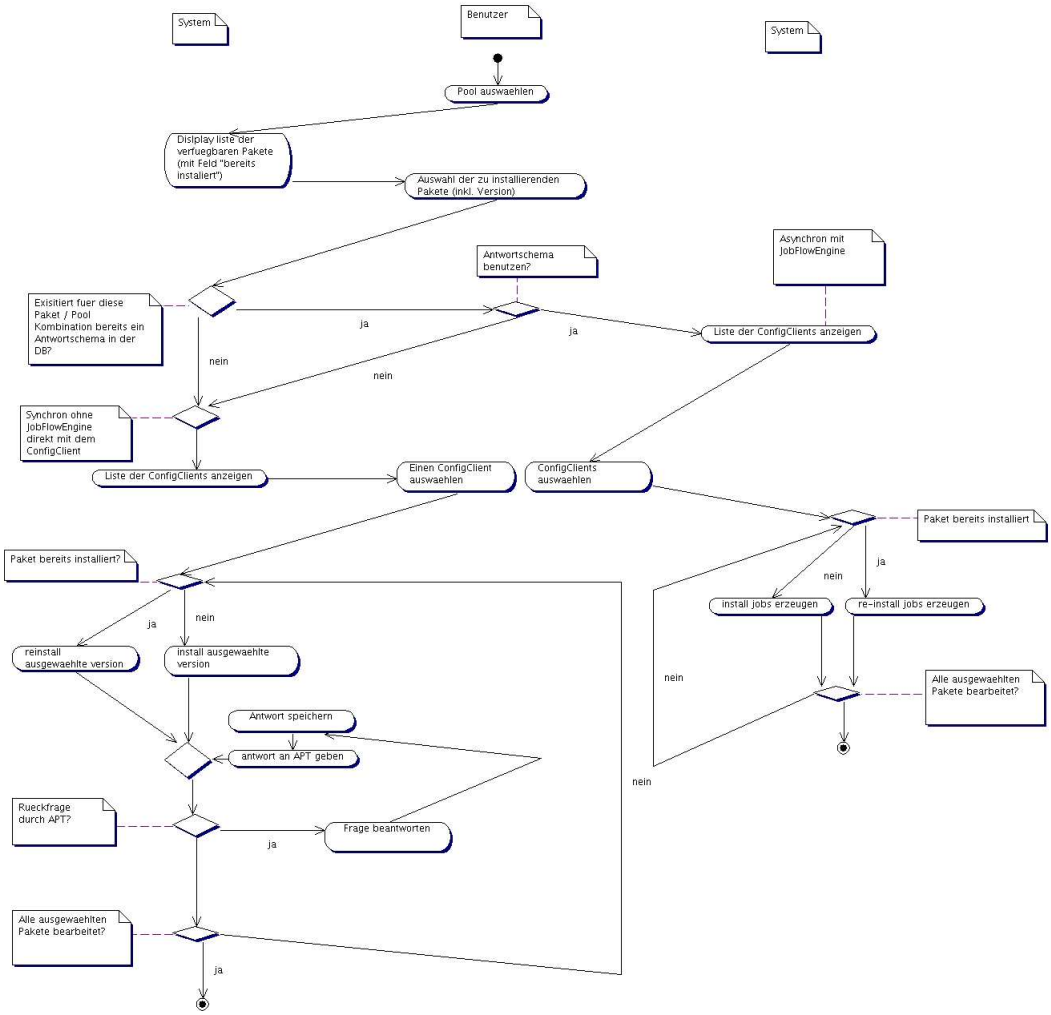

Abbildung 6.6: APT - Aktivitätsdigramm: APT-(re)install

apt-(re-)install-activity:
Bei diesem Teil des APT-Moduls kommen zwei Fälle vor. Zum einem bezieht sich diese Funktion auf ein Reinstallieren vorhandener Pakete, um diese neu zu installieren. Diese Funktion kann hilfreich sein, wenn neuere Versionen vorhandener Pakete verfügbar sind oder installierte Pakete defekt sind. Zum Anderen bezieht sich diese Funktion auf das Neuinstallieren von Paketen (siehe Abb. 6.6). Beide Fälle unterscheiden sich also nur in einem Punkt: wenn die gewählten Pakete schon installiert sind, wird ein reinstall, sonst ein install ausgeführt. Beide Fälle können auch vermischt auftreten. Somit können beide Fälle analog beschrieben werden. Eine Abfrage im Ablauf stellt sicher, ob Pakete vorhanden sind oder installiert werden müssen. In beiden Fällen gilt es allerdings zu beachten, ob es schon ein sogenanntes Antwortschema in der Datenbank gibt. Sollte es kein solches Schema geben, so muss eins erstellt werden, was wiederum eine synchrone Abarbeitung des Falles mit sich bringt. Sollte schon ein Antwortschema vorhanden sein, so kann der Benutzer entscheiden dieses zu verwenden, was ein asynchrones Abarbeiten des Reinstallieren bzw. Neuinstallieren ermöglicht. Dieses vorhandene Antwortschema kann allerdings auch verworfen werden, womit wieder der erste Unterfall ausgeführt wird (synchroner Ablauf). Beide Fälle starten mit der Wahl eines Pools und dem Anzeigen von den verfügbaren Paketen inklusive eines Feldes der bereits im Pool/ConfigClient installierten Pakete. Im nächsten Schritt wählt der Benutzer die zu (re-)installierenden Pakete anhand einer Liste mit den verfügbaren Paketen aus.
Gibt es für die getroffene Auswahl kein Antwortschema oder wurde das vorhandene Antwortschema vom Benutzer abgelehnt, so wird synchron installiert, d.h. es wird direkt ein ConfigClient ausgewählt, via SSH verbunden und ohne JobFlowEngine installiert. Nach der Wahl eines ConfigClients wird nun für jedes gewählte Paket geprüft, ob es bereits vorhanden ist (reinstall) oder erst noch installiert werden muss. Wenn während der (Re-)Installation Rückfragen vom APT gestellt werden, so wird der Benutzer aufgefordert diese zu beantworten. Die Antworten werden gespeichert und im Anschluss an die gesamte (Re-)Installation in die Datenbank geschrieben. Sie dienen als neues Antwortschema. Sollten noch nicht alle Pakete (re-)installiert sein, startet die Schleife mit dem nächsten Paket erneut.
Wenn der Benutzer auf ein in der Datenbank gespeichertes Antwortschema zurückgreift, kann er im Folgenden die Liste der ConfigClients einsehen und diejenigen wählen, die ein (re-)install der gewählten Pakete erhalten sollen. Nach der Wahl wird erneut jedes Paket mittels einer Schleife durchlaufen, diesmal nur mit der JobFlowEngine auf einem asynchronen Weg. D.h. es werden Jobs erstellt, die das (re-)install ausführen. Die APT-Rückfragen werden direkt mit Hilfe des Antwortschemas beantwortet. Sobald alle Pakete (re-)installiert wurden, endet die Schleife und die JobFlowEngine aktualisiert den Status der generierten Jobs.

6.1.2 Implementierung

Um die beschriebenen Anwendungsfälle zu realisieren waren zunächst noch fehlende SIBs für ABC zu entwickeln. Anschließend konnte dann der Dienst in Form eines Service-Logic-Graphes implementiert werden.
Da eine ausführliche Erklärung aller Komponenten des Graphen den Rahmen dieses Endberichtes überschreiten würde, wird hier nur auf einen der aufwändigsten Anwendungsfälle, die Installation von Paketen, eingegangen.
Abbildung 6.7 zeigt den für diesen Anwendungsfall relevanten Teil des Graphen des

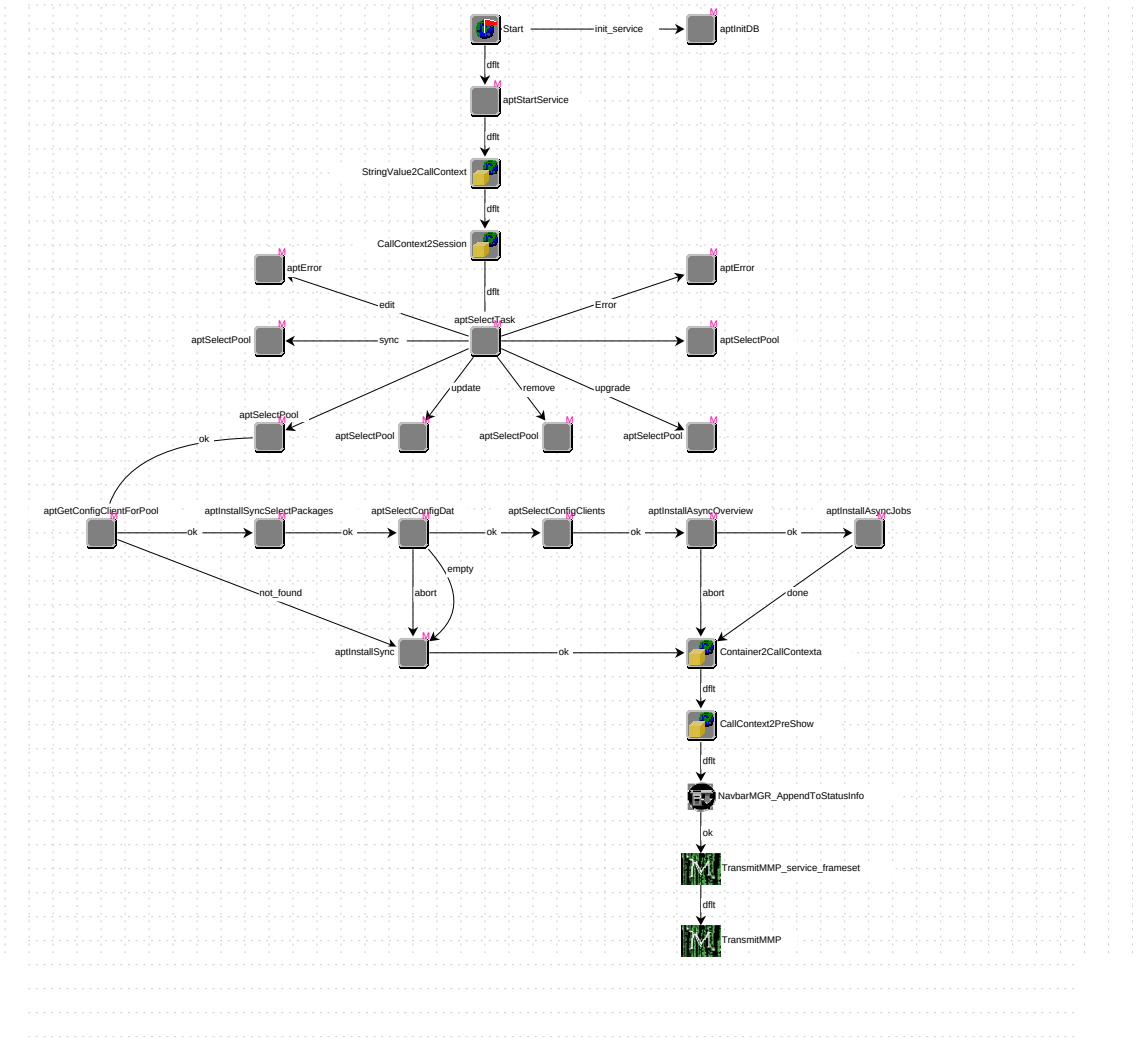

Abbildung 6.7: APT - SLG des Basisdienstes

Basisdienstes.
Wie hier zu sehen ist, wird zunächst das `aptInitDB-Makro` über den Branch init_service angesprungen. Dieses Makro initialisiert die Datenbankschnittstelle, so dass diese von

unserem Dienst genutzt werden kann.
Der default-Branch des Start-SIBs mündet in dem Makro `aptStartService` welches einige initiale Aufgaben übernimmt die bei jedem Aufruf des Dienstes erforderlich sind.
Das nächste interessante Makro ist `selectTask`. Dieses Makro zeigt eine Liste verfügbarer Teil-Services über den WebAgent an. Der vom Benutzer gewählte Teil-Service bestimmt dann den weiteren Ablauf.
Hat der Benutzer nun den Service install ausgewählt, um Pakete auf ConfigClients zu installieren, folgt der Workflow dem install-Branch und trifft hier auf das Makro `selectPool`. Dieses Makro fordert den Benutzer auf, einen Pool auszuwählen.
Anschließend wird `aptGetConfigClientForPool` angesprungen. In diesem Makro wird überprüft, ob für den gewählten Pool ein Referenz-ConfigClient in der Datenbank eingetragen ist. Wird der Dienst zum ersten Mal ausgeführt, ist kein solcher Eintrag in der Datenbank vorhanden und das Makro wird über den Branch not_found verlassen.
Das nun verwendete Makro `aptInstallSync` ist in Abbildung 6.8 zu sehen. Es beginnt

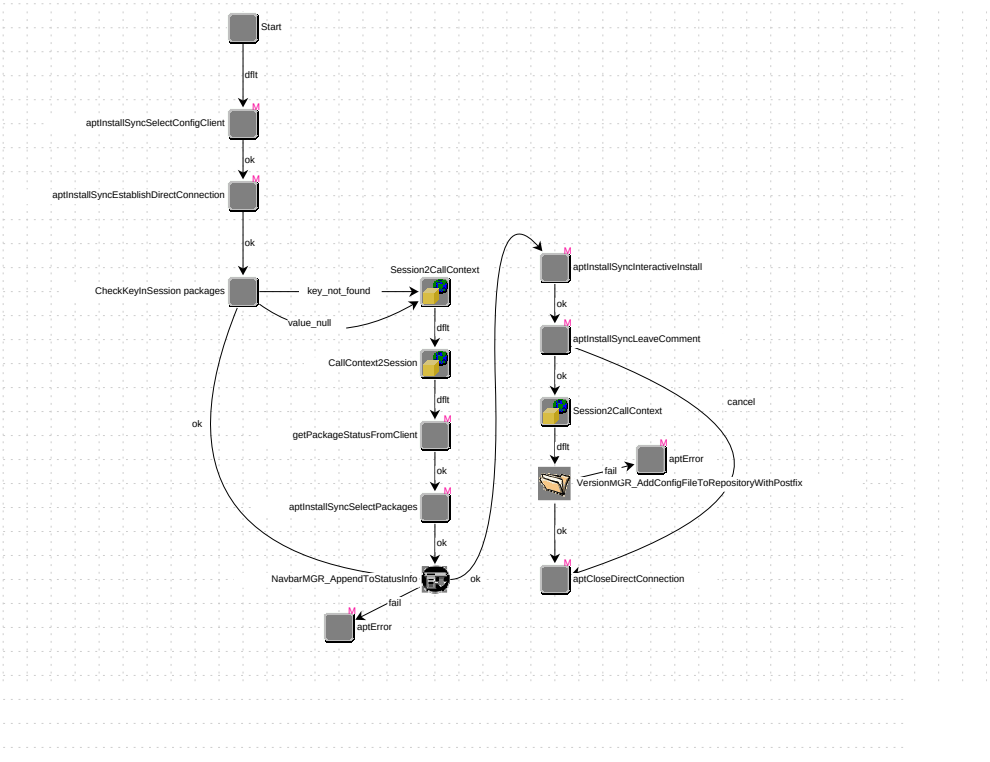

Abbildung 6.8: APT - SLG des Makros aptInstallSync

mit dem Makro `aptInstallSyncSelectConfigClient` welches von dem Benutzer einen ConfigClient des Pools auswählen lässt, auf welchem die Pakete über eine synchrone Verbindung installiert werden sollen.
Über das Makro `aptInstallSyncEstablishDirectConnection` wird eine direkte Verbindung zu dem gewählten ConfigClient aufgebaut. Anschließend wird überprüft, ob

bereits eine Paketauswahl für die Installation getroffen wurde. Dies ist nicht der Fall und das Makro `getPackageStatusFromClient` wird ausgeführt. Dieses Makro lässt auf dem ConfigClient eine Datei erstellen, welche alle verfügbaren und installierten Pakete des ConfigClients enthält. Der Inhalt dieser Datei wird dann an unseren Dienst übertragen, welcher die darin enthaltenen Paketinformationen aufbereitet und in der Datenbank hinterlegt. Außerdem wird der soeben benutze ConfigClient als Referenz-ConfigClient für alle Pools, in welchen dieser sich befindet, in die Datenbank eingetragen. Der Referenz-ConfigClient ist also der ConfigClient, dessen Paketinformationen als letztes in die Datenbank eingetragen wurden und somit am aktuellsten sind.
Nun wird durch das Makro `aptInstallSyncSelectPackages` eine Liste der Pakete erstellt, welche auf dem ConfigClient verfügbar, aber nicht installiert sind. Aus dieser Liste kann der Benutzer nun eine beliebige Anzahl auswählen.
Der nächste Schritt besteht in der interaktiven Installation der gewählten Pakete auf dem gewählten ConfigClient in dem Makro `aptInstalLSyncInteractiveInstall`. Hierbei werden dem Benutzer die Ausgaben des ausgeführten apt-Kommandos angezeigt und der Benutzer kann auf Rückfragen reagieren.
Ist die interaktive Installation abgeschlossen, bekommt der Benutzer die Möglichkeit die soeben durchgeführte Installation zu kommentieren. Dies geschieht in dem Makro `aptInstallSyncLeaveComment`. Wird der Vorgang an dieser Stelle nicht auf Wunsch des Benutzers abgebrochen, wird die bei der Installation erstellte Konfiguration der Pakete in das CVS eingecheckt und die zugehörigen Kommentare in der Datenbank gespeichert.
Anschließend wird die direkte Verbindung wieder getrennt und der Dienst wird somit erfolgreich beendet. Wählt der Benutzer den Abbruch, wird weder die erstellte Konfiguration noch werden die eingegebenen Kommentare gesichert.
Der hier beschriebene Ablauf wird auch als synchroner Installationsmodus bezeichnet da hier eine direkte Verbindung zum ConfigClient zur Laufzeit des Dienstes erforderlich ist und der Benutzer auf Rückfragen mit Eingaben reagieren muss.
Der synchrone Modus bietet der herkömmlichen Installation per Hand gegenüber nur einige wenige Vorteile. Die wirkliche Stärke der MaTRICS und unseres Dienstes liegt in dem asynchronen Modus welcher im Folgenden beschrieben wird.
Zurück in den Basisdienst von Abbildung 6.7. Der Service wird neu gestartet, als Teilservice wird die Installation auf demselben Pool wie vorher ausgewählt. Nun existiert in der Datenbank ein Eintrag für einen Referenz ConfigClient für diesen Pool.
Die Liste der verfügbaren aber nicht installierten Pakete kann nun durch die vorher erstellten Datenbankeinträge erzeugt werden.
Nachdem die Pakete ausgewählt wurden, wird in dem Makro `aptSelectConfigDat` anhand der Datenbank überprüft, ob für diese Paketauswahl bereits eine Installation durchgeführt wurde. Wird eine solche Installation nicht gefunden, so müssen wir wieder in den synchronen Modus wechseln. Andernfalls wird eine Liste aller vorangegangenen Installationen, welche die gewählten Pakete umfassten, zusammen mit den dabei eingegebenen Kommentaren angezeigt. Der Benutzer kann sich nun dazu entschließen die Konfiguration, die bei einer dieser Installationen erstellt wurde, zu verwenden oder den synchronen Modus zu benutzen.

Entscheidet sich der Benutzer dafür, eine der vorhandenen Konfigurationen zu verwenden, wird das Makro `selectConfigClients` angesprungen. Hier hat der Benutzer nun die Möglichkeit mehrere ConfigClients des Pools auszuwählen auf welchen die Pakete installiert werden sollen.
Nachdem der Benutzer noch einmal alle gewählten Parameter in dem Makro `aptInstallAsyncOverwiew` bestätigt hat, wird das Makro `aptInstalLAsyncJobs` aufgerufen. In diesem Makro werden nun mit Hilfe der ScriptingFactory Jobs für die Installation erzeugt und an die JobFlowEngine übergeben.
Für den Benutzer ist der Dienst an dieser Stelle beendet. Von nun an liegt es in der Verantwortung der JobFlowEngine zu gegebener Zeit Kontakt zu den gewählten ConfigClients aufzunehmen und auf diesen die gewählten Pakete mit der gewählten Konfiguration zu installieren.
Die Stärke liegt ganz klar darin, dass der Benutzer neue Pakete auf ganzen Pools installieren kann ohne jedesmal den langwierigen interaktiven Installationsprozess durchführen zu müssen. Außerdem müssen die ConfigClients zu der Zeit, zu welcher die Jobs erzeugt werden, nicht einmal eingeschaltet sein. Die JobFlow-Engine erkennt einen nicht erreichbaren ConfigClient und versucht es später erneut.

Neben der Installation von Paketen unterstützt unser Dienst das Entfernen von ausgewählten Paketen ebenso wie deren Aktualisierung auf eine neuere Version. Außerdem können alle Pakete des Systems gleichzeitig auf den neusten Stand gebracht werden.
Grundlage des apt-Systems sind die Repositories, aus denen die Pakete bezogen werden. Die Liste dieser Repositories kann mit Hilfe unseres Dienstes bearbeitet werden. Die Liste der verfügbaren Pakete kann über den Anwendungsfall update auf den neusten Stand gebracht werden.
Wurde der Dienst längere Zeit nicht genutzt, bietet es sich außerdem an, die Status der Pakete in der Datenbank mit den Stati auf einem Referenz ConfigClient zu synchronisieren. Auch diese Aufgabe wird von unserem Dienst unterstützt.

6.1.3 Zusammenfassung

Als Beispiel für einen Workflow wird das Installieren von Paketen auf einem oder mehreren ConfigClients via APT betrachtet.
Nachdem man den Dienst über das Auswahlmenü gestartet hat, wird man dazu aufgefordert, den Pool auszuwählen, auf dem man Pakete installieren möchte. Hierbei sind bereits diejenigen ConfigClients, die den gewünschten Dienst unterstützen grün hinterlegt. Die rot hinterlegten ConfigClients unterstützen den ausgewählten Dienst nicht.
Als nächstes erfolgt die Auswahl der zu installierenden Pakete. Diese werden hier incl. Distribution (stable/testing/unstable) und Versionsnummer aufgelistet. Die erforderlichen Daten werden dazu aus einer Datenbank ausgelesen und entsprechend aufbereitet (bereits installierte Pakete werden nicht aufgelistet).

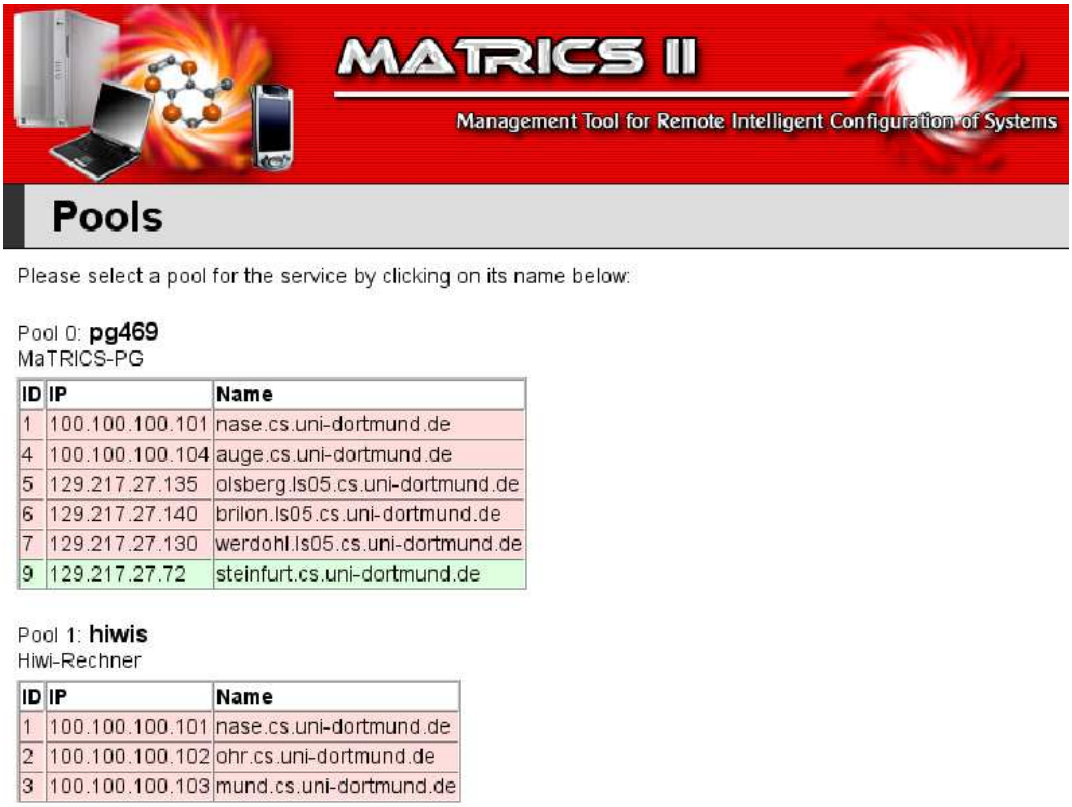

Abbildung 6.9: APT – Workflow: Pool auswählen

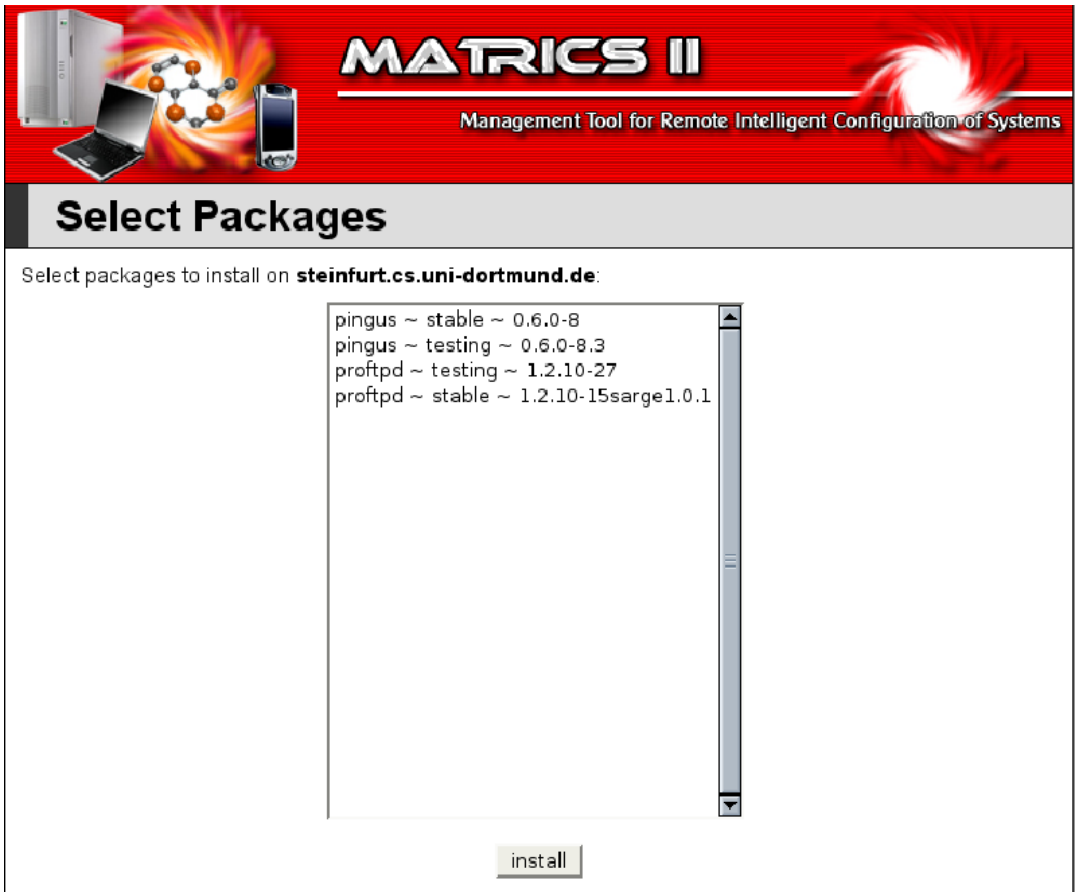

Abbildung 6.10: APT – Workflow: Pakete wählen

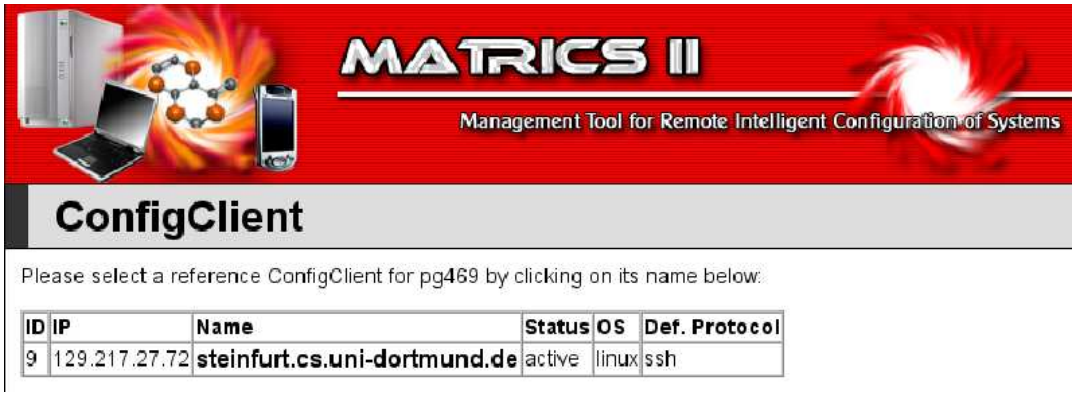

Abbildung 6.11: APT – Workflow: ConfigClient für synchrone Installation wählen via SSH

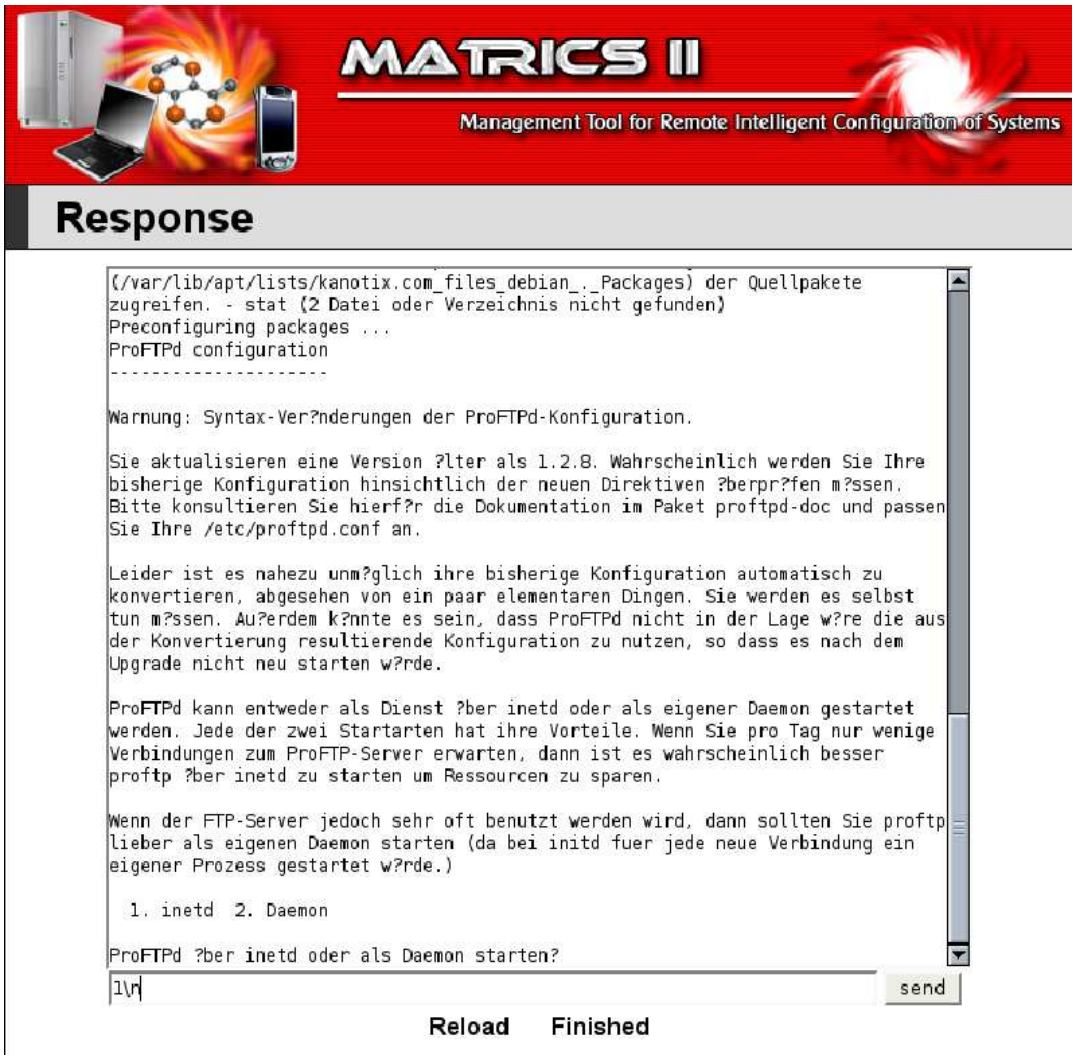

Abbildung 6.12: APT – Workflow: Antwortschema

Sollte nun für die gewählten Pakete bereits eine Konfiguration vorliegen, so kann die Installation synchron über die JobFlow-Engine ablaufen. Dieser Fall wird aber erst später betrachtet. Liegen also keine entsprechenden Informationen für die gewählten Pakete vor, so muss der ConfigClient ausgewählt werden, auf dem man direkt (via SSH) die gewünschten Pakete installieren möchte. Es wird nun eine direkte Verbindung zu dem gewünschten ConfigClient aufgebaut und das entsprechende Kommando (im Beispiel: apt-get install proftpd) direkt abgeschickt. Den Ablauf kann man direkt in dem angezeigten Fenster verfolgen. Mit Hilfe des Edit-Bereiches unten kann man direkt in den Ablauf eingreifen, um z.B. das gewünschte Paket zu konfigurieren. Dies geschieht durch Eingabe der jeweiligen Option (im Beispiel: 1) und eines anschließenden \n, um die Zeile zu beenden. Anschlie-

ßend kann das Kommando abgeschickt werden. Sollte nun das Installieren länger dauern, so kann man mit Hilfe des Reload Buttons die Anzeige aktualisieren und mit Hilfe des Finish Buttons läßt sich der Vorgang jederzeit abbrechen. Ist das gewünschte Paket installiert, so erhält man zum Abschluss noch mal einen kompletten Überblick über den gesamten Ablauf. Hier sind aus Sicherheitsgründen aber keine Eingaben mehr erlaubt. Zum Abschluss der synchronen Installation besteht nun noch die Möglichkeit, jedes der

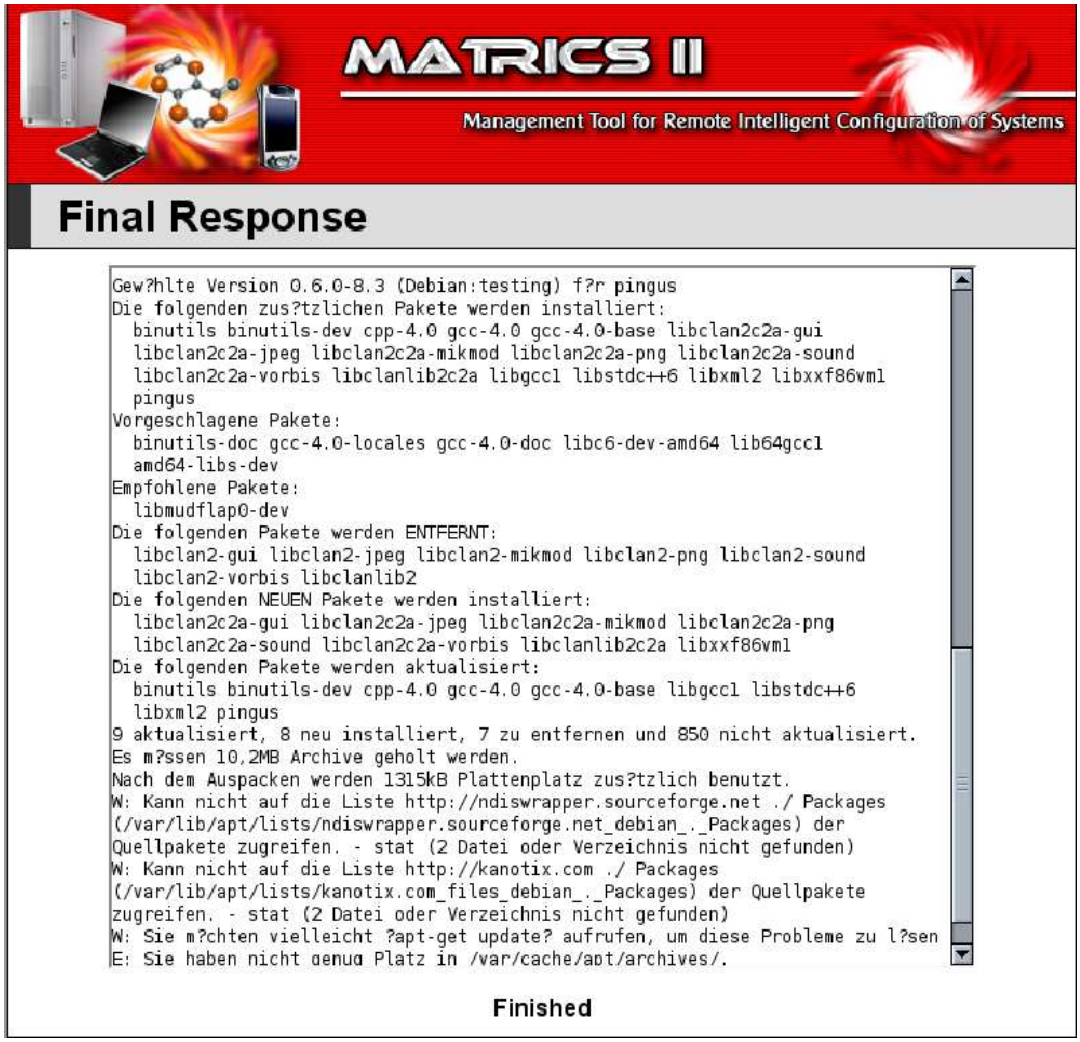

Abbildung 6.13: APT – Workflow: finales Antwortschema

Pakete zu kommentieren, sowie einen Gesamtkommentar zu hinterlassen. Diese werden zusammen mit der gewählten Konfiguration in der Datenbank abgelegt und später beim asynchronen Weg wieder angezeigt, um ggf. die benutzten Optionen wiedergeben zu können. Sollte nun bereits eine Konfiguration für die gewählten Pakete vorliegen, so wird diese auf der folgenden Seite angezeigt. Neben den Namen der zu installierenden Pakete werden hier auch die jeweiligen Kommentare zu jedem Paket und der Gesamtkommentar

angezeigt. An dieser Stelle kann man nun eine der vorgeschlagenen Konfigurationen auswählen, oder sich für den bereits oben beschriebenen synchronen Weg entscheiden. Auf der folgenden Seite hat man nun noch die Möglichkeit aus dem gewählten Pool einzelne ConfigClients auszuwählen, auf denen die gewünschten Pakete installiert werden sollen.

Abschließend erhält man eine Übersicht über die soeben gewählten Schritte. Hier erhält

Abbildung 6.14: APT – Workflow: installierte Pakete kommentieren

man noch einmal einen Überlick auf welchem ConfigClient welche Pakete mit welchen Konfigurationen installiert werden. Man hat nun die Möglichkeit, die gewählte Transaktion abzubrechen, oder die gewählten Pakete zu installieren. Wählt man nun send, so wird die angezeigte Konfiguration an die JobFlow-Engine übertragen, die anschließend die Pakete auf den jeweiligen ConfigClients installiert.

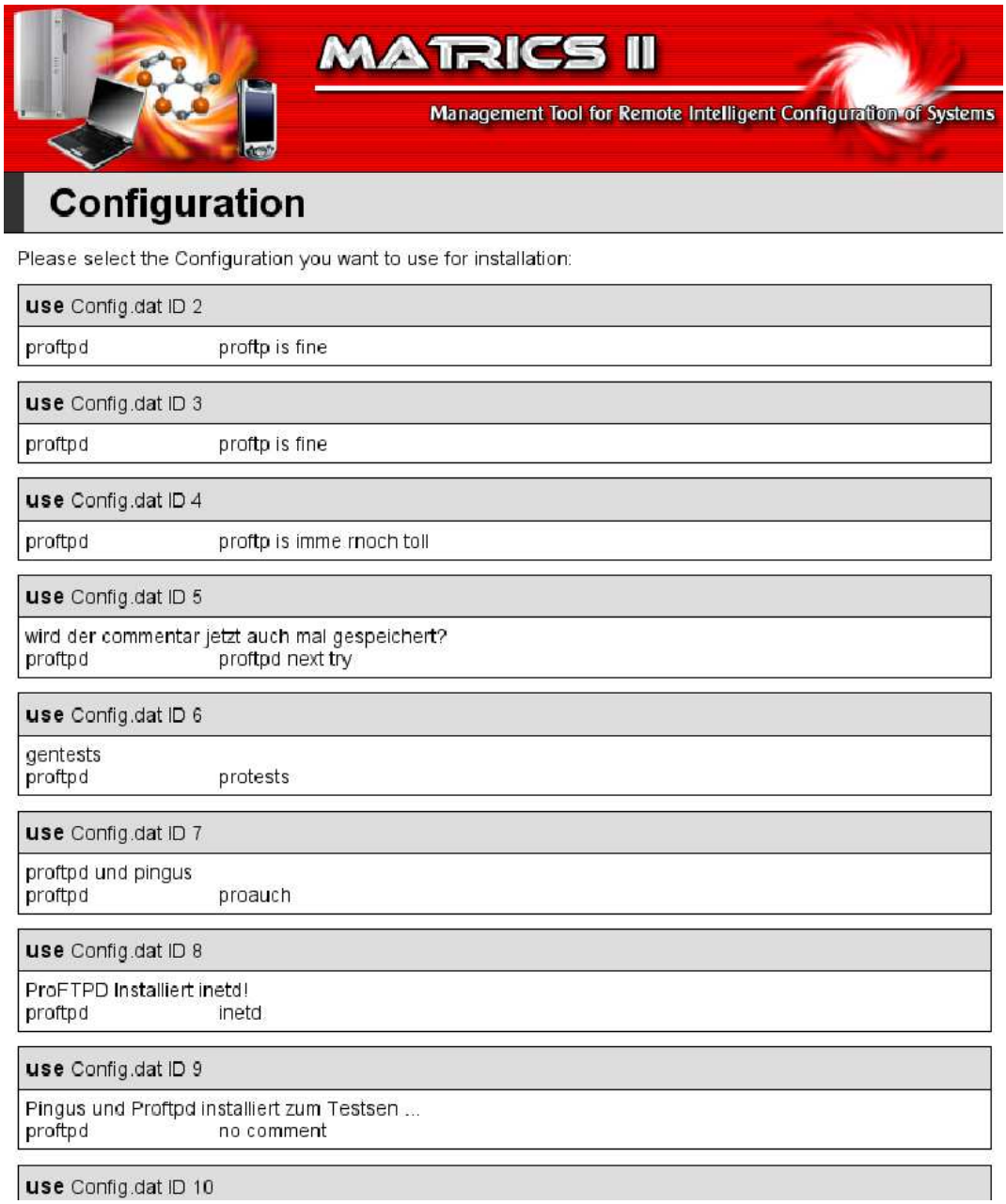

Abbildung 6.15: APT–Workflow: verfügbare Konfiguration wählen

Abbildung 6.16: APT–Workflow: Übersicht

6.2 Unix Usermanagement

Parallel zum AptService-Konfigurationsdienst wurde innerhalb der MaTRICS ein weiterer Dienst entwickelt. Mit Hilfe dieses *UnixUserManagement*-Dienstes (kurz: UUM-Dienst) läßt sich eine Benutzer- und Gruppenverwaltung auf Unixrechnern durchführen, welche sowohl auf einzelnen Client-Rechnern wie auch auf ganzen Rechnerpools durchgeführt werden kann. Der Dienst bietet neben dem Anlegen und Löschen von Benutzern eine Rechteverwaltung über (Netz-)Gruppen und läßt eine Änderung von Benutzerdaten (Passwortänderung, Verzeichnisse...) über die webbasierte Benutzerschnittstelle zu.

Der UUM-Dienst baut auf den Core-Komponenten der MaTRICS auf. Die relevanten Konfigurationsdateien (`passwd`, `group`, `netgroup` und `shadow`) werden über den VersionManager von den Clients geholt (und geändert auch wieder zurückkopiert) und in einem CVS-System gehalten. Intern wird nun auf den im CVS gespeicherten Versionen der Dateien gearbeitet, wodurch Änderungen auch wieder rückgängig gemacht werden können. Die Manipulation der Konfigrationsdateien geschieht im XML-Format. Hierzu wird die XML-SIB-Bibliothek aus den Kurzthemen benutzt. Um allerdings auf Konfigurationen im XML-Format zu arbeiten, müssen die Dateien auch als XML vorliegen. Hierfür ist ebenfalls der VersionManager zuständig, der seit den Zwischenthemen eine entsprechende Funktionalität bereitstellt. Der Kontakt zu den Clients und Rechnerpools wird über den HostManager hergestellt. Eine genauere Beschreibung der einzelnen Komponenten des UUM-Dienstes wird in den folgenden Abschnitten gegeben.

6.2.1 Modellierung

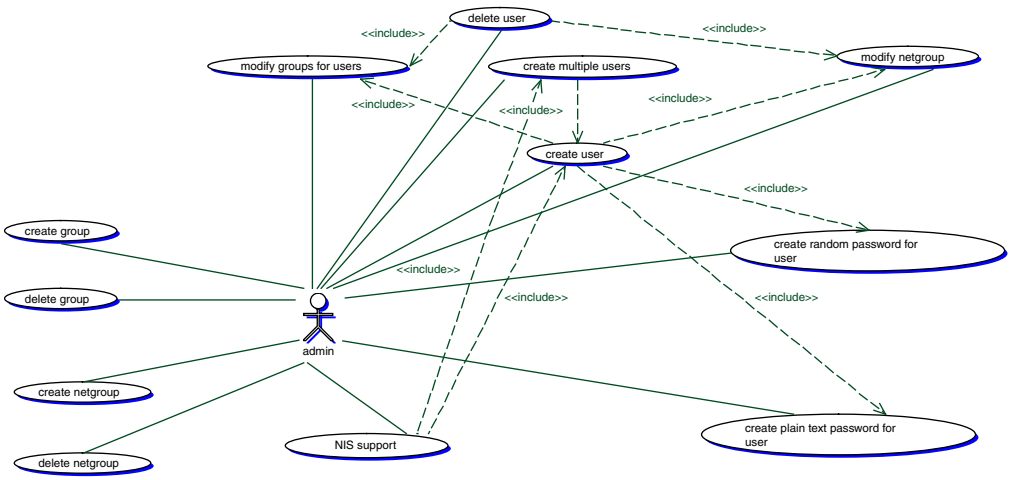

Abbildung 6.17: UUM-Service - Anwendungsfalldiagramm

Die Modellierung unserer Aufgabe erfolgte in UML. Es wurden Anwendungsfalldiagramme und Aktivitätsdiagramme erstellt, die die Basis für unsere Implementierung dienten.

Anwendungsfalldiagramm Anhand des Anwendungsfalldiagramms aus Abb.6.17 lassen sich die wichtigsten Anwendungsfälle beschreiben. Der Administrator kann einen neuen Benutzer anlegen oder einen vorhandenen Benutzer löschen, sowie die Rechte der Benutzer über Gruppen und Netzgruppen festlegen. Ebenfalls kann er sämtliche vorhandenen Einstellungen nachträglich modifizieren. Im Folgenden wird beispielhaft der Anwendungsfall ”Create User” näher beschrieben.

Beispiel: Anlegen eines neuen Benutzeres Bei Anlegen eines oder mehrerer neuen Benutzer werden der Benutzername, sowie benutzerspezifische Daten (Homeverzeichnis, Shell...) angegeben. Werden mehrere Benutzer erstellt, können diese über einen Platzhalter automatisch durchnummeriert werden. Zu beachten ist hierbei, dass der Benutzername automatisch auf die unix-typischen 8 Zeichen reduziert wird. Im selben Schritt kann der Administrator auch festlegen, in welchen (Netz-)Gruppen die Benutzer vorhanden sein sollen. Im nächsten Schritt werden für alle zu erstellenden Benutzer (auf Wunsch automatisch) Passwörter festgelegt und eine Auflistung aller vorzunehmenden Änderungen gegeben. Diese erfordert abschliessend noch eine Bestätigung, bevor die Änderungen übernommen werden, um sie später an die Jobflow-Engine zu übergeben.

Aktivitätsdiagramm: Netzgruppenmanagement Im unten stehenden Aktivitätsdiagramm (Abb. 6.18) wird der genauere Ablauf des Anwendungsfalls ”Netzgruppenmanagement” beschrieben. Nachdem die Konfigurationsdatei vom VersionManager vom ConfigClient geholt und automatisch nach XML konvertiert wurde, kann der Administrator wählen, ob er Netzgruppen erstellen oder Löschen möchte. Zum Erstellen muss der Name angegeben werden, optional können auch sofort Einträge der Netzgruppe hinzugefügt werden. Sämtliche Änderungen erfolgen in der XML-Repräsentation der eigentlichen Konfigurationsdatei und werden bei Abschluss der Konfiguration zurückkonvertiert und zum ConfigClient transportiert.

6.2.2 Implementierung

In diesem Abschnitt beschreiben wir den Workflow des Unix-User-Managements. Auf Basis des im vorherigen Kapitel vorgestellten Anwendungsfalldiagramms sowie den zugehörigen Aktivitätsdiagrammen entwickelten wir mit Hilfe von Service-Logic-Graphen Teildienste für verschiedene Aktionen sowie einen Rahmendienst, über den diese Teildienste angesprochen und die MaTRICS eingebunden werden können.
Da wir mit unserem Service ausschießlich Unix-Systeme unterstützen, besteht die User-, Gruppen- und Netzgruppen-Konfiguration eines Systems aus Einträgen in den vier Konfigurationdateien Passwd, Shadow, Group und Netgroup. Erstere enthält Einträge der User auf dem System, wobei für jeden User eine eindeutige ID, ein Login-Name, eine Hauptgruppe sowie einen Hinweis für die existenz eines Passwortes. Exisitiert ein Passwort für einen User, gibt es einen entsprechenden Eintrag in der Shadow-Datei, wobei das Passwort mittels MD5 verschlüsselt ist. Die Group- und Netgroup-Datei beinhalten jeweils Einträge der auf dem System existierenden Gruppen und Netzgruppen, wobei zu jedem Eintrag auch die Gruppenmitglieder bzw. die enthaltenen User aufgeführt sind.

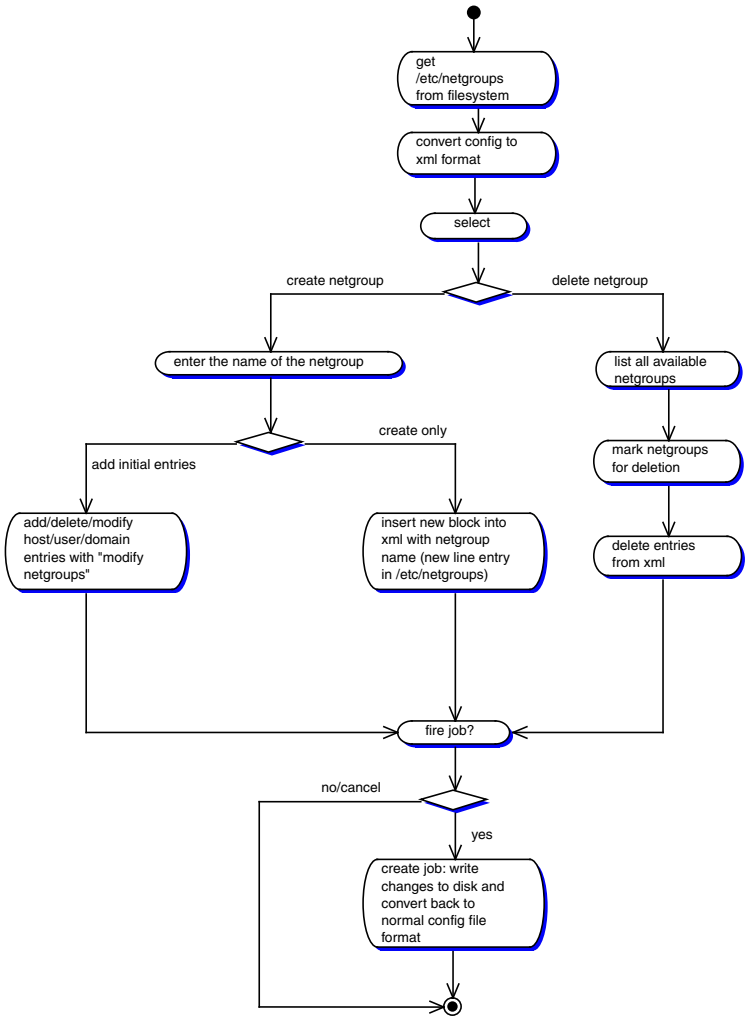

Abbildung 6.18: UUM-Service - Netzgruppenmanagement

Um jetzt die Konfiguration eines solchen Systems zu speichern, zu modifizieren oder wiederherzustellen, bedarf es lediglich der Anpassung dieser vier Konfigurationsdateien. Da die MaTRICS in jüngster Zeit um die Hin- und Rückkonvertierung von Konfigurationsdateien in XML erweitert wurde, und in Verbindung mit dem Version-Manager Konfigurationsdateien automatisch in XML konvertiert und in die Versionsverwaltung CVS einspeist, konnten wir uns bei der Entwicklung unserer (Teil)-Dienste auf das Modifizieren von Konfigurationsdateien beschränken, die als XML-Datei vorliegen. Und auch hierfür standen uns bereits erste SIBs zur Verfügung (siehe Kurzthemen, XML-Modification). Aufgabe der einzelnen Teildienste soll es nun sein, solche XML-Dateien entsprechend zu verändern und diese zurück an den Rahmendienst zu liefern, so dass Konfigurationsänderungen aufeinander aufbauen können.
Wir beginnen nun zunächst mit der Beschreibung der Service-Logik-Graphen der einzenen Teildienste, beginnend mit dem Anwendungsfall Create-User. Anschließend stellen wir dann den Rahmendienst vor, über den diese Dienste angesprochen werden können. Schließlich folgt eine Vorstellung der erzielten Ergebnisse, mit denen wir dieses Kapitel abschließen.

User erstellen

Wie sich bereits aus der Namensgebung erkennen läßt, wurden die Use-Cases der Designphase nicht direkt in Services umgesetzt. In diesem Fall wurden die Use-Cases *Create-User*, *Create-Password-for-User* sowie *Modify-Groups-for-User* gemeinsam dafür genutzt, einen Teildienst zu erstellen, der es ermöglicht, einen oder mehrere User auf einer oder mehreren Maschinen zu erstellen. Gemäß den vorangegangenen Erklärungen, werden zu Beginn eines solchen Dienst-Graphen die benötigten Konfigurationsdateien eingelesen. In diesem Fall liefert uns der Rahmendienst, über den die zu konfigurierenden ConfigClients ausgewählt wurden, die entsprechenden Konfigurationsdateien in Form einer HashTable. Wie dies genau funktioniert, sehen wir später in der Beschreibung des Rahmendienstes. Wenn wir beispielsweise User auf 10 verschiedenen ConfigClients erstellen wollen, erhalten wir insgesamt 40 Konfigurationsdateien, vier Stück je ConfigClient.
Der Create-User Dienst beginnt mit einer Eingabemaske für den/die neuen User, wobei angegeben werden muss, wieviele User erstellt werden sollen, wie deren Home-Verzeichnis lautet, wo sich die Shell des Users befindet und wie deren Login lauten soll (Abb. 6.19). Die Eingabe des Kürzels #N stellt einen Platzhalter dar, der beim Erstellen mehrerer User durch eine Nummerierung ersetzt wird. Desweiteren ist die Auswahl einer Hauptgruppe sowie eine Auswahl von zusätzlichen Gruppen und Netzgruppen, in denen der/die User Mitglied sein soll(en) erforderlich.

Da hierfür bekannt sein muss, welche Gruppen und Netzgruppen den neuen Usern überhaupt zur Verfügung stehen, müssen diese zunächst aus den Konfigurationsdateien Group und Netgroup extrahiert werden. Da es möglich ist, dass auf den verschiedenen ConfigClients auch verschiedene Gruppen und Netzgruppen existieren, müssen diese je ConfigClient ausgelesen und darüber die Schnittmenge gebildet werden. Dies hat zur Folge, dass nur noch Gruppen und Netzgruppen aufgelistet und ausgewählt werden können, die auf

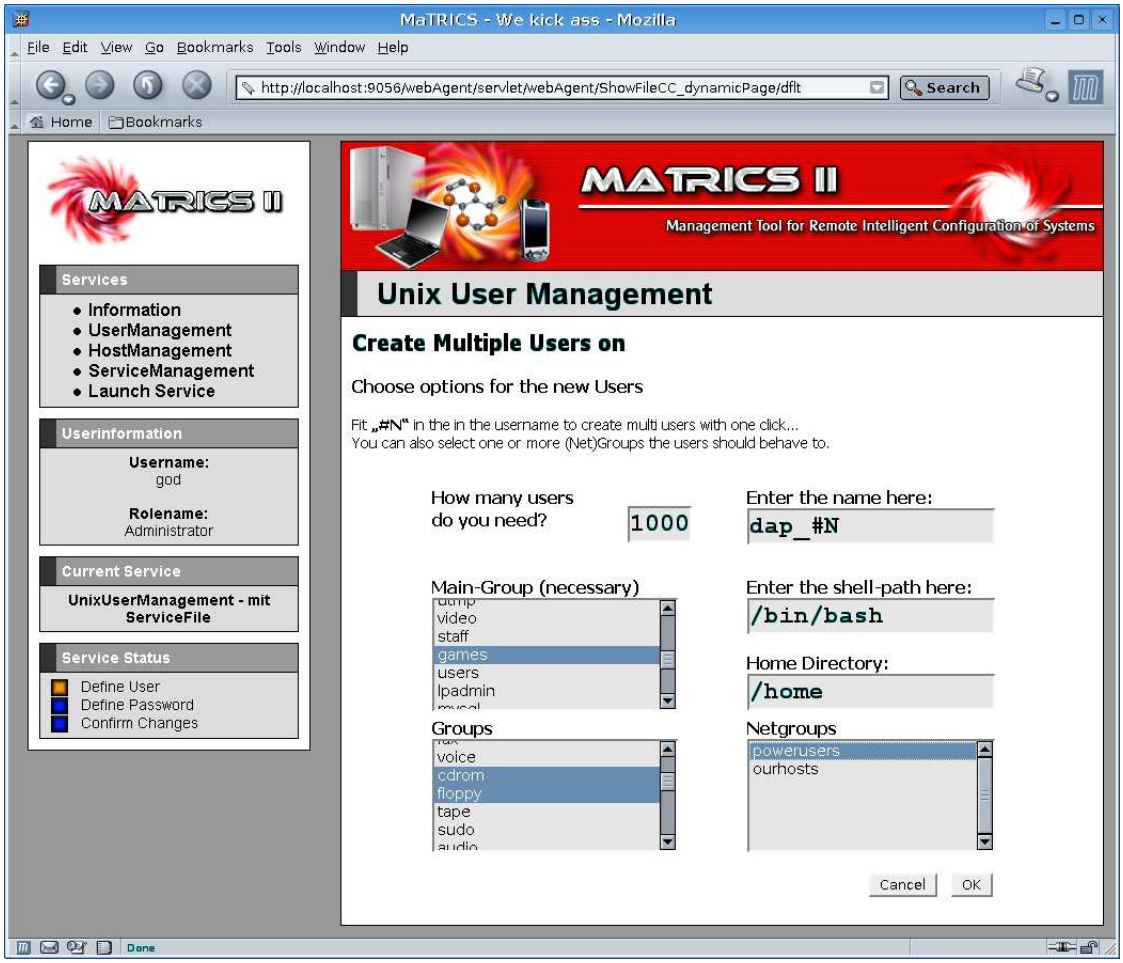

Abbildung 6.19: UUM-Service - User erstellen

allen ConfigClients zur Verfügung stehen. Umgesetzt wird dieses Vorgehen mittels einer Iteration über den Inhalt einer gegebenen HashTable, die sämtliche Konfigurationsdateien aller ausgewählten ConfigClients in XML enthält und vom Rahmendienst generiert und zur Verfügung gestellt wird. In jedem Durchgang wird die entsprechende Konfigurationsdatei aus der HashTable ausgelesen und mittels eines XML-Parser-SIBs zu einem JDOM-Dokument gewandelt. Im folgenden werden dann die Name-Elemente der einzelnen Gruppen ausgelesen, in ein Array konvertiert und mit der bisherigen Schnittmenge geschnitten (diese ist im ersten Durchgang leer). Wurden die Konfigurationsdateien aller ConfigClients abgearbeitet, liegt die Schnittmenge aller Gruppennamen der ConfigClients vor. Entsprechend wird auch bei den Netzgruppen vorgegeangen.
Bestätigt der User seine Eingaben, werden diese zunächst auf Fehleingaben geprüft. Leere Eingabefelder werden ebenso abgefangen wie eine unkorrekte Syntax. Beim Erstellen mehrerer User ist die Angabe eines #N innerhalb des Login-Namens beispielsweise ein Muss. Auch die Auswahl einer Hauptgruppe ist erforderlich. Je nach Fehleingabe wird eine Fehlermeldung festgelegt, die beim erneuten Aufruf der Eingabemaske dynamisch eingebunden wird.
Waren alle Eingaben korrekt, gelangt man zur Passworteingabe. Hier hat man die Auswahl zwischen automatischen und Plaintext-Passwörtern. Ähnlich zur Login-Eingabe müssen Plaintext-Passwörter mittels #N durchnummeriert werden, falls mehr als ein User erstellt wird. Für die Zufallspasswörter hingegen läßt sich im Graphen die Länge der automatisch erzeugten Passwörter festlegen. Die Generierung der Login-Namen, wie auch der Passwörter erfolgt mit Hilfe des SIBs AutoWord. Insbesondere ist dieser SIB für die korrekte Nummerierung verantwortlich, falls mehrere User erstellt werden sollen. Dabei wird auch die Länge der Nummerierung dynamisch angepasst, so dass Zahlen ggf. mit führenden Nullen aufgefüllt werden.
Im Folgenden soll dem User eine Übersicht der neuen Konfigurationsaspekte gegeben werden. Speziell in Bezug auf automatische Passwörter, die randomisiert erfolgen ist es auch dringend erforderlich, die neuen Passwörter zu notieren oder zu drucken. Desweiteren sollen Konflikte jeglicher Art mit der bestehenden Konfiguration rot markiert werden und eine Fortsetzung nur mittels einer Korrektur ermöglichen. Dazu wird wie folgt vorgegangen: Zunächst werden die Userinformationen durch die User-IDs vervollständigt. Dies geschieht mit Hilfe eines AutoID-SIBs, der als Eingabe alle bereits belegten IDs erfordert und mit diesen Informationen n neue IDs erstellt (lückenstopfend). Da wir im allgemeinen auf Konfigurationen verschiedener ConfigClients arbeiten, müssen die bestehenden IDs aller ConfigClients erneut kombiniert werden. Diesmal bilden wir jedoch nicht die Schnitt- sondern die Vereinigungsmenge, der jeweiligen ID-Arrays. In der Praxis bedeutet dies, dass wir eine ID als belegt ansehen, sobald sie bereits auf mindestens einem ConfigClient verwendet wird. Als nächstes werden sogenannte Konflikt-Listen erzeugt, die aus einem Abgleich der bisherigen Eingaben mit den bestehenden Konfigurationen erstellt werden. Diese Konfliktlisten dienen der Webseite zur Darstellung von Konflikten, die eine Korrektur erfordern. Diese Abgleiche werden sowohl für die Login-Namen als auch für die IDs durchgeführt. Hierfür ist es erneut erforderlich, eine Vereinigungsmenge zu bilden, nämlich die der Login-Namen auf den einzelnen ConfigClients. In jedem Fall gelangt der

User zu der angesprochenen Übersicht. Hier wurde besonders darauf geachtet, dass alle Werte nachträglich editierbar sind und verändert werden können. Wurden Konflikte gefunden, wie beispielsweise überlapptende Logins, so werden diese rot unterlegt. Behebt man solche Konflikte und bestätigt diese, werden die Abgleiche erneut durchgeführt. Treten keine Konflikte auf und die Eingaben werden bestätigt, erfolgt ein letzter Abgleich und es folgt die Passwort-Kodierung sowie die Generierung der User. Dieser letzte Abgleich soll verhindern, dass der Anwender eine konfliktfreie Übersicht ändert, dabei einen neuen Konflikt erzeugt und bestätigt. Als editierbare Felder werden die User-Logins, die Passwörter, die IDs, die Hauptgruppe sowie die Speicherorte vom Heimverzeichnis und der Shell angeboten.
Die Passwort-Kodierung erfolgt mittels eines extra SIBs, der beliebig viele Passwörter zugleich kodieren kann. Der Codierungsvorgang selbst erfolgt mittels eines MD5-Algorithmus.
Der Generierungsprozess erfolgt erneut mittels einer Iterationsschleife. In jedem Durchlauf werden die vier Konfigurationsdateien des entsprechenden ConfigClients erneut ausgelesen und gemäß den Eingaben modifiziert. Hierbei ist dringenst zu beachten, der Hauptgruppen-Eintrag in der Passwd aus der entsprechenden Group-ID und nicht deren Name besteht. Da diese ID auf jedem ConfigClient verschieden sein kann, darf der Gruppenname erst innerhalb der Schleife in eine ID umgesetzt werden.
Das eigentliche Modifizieren der Konfigurationsdateien erfolgt mit jeweils einem speziellen SIB, die als Eingabe alle nötigen Informationen zur Modifizierung erfordern. Das geänderte JDOM-XML-Dokument wird von den SIBs jeweils zurückgegeben. Wurden alle vier Modifizerungs-SIBs durchlaufen, werden die Konfigurationsdateien des entsprechenden ConfigClients zurück in die HashTable gelegt. Wurde die Iteration über alle HashTable-Einträge abgeschlossen, wird der Dienst beendet und liefert dem Rahmendienst die geänderte HashTable zurück.

User bearbeiten

User, die bereits angelegt wurden, erfordern häufig eine Umkonfigurierung oder Korrektur. Beispielsweise könnte ein Passwort vergessen worden sein, ohne das der User-Account nicht mehr zugänglich ist. Um solche Korrekturen und Änderungen zu ermöglichen, entschieden wir uns für einen Teilservice, basierend auf dem Use-Case *Create-User* und *Change Groups for User*. Im Service gelangt der Anwender zunächst zu einer Auswahlliste der existierenden User. Hierzu wird die übergebene Passwd-Datei als JDOM-Dokument gewandelt. Anschließend werden die User-Name-Elemente extrahiert und ihre Texte als Array ausgegeben. Wählt der Anwender einen User aus, gelangt er zu einer Eingabemaske mit den bisherigen Werten (abgesehen vom Passwort und den Netzgruppen). Beispielsweise kann ein neues Passwort eingeben werden, wobei ein leeres Textfeld darauf hindeutet, das alte Passwort beizubehalten. Desweiteren läßt sich die ID und die Haupt-Gruppe des Users neu festlegen. Eine Änderung des Home-Verzeichnisses sowie die Lokalisierung der zuverwendenen Shell ist ebenfalls möglich. Eine Änderung des Logins ist absichtlicherweise nicht möglich, da es sich so nun um einen anderen User handeln würde. Hierzu wird darauf verwiesen, den User zu löschen und einen neuen zu erstellen.

Die Auswahlfelder der zur Verfügung stehenden IDs und Gruppen werden - wie auch beim Fall Create-User - durch das Auslesen der entsprechenden Konfigurationsdateien erzielt. Beispielsweise liefert die Passwd-Datei alle bislang verwendeten IDs. Erstellt man darauf basierend mit dem bereits angesprochenen SIB GetAutoID 65535 neue IDs (mehr sind unter Unix-Systemen nicht möglich), so erhält man eine Liste aller der IDs, die noch möglich sind. Vormarkiert wird natürlich die ID, die der User bislang besaß. Genauso verhält es sich auch bei dem Auswahlfenster der Hauptgruppe. Zunächst werden alle existierenden Gruppennamen ausgelesen und danach werden diese in das Auswahlfeld eingefügt. Selektiert wird dann noch die Gruppe, die der User bislang als Hauptgruppe definiert hatte.
Bei der Auswahl der zusätzlichen Gruppenzugehörigkeiten des Users kamen wir mit einem solchen Vorgehen leider nicht mehr aus, da man später bei der Abspeicherung der neuen Gruppenzugehörigkeiten hätte vergleichen müssen, ob es eine Mitgliedschaft in dieser Gruppe auch voher schon gegeben hat oder nicht. Aus diesem Grund wird die Group-Datei des ConfigClients zunächst in JDOM gewandelt und dann durchlaufen. Gruppen, in denen der User Mitglied ist, werden identifiziert und in ein neues JDOM-Dokument gespeichert. Die Einträge des Users werden dabei aus dem Gruppeneintrag gelöscht, so dass der User temporär in keiner Gruppe Mitglied ist. Bitte beachten Sie, dass weder ein ungewollter Programmabsturz noch ein Benutzerabbruch dazu führen können, dass der Benutzer später ohne Gruppe darsteht. Wir befinden uns nicht - wie man annehmen könnte - in einem Hard-State, da die Konfigurationsdateien nur als Kopie innerhalb des Teildienstes geändert werden und erst nach Abschluß des gesamten Teildienstes wieder zurückgeschrieben werden (vgl. Transaktion). Mit Hilfe der gesondert gespeicherten Gruppen-Namen, kann auf der Webseite vorselektiert werden, welche Gruppen bislang für den User definiert waren. Diese Auswahl kann beliebig geändert werden. Bestätigt der User die Eingaben, kann der User in den selektieren Gruppen hinzugefügt werden, ohne dabei etwas beachten zu müssen. Alle Eingaben können ohne Fehlerprüfung übernommen werden.
Bei der Generierung der Konfigurationsdateien wird bei einem nicht-leeren Passwort-Feld der eingegebene Text neu kodiert und gegen das alte Passwort getauscht. Hierzu wird in der Shadow-Datei der entsprechende User-Eintrag gesucht und das Element Passwort neu gesetzt. Ähnlich verhält es sich mit den übrigen Eingaben, da in jedem Fall erst das User-Element mit dem entsprechenden Usernamen gesucht wird und von diesem Element ausgehend die Geschwister-Elemente wie GID (ID der Hauptgruppe), HomeDir, Shell geholt und deren Text-Werte neu gesetzt werden kann. Die selektieren Gruppen erhalten, entsprechend dem Vorgehen beim Create-User, einen neuen User-Eintrag im jeweiligen Abschnitt dieser Gruppen.
Abschließend werden die Konfigurationsdateien wieder in die Session gelegt und können darauf basierend weiter verändert werden.

User löschen

Um das gesamte Angebot zu vervollständigen, wird dem Anwender auch die Möglichkeit angeboten, einen oder mehrere User zu entfernen (Abb. 6.20). Dieses kann je nach Auswahl der Clients im Rahmendienst auf einem oder auf mehreren ConfigClients gleichzeitig

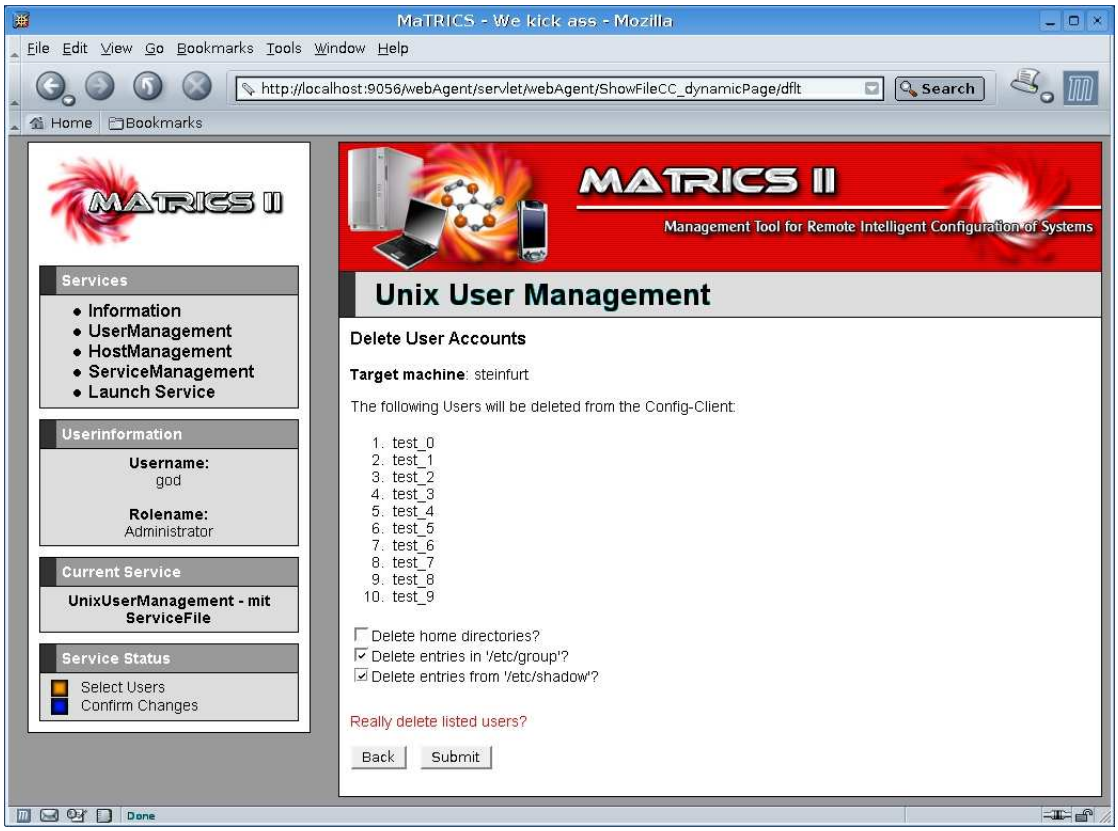

Abbildung 6.20: UUM-Service - Löschen von Usern

durchgeführt werden. Auch dieser Teildienst beginnt mit dem Auslesen der Konfigurationsdateien, die ihm aus der Session zur Verfügung stehen. Mit dem bereits erklährten Merging, werden auch hier innerhalb einer Iterationsschleife die vorhanden Userlisten aller Clients germerged. Folglich werden dem Anwender nur noch diejenigen User zur Auswahl angeboten, die auf allen ConfigClients vorhanden sind. Wahlweise läßt sich der Graph auch mittels des Selection-Parameters im SIB MergeStringArrays auf die Vereinigungsmenge der Namen umstellen, so dass auch User zur Auswahl angeboten werden, die auf mindestens einem ConfigClient definiert sind. Dies kann von Vorteil sein, kann aber auch Probleme bringen, bei dem Versuch einen User auf einem Client zu löschen, auf dem er garnicht exisitiert. Neben der Auswahl des Users, der gelöscht werden soll, kann der Anwender auch entscheiden, ob auch dessen Shadow, Group- und Netgroup-Einträge gelöscht werden sollen. Eine weitere Option legt fest, ob auch das Home-Verzeichnis des Users entfernt werden soll.
Wird die Auswahl bestätigt, werden zunächst die Heimverzeichnisse der User gesichert, um später ggf. entfernt zu werden. Anschließend werden die Einträge der User aus der Passwd-Datei entfernt. Für den Fall, dass auch die Homeverzeichnisse gelöscht werden sollen, werden an dieser Stelle die entsprechenden Befehle mit Hilfe der Script-Factory erzeugt. Hat der User selektiert, dass auch die Shadow-, Group- oder Netgroup-Einträge entfernt werden sollen, werden auch diese mit Hilfe einer Iteration entfernt. Die neu resultierenden Dokumente werden nach jeder Iteration zurück in die HashTable gelegt. Das Entfernen der Einträge aus den XML-Dateien erfolgt jeweils erneut mittels einer Iteration über die die Einträge bisherigen Einträge in den Konfigurationsdateien. Beispielsweise wird über die enthaltenen Gruppen-IDs iteriert, bis diejenige Gruppe gefunden wurde, die die Maingroup des Users darstellt. Anschließend können die gefundenen Elemente dann entfernt werden. Dies geschieht mit Hilfe der bereits genannten XML-SIBs. Konkret wird beispielsweise in der Passwd-Datei zunächst das Name-Element gesucht, dessen Text mit dem gewälten Usernamen übereinstimmt. Von diesem Element ausgehend wird mittels des Vaterelements der gesamte Block des Usereintrages identifiziert und aus der Passwd-Datei entfernt. Auf gleiche Weise erfolgt das Entfernen des Users aus der Shadow-Datei. Der gesamte Löschprozess wird mit einer großen Schleife über alle im Rahmendienst ausgewählten ConfigClients durchgeführt.

Gruppen erstellen

Ähnlich zu den Teildiensten bezogen auf den User, bieten wir dem User auch solche für die Verwaltung von Gruppen an. Jedoch verzichten wir auf den Fall, eine Gruppe zu modifizeren und beschränken uns auf das Erstellen und das Löschen von Gruppen. Das Erstellen einer Gruppe gestaltet sich als recht einfach. In dem zugehörigen Service-Logik-Graphen werden zunächst die Userinformationen aus der Konfigurationsdatei Passwd ausgelesen, und deren Schnittmenge gebildet, um nur diejenigen User zur Auswahl anzubieten, die auf allen ConfigClients existieren. Dies geschieht innerhalb des Graphen, wie bereits mehrfach zuvor beschrieben. Im folgenden wird ein Gruppenname verlangt. Diese Eingabe wird innerhalb des Graphen mit den bisherigen Konfigurationen verglichen. Wird der Name bereits verwendert, erfolgt ein Rücksprung zurück zur Eingabemaske. Ansonsten wird

mit der Vergabe einer Group-ID fortgefahren. Diese wird zunächst mittels des AutoID-SIBs automatisch generiert. Möchte der Anwender der Gruppe eine ID manuel zuweisen, kann er dies an dieser Stelle tun. Die Eingaben werden eingelesen und erneut verglichen. Bei Konflikten wird zur Eingabemaske zurückgesprungen. Andernfalls wird ein letzter Bestätigungsbildschirm ausgegeben. Wird auch hier bestätigt, erfolgt die Generierung der Gruppe.

Hierzu wird zunächst ein neues JDOM-Element erzeugt, das den neuen Gruppenblock darstellt. Im folgenden werden dann gemäß den eingelesenen Werten neue Elemente für diese Gruppe angelegt und unter dieses Block-Element gehangen. Da für jeden User ein einzelner Gruppeneintrag vorgenommen werden muss, wird auch hier eine Iteration verwendet. In jedem Interationsschritt wird einer der selektierten User einem neuen User-Element zugewiesen und unter das Block-Element der Gruppe gehangen. Abschließend wird das endgültige Blockelement der Konfigurationsdatei Group angefügt. Diese kann daraufhin zurück in die Session geschrieben werden und steht somit den folgenden Teildiensten zur Verfügung.

Gruppen löschen

Schließlich und endlich fehlt uns noch der Teildienst Gruppen löschen, welcher hier kurz erläutert werden soll. Mittels der bereits beschriebenen Techniken, die zur Verfügung gestellten XML-Dateien zu parsen und auszulesen wird hier wie folgt vorgegangen: Dem User wird zu Beginn eine Liste der zur Verfügung stehenden Gruppen zur Auswahl angeboten. Diese wird mittels der gelieferten Konfigurationsdateien recht leicht realisiert, besonders im Hinblick auf die Tatsache, dass dieser Teildienst aus technischen Möglichkeiten nur das Löschen von Gruppen auf genau einer ConfigClient-Maschine erlaubt. Wurde der User selektiert und bestätigt, erfolgt eine Prüfung darauf, ob irgendein User auf der Maschine diese Gruppe als seine Hauptgruppe eingetragen hat. Zu diesem Zweck wird die Passwd Konfigurationsdatei durchlaufen und das GID-Element der User wird mit dem der Gruppe verglichen. Treten keine Konflikte auf, wird die Gruppe sofort gelöscht, andernfalls wird zunächst eine hinweisende Seite ausgegeben, über die der Vorgang zu diesem Zeitpunkt noch einmal abgebrochen werden kann.
Der Löschprozess selbst gestaltet sich so, dass zunächst das Name-Element in der Group-Datei ausfindig gemacht wird, dessen Text mit dem Gruppennamen übereinstimmt, also den Eintrag der gewählten Gruppe darstellt. Danach wird mittels des Vater-Elements der zugehörige Block identifiziert, der daraufhin lediglich gelöscht werden muss, um den Vorgang abzuschließen.

Netzgruppen verwalten

Dieser Teildienst beschäftigt sich mit der Konfiguration von Netgroup-Dateien. Diese haben folgende Struktur:
`netgroup (host,user,domain) (host,user,domain) ..`
So würde z.B. die Definition einer Netzgruppe `powerusers` mit den eingetragenen Benutzern `miquels`, `torvalds`, `fubar` als folgender Eintrag in der Netgroup-Datei repräsentiert:

```
powerusers (,miquels,) (,torvalds,) (,fubar,)
```

Nach der Auswahl des Teildienstes werden aktuelle Gruppen aus der Konfigurationsdatei ausgelesen und dem Benutzer präsentiert, der nun zwischen drei Möglichkeiten auswählen kann:

- eine neue Netzgruppe erzeugen
- eine bestehende Netzgruppe modifizieren
- eine bestehende Netzgruppe löschen

Neue Netzgruppe erzeugen
Der Benutzer gibt den Namen der neuen Netzgruppe ein und wechselt durch ein Klick auf eine Bestätigungsschaltfläche zum Modifikationsmodus für Netzgruppen, welcher auf der Seite 116 näher beschrieben wird.

Bestehende Netzgruppe modifizieren
Aus der Liste bereits bestehender Netzgruppen wird eine Netzgruppe ausgewählt. Mit einem Klick auf die entsprechende Schaltfläche wird in den Modifikationsmodus für diese Netzgruppe gewechselt.

Bestehende Netzgruppe löschen
Aus der oben erwähnten Auswahlliste wird nun eine Netzgruppe ausgewählt. Durch ein Klick auf die entsprechende Schaltflöche wird der in der entsprechende Eintrag in der Netgroup-Datei (nach einer Sicherheitsabfrage) gelöscht.

Modifikationsmodus für Netzgruppen

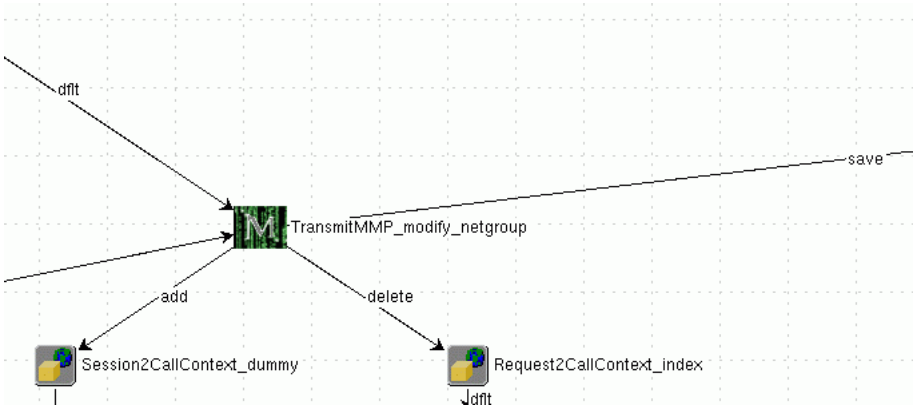

Abbildung 6.21: UUM-Service - Block einer Netzgruppe hinzufügen/löschen

Der Modifikationsmodus ist das Kernstück dieses Teildienstes. Die Auswahlmaske enthält drei Auswahllisten - jeweils eine für Benutzer, Hosts und Domains. Über diese Listen sind

drei Textfelder platziert. Am unteren Rand der Auswahlmaske werden bereits vorhandene Blöcke der Netzgruppe angezeigt. Der Benutzer kann nun neue Blöcke für die Netzgruppe folgendermaßen erfassen:
Relevante Einträge für einen Tripel werden entweder aus den Auswahllisten selektiert oder direkt in die Textfelder eingegeben. Der Klick auf die Schaltfläche zum Hinzufügen eines Blocks fügt den Block zur aktuellen Netzgruppe hinzu. Die Eingaben in den Textfelder haben höhere Priorität als Selektionen aus den Auswahllisten. Die Liste der Blöcke wird aktualisiert.
Alternativ kann ein vorhandener Block durch einen Klick auf `delete` Link aus der Netzgruppenkonfiguration entfernt werden.
Ist die Modifikation der Netzgruppe vollendet, klickt der Benutzer auf die Schaltfläche zum Speichern der Konfiguration (Abb. 6.21).

Auf Graphebene wird bei jedem Klick auf die Schaltfläche zum Hinzufügen eines Blocks eine Schleife durchlaufen. Während des Durchlaufs wird überprüft, welche Eingaben der Benutzer gemacht hat. Leere Eingaben werden ingoriert, Textfeldeingaben haben Vorrang vor den Listenselektionen.
Einzelne Zeichenketten werden nun zu einem Block zusammengefasst und als ein neuer XML-Element dem JDOM-Dokument an der entsprechender Stelle als neuer Trippel-Eintrag zu dieser Netzgruppe hinzugefügt (Abb. 6.22).

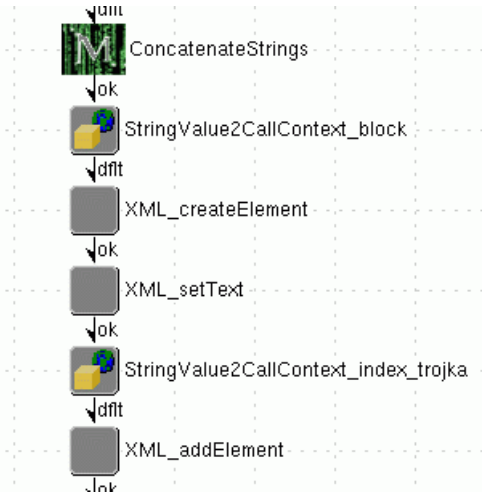

Abbildung 6.22: UUM-Service - Konkatenation einzelner Eingaben zu einem neuen Block und Erzeugung eines neuen XML-Elementes

Konfiguration speichern

Um eine geänderte Konfiguration zu aktivieren und mittels Versionsverwaltung zu speichern, wurde ein eigener Teilservice entwickelt, anstelle eines Speicherns nach jeder Modifizierung. Auf diese Art ist es möglich, mehrere Konfigurationsänderungen hintereinander durchzuführen, ehe eine Speicherung erfolgt. Dieses Vorgehen ist in soweit sinnvoll,

da es sich bei Konfigurationsänderungen für gewöhnlich um Transaktionen handelt, die aus mehreren Teilschritten bestehen.
Zu Beginn dieses UseCases wird der Anwender aufgefordert, einen Vermerk der aktuellen Konfiguration einzugeben. So kann im Falle einer Wiederherstellung die richtige Version leichter gefunden werden. Desweiteren bekommt der User angezeigt, welche Änderungen bei diesem Vorgang in Kraft treten (siehe auch History-Feld im Rahmendienst). Wie zuvor arbeitet auch dieser Dienst auf die, in der Session liegenden HashTable, gefüllt mit Konfigurationsdateien für jeden bearbeiteten ConfigClient. Auch hier wird über diese HashTable iteriert. In jeder Iteration wird zunächst der zugehörige ConfigClient identifiziert. Anschließend werden seine Konfigurationsdateien in das CVS eingechecked. Im dritten und letzten Schritt sorgt ein Datenbankeintrag dafür, dass die Konfiguration später wiederhergestellt werden kann. Zu diesem Zweck werden die jeweiligen CVS-Revisions der vier Konfigurationsdateien zusammen mit ConfigClient-ID, Pool-ID und Zeitstempel in einer Datenbankzeile abgelegt.

Da der Zeitstempel vor der Iteration erstellt wird, ist dieser für alle ConfigClients gleich. Auf diese Weise ist es möglich, Konfigurationen einzelner Clients von denen mehrerer Clients zu unterscheiden. Konfiguriert man beispielsweise einen Pool, der beliebig viele ConfigClients enthält, so enthalten alle Datenbankeinträge dieser Clients die selbe Pool-ID und vor allem denselben Zeitstempel. Daran kann erkannt werden, dass es sich um eine gemeinsame Konfigurierung gehandelt haben muss. Nach dem Speichern wird der Teildienst verlassen und der Rahmendienst startet neu.

Der Rahmendienst

Wie bereits in den vorherigen Kapitel angedeutet, besteht die Aufgabe des Rahmendienstes darin, die Teildienste zu einem gemeinsamen Dienst zusammenzuführen und in Form eines zentralen Dienstes in der MaTRICS anzubieten. Dabei übernimmt der Rahmendienst sämtliche Verwaltungsaufgaben der Teildienste und stellt somit eine Schnittstelle zwischen dem Config-Manger der MaTRICS und den Teildiensten dar. Zu diesen Aufgaben gehört beispielsweise die Initialisierung einer Dienstspezifischen Datenbank, welche zu Beginn des Graphen initialisiert wird.

Eine weitere Anforderung der Teildienste, die wir bereits kennengelernt haben, ist die Vorauswahl der ConfigClients bzw. des Pools, die konfiguriert werden sollen. Zu diesem Zweck bietet der Rahmendienst dem User zwei Auswahlfenster an, in dem die vorhandenen Pools und ConfigClients angezeigt werden. Bitte beachten Sie, daß ein ConfigClient nur dann in der Liste verfügbar ist, wenn er auch für den Unix-User-Management-Dienst eingerichtet ist.

Pools werden entsprechend nur dann zur Auswahl angeboten, wenn dies für alle enthaltenen ConfigClients gilt. Bei der Auswahl kann der Anwender grundsätzlich zwischen zwei Konfigurationstypen entscheiden. Zum einen können einzelne ConfigClients konfiguriert werden, zum anderen ganze Pools, die in der Regel mehrere ConfigClients enthalten. Wählt der Benutzer einen ConfigClient aus und bestätigt mit dem Button *Configure*, so gelangt er in das Konfigurationsmenü eines einzelnen ConfigClients. Es stehen dem

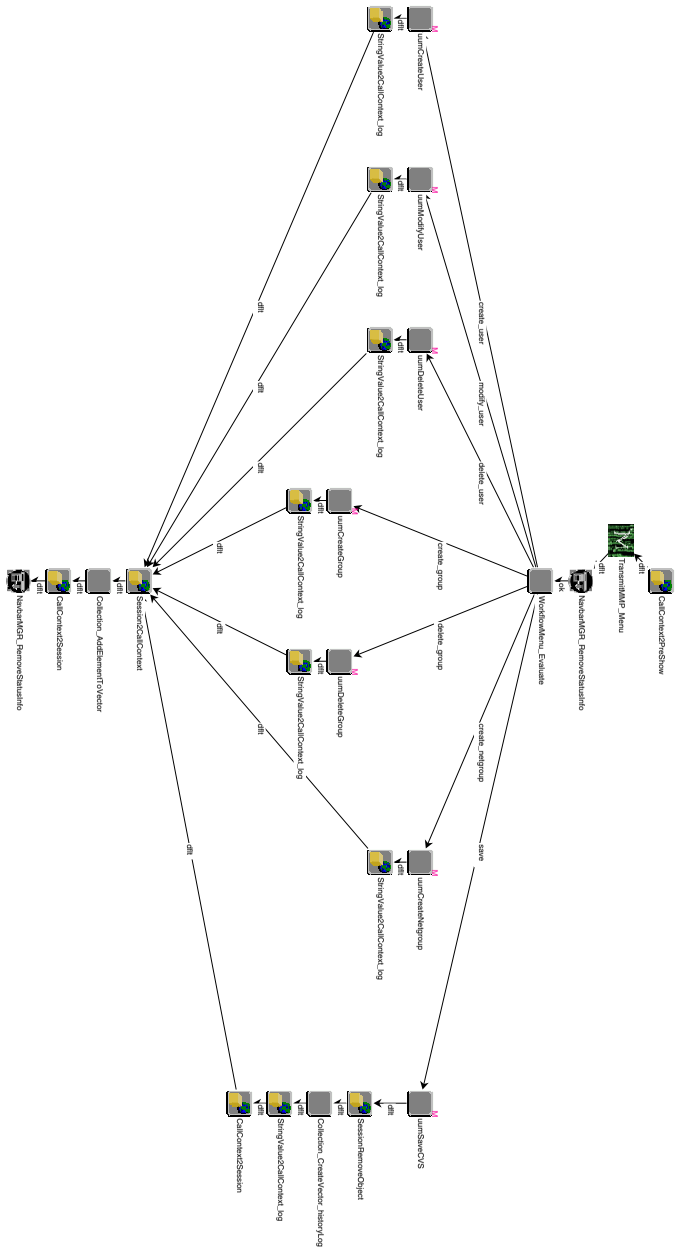

Abbildung 6.23: UUM-Service - Die Dienstweiche

Benutzer in diesem Fall dann alle bislang beschriebenen Teildienste zur Verfügung. Selektiert der Anwender hingegen einen Pool und wählt *Configure Pool*, so stehen nur die Dienste zur Verfügung, die die Verarbeitung mehrerer ConfigClients zugleich erlauben.

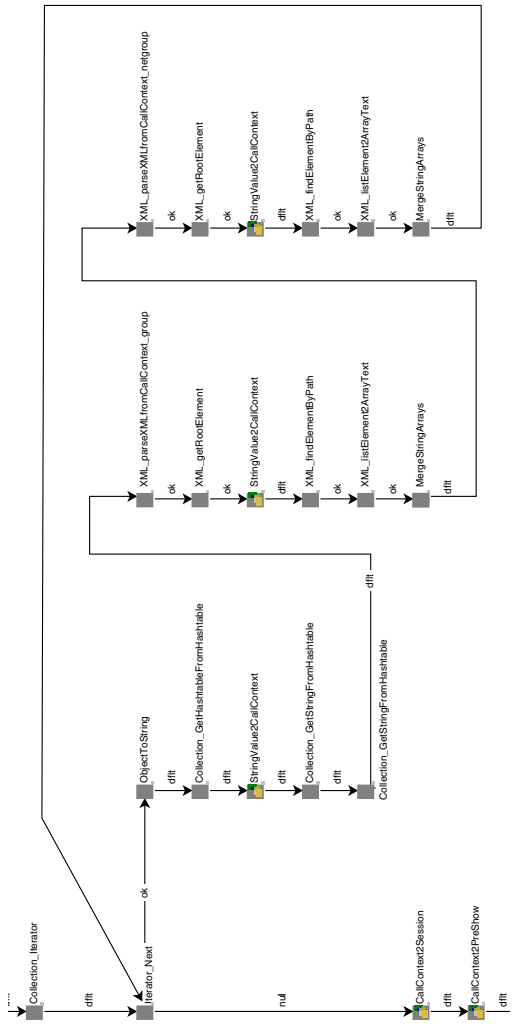

Abbildung 6.24: UUM-Service - Einlesen und Mergen von Konfigurationsdateien per Iteration

Im Service-Logik-Graphen werden mit der Auswahl des Pools bzw. des ConfigClients auch sofort deren Konfigurationsdateien vom Versionmanager angefordert. Dieser arbeitet so, daß er die vier Konfigurationsdateien jedes ConfigClients ausliest und in das CVS-Repository ablegt, falls diese dort noch nicht vorliegen. Desweiteren wird die Datei - wenn möglich - bereits an dieser Stelle in das XML formatiert. Der Rahmendienst arbeitet nahtlos mit dem Versionmanager zusammen, da anstelle der Konfigurationsdateien selbst direkt die konvertierten XML-Dateien angefordert werden können. Zudem bietet

das System mittels der Versionsverwaltung CVS alle Vorraussetzungen dafür, daß Versionen später wiederhergestellt werden können. Um die Versionsverwaltung später einbauen zu können, wurde zusätzlich eine Datenbank angelegt, die nach jedem Speichervorgang *SaveCVS* mit neuen Einträgen der einzelnen ConfigClients gefüllt wird. Diese Einträge enthalten neben der ID des jeweiligen ConfigClients bzw. des Pools auch die Revisions der zugehörigen Konfigurationsdateien. Ein zusätzlicher TimeStamp bringt den Vorteil, daß Konfigurationen von Pools später dadurch identifiziert werden können, daß sie zum einen die selbe Pool-ID, zum anderen aber auch den gleichen TimeStamp besitzen.

Das Menü selbst enthält neben Buttons zum Aufruf der einzelnen Teildienste auch ein Verlaufsfenster. Wurde ein Teildienst abgeschloßen, wird ein entsprechender Eintrag in diesem Fenster vorgenommen, um dem Anwender später eine bessere Übersicht zu gewährleisten. Nach jedem Speichervorgang mittels *SaveCVS* wird das Fenster zurückgesetzt. Die eigentliche Verzweigung bei Auswahl eines Teildienstes läß sich gut in der letzten Abbildung erkennen. In jedem Fall wird nach der Abarbeitung des Teildienstes wieder zurück ins Menü geführt. So ist es möglich, mehrere Konfigurationen hintereinander auszuführen, bevor eine Konfiguration gespeichert wird.

Das Bereitstellen der Konfigurationsdateien für die einzelnen Teildienste erfolgt mittels einer Hashtable, die für jeden ConfigClient das zugehörige Business-Objekt sowie eine Hashtable der einzelnen vier Konfigurationsdateien enthält. Der Rahmendienst ist so konstruiert, daß er jederzeit völlig dynamisch mit Teildiensten erweitert werden kann. Dank den Vorbereitungen in Bezug auf die Versionsverwaltung, sollte es später auch auf einfache Art und Weise möglich sein, ein Rollback zu installieren, bei dem Versionen wiederhergestellt werden können.

6.2.3 Zusammenfassung

Im Rahmen unserer Arbeit ist möglich geworden, über die MaTRICS Benutzer- und Gruppenverwaltung sowohl auf einzelnen Rechnern als auch auf kompletten Pools durchführen zu können.

Mit Hilfe des UnixUserManagement-Dienstes können einzelne Benutzer auf einem oder mehreren Rechnern angelegt werden. Es können Home-Verzeichnise, sowie deren Login-Shells definiert werden. Passwörter werden nach Wahl entweder im Klartext vorgegeben oder aber vom UnixUserManagement-Dienst automatisch generiert. Die Gruppenzugehörigkeit anzulegender Benutzer kann ebenfalls definiert werden.

In Anbetracht der Benutzerverwaltung von z.B. Übungsgruppen ist besonders hervorzuheben, dass standardisierte Benutzer angelegt werden können, deren Login-Name sich z.B. in einer laufenden Nummer unterscheidet. Somit ist ein einfaches Anlegen von Benutzeraccounts für eine vorlesungsbegleitende Übung realisiert. Die automatisch generierten Benutzeraccounts und Passwörter können danach bequem in Form einer Liste ausgedruckt und unter den Übungsteilnehmern verteilt werden. Nach dem Ende der Veranstaltung können die vorhandenen Accounts ebenfalls mit dem UnixUserManagement-Dienst wieder

entfernt werden. Gruppen- und Netzgruppenzugehörigkeiten können jederzeit angepasst werden.

Um ein Rückgängigmachen von unerwünschten Änderungen zu realisieren, wurde als Grundlage eine Datenbankverwaltung der wichtigen Konfigurationsdateien und deren Stand zu gewissen Konfigurationszeitpunkten realisiert.

Dank seines modularen Aufbaus kann der Dienst leicht um weitere Funktionalitäten (z.B. Rollback, NIS-Unterstützung) erweitert werden.

Kapitel 7

Fazit

Die vorangegangenen Kapitel haben auf objektive und technisch orientierte Weise die Arbeit unserer Projektgruppe dargestellt. Dieses letzte Kapitel möchten wir nutzen, um die Projektgruppe auf subjektive Weise zu resümieren und ein allgemeines Fazit zu ziehen.

Unsere Projektgruppe begann mit einem Seminar, welches den Teilnehmern die Möglichkeit bot sich kennenzulernen und sich in ersten Schritten dem Thema anzunähern. Die auswärtige Durchführung des Seminars im Universitätskolleg Bommerholz führte schnell zu persönlichem Kontakt unter den Teilnehmern und legte erste Grundsteine für die Bildung eines wir-Gefühls.

Die Kurzthemen gaben uns die Gelegenheit erste Erfahrungen mit dem Arbeiten in einer Gruppe und der Entwicklungsumgebung zu machen. Durch den vorbereitenden Charakter der Seminarthemen und die gute Betreuung durch den Lehrstuhl war das Niveau der Kurzthemen angemessen um die Entwicklungsumgebung kennenzulernen und die Arbeiten in der vorgegebenen Zeit abzuschließen.

Der Zwischenbericht bot der Gruppe die erste Gelegenheit die Organisation innerhalb der Gruppe aktiv mitzugestalten. Die Betreuer gaben lediglich einen finalen Abgabetermin vor und überließen es uns diesen Termin einzuhalten.

Nachdem wir uns mit den Kurzthemen in die Entwicklungsumgebung eingearbeitet hatten, konnten wir die Zwischenthemen dafür verwenden uns in die bestehenden Komponenten der MaTRICS einzuarbeiten, diese zu verstehen und durch deren Weiterentwicklung das erste Ziel unserer PG zu erreichen.
Diesesmal wurden die zeitlichen Rahmenbedingungen zwischen den Betreuern und den Teilnehmern ausgemacht und die interne Organisation wurde in die Hände der Teilnehmer gelegt.
Leider führte dies zu weniger erfreulichen Ergebnissen. Die Kleingruppen neigten dazu die Aufgaben zu Beginn der Bearbeitungszeit aufzuteilen und diese Aufteilung später strikt beizubehalten. Da jedoch die Komplexität der Themen zu Beginn noch nicht ab-

zusehen war, kam es natürlich dazu, dass einige Teilnehmer früher fertig waren als andere. Einige Teilnehmer verhielten sich gruppengerecht und unterstützten die Teilnehmer die langwierigere oder komplexere Teilaufgaben übernommen hatten. Andere Teilnehmer vertraten allerdings die Meinung, dass sie ihren Teil der Aufgabe erledigt hätten und mit dem Rest nichts mehr zu tun hätten. Dies führte unweigerlich zu der Nichteinhaltung der zeitlichen Vorgaben.

Nicht entmutigt sondern von dem Leitspruch - aus Fehlern lernen wir - angetrieben begaben wir uns an die Hauptthemen. Hier hatten wir die Gelegenheit die fachlichen und organisatorischen Erfahrungen aus den vorangegangenen Phasen umzusetzen. Die Organisation wurde nun vollständig in die Hände der Teilnehmer gelegt. Dies stellte sich aufgrund der verdoppelten Gruppengröße als problematischer heraus als zunächst angenommen. Die Betreuer waren weiterhin sehr hilfsbereit und standen immer als Berater und Helfer in der Not zur Verfügung.

Die Projektgruppe gab uns die Möglichkeit in kontrollierter Umgebung den Alltag eines Informatikers zu erleben und uns auf eben diesen vorzubereiten.
Um eine neue Aufgabe zu bewältigen, mussten wir uns in neue Themen und in neue Entwicklungsumgebungen einarbeiten. Wir konnten erleben, dass Aufgaben mit steigender Komplexität nicht mehr von einzelnen Personen bewältigt werden können, sondern vielmehr die Zusammenarbeit in einer Gruppe erfordern.
Neben diesen wichtigen Erfahrungen für die Zukunft konnten wir den praktischen Nutzen einiger von uns besuchter Vorlesungen erkennen.

Werfen wir nun einen letzten Blick auf die definierten Minimalziele und in wie weit diese von der Projektgruppe erreicht wurden.

- Weiterentwicklung der Basiskomponenten des ConfigManager. Dieses Minimalziel wurde in den Zwischenthemen erreicht.
- Einsatz einer universellen Beschreibungssprache zur Spezifikation der GUI beim WebAgent. Die Entwicklung der universellen Beschreibungssprache ist derzeit Teil einer Diplomarbeit am Lehrstuhl 5 des Fachbereichs Informatik an der Universität Dortmund. Den Grundstein für die Verwendung dieser Sprache haben wir mit Entwicklung des MMP Tunnels gelegt.
- Modellierung von Konfigurationsdiensten, die in MaTRICS bereitgestellt werden. Dieses Ziel wurde durch die Hauptthemen erreicht.
- Entwicklung eines Protokolls für autonome ConfigClients. Dieses Ziel wurde zu Gunsten eines wichtigeren Themas, der internen Verarbeitung der Konfigurationsdateien im XML Format, aufgegeben. Dieses neu gesteckte Ziel wurde in den Kurz- und in den Zwischenthemen bearbeitet und erreicht.

Mit Blick auf das vergangene Jahr und auf die Ergebnisse kann die Projektgruppe nur als Erfolg und als wertvolles Element unseres Studiums bezeichnet werden.

Literaturverzeichnis

[1] Bastian Raining *Design von Webapplikationen mit ABC/EWIS,*Ausarbeitung in PG MaTRICS1

[2] *METAFrame Technologies GmbH The Agent Biulding Center Users Guide*

[3] *Apache Software Foundation, http://www.apache.org*

[4] *Tomcat - Webserver und JSP/Servlet Container, http://tomcat.apache.org*

[5] *Cisco SNMP Guide, http//www.cisco.com*

[6] *Endbericht MaTRICS, PG 451*

[7] *http://www.borland.com/together*

[8] *http://www.borland.com*

[9] Internet *Stand 2005*, offizielle Adresse: www.eclipse.org

[10] METAFrame Technologies GmbH *The Agent Building Center: Bringing a web application into operation*

[11] C. Beerens, J. H. Jellesma, M. Krücken *2005 Dokumentation der SIB Bibliothek WebTesting*