Endbericht

Entwicklung eines Konfigurations-werkzeugs für Computersysteme

Azadeh Alebrahim, Mohamed Babiker, Daniel da Silva Lopes,
Chadi El-Charif, Martin Gernandt, Miloud Hammouch, Sebastian Hundt,
Jörg Knoche, Timo Kockert, Andre Lagerpusch,
Illya Sokolov, Thomas Stegemann, Thi Bich Lien Tran

PG 488
am Fachbereich Informatik
Universität Dortmund
Lehrstuhl V

01. April 2007

Veranstalter:

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

Inhaltsverzeichnis

1 Einleitung / Übersicht **1**

1.1 Einleitung 1

1.2 Was ist die MaTRICS? 2

1.3 Ziele der PG 3

2 Seminarphase **5**

2.1 Objektorientiertes Design auf Basis von UML 5

2.2 Entwicklung von Business Applikationen nach dem J2EE Standard 7

2.3 SOA Entwicklung von Web-Services nach dem Modell WSDL 9

2.4 Konzepte von SSH 10

2.5 Technologie von Java Server Faces(JSF) 10

2.6 Kommunikation auf Basis von Bluetooth in Java 11

2.7 Datenbanken in Java mit Hibernate und JDBC 11

2.8 Grundlagen von LDAP 12

2.9 Die Entwicklungsumgebung ABC mit EWIS 13

2.10 Die neue Entwicklungsumgebung jABC mit jEWIS 14

2.11 Bash-Programmierung unter Linux 14

2.12 Service Availability und Hochverfügbarkeits-Systeme 15

2.13 Kryptographie und Verschlüsselung in Java 15

3 Aufwärmphase **17**

3.1 Kniffel 17

3.2 Organizer 19

3.3 Währungskonverter 21

3.4 Schiffe versenken 23

3.5 Taschenrechner 25
3.6 Kalender 28
3.7 Telefonbuch 30

4 Kurzthemen **33**

4.1 Design und Implementierung einer SIB Bibliothek zur Kommunikation auf Basis von Bluetooth 33
4.1.1 Einleitung 33
4.1.2 Design 34
4.1.2.1 Anwendungsfalldiagramm 34
4.1.2.2 Aktivitäts-Diagramm 36
4.1.2.3 Sequenz-Diagramm 36
4.1.2.4 Klassen-Diagramm 39
4.1.3 Implementierung 43
4.1.4 Beispielapplikation BluChat 43
4.1.5 Benötigte Bibliothek 45
4.2 Design und Implementierung einer SIB Bibliothek für die Ausführung von BASH-Befehlen 47
4.2.1 Einleitung 47
4.2.2 Design 47
4.2.3 Implementierung und Test 49
4.2.4 Beispielanwendung 51
4.3 Design und Implementierung einer SIB-Bibliothek zum Testen und zur Diagnose von Netzwerken basierend auf Network-Sockets 54
4.3.1 Einleitung 54
4.3.2 Design 55
4.3.2.1 Usecase-Diagramm 55
4.3.2.2 Activity-Diagramm 56
4.3.2.3 Sequence-Diagramm 58
4.3.3 Implementierung und Test 58
4.3.3.1 Übersicht 58
4.3.3.2 Benutzung 59
4.3.4 Beispiel 59
4.3.4.1 Test des CVS-Protokolls 59

4.3.4.2 Aufruf eines externen Befehls 60
4.3.4.3 Ausführung des Beispiels 61
4.4 Design und Implementierung einer SIB Bibliothek zur Kommunikation und Verwaltung von LDAP 62
4.4.1 Einleitung 62
4.4.2 Design 62
4.4.2.1 UseCase Diagramm 62
4.4.2.2 Aktivitätsdiagramm 63
4.4.3 Implementation 64
4.4.3.1 Überblick 64
4.4.3.2 Anwendung 65
4.4.3.3 Beispielanwendung 65

5 Zwischenthemen **69**
5.1 Integration einer Notification Komponente 69
5.1.1 Einleitung 69
5.1.2 Design 70
5.1.2.1 Anwendungsfalldiagramm: Notification Dienst 70
5.1.2.2 Aktivitätsdiagramm: Send Notification 70
5.1.3 Aktivitätsdiagramm: Notification Management 73
5.1.4 Implementierung 73
5.1.4.1 Notification Komponente 73
5.1.4.2 Notification senden 76
5.1.4.3 ClientProtocolManager 76
5.1.5 Ausblick 77
5.2 Erweiterung der ConfigAgents um eine allgemeine GUI Beschreibungssprache 78
5.2.1 Einleitung 78
5.2.2 Aufgabenbeschreibung 78
5.2.3 Design 78
5.2.3.1 Anwendungsfalldiagramm 78
5.2.3.2 Sequenzdiagramm 79
5.2.4 Implementierung und Test 81

5.2.5 WebAgent 84
5.2.6 EmailAgent 84
5.2.7 Anwendungsbeispiel 89
5.3 Entwicklung einer autonomen Ausführungskomponente für ConfigClients 96
5.3.1 Einleitung 96
5.3.2 Aufgabenbeschreibung 96
5.3.3 Design 96
5.3.3.1 ConfigClient/ConfigManager Use Case Diagramm . . . 96
5.3.3.2 MCPServer Use Case Diagram 97
5.3.3.3 Sequenz Diagramm ExecuteJob 98
5.3.3.4 Klassendiagramm, MCP 98
5.3.4 Implementierung und Test 101
5.3.5 Beispielanwendung 102
5.4 Design einer Monitoringbibliothek auf Basis von autonomen ConfigClients 104
5.4.1 Einleitung 104
5.4.2 Aufgabenbeschreibung 104
5.4.3 Design 104
5.4.3.1 Use Case Diagramm 104
5.4.3.2 Activity Diagramm 104
5.4.3.3 Sequence Diagramm 107
5.4.3.4 Class Diagramm 108
5.4.4 Implementierung 108
5.4.4.1 Erstellte JAVA Klassen 108
5.4.4.2 Dateiverwaltungs SIBs 108
5.4.4.3 Threading SIBs 109
5.4.4.4 Application SIBs 110
5.4.4.5 ConfigClient SIBs 113
5.4.5 jABC-Graphen 113
5.4.6 Ausblick 114

6 Hauptthema - Entwicklung eines Watchdog-Systems zur Diagnose und Steuerung von Unix-Prozessen 117

6.1 Einleitung . . . 117
6.2 Aufgabenbeschreibung . . . 118
6.3 Design . . . 118
6.3.1 Anwendungsfalldiagramm . . . 118
6.3.1.1 Watchdogmanagement . . . 120
6.3.1.2 Watchdog Thread . . . 120
6.3.1.3 Messung . . . 120
6.3.2 Anwendungsfalldiagramm ConfigClient . . . 121
6.3.3 Aktivitätsdiagramm, Watchdog Engine . . . 123
6.3.4 Aktivitätsdiagramm, Profile Management . . . 123
6.3.5 Aktivitätsdiagramm, Prozess Management . . . 123
6.3.6 Aktivitätsdiagramm, Freier Speicherplatz . . . 126
6.3.7 Sequenzdiagramm, Watchdog Engine . . . 126
6.3.8 Klassendiagramm . . . 128
6.3.8.1 Management . . . 129
6.3.8.2 Reaction . . . 129
6.3.8.3 Result . . . 130
6.4 Implementierung Management . . . 130
6.4.1 SIB Implementierung . . . 130
6.4.2 WatchdogManagement . . . 131
6.4.2.1 Create Profile . . . 132
6.4.2.2 View Profiles . . . 135
6.4.2.3 Create Process . . . 135
6.4.2.4 View Processes . . . 141
6.5 Implementierung Service . . . 144
6.5.1 SIB Implementierung . . . 144
6.5.2 Service Graph . . . 147
6.6 Implementierung ConfigClient . . . 154
6.7 Beispielanwendung . . . 158
6.7.1 WatchdogManagement . . . 158
6.7.1.1 Create Profile . . . 158

6.7.1.2 View Profiles 163
6.7.1.3 Create Process 163
6.7.1.4 View Processes 163
6.7.2 Test-Graphen 166

7 Fazit **171**

Literaturverzeichnis **173**

Abbildungsverzeichnis

1.1 Aufbau der MaTRICS 3

2.1 J2EE, Übersicht 8

3.1 Kniffel, Webseite 18
3.2 Kniffel, jABC-Graph 19
3.3 Organizer, jABC-Graph 20
3.4 Organizer, Startseite 21
3.5 Währungskonverter, jABC-Graph 22
3.6 Währungskonverter, Startseite 22
3.7 Schiffe Versenken, jABC-Graph 23
3.8 Schiffe Versenken, Start 24
3.9 Schiffe Versenken, Spiel 24
3.10 Schiffe Versenken, Ergebnis 25
3.11 Taschenrechner, Hauptseite 26
3.12 Taschenrechner, jABC-Graph 27
3.13 Kalender, jABC-Graph 28
3.14 Kalender, Ergebnis 29
3.15 Telefonbuch, Eingabe 30
3.16 Telefonbuch, Eintrag 31

4.1 Bluetooth, Usecase-Diagram 35
4.2 Bluetooth, ActivityDiagramClient 37
4.3 Bluetooth, ActivityDiagramServer 38
4.4 Bluetooth, SequenceDiagramClient 40
4.5 Bluetooth, SequenceDiagramServer 41

4.6 Bluetooth, Class-Diagram 42
4.7 Bluetooth, BluChat Startseite 44
4.8 Bluetooth, BluChat jABC- Graph 46
4.9 Bash, Usecase-Diagramm 48
4.10 Bash, Aktivitätsdiagramm 52
4.11 Bash, Beispiel jABC-Graph 53
4.12 Sockets, Usecase-Diagramm 56
4.13 Sockets, Activity-Chart 57
4.14 Sockets, Sequence-Chart 58
4.15 Sockets, jABC-Graph CVS-Test 60
4.16 Sockets, jABC-Graph Aufruf Externer Befehl 61
4.17 LDAP, Usecase-Diagramm 62
4.18 LDAP, Aktivitätsdiagramm 64
4.19 LDAP, jABC-Graph zum Löschen eines Eintrags 66
4.20 LDAP, Verzeichnisstruktur vor dem Löschen eines Eintrags 67
4.21 LDAP, Verzeichnisstruktur nach dem Löschen eines Eintrags 68

5.1 Notification, Usecase-Diagramm 71
5.2 Notification, Aktivitätsdiagramm: Send Notification 72
5.3 Notification, Aktivitätsdiagramm: Notification Management 74
5.4 Notification, jABC Graph - Macro ConvertAndStore 76
5.5 GUI, Usecase-Diagramm 80
5.6 GUI, initializeGUIengine-server 81
5.7 GUI, initializeGUIengine-client 82
5.8 GUI, generateOutput 82
5.9 GUI, handleRequest 83
5.10 GUI, WebAgent 85
5.11 GUI, WebAgent:Nach der Initialisierung 86
5.12 GUI, WebAgent:Generierung der Ausgabe 87
5.13 GUI, WebAgent:Parst den Request und übergibt ein xml-String 87
5.14 GUI, EmailAgent: Init service 88
5.15 GUI, EmailAgent: Kreislauf zum schauen nach neuen Mails 90
5.16 GUI, EmailAgent: Erste Mail 91

5.17 GUI, EmailAgent: Antworten auf die erste Mail und Auswahl des Dienstes 92
5.18 GUI, EmailAgent: executeNextStep 93
5.19 GUI, GUI_DL_Test Graph 94
5.20 Ausführungskomponente, Usecase-Diagramm Configclient/Configmanager 97
5.21 Ausführungskomponente, Usecase-Diagramm MCPServer 98
5.22 Ausführungskomponente, Sequenzdiagramm ExecuteJob 99
5.23 Ausführungskomponente, Klassendiagramm MCP 100
5.24 Ausführungskomponente, MCPThread jABC-Graph 103
5.25 Monitoring, Usecase-Chart 105
5.26 Monitoring, Aktivitätsdiagramm der Ausführung eines Monitorings . . . 106
5.27 Monitoring, Sequence-Chart von der Erstellung eines MonitoringJobs bis zum Senden an den ConfigClient via MCP 107
5.28 Monitoring, MonitoringManagement Dienst 114
5.29 Monitoring, jABC-Graph des MonitoringWorkerThread auf ConfigManager-Seite 115
5.30 Monitoring, jABC-Graph des MonitoringControllerThread auf ConfigManager-Seite 116

6.1 Watchdog, Anwendungsfalldiagramm 119
6.2 Watchdog, Anwendungsfalldiagramm ConfigClient 121
6.3 Watchdog, Watchdog Engine, Aktivitätsdiagramm 122
6.4 Watchdog, Watchdog Profile Management, Aktivitätsdiagramm 124
6.5 Watchdog, Watchdog Prozess Management, Aktivitätsdiagramm 125
6.6 Watchdog, Aktivitätsdiagramm Freier Speicherplatz 126
6.7 Watchdog, Watchdog Engine, Sequenzdiagramm 127
6.8 Watchdog, Klassendiagramm 128
6.9 jABC Graph - WatchdogManagement Initial Part 131
6.10 jABC Graph - WatchdogManagement CreateProfile - CreateContainerProfile 133
6.11 jABC Graph - WatchdogManagement CreateProfile - CreateParameterContainerList 134
6.12 jABC Graph - WatchdogManagement CreateProfile - Select Reactions . . 135
6.13 jABC Graph - WatchdogManagement - Macro ”Watchdog_SelectReactions” 136
6.14 jABC Graph - WatchdogManagement CreateProfile - StoreContainers2DB 137
6.15 jABC Graph - WatchdogManagement ViewProfiles 138

6.16 jABC Graph - WatchdogManagement CreateProcess - Select ConfigClient 139
6.17 jABC Graph - WatchdogManagement CreateProcess - Select Profile . . . 140
6.18 jABC Graph - WatchdogManagement CreateProcess - Bestätigen 142
6.19 jABC Graph - WatchdogManagement CreateProcess - Finish 143
6.20 Watchdog, jABC Graph - WatchdogThread 146
6.21 Watchdog, jABC Graph - StatisticQueue öffnen 147
6.22 Watchdog, jABC Graph - Statistik-ID aus der Queue lesen 148
6.23 Watchdog, jABC Graph - Statistikdaten lesen 149
6.24 Watchdog, jABC Graph - Reaktion lesen 150
6.25 Watchdog, jABC Graph - Reaktion ausführen 151
6.26 Watchdog, jABC Graph - Watchdog Benachrichtigung verschicken 151
6.27 Watchdog, jABC Graph - Watchdog Script ausführen 152
6.28 Watchdog, jABC Graph - Watchdog Process starten/stoppen 153
6.29 jABC Graph - Measurementgraph(Memory) 157
6.30 Example Application - WatchdogManagement - Main 158
6.31 Example Application - WatchdogManagement - Create Profile 159
6.32 Example Application - WatchdogManagement - Create Profile 159
6.33 Example Application - WatchdogManagement - Create Profile 160
6.34 Example Application - WatchdogManagement - Create Profile 160
6.35 Example Application - WatchdogManagement - Create Profile 161
6.36 Example Application - WatchdogManagement - Create Profile 162
6.37 Example Application - WatchdogManagement - View Profiles 163
6.38 Example Application - WatchdogManagement - Create Process 164
6.39 Example Application - WatchdogManagement - Create Process 164
6.40 Example Application - WatchdogManagement - Create Process 165
6.41 Example Application - WatchdogManagement - Create Process 165
6.42 Example Application - WatchdogManagement - View Process 165
6.43 Watchdog, jABC Graph - CPU-Measurement 166
6.44 Watchdog, jABC Graph - UnpackHashMap 167
6.45 Watchdog, jABC Graph - GetParameterFromHashMap 168
6.46 Watchdog, jABC Graph - CreateHashMapCPU 168
6.47 Watchdog, jABC Graph - Get Warn/Crit Parameter 169
6.48 Watchdog, jABC Graph - Set Levels 169

Kapitel 1

Einleitung / Übersicht

1.1 Einleitung

Die Projektgruppe 488 an der Universität Dortmund wird die Entwicklung des MaTRICS-Systems, das von der Projektgruppe 451 entwickelt und von der Projektgruppe 469 weiterentwickelt wurde, fortführen. MaTRICS ist die Abkürzung für „Management Tool für Remote Intelligent Configuration of Systems“ und stellt ein verteiltes intelligentes Konfigurationssystem für Computersysteme dar. Von den Projektgruppen 451 und 469 sind folgende Komponenten entwickelt worden:

- Adressierung der ConfigClients
- Bereitstellung der Konfigurationsdienste
- Weiterleitung von Aktionen an die ConfigClients
- Verwaltung von Konfigurationsdaten
- Bereitstellung der Kommunikationsprotokolle
- Entwicklung des Version Managers
- Bereitstellung der Jobflow-Engine

Das Ziel der Projektgruppe 488 ist es, noch fehlende Basiskomponenten, wie z.B. ein NotificationManagement zu entwickeln und damit zusätzliche Funktionen in der MaTRICS zur Verfügung zu stellen. Zum anderen sollen aber auch Komponenten erweitert werden, z.B. das jABC, und weiterhin werden noch weitere Konfigurationsdienste in die MaTRICS integriert.

1.2 Was ist die MaTRICS?

MaTRICS soll die Administratoren von der Komplexität der eingesetzten Server- und Softwaresysteme entlasten. Dabei wird der gesamte Lebenszyklus der Server-Systeme als auch deren Software-Konfiguration transparent überwacht und intelligent konfiguriert. Alle Installations- und Konfigurationsvorgänge sollten einfach und, unterstützt durch einen personalisierten Agenten, in einer Konfigurationsumgebung durchführbar sein. Dabei sollen alle Aktivitäten an den Konfigurationsdateien und Prozessen der zu konfigurierenden Maschine analysiert werden, um auftretende Probleme beim Betrieb zu minimieren und eine erhöhte Verfügbarkeit der Server-Systeme zu garantieren. Mit MaTRICS wollen wir eine einheitliche Konfigurationsumgebung entwickeln, die den Zugang über verschiedenste Kommunikationswege (Webfrontend, E-Mail, SMS, etc.) ermöglicht. MaTRICS besteht aus 3 verteilten Komponenten, siehe Abbildung 1.1:

- **ConfigAgents**: sie sind spezialisierte Agenten zur Unterstützung der Menge an Kommunikationsmedien der MaTRICS zwischen dem Client auf Benutzerseite und dem ConfigManager. Sie stellen somit eine Schnittstelle zwischen Benutzer und dem ConfigManager dar. Ein rudimentärer WebAgent wurde bereits entwickelt. Dabei wurde die GUI in Form von HTML-Seiten starr an den Ausgabekanal gekoppelt. Eine am Lehrstuhl entwickelte Visualisierungs-Komponente auf Basis von XML erlaubt es, die Darstellung unabhängig von einem Ausgabeformat zu beschreiben.

- **ConfigClient**: er verarbeitet auf den Zielmaschinen die Aktionen, die vom ConfigManager initiiert wurden. ConfigClients sind daher relativ einfach gehalten - sie stellen eine Systembibliothek bereit, die einen Zugriff auf das Betriebssystem und die Prozesse der Zielmaschine ermöglichen. Instanzen der ConfigClients realisieren die Schnittstelle zwischen Server und ConfigManager. Dabei steuert der zentrale ConfigManager die Akteure fern (remote management).

- **ConfigManager**: er ist das Bindeglied zwischen ConfigAgent und ConfigClient, und stellt somit den Kern der Konfigurationsumgebung der verteilten Serverprozesse dar. Seine Aufgaben sind:

  - Validierung aller Aktionen der ConfigAgents
  - Realisierung des Nutzer/Rollen Management
  - Bereitstellung aller Konfigurationsdienste
  - Delegation von Konfigurations-Aufträgen an die ConfigClients
  - Bereitstellung einer JobFlow-Engine für Konfigurations-Aufträge
  - Verwaltung und Versionierung von Konfigurationsdaten
  - Bereitstellung der Kommunikationsprotokolle
  - Erkennung von Konflikten und Abhängigkeiten bei der Konfiguration eines Servers

– Realisierung der einzelnen Workflows zu den Administrations- und Konfigurationsvorgängen eines Servers

Die Akteure der MaTRICS sind die zu konfigurierenden Server.

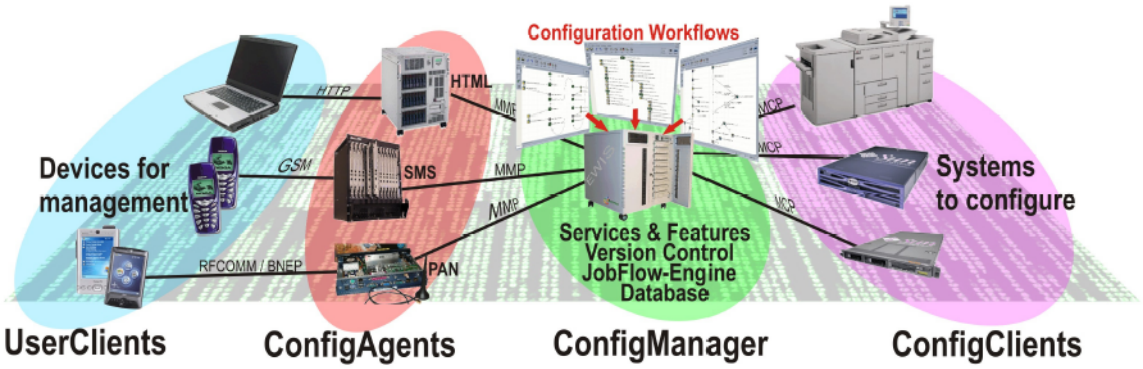

Abbildung 1.1: Aufbau der MaTRICS

1.3 Ziele der PG

Die Ziele unserer PG sind u.a.:

- die Modellierung von Konfigurationsdiensten, die in MaTRICS bereitgestellt werden
- die Entwicklung weiterer ConfigAgents (SMSAgent, E-MailAgent)
- Erstellung eines NotificationManagements
- die Migration auf die neue Entwicklungsumgebung jABC

Kapitel 2

Seminarphase

Der Sinn der Seminarphase bestand hauptsächlich darin, sich in die Themen einzuarbeiten, mit denen wir uns in der Projektgruppe MaTRICS III beschäftigen werden. Ein Ziel war auch das Kennenlernen der PG-Teilnehmer selbst. Insgesamt wurden vierzehn Seminarthemen vorgestellt, die jeweils von einem Teilnehmer bearbeitet wurden. In der Vorbereitung wurden die Folien der Präsentation und die entsprechenden Ausarbeitungen erstellt. Beim Seminar wurden die Themen in einer 30-minütigen Präsentation mit anschließender Diskussion vorgetragen. Die Seminarphase fand am 20. und 21. März 2006 im Gästehaus der Universität Dortmund im Haus Bommerholz in der Nähe von Witten statt. In diesem Kapitel des Zwischenberichts findet sich eine kurze Zusammenfassung der einzelnen Seminarthemen.

2.1 Objektorientiertes Design auf Basis von UML

In dieser Ausarbeitung geht es um das Objektorientierte Design auf Basis von UML. Am Anfang wird UML und Ihre Entstehungsgeschichte vorgestellt. Danach werden die 13 UML Diagrammformen aufgezählt. Sie werden in Verhaltensdiagrammtypen und Strukturdiagrammtypen aufgeteilt. Ein paar Beispiele von ihnen werden vorgestellt und erklärt - das Anwendungsfalldiagramm, Aktivitätsdiagramm, Sequenzdiagramm und Zustandsdiagramm. Danach kommen wir zum Objektorientierten Design in Java in Zusammenhang mit Klassendiagrammen. Am Ende wird ein Beispiel für die Modellierung mit Together erklärt.

Was ist die UML?

UML ist eine Abkürzung für Unified Modeling Language, auf Deutsch: Die Einheitliche Modellierungssprache. Sie ermöglicht es uns Systeme in Worten und Bildern zu beschreiben. Im Sinne einer Sprache definiert die UML dabei Bezeichner für die meisten Begriffe, die für die Modellierung wichtig sind, und legt mögliche Beziehungen zwischen diesen Begriffen fest. Verschiedenartige Syteme können modelliert werden z.B. Software-Systeme, Geschäftssysteme oder beliebige andere Systeme. Die Diagramme von UML

sind nicht grundsächtzlich neu. Neu an der UML ist die **Weltweite Einigung** auf eine Modellierungssprache, die von der **Object Management Group (OMG)** standardisiert wurde.

Vorteile von UML:

UML als Modellierungssprache zu benutzen hat mehrere Vorteile, einige davon sind:

- Sie erleichtert die Verständigung und Austausch von Modellen.
- Sie wächst mit ihren Anforderungen an die Modellierung.
- Sie ist Praxisnah.
- auf UML basierende Angebote lassen sich leicht vergleichen.

Was ist ein Modell?

Um existierende oder zukünftige Systeme besser zu verstehen muss man ein Modell erstellen. Ein Modell entspricht nicht genau der Realität. Es hebt die wesentlichen Details hervor und lässt die unwichtigen Details weg. Was wesentlich und was unwichtig ist hängt davon ab, was mit dem Modell für Ziele verfolgt werden und welche Zielgruppe diesen Modell betrachten wird. Je mehr Informationen in einem Modell gezeigt werden sollen, desto komplexer und schwieriger wird es. Es werden verschiedene Sichten auf den betrachteten Gegenstand gebildet, die miteinander verbunden sind. Wenn ein Sicht geändert wird, müssen alle anderen Sichten ebenfalls angepasst werden.

Warum betrachtet man Modelle?

Das Modell eines System muss folgende Aufgaben erfüllen:

- Die **Kommunikation** zwischen allen Beteiligten ermöglichen.
- Die Fakten für Auftraggeber, Fachexperten und Anwender **visualisieren**.
- Die Fakten bzgl. Vollständigkeit, Widerspruchsfreiheit und Korrektheit **überprüfen**.

Anwendungsfalldiagramm (Use-case diagram oder Nutzungsfalldiagramm)

In einem Anwendungsfalldiagramm werden Akteure, Anwendungsfälle und deren Beziehungen dargestellt. Es beschreibt die Zusammenhänge zwischen verschiedenen Anwendungsfällen untereinander und zwischen Anwendungsfällen und den beteiligten Akteuren. Die Beziehung der Akteure zu den Anwendungsfällen sagt aus, dass ein Akteur eine bestimmte Dienstleistung des Systems nutzen kann.

Aktivitätsdiagramm

In einem Aktivitätsdiagramm werden die Objekte eines Programmes mittels der Aktivitäten, die sie während des Programmablaufes vollführen, beschrieben. In einem Programmablauf durchläuft ein Modellelement eine Vielzahl von Aktivitäten, d.h. Zuständen, die eine interne Aktion und mindestens eine daraus resultierende Transition enthalten. Die ausgehende Transition impliziert den Abschluss der Aktion und den Übergang des Modellelementes in einen neuen Zustand bzw. eine neue Aktivität. Diese Aktivitäten können

in ein Zustandsdiagramm integriert werden oder besser in einem eigenen Aktivitätsdiagramm visualisiert werden.

Sequenzdiagramm (Ereignispfaddiagramm oder Nachrichtendiagramm)
Das Sequenzdiagramm beschreibt die zeitliche Abfolge von Interaktionen zwischen einer Menge von Objekten innerhalb eines zeitlich begrenzten Kontextes. Mittels des Sequenzdiagrammes beschreibt man die Interaktionen zwischen den Modellelementen ähnlich, wie bei einem Kollaborationsdiagramm, jedoch in Kollaborationsdiagramme stehen die Beziehungen der einzelnen Objekte und deren Topologie im Vordergrund, während beim Sequenzdiagramm der zeitliche Verlauf des Nachrichtenaustausches im Vordergrund steht.

Zustandsdiagramm
Ein Objekt kann in seinem Leben verschiedenartige Zustände annehmen. Mit Hilfe des Zustandsdiagrammes visualisiert man diese, sowie Funktionen, die zu Zustandsänderungen des Objektes führen. Ein Zustandsdiagramm beschreibt eine hypothetische Maschine, die sich zu jedem Zeitpunkt in einer Menge endlicher Zustände befindet. Sie besteht aus:

- einem Anfangszustand
- einer endlichen Menge von Zuständen
- einer endlichen Menge von Ereignissen
- einer endlichen Anzahl von Transitionen, die den Übergang des Objektes von einem zum nächsten Zustand beschreiben
- einem oder mehreren Endzuständen

2.2 Entwicklung von Business Applikationen nach dem J2EE Standard

Einleitung
Java 2 Platform Enterprise Edition (J2EE) definiert die Schnittstellen und das Verhalten eines auf Java basierenden Applikationsservers. Sie definiert die Dienste, die eine J2EE-Plattform zur Verfügung stellen muss, die Schnittstellen und das Verhalten, das die Applikationskomponenten implementieren müssen, damit sie von der Plattform verwaltet werden können, und sie definiert Schnittstellen, damit vorhandene Standardkomponenten, wie zum Beispiel Datenbanksysteme oder Verzeichnisdienste, in die Plattform eingebunden werden können.

Übersicht
Da im Seminarthema die Entwicklung nach J2EE im Vordergrund steht, wird erst der grundlegende Aufbau einer J2EE Plattform beschrieben. Insbesondere wird der Zusammenhang zwischen den Containern - Web-Container, EJB-Container und Application-Client-Container, ihren Komponenten - Bean, Servlet, Client-Applikation - und den Diensten, die die Container zur Verfügung stellen, dargestellt. Dabei werden auch die drei

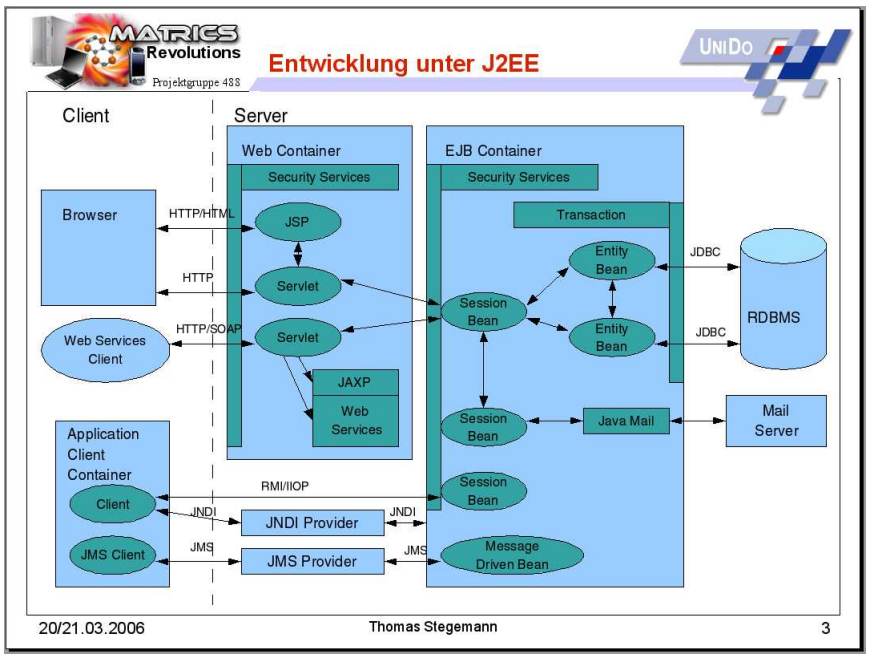

Abbildung 2.1: J2EE, Übersicht

Schichten einer J2EE Applikation erklärt - Darstellungsschicht, Anwendungsschicht und Datenhaltungsschicht.

Dienste

Danach wird kurz die Verwendung des Dienstes JNDI (Java Naming and Directory Interface) beschrieben. Dieser Dienst wird im Seminarthema Grundlagen von LDAP noch angesprochen.

Die Verwendung des Dienstes JMS (Java Messaging Service) wird genauer erklärt. Insbesondere wird auf die Art der Verbindungen - Point-To-Point oder Publish Subscribe, die Art der Nachrichten - Text, Binary, Map, ... - und auch die Möglichkeit Transaktionen zu verwenden, bzw. Nachrichten über ihre Eigenschaften zu Filtern eingegangen.

Komponenten

Auch die Komponenten des Web-Containers - die Servlets - und die Komponenten des EJB-Containers - Beans - werden noch beschrieben. Bei den Servlets wird beschrieben, wie sie Anfragen erhalten und Antworten erstellen können, aber auch wie sie Daten zwischen zwei Anfragen oder zwischen zwei Sessions speichern sollten.

Bei den Beans werden die verschiedenen Typen beschrieben - Session Beans, Entity Beans und Message Driven Beans. Für alle Beans werden ihre möglichen Zustände und die Schnittstelle, die sie implementieren müssen, erläutert. Für die Session Beans und Entity Beans wird auch der Zugriff über RMI (Remote Method Invocation) erläutert.

2.3 SOA Entwicklung von Web-Services nach dem Modell WSDL

Dieses Seminarthema behandelt Web-Services, die im Zusammenhang mit Software oriented Architecture (SOA) genutzt werden.
Unter Web-Services versteht man Softwarekomponenten, die auf verschiedenen Rechnern laufen und über XML- gekapselte Standard-Internetprotokolle und Schnittstellen miteinander mittels XML-Messages kommunizieren und zu einer eigenständigen Anwendung kombiniert werden können. Die Kommunikation zwischen den einzelnen Softwarekomponenten erfolgt über ein Netzwerk, vor allem übers Internet und basiert auf den drei Standards WSDL, SOAP (oder XML-RPC) und UDDI, die jeweils auf XML basieren. Web-Services werden mit einem Uniform Resource Identifier (URI) eindeutig identifiziert.
Im diesem Seminarthema wird ausführlich auf die Kapitel ”Web-Services”, ”SOA”, ”WSDL” und ”SOAP” eingegangen. Eine Service orientierte Architektur (SOA) verteilter Systeme kann mit folgenden Eigenschaften charakterisiert werden: die Services stellen abstrakte, logische Sichten auf konkrete Programme, Datenbanken und Prozesse dar; die Services sind formal durch Nachrichtenaustausch zwischen Anbietern und Empfängern von Diensten definiert; die interne Struktur der einzelnen Komponenten wird abstrahiert, so dass man die Wahl der Programmiersprache und Protokolle offen lässt. Die interne Struktur der Komponenten bleibt verborgen. Eine Service orientierte Architektur kann ganz praktisch im Zusammenhang mit Web-Services zum Einsatz kommen. Die SOA ist für eine bestimmte Anwendung konzipiert und Web Services sollten ausschließlich innerhalb dieser Anwendung laufen. Innerhalb dieser Architektur existieren Service-Anfrager und Service-Anbieter. Services werden implementiert und von Service- Anbietern publiziert und können später von Service-Anfragern entdeckt und aufgerufen werden.
WSDL (Web Services Description Language) ist eine Beschreibungssprache für Web- Services, die auf XML-Format basiert. Eine WSDL-Datei stellt für ein Web-Service seine Schnittstelle und seine abstrakte Beschreibung dar. Somit kann ein Web-Service in mehreren Hochsprachen realisiert werden. Wenn ein Client auf ein Web-Service zugreift, dann liest er seine WSDL-Datei und stellt anhand dieser seine Anfragen an den Web-Service.
SOAP (Simple Object Access Protokoll) ist ein XML-basiertes Protokoll zum Austausch von XML-Nachrichten zwischen Web-Services über ein Netzwerk. SOAP kann mit verschiedenen Transportprotokollen verwendet werden, meistens wird aber HTTP gewählt.Das SOAP Protokoll konvertiert die Anfragen von Clients und Rückmeldungen von Web-Services in XML-Nachrichten und gleichzeitig konvertiert SOAP XML-Nachrichten bei Endbenutzern in Daten.
Der große Vorteil von Web Services liegt in offenen Standards, für die keine Lizenzkosten anfallen. Die Entwicklung von Web Services kann somit auch leichter erlernt werden. Dank offenen Standards können Web-Services unabhängig von der jeweiligen Plattform, Programmiersprachen und Protokollen implementiert werden. Bei der Performance von Web- Services kann man auch einige Schwierigkeiten bekommen. Diese wird durch XML,

Parsen und Dateigrößen verlangsamt. Bei komplexen verteilten Anwendungen ist auch mit hohem Verwaltungsaufwand und zusätzlichem Overhead zu rechnen.

2.4 Konzepte von SSH

Bei diesem Seminarthema ging es um SSH (Secure Shell). Es wurde erklärt, worum es sich bei SSH eigentlich handelt, also wozu es dient, wie es das bewerkstelligt und wie man es handhabt. Von Interesse war dieses Thema, da zum einen in der MaTRICS ConfigClients über SSH-gestützte Verbindungen kommunizieren, zum anderen auf die Applikationsserver am Lehrstuhl über SSH zugegriffen werden kann.

Dazu wurde der Protokollaufbau und -ablauf erklärt, indem die zugrundeliegende Verschlüsselung nach dem Seminarthema 'Kryptographie' aufgefrischt wurde. Anschließend wurde, da jeder SSH-User hier mit dem Protokoll in direkten Kontakt kommt, besprochen, wie man sich gegenüber einem SSH-Server authentifiziert. Zu guter Letzt wurde in diesem Abschnitt noch erwähnt aus welchen Schichten das Protokoll aufgebaut ist.

Um begreiflich zu machen, was man denn mit SSH machen kann, wurden die enthaltenen Tools wie die X11-Umleitung oder sftp angesprochen und erklärt.

Damit ersichtlicht wurde wie SSH in der Praxis in eigenen Projekten zu verwenden ist, wurde anhand von wenigen Code-Zeilen die Initialisierung einer Verbindung, eine notwendige Authentifizierung dem Server gegenüber, und die Benutzung eines Session Channels auf Basis von Java mit j2SSH geboten.

Bevor in einer kleinen Demo die Authentifizierung und der Fernzugriff auf einen anderen Rechner über SSH mit zwei Laptops gezeigt wurde besprachen wir noch die Konfiguration von SSH durch die notwendigen Dateien wie z.B. /etc/ssh/ssh_config.

2.5 Technologie von Java Server Faces(JSF)

Java Server Faces ist ein, basierend auf den Servlet und JSP Technologien, Framework, das für die Entwicklung von Webapplikationen benutzt wird. JSF arbeitet ereignisgesteuert und komponentenbasiert. Die Erzeugung und Verwendung von Benutzeroberflächen (User Interfaces) werden dabei stark vereinfacht. Der Transfer von Daten zwischen Komponenten und die Navigation zwischen Seiten werden unkomplizierter.
Am Anfang der Seminararbeit wird Java Server Faces vorgestellt, dann wird erklärt, warum man mit dieser neuen Technologie arbeiten sollte und danach werden die Vorteile aufgezählt.
Im nächsten Kapitel wird die Model-View-Controller (MVC) Architektur präsentiert. Es wird gezeigt, dass JSF genau auf dieser Architektur basiert. Durch ein flexibles Design können nämlich spätere Änderungen und Erweiterungen leicht durchgeführt werden. Die Trennung der Schichten bringt Übersicht und Ordnung mit sich.

Im dritten Kapitel wird auf den Aufbau von Java Server Faces eingegangen, dabei werden die unterschiedichen Komponenten erläutert: das User-Interface-Komponenten-Modell (UI-Komponentenklassen, Rendering-Modell, Event- und Listener-Modell, Konvertierung und Validierung), die Managed-Beans (Definition und Verwendung) und die Navigation (statisch und dynamisch).
Java Server Faces benutzt ein Request-Response basiertes Verarbeitungsmodell mit mehreren Verarbeitungsphasen. Wenn eine Anfrage beim Server ankommt, verarbeitet JSF diese, dem Request-Lebenszyklus nach, in sechs Phasen. Mit dem Lebenszyklus wird hier erklärt wie ein Zusammenhang zwischen den Komponenten besteht.
Zum Schluss wird ein Beispiel einer Applikation mit JSF demonstriert. Dabei werden Implementierung und Konfiguration erläutert und schließlich die Ergebnisse der Ausführung gezeigt.

2.6 Kommunikation auf Basis von Bluetooth in Java

Bluetooth ist eine Funktechnologie zur kabellosen Übertragung von Sprache und Daten über kurze Entfernungen. Es wurde hauptsächlich entwickelt um verschiedene Geräte unterschiedlicher Hersteller einheitlich miteinander verbinden zu können. So kann man z. B. (bluetooth-fähige Geräte vorausgesetzt) mit einer Digitalkamera ohne Verkabelung Bilder zu einem Drucker schicken oder per Headset kabellos über ein Handy telefonieren. Um eine problemlose Kommunikation zwischen den Geräten sicherzustellen, wurden sogenannte Profile definiert, die bestimmte Anforderungen festlegen. Geräte, die erfolgreich eine Verbindung herstellen möchte, müssen jeweils das gleiche Profil unterstützen.

In J2ME ist die Bluetooth-Unterstützung (JSR-82) bereits von Haus aus integriert, für die Standard Edition muss man sich zunächst auf die Suche nach einem passenden Paket begeben. Die für Linux kostenlose JSR-82 Implementierung avetanaBluetooth ist zwar noch nicht vollständig, erfüllt aber alle Anforderung im Zusammenhang mit der MaTRICS. Mit ihrer Hilfe lässt sich auf einem PC mit Bluetooth-Stick nun per Java nach bluetoothfähigen Geräten in der Umgebung suchen. Einmal gefunden, kann man anschließend die verfügbaren Dienste der Gegenstelle abfragen und gegebenenfalls eine Verbindung herstellen. Der PC lässt sich außerdem als Server konfigurieren, so dass er selbst Dienste anbietet, zu denen sich entfernte Geräte verbinden können. Im Seminar wurde in diesem Zusammenhang gezeigt, wie zwei Handys über eine Java Anwendung eine Verbindung per Bluetooth herstellen und über diese anschließend einen Text von einem Gerät zum anderen senden können.

2.7 Datenbanken in Java mit Hibernate und JDBC

Da innerhalb der MaTRICS zur Speicherung von Daten auch Datenbanken eingesetzt werden, wird in dieser Seminararbeit ein Überblick über Datenbanksysteme vorgestellt. Zuerst

werden kurz Grundlagen und Konzepte von relationalen und objektrelationalen Datenbanken erklärt. Danach wird das frei verfügbare objektrelationale Datenbankmanagementsystem (ORDBMS) PostgreSQL vorgestellt, welches wie gefordert den SQL99 Standard unterstützt und viele Erweiterungen bietet. Die ganzen Leistungsmerkmale von PostgreSQL werden aufgezählt, dessen Serveranwendung ”postmaster” sowie dessen einfacher Datenbankmonitor ”psql” erklärt.
Im nächsten Kapitel wird auf die Query-Sprache SQL eingegangen. Die relationale Algebra und das relationale Datenmodell, die Grundlage von SQL wird dabei kurz erläutert. Die wichtigsten SQL-Befehle werden an Hand von konkreten Beispielen vorgestellt.
In den beiden folgenden Kapiteln werden die zwei Datenbankschnittstellen JDBC und Hibernate jeweils kurz beschrieben. JDBC ist eine Standard-Datenbankschnittstelle, welche in Java integriert ist. Hibernate ist eine kostenlose objektorientierte Datenbankschnittstelle in Java unter Verwendung von JDBC. Jede Zeile einer Tabelle entspricht dabei eine Instanz einer POJO-Klasse und umgekehrt. Die Tabelle selbst wird dagegen durch eine XML-Datei definiert. Die Anbindung der einzelnen Tabellen wird über eine XML-Konfigurationsdatei durchgeführt. Am Ende wird dann demonstriert, wie eine Datenbankanbindung mit Hilfe der beiden Datenbankschnittstellen und PostgreSQL jeweils implementiert und ausgeführt werden kann.
Diese kurze Seminararbeit ist jedoch nur ein Einblick und keine Einführung in Datenbanken, um in der MaTRICS mit einer ähnlichen Datenbankschnittstelle und PostgreSQL arbeiten zu können. Dazu ist eine ausführliche Einführung nötig.

2.8 Grundlagen von LDAP

LDAP ist die Abkürzung für das Lightweight Directory Access Protocol. Wie der Name schon sagt, unterstützt dieses Protokoll einen Verzeichnisdienst (Directory). In vielen Unternehmen und Organisationen haben Verzeichnisse zur Speicherung und effizienten Bereitstellung von Informationen eine strategische Bedeutung. Notwendige Informationen müssen für einen schnellen und leichten Zugriff katalogisiert werden. LDAP ist der vorherrschende Zugriffsmechanismus auf Verzeichnisdienste und wird von allen gängigen freien und kommerziellen Softwarelösungen unterstützt.

In der Seminararbeit wird nach einer kleinen Einführung zunächst auf die Geschichte von LDAP eingegangen und einen Zusammenhang zwischen den X.500 Standards hergestellt. LDAP stellt ein vereinfachtes Protokoll dar, das vorher sehr umfangreich und schwierig in den X.500 Standards spezifiziert wurde. Danach wird noch einmal intensiver auf Verzeichnisdienste eingegangen und erklärt, was Verzeichnisdienste sind und welche Vorteile sie haben.

Anschließend geht es um das X.500 Datenmodell. Hier werden unter anderem die Elemente erläutert, aus denen ein Verzeichnisdienst besteht. Dazu zählen Begriffe wie Objekte, Attribute, Vererbung, Namengebung und Standardisierungen. Schließlich wird kurz auf die Sicherheit in LDAP eingegangen.

Ein weiteres wichtiges Merkmal von LDAP sind Schemas und deren Aufbau. Alle Attribute und Objekte (die man in Objektklassen zusammenfasst) müssen erst durch Schemas bekannt gemacht werden, damit man diese dann in seiner Verzeichnisstruktur benutzen kann.

Mit LDIF schließlich wird ein Textbasiertes Format vorgestellt, mit denen man ganze Verzeichnisstrukturen beschreiben kann. Dadurch lassen sich auf einfache weise solche Verzeichnisstrukturen im- und exportieren.

Danach wird auf JNDI eingegangen. JNDI ist ein Interface für Namens- und Verzeichnisdienste in Java. Zum Schluss wird auf die Installation und Konfiguration von OpenLDAP eingegangen. Außerdem wird eine Beispielanwendung gezeigt.

2.9 Die Entwicklungsumgebung ABC mit EWIS

ABC ist eine Umgebung der Firma MetaFrame und steht für Agent Building Center. Wir benutzen mittlerweile JABC als Umgebung zur Entwicklung von Webapplikationen. JABC ist der Nachfolger von ABC. Da JABC auf ABC aufgebaut und nur eine Weiterentwicklung von ABC ist, wird in dieser Seminararbeit auf die Konzepte, Grundlagen und Funktionen von ABC eingegangen. Dann werden die Grundlagen von EWIS beschrieben. Als Konzepte von ABC werden die Begriffe SIBs (Service Independent Building Blocks), SLG (Service Logic Graph) und Macros erläutert. Danach wird beschrieben, wie ABC gestartet, das Hauptfenster Service Logic Editor geöffnet wird und worauf die Graphen gebaut werden.
Es wird erklärt wie die Parameter und Branches für ein SIB eingestellt werden können. Es gibt verschiedene Inspektoren bei ABC, wie Sugyiama Inspector(für das Layout), Graph Inspector, Node Inspector und Edge Inspector, die kurz erklärt werden. Es werden danach Lokal- und Model Checker vorgestellt.
Im Kapitel 3 der Seminararbeit wird mit einem einfachen Beispiel illustriert, wie Webapplikationen mit ABC/EWIS entwickelt werden. Dazu braucht man einen Webserver und einen Servlet Container . Hier wird der Tomcat Servlet Container benutzt. Es muss dann eine Verzeichnisstruktur erzeugt werden. Danach muss ABC konfiguriert werden, für EWIS Applikationen und für Tomcat. Anschließend wird das Webprojekt zu ABC hinzugefügt. Als Beispiel für die Webapplikation werden zwei einfache HTML Seiten erstellt. Auf der ersten werden zwei beliebigen Zahlen vom User eingegeben. Die Zahlen werden miteinander verglichen und das Ergebnis des Vergleiches auf der zweiten Seite angezeigt. Manchmal werden für die Applikationen SIBs benötigt, die nicht existieren. Diese müssen dann implementiert und erstellt werden. Für unser Beispiel brauchen wir ein SIB, dass die eingegebenen Zahlen miteinander vergleicht und das Resultat in einer HTML Seite anzeigt. Nachdem der Service vollständig definiert ist, kann die Applikation kompiliert, installiert und gestartet werden. Dafür müssen zuerst Compiler und Webserver konfiguriert werden. Nachdem die Applikation installiert ist, starten wir Tomcat und anschließend die Applikation.

2.10 Die neue Entwicklungsumgebung jABC mit jEWIS

Wie das ABC, so ist auch das JavaABC eine Entwicklungsumgebung zur Modellierung und Konstruktion eines beliebigen Softwaresystems. Der Anwender geht bei der Entwicklung graphenorientiert vor, indem er sogenannte SIBs (Service Independent Building Blocks) graphisch miteinander in Verbindung setzt. Der nach diesem Muster erstellte Graph stellt dann letztendlich das Softwaresystem dar. Solange dem Anwender alle für seine Arbeit erforderlichen SIBs vorliegen, kann er mit dem jABC prinzipiell ein Softwaresystem ohne Kenntnisse einer Programmiersprache erstellen.
Die Benutzeroberfläche des jABC besteht dabei im wesentlichen aus 3 Teilen: Dem Browserpanel, in welchem aktuelle Projekte und zur Verfügung stehende SIBs angezeigt werden, dem Inspektorenpanel, welcher es dem Benutzer erlaubt, Parameter des aktuellen Graphen oder eines ausgewählen SIBs zu editieren, sowie der Graphzeichenfläche, auf der der eigentliche Graph selber erstellt wird.
Eine wichtige Eigenschaft des jABC ist dessen modularer Aufbau, wodurch es leicht durch Plugins erweitert werden kann. Plugins können hierbei neue SIBs, neue Semantiken oder auch einfach nur Verbesserungen und Erweiterungen der Benutzeroberfläche enthalten. Von besonderer Wichtigkeit für die PG488 ist dabei das jEWIS, welches es dem Anwender erlaubt, graphenorientiert Webapplikationen zu erstellen.

2.11 Bash-Programmierung unter Linux

Da die MaTRICS zum grossen Teil auf Linux Systemen arbeitet, wurde in diesem Seminarthema unter anderem der grundlegende Umgang mit Linux und damit verbunden der Bash als ein Kommandozeileninterpreter erklärt. Der Hauptteil der Seminararbeit liegt allerdings auf der Automatisierung von Befehlsabläufen innerhalb der Bash auch bekannt als Bash-Programmierung. Es gab eine kurze Einführung in einfache täglich benutzte Befehle und dem Aufbau des Linux-Dateisystems. Ausserdem wurde der Umgang mit speziellen Dateien (welche ”Geräte” oder Sonderfunktionen darstellen) erklärt. Danach wurde die Bash-Programmierung erläutert. Unterkapitel hiervon waren die Definition von Variablen und Auswertung dieser; Programmierung von Schleifen; Realisierung von if-Abfragen; die Definition von Funktionen und das Debugging von Skripten.
Außerdem wurden einige nützliche Programme erklärt und ihre Funktionsweise näher betrachtet. Dazu gehörten u.a. ”sed” um Texte nicht-interaktiv zu editieren; awk – eine Programmiersprache zur Bearbeitung und Auswertung einfacher Textdateien; ”grep” – um nach Mustern in einem Text zu suchen; ”wc”, ”tr” und ”cut” um Text zu schneiden und die Anzahl der Wörter zu zählen sowie ”dialog” und ”whiptail” zur Realisierung von einfachen graphischen Oberflächen.

2.12 Service Availability und Hochverfügbarkeits-Systeme

Da die MaTRCIS unter anderem dazu genutzt werden soll, größere Systeme zu administrieren und damit diese verfügbar zu machen und zu halten, sind natürlich die Themen Service Availability und Hochverfügbarkeits-Systeme sehr wichtig. Dieses Seminarthema sollte genau diese Punkte behandeln und sie näher bringen.
Dazu wurde zuerst einmal erklärt, was der Begriff Service Availability bedeutet und seine Position in der IT-Branche definiert. Grob ist sie ein Maß für die Wahrscheinlichkeit, dass ein System oder Dienst innerhalb eines vorher festgelegten Zeitraums dann auch tatsächlich verfügbar ist. Z.B. ein System, das 365 Tage im Jahr rund um die Uhr läuft, dürfte bei einer Verfügbarkeit von 99%, einen Ausfall von 5256 Minuten oder 87,6 Stunden oder 3,65 Tagen haben. Alle Optionen wie z.B. Verfügbarkeit oder Strafen bei nicht Einhaltung werden im Service Level Agreement (SLA) zwischen Kunde und Dienstleister festgelegt.
Als nächstes wurde erklärt, was ein Hochverfügbarkeits-System ist und welche Konzepte es dafür gibt. Dabei wird ein solches System dadurch definiert, dass es eine sehr geringe Downtime hat. Das Verhältnis zwischen operativer Zeit und dem vereinbarten Zeitraum tendiert damit immer mehr zu 1. Um Verfügbarkeit besser einordnen zu können, wurden entsprechende Klassen entwickelt. Dabei gilt, je höher die Klasse, desto weniger fehleranfällig und höher verfügbar ist das System. In Zahlen heißt das, es stehen immer mehr 9 hinter dem Komma. Also z.B. bedeutet 99,999999% eine sehr hohe Verfügbarkeit. Je höher ein Dienst oder ein System in eine solche Klasse eingestuft wird, desto besser ist seine Verfügbarkeit. Als Konzepte für Hochverfügbarkeits-Systeme wurden kurz Fehlertoleranz, Verfügbarkeitsverbund, Redundanz und Cluster -bzw. Parallel-Computing vorgestellt.
Zuletzt wurden noch ein paar Beispiele für solche Systeme geliefert. So gibt es beispielsweise Parallel Virtual Machine (PVM) und Message Passing Interface (MPI) als Frameworks für parallele und verteilte Anwendungen. Als Cluster-System wurde auf Beowulf bzw. Wolfpack eingegangen. Linux Virtual Server (LVS) wurde als Beispiel für Load Balancer erklärt und danach noch auf RTP Continuous Services als Framework für hochverfügbare, parallele und verteilte Anwendungen im Enterprise -bzw. Telekommunikations-Bereich verwiesen.
Abschließend wurde dann das Fazit gezogen, dass Service Availability und Hochverfügbarkeits-Systeme wichtige Bestandteile der IT-Landschaft sind, denen verstärkt Aufmerksamkeit gewidmet wird.

2.13 Kryptographie und Verschlüsselung in Java

Das Seminarthema beschäftigt sich mit dem Thema Kryptographie und Verschlüsselung in Java. Dabei sollen zum einen Kryptographie und kryptographische Grundlagen allgemein erläutert werden, zum anderen speziell auf die kryptographischen Fähigkeiten und

Möglichkeiten in Java eingegangen werden. Dabei wird zuerst auf die Frage eingegangen, wieso Kryptographie heute eine so wichtige Rolle spielt. An welchen Stellen im Alltag arbeiten wir mit Kryptographie, z.B. Email-, WLAN-Verschlüsselung, usw. und wo benutzen wir selbst Kryptographie, z.B. Online-Banking.
Danach wird auf die kryptographischen Grundlagen eingegangen. Hier werden zuerst wichtige Begriffe für die Kryptographie erläutert und grundlegende Definitionen gemacht, die für das Verständnis der weiteren Kapitel notwendig sind. Es werden allgemeine Methoden, wie symmetrische und asymmetrische Verfahren inklusive ihrer Vor- und Nachteile vorgestellt. Es wird auf die dazugehörigen Algorithmen, z.B. bei den symmetrischen Verfahren auf Substitutionsverschlüsselung, Exklusive-Oder, DES-Verschlüsselung und Triple-DES-Verschlüsselung, sowie bei den asymmetrischen Verfahren auf den RSA-Algorithmus eingegangen. Die Sicherheitsfrage zu den Algorithmen wird erklärt, und Vor- und Nachteile der Verschlüsselungsverfahren werden aufgezeigt. Es folgt eine Beschreibung der Hashfunktionen und der dazugehörigen Verfahren der MDx Reihe.
In Kapitel fünf wird der Aufbau und Umgang mit den Zertifikaten, speziell des X.509 Zertifikats und die entsprechende Benutzung in der Public-Key Infrastruktur gezeigt. Hier wird ebenfalls auf das Konkurrenzverfahren, das Web Of Trust, eingegangen.
In Kapitel sechs folgt eine Beschreibung des SSL Protokolls. Der Aufbau von SSL Verbindungen, die verschiedenen Protokollschichten und der Einsatz in der Praxis werden hier erklärt. Am Ende wird auf die Java Cryptographie Architecture eingegangen. Hier wird gezeigt, wie in Java Algorithmen aufgerufen und registriert werden können, wie Hashfunktionen in Java benutzt werden, wie die Schlüssel für die symmetrischen und asymmetrischen Verschlüsselungsverfahren erstellt werden und wie damit umgegangen wird. Es wird gezeigt, wie digitale Signaturen und „echte“ Zufallszahlen erzeugt werden können. Zum Schluss wird eine Beispielimplementierung mit dem DES-Verschlüsselungsverfahren erstellt und im Einzelnen erläutert.

Kapitel 3

Aufwärmphase

Die Aufwärmphase hatte weniger den Sinn aufwendige und ausgefeilte Projekte abzuliefern, sondern hier sollten sich die Teilnehmer der Projektgruppe vorallem mit dem Umgang der einzelnen Tools vertraut machen, z.B. jABC, jEWIS, Eclipse, vi, tomcat. Die Dauer der Projekte war auf 2 Wochen ausgelegt. Die Teilnehmer sollten hier den Umgang mit dem jABC und dem jEWIS kennenlernen. Dabei sollte vorallem verstanden werden, wie das jEWIS konfiguriert wird, welche verschiedenen Typen von SIBs es gibt, wie die einzelnen SIBs zusammenhängen bzw. wie die Daten durch die einzelnen Kontexte im jABC Graphen geschleust werden. Wichtig war außerdem, wie eine HTML Seite am Anfang des jABC-Graphen erzeugt wird und wie auf die Daten, die in der HTML-Seite eingegeben werden, per Velocity-Engine zugegriffen werden kann. Wie werden die Daten dann in den einzelnen SIBs weiterverarbeitet und wie werden die bearbeiteten Daten am Ende wieder auf der HTML-Seite angezeigt.
Dazu wurden insgesamt sieben Themen vorgestellt, die von jeweils zwei Teilnehmern bearbeitet wurden.

3.1 Kniffel

Die Aufgabe war es mit Hilfe des jABC eine vereinfachte Kniffel-Webapplikation zu erstellen. In dieser sollte es dem Anwender möglich sein zu würfeln, eine gewisse Auswahl an Würfeln zu machen und mit den verbliebenen Würfeln wiederum zu würfeln; die Applikation sollte zudem nach 3 Würfen das Spiel automatisch beenden.
Zur Erfüllung dieser Anforderungen war es notwendig HTML-Seiten, einen neuen eigenen SIB sowie einen jABC-Graph zu erstellen. Abbildung 3.1 zeigt das Design der Kniffel-Webseite. Die Webseite zeigt 5 Würfel mit jeweils einer dazugehörigen Checkbox sowie 2 Buttons, mit deren Hilfe der Benutzer den nächsten Wurf starten oder die Runde mit den aktuell liegenden Würfeln beenden kann. Der Anwender kann mit Hilfe der Checkboxen auswählen, welche Würfel im nächsten Wurf nochmals gewürfelt werden. Oberhalb der Würfel wird zudem angezeigt, wieviele Würfe schon absolviert worden sind. Ferner

überprüft die Applikation zudem, was für ein Bild die Würfel zeigen und teilt dieses dem Benutzer unterhalb der Checkboxen mit. Abbildung 3.2 zeigt den jABC-Graphen der

Abbildung 3.1: Kniffel, Webseite

Kniffel-Applikation. Herzstück des Graphen ist der selbst erstellte randomizeEWIS-SIB, welcher für das Würfeln und das Berechnen des aktuellen Bildes zuständig ist. Dieser SIB besitzt 3 Branches ’done’, ’empty’ und ’toomanyrolls’. Der ’done’ Branch ist der Standardfall und wird verfolgt, falls der Benutzer eine zulässige Eingabe auf der Webseite gemacht hat und noch keine 3 Würfe absolviert worden sind. Der ’empty’ Branch hingegen wird gewählt, falls der Benutzer auf der Webseite keinen Würfel ausgewählt hat; in diesem Fall wird er aufgefordert dies nachzuholen. Hat der Benutzer bereits 3 mal gewürfelt, so wird der ’toomanyrolls’ Branch genommen.
Nachdem der Anwender eine Runde beendet hat, kann er sofort eine weitere starten.

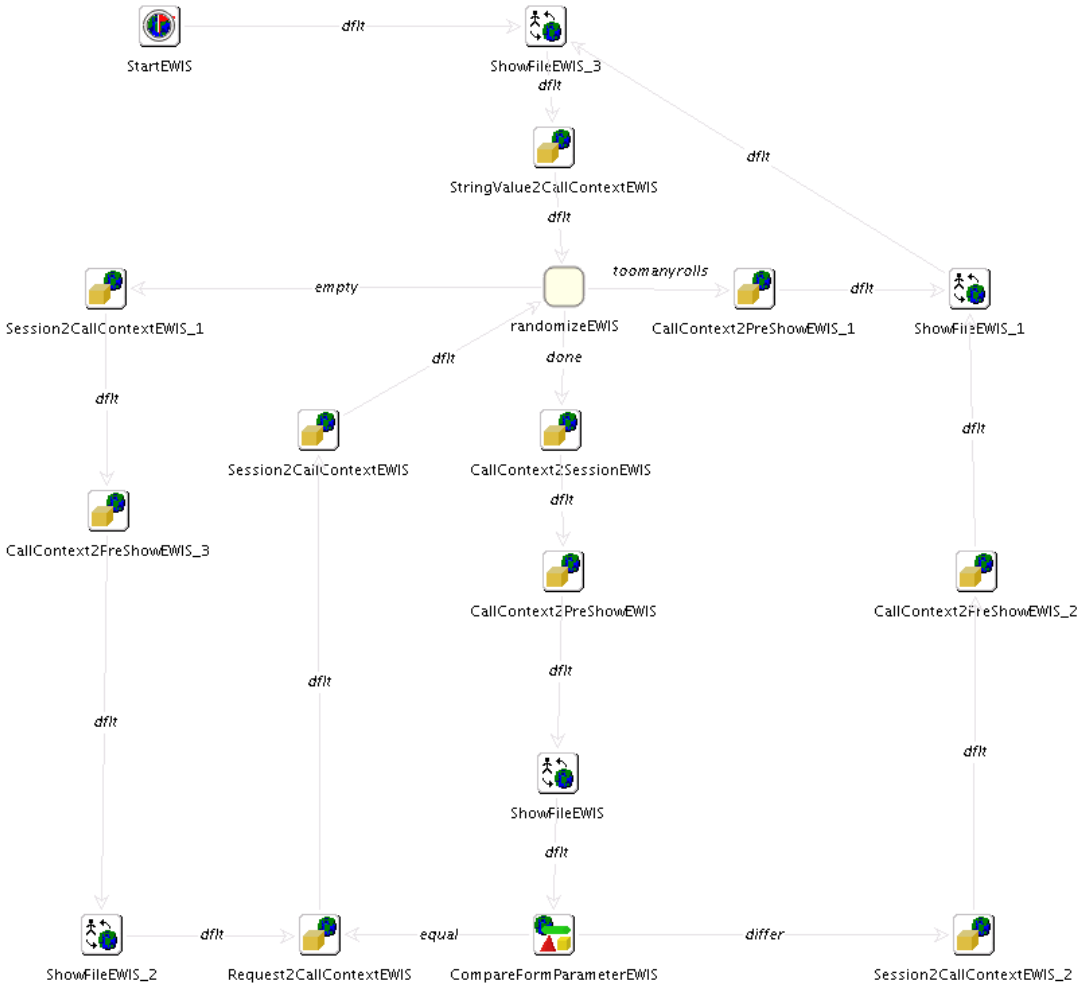

Abbildung 3.2: Kniffel, jABC-Graph

3.2 Organizer

Die Aufgabe war es einen simplen Termin-Organizer zu erstellen, in dem man Termine für einen Tag mit Uhrzeit und Beschreibung eintragen und wieder löschen kann. Zuerst wurde eine HTML-Seite erstellt (siehe Abbildung 3.4), in der man mit Hilfe von Auswahlboxen die Anfangs- und die Endzeit eines Termins auswählen und danach in einer Textbox eine Beschreibung für diesen Termin eingeben kann. Ausserdem wurden mit Hilfe der Velocity-Engine die bisher eingetragenen Termine aufgelistet. Danach wurde der Graph gebaut (siehe Abbildung 3.3), der den logischen Ablauf enthält. Als erstes wird einmalig der Daten-Vektor im InitVecEWIS initialisiert. Danach wird eine Webseite angezeigt, in der die Termine eingegeben werden können. Dann werden die Daten von der

Webseite entgegen genommen und im UpdateVecEWIS in den Daten-Vektor einsortiert, so dass alle Termine in zeitlich korrekter Reihenfolge ausgegeben werden können.

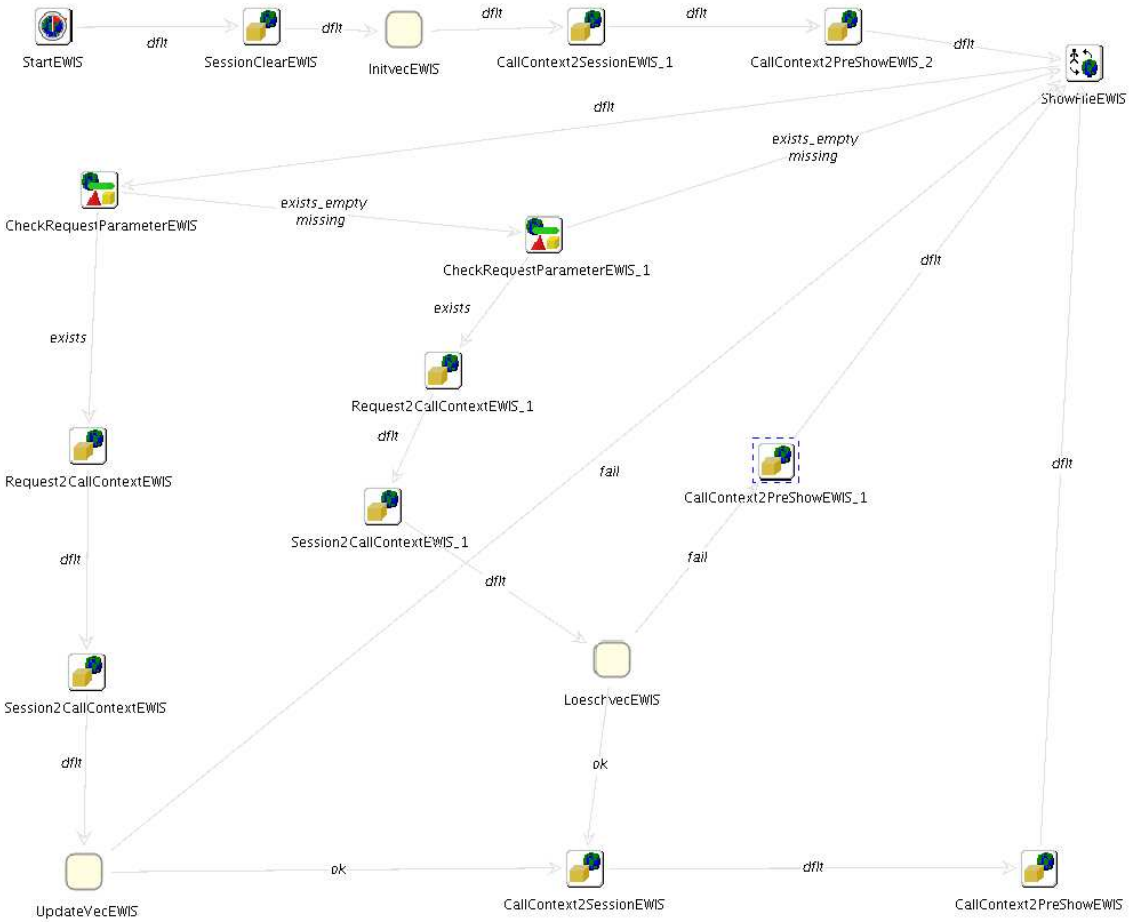

Abbildung 3.3: Organizer, jABC-Graph

Der Vektor wird daraufhin im Container-Context gespeichert, da nicht alle Daten über die Webseite gespeichert werden sollten. Danach wird der Vektor der Webseite zurückübergeben und mit Hilfe der Velocity-Engine angezeigt. Soll ein Termin gelöscht werden muss man den Termin in die Maske eintragen und auf ”löschen” klicken. Daraufhin wird der Vektor mit dem LoescheVecEWIS durchsucht und wenn der Termin gefunden wurde aus dem Vektor gelöscht.

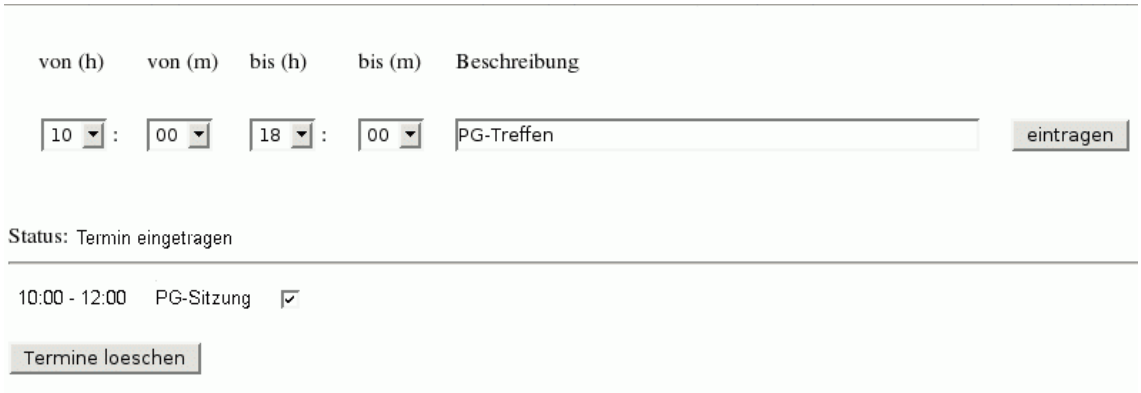

Abbildung 3.4: Organizer, Startseite

3.3 Währungskonverter

In dieser Aufgabe soll eine Webapplikation für einen Währungskonverter erstellt werden. Da nur eine Webapplikation erstellt werden sollte, um die Tools kennenzulernen, wurde der Währungskonverter sehr schlank gehalten. Es wurden dazu HTML-Seiten für die Ein- und Ausgaben sowie für Eingabefehlerausgaben erstellt und ein SIB CalcWaehrung implementiert, der die Umrechnung durchführt. Auf der Startseite kann über Auswahlboxen jeweils die Quell- bzw. Zielwährung ausgewählt werden. Der Betrag kann über ein Textfeld eingegeben werden. Wobei Zahlen mit Punkt oder mit Komma eingegeben werden können. Jedoch muss der Betrag eine Zahl sein. Eine falsche Eingabe wird auf eine HTML-Seite als Eingabefehler deklariert, mit der Aufforderung nur Zahlen einzugeben. Die Wechselkurse sind statisch im CalcWaehrung-SIB eingetragen. Der eingegebene Betrag wird in dem CalcWaehrung-SIB mit dem entsprechenden Wechselkurs multipliziert. Dann wird der umgerechnete Betrag zurückgegeben, der anschließend auf einer HTML-Seite angezeigt wird. Von dieser Ergebnis-Seite kann der Benutzer wieder zur Startseite zurückkehren und einen neuen Betrag umrechnen lassen. In Abbildung 3.5 ist der entsprechende Graph für den Währungskonverter zu sehen. Beginnend vom Startknoten geht es zu einer HTML-Seite, in welcher der Betrag und die umzurechnenden Währungen eingetragen bzw. ausgewählt werden. Man kommt auf die Seite mit der Fehlermeldung, wenn man nichts oder keine Zahl eingetragen hat. Die Werte werden übernommen und in den CallContext gespeichert. Aus dem CallContext holt sich der SIB CalcWaehrung die benötigten Parameter. Es wird überprüft, ob die Eingabe eine Zahl ist oder nicht, wenn ja dann wird die Umrechnung durchgeführt. Abbildung 3.6 zeigt einen Screenshot von unserem einfachen Währungskonverter.

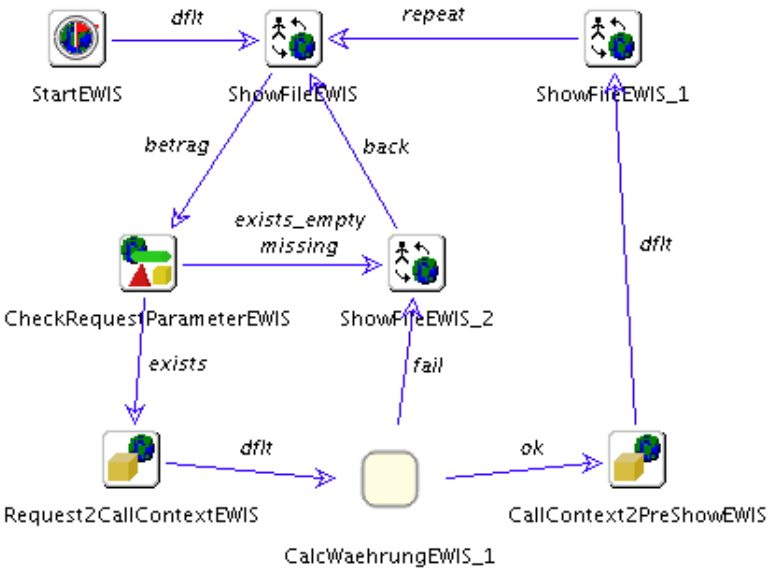

Abbildung 3.5: Währungskonverter, jABC-Graph

Abbildung 3.6: Währungskonverter, Startseite

3.4 Schiffe versenken

Bei der einfachen Variante des Spiels Schiffe versenken muss der Spieler die zufällig verteilten Schiffe auf dem Spielfeld finden.
In Abbildung 3.7 kann man den SIB-Graph des Spiels mit den beiden neuen SIBs BattleShipsinit und BattleShipsplay sehen.
Zu Beginn wird die Startseite angezeigt (ShowBattleShips_Start). Hier wählt der Spieler, die Größe des Spielfelds, die Anzahl der Schiffe und die Anzahl der Schüsse aus (Abbildung 3.8).
Diese Einstellungen werden im SIB BattleShipsinit ausgewertet. Kann damit kein gültiges Spielfeld erzeugt werden, muss der Spieler die Einstellungen korrigieren (ShowBattleShips_StartError), sind die Einstellungen gültig, wird das Spielfeld angezeigt (ShowBattleShips_Result). Auf dem Spielfeld kann der Spieler die Felder auswählen, auf denen er ein Schiff vermutet. Schickt er seine Auswahl ab, wird diese im SIB BattleShipsplay ausgewertet, und es wird wieder das Spielfeld angezeigt (ShowBattleShips_Result). Der

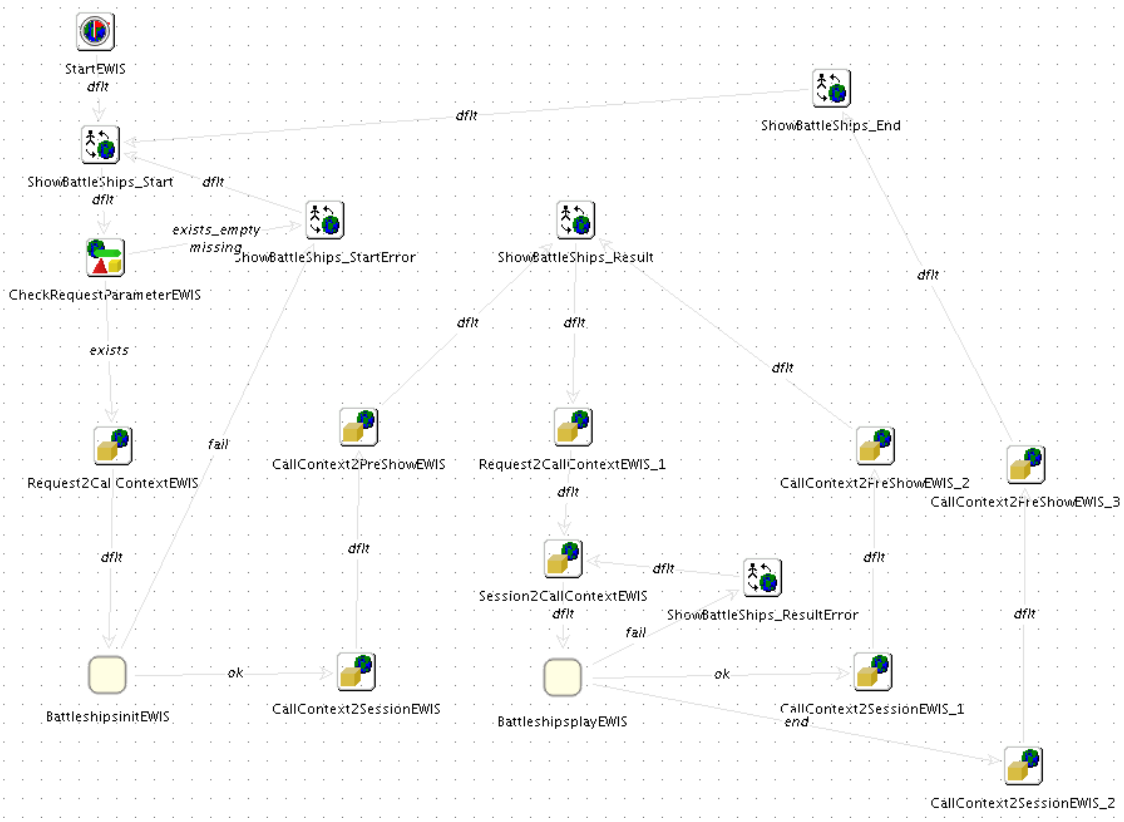

Abbildung 3.7: Schiffe Versenken, jABC-Graph

Spieler erhält hier eine Rückmeldung über die Treffer und Fehlschläge (Abbildung 3.9).
Hat der Spieler alle Schiffe gefunden, oder keinen Versuch mehr frei, wählt der SIB

Abbildung 3.8: Schiffe Versenken, Start

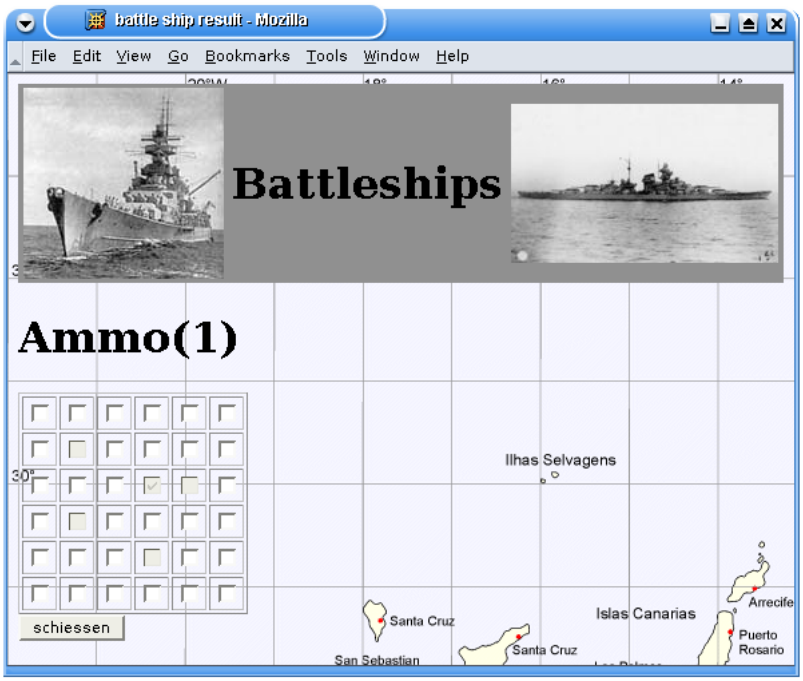

Abbildung 3.9: Schiffe Versenken, Spiel

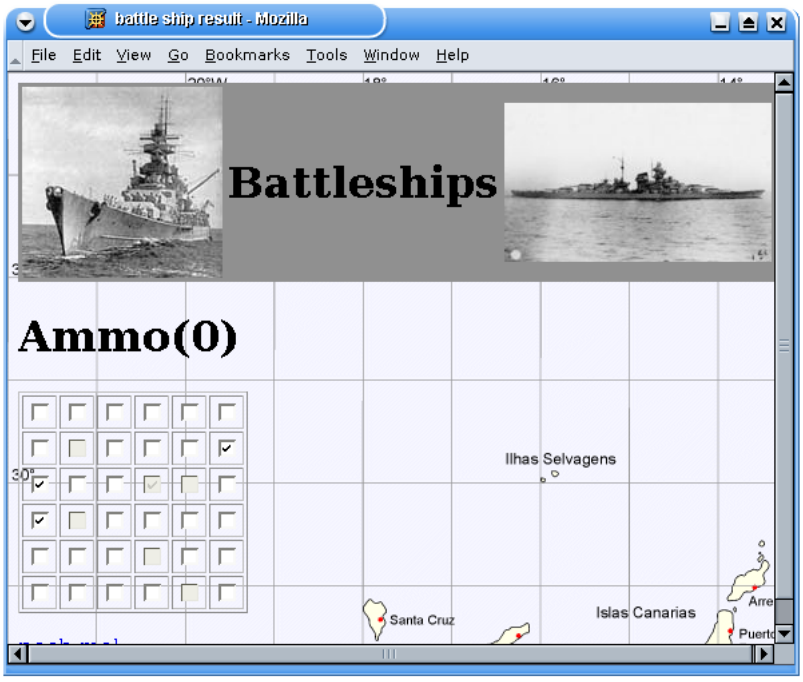

Abbildung 3.10: Schiffe Versenken, Ergebnis

BattleShipsplay den Branch end und das Spielfeld mit dem Endergebnis wird angezeigt (ShowBattleShips_End). Hier werden dem Spieler alle Treffer, alle Fehlschläge und alle nicht getroffenen Schiffe angezeigt (Abbildung 3.10). Danach kann der Spieler das Spiel neu starten (Show BattleShips_Start).

3.5 Taschenrechner

Die Aufgabe bestand darin, einen einfachen Taschenrechner zu erstellen. Man hat zwei Eingabefelder für die Zahlen und kann dann eine von vier Operationen auswählen.

In der Abbildung 3.11 sieht man die Hauptseite des Taschenrechners.

Die Realisierung in jABC ist in Abbildung 3.12 dargestellt.

Abbildung 3.11: Taschenrechner, Hauptseite

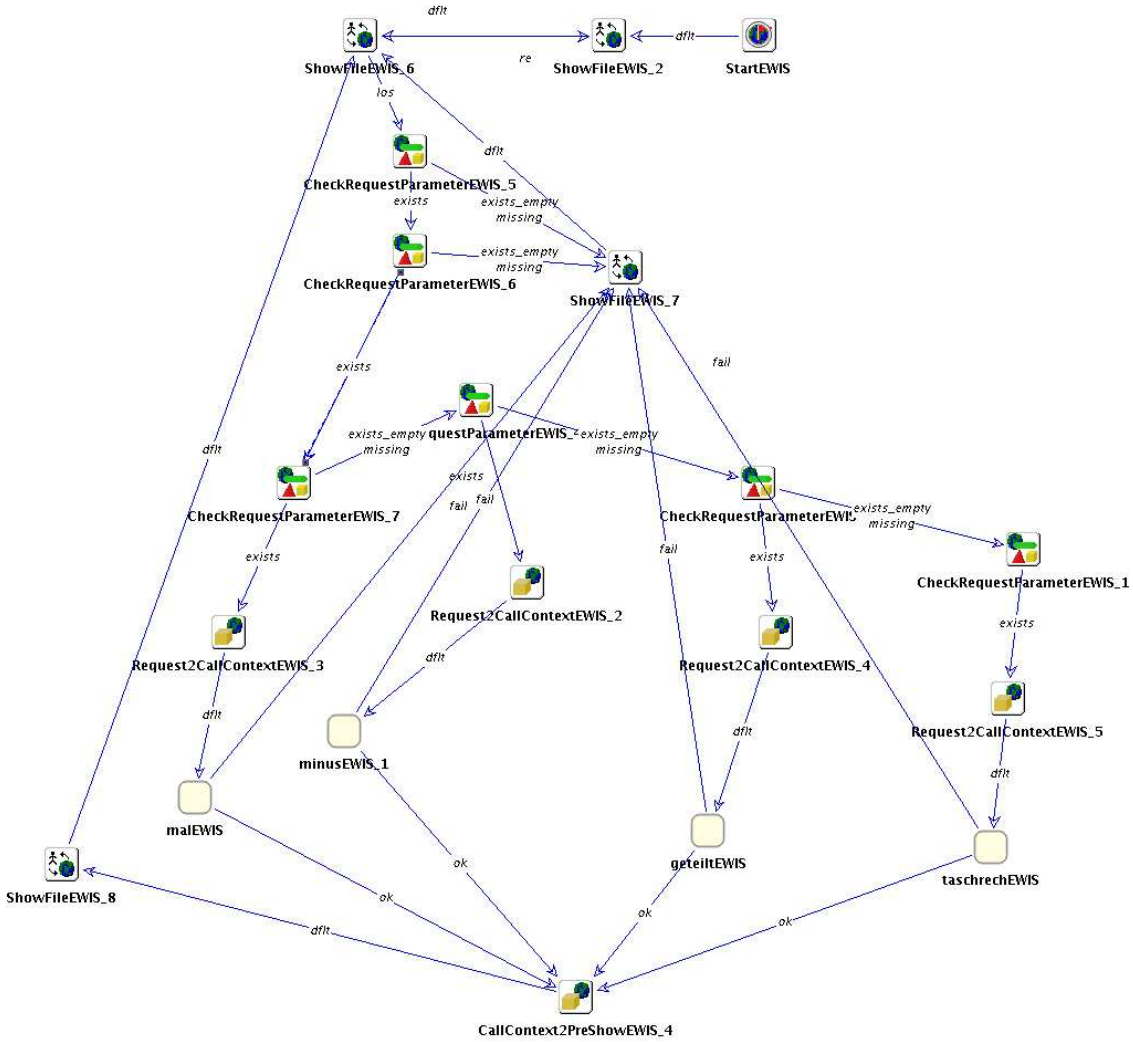

Abbildung 3.12: Taschenrechner, jABC-Graph

Der Start-SIB zeigt auf die Anfangsseite (ShowFileEWIS_2). Dort befindet sich ein Link, der auf die Hauptseite (ShowFileEWIS_6) des Taschenrechners zeigt (Abbildung 3.11).

Hat man nun in der Hauptseite die Eingaben eingegeben und auf eine Operation geklickt (los-branch von ShowFileEWIS_6), wird als erstes überprüft, ob beide Zahlen vorhanden sind. Ist dies nicht der Fall, gelangt man sofort zur Fehlerseite des Projektes (ShowFileEWIS_7).

Sind jedoch beide Eingabefelder gefüllt, wird im zweiten Schritt überprüft, welche der vier Operationen angeklickt wurde. Danach werden im SIB Request2CallContext die beiden Zahlen und die Operation in den CallContext gespeichert. Als nächstes gelangt man zu einer der vier Rechen-SIBs (malSIB, minusSIB, geteiltSIB und taschrechSIB) für die Multiplikation, Subtraktion, Division und Addition.

In diesen vier Rechen-SIBs wird überprüft, ob die Eingaben korrekt sind. Hat jemand z.B. Buchstaben statt Zahlen eingegeben, gelangt man von den Rechen-SIBs zu der Fehlerseite (ShowFileEWIS_7).

Sind die Eingaben korrekt, gelangt man zum SIB CallContext2PreShowEWIS_4. Dieser SIB speichert alle Informationen vom CallContext in den PreShow. Im CallContext befinden sich zu dieser Zeit die beiden Zahleneingaben, das Ergebnis und die ausgewählte Operation. Diese Informationen werden dann auf der Ergebnisseite angezeigt (ShowFileEWIS_8). Von dort gibt es einen Link, der wieder auf den Anfang verweist (Hauptseite Abbildung 3.11).

3.6 Kalender

Um sich mit den Tools wie *Eclipse* oder *jABC* vertraut zu machen, soll hier eine kleine Webapplikation erstellt werden. In diesem Fall wird ein Kalender implementiert: Auf einer HTML-Seite werden das Jahr, der Monat und die Tage des Monats angezeigt. Es besteht die Möglichkeit den nächsten oder den vorherigen Monat anzuklicken.
Die Anwendung wird mit einem SIB-Graph unter jABC gebaut. Ein neuer SIB wurde hierfür implementiert (BerechneKalenderEWIS), er berechnet für den Kalender die benötigten Daten. Die Abbildung 3.13 zeigt den jABC-Graph für die Webapplikation.

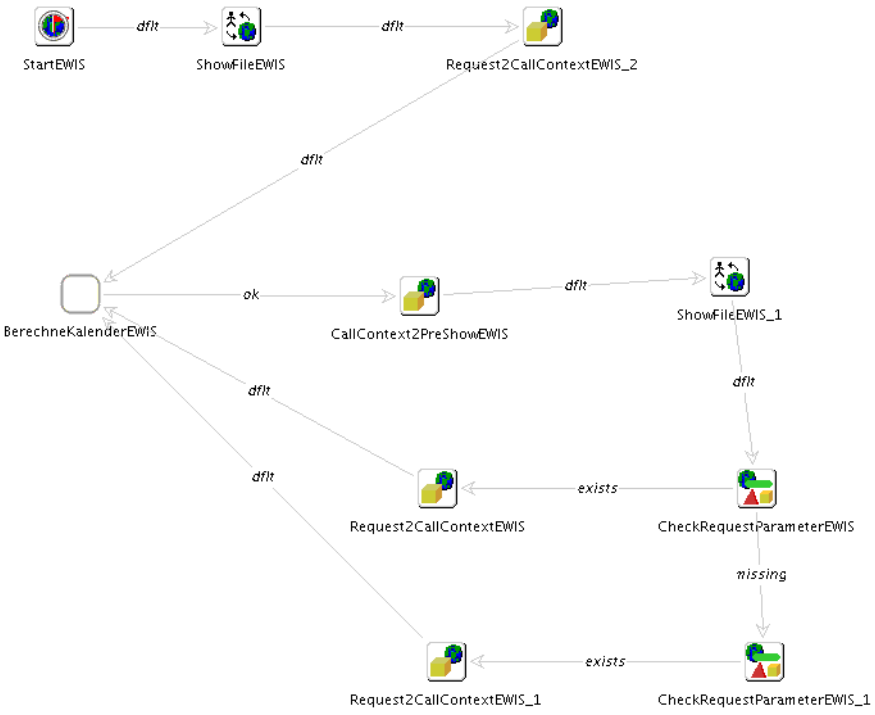

Abbildung 3.13: Kalender, jABC-Graph

Wenn man auf der ersten Seite auf *weiter zum Kalender* klickt, wird der SIB *BerechneKalenderEWIS* aufgerufen. Dieser SIB berechnet das aktuelle Jahr, den aktuellen Monat und die entsprechenden Tage. Die berechneten Daten werden auf der nächsten Seite angezeigt. Man kann das Ergebnis in der Abbildung 3.14 betrachten.

Abbildung 3.14: Kalender, Ergebnis

Oben stehen Monat und Jahr, unten sind die Tage in Wochenzeilen aufgeteilt. Für die Navigation stehen zwei Knöpfe *zurueck* und *weiter* zur Verfügung. Mit den SIBs *CheckRequestParameterEWIS* und *CheckRequestParameterEWIS_1* wird entschieden, ob man auf *zurueck* oder auf *weiter* geklickt hat. Danach berechnet der SIB *BerechneKalenderEWIS* entsprechend der Auswahl den nächsten oder den vorherigen Monat und dieser wird auf der nächsten Seite angezeigt. Wichtig dabei ist es, dass für die Berechnung bei den Monaten Januar und Dezember auch der Jahreswechsel berücksichtigt wird.

3.7 Telefonbuch

Die Aufgabe bestand darin, mit Hilfe des jABC ein einfaches Telefonbuch in Form einer Web-Applikation zu entwickeln. Der Anwender sollte die Möglichkeit haben, seine Telefonnummer und seinen Vor- und Nachnamen einzutragen, zu editieren und zu löschen. Dabei sollten Eingabefehler erkannt und sichtbar zur Neueingabe dargestellt werden.

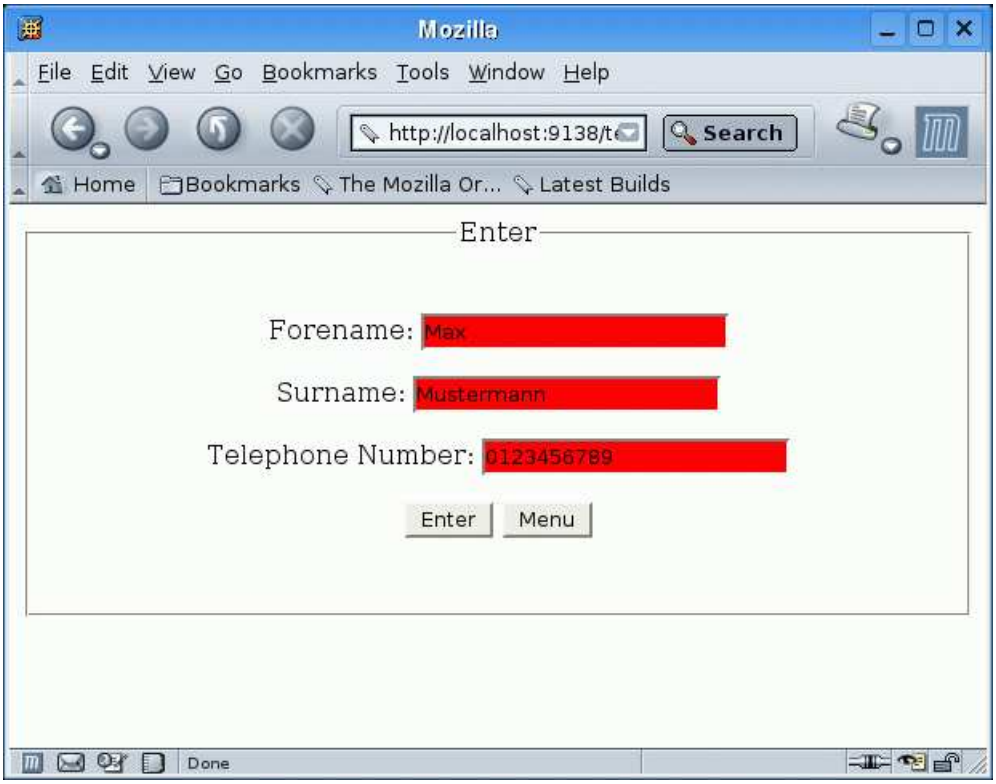

Abbildung 3.15: Telefonbuch, Eingabe

In Abbildung 3.15 sieht man die Eingabemaske. Hierhin gelangt man, wenn man neue Einträge tätigen oder alte Einträge editieren will. Die Werte der Felder werden auf Eingabefehler überprüft und die Eingabefehler farblich dargestellt. So zum Beispiel wenn im Telefonnummernfeld nicht nur Zahlen eingegeben werden.

Abbildung 3.16 zeigt einen exemplarischen Eintrag im Telefonbuch.

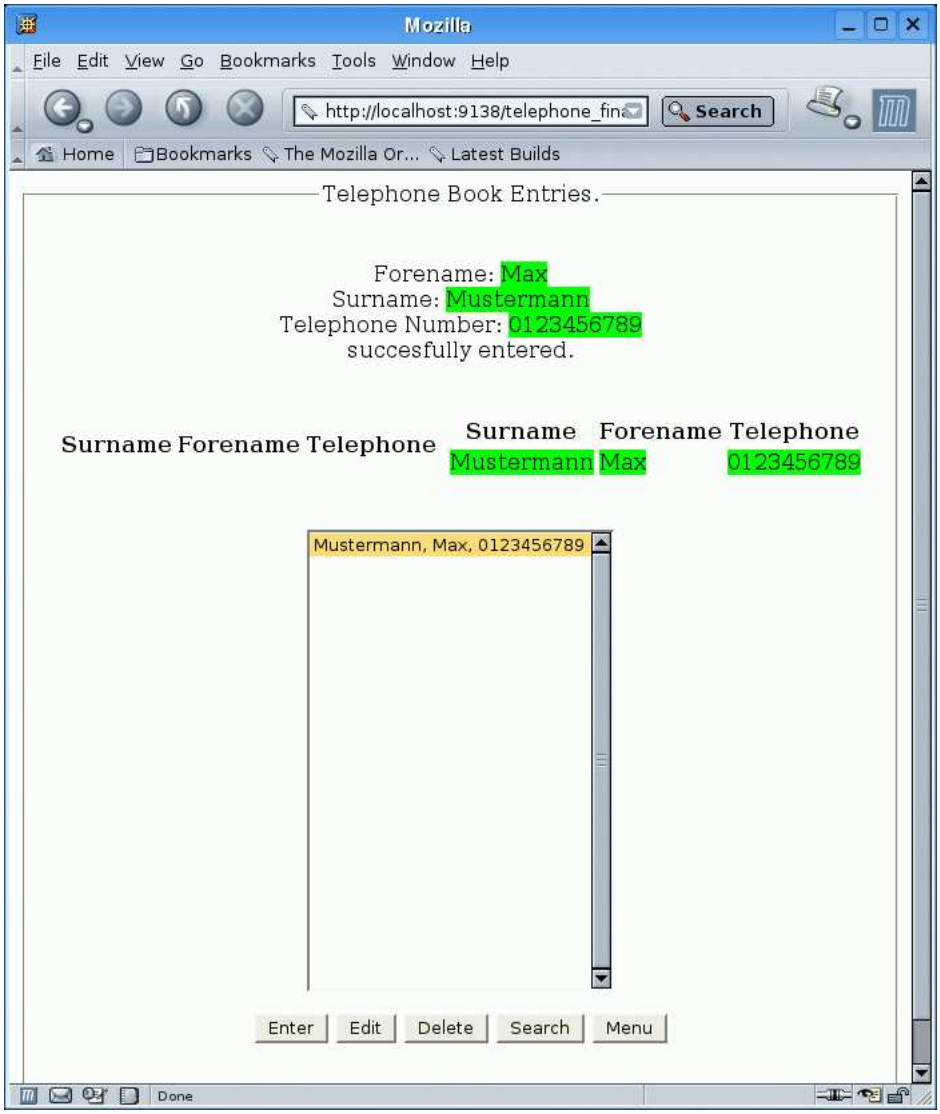

Abbildung 3.16: Telefonbuch, Eintrag

Die zentralen SIBs sind zum einen CheckValuesEWIS, welcher die Eingaben überprüft und entsprechend reagiert und zum anderen TelephoneBookEntryEWIS, welcher den Eintrag in das Telefonbuch vornimmt oder entsprechend reagiert, wenn ein Fehler auftritt z.B. bei einem duplizierten Eintrag.

Kapitel 4

Kurzthemen

Bei den Kurzthemen sollten in dreier bzw. vierer Gruppen jeweils eine SIB-Bibliothek

- zur Kommunikation auf Basis von Bluetooth
- für die Ausführung von BASH-Befehlen
- zur Diagnose von Netzwerken auf Basis von Network-Sockets
- zur Kommunikation und Verwaltung von LDAP

designt und implementiert werden. Dafür hatten wir 6 Wochen Zeit. Zuerst mussten alle das Design mit Hilfe von UML erstellen. Dazu gehörte das Erstellen von Use-Case-Diagrammen, Klassendiagrammen sowie Aktivitätsdiagrammen. Darauf aufbauend wurden dann die SIBs implementiert und getestet. Die Verwendung der einzelnen SIBs wurde an Hand von kleinen Beispiel-SLGs demonstriert. Als Abschluss wurde eine englische Dokumentation erstellt, die das Design der SIB-Bibliothek, eine kurze Beschreibung und die Verwendung der einzelnen SIBs und ein einfach zu verstehendes Beispiel enthielt. Die SIB-Bibliotheken wurden völlig unabhängig von der MaTRICS entwickelt. Kenntnisse über die MaTRICS waren nicht notwendig. Die SIB-Bibliotheken können jedoch problemlos in die MaTRICS integriert werden.

4.1 Design und Implementierung einer SIB Bibliothek zur Kommunikation auf Basis von Bluetooth

4.1.1 Einleitung

Viele mobile Endgeräte bieten die Möglichkeit über Bluetooth zu kommunizieren. Mit einer Bluetooth Bibliothek kann die MaTRICS neben dem webbasierten Zugang, den

Agents auch einen Zugang auf Basis von Bluetooth ermöglichen. Auch auf Seite der Configuration Clients kann mit einer Bluetooth Bibliothek die Kommunikationsschnittstelle erweitert werden, und so können auch mobile Endgeräte mit Hilfe der MaTRICS über Bluetooth konfiguriert werden. Zusätzlich kann die MaTRICS über Bluetooth auch die Dienste von mobilen Endgeräten verwenden, etwa den Versand von SMS.

Das Thema im Rahmen dieser Kurzaufgabe bestand darin eine SIB Palette zu implementieren, die eine Bluetooth-basierte Kommunikation ermöglicht. Hierzu waren insbesondere SIBs zur Entdeckung von Geräten, zur Suche von Services, zum Verbindungsmanagement sowie zur Verwaltung des Datentransfers zu entwickeln. Um alle erforderlichen Funktionen zu realisieren, bedienten wir uns der Avetana Bluetooth API [1] für Linuxsysteme, welche für nicht kommerzielle Zwecke frei zu bekommen ist. Die Avetana API baut dabei auf dem offiziellen BlueZ [3] Bluetooth Stack auf.

4.1.2 Design

4.1.2.1 Anwendungsfalldiagramm

In unserem Anwendungsfalldiagramm (siehe Abbildung 4.1) unterscheiden wir zwischen Anwendungsfällen für Bluetooth Client und Bluetooth Server. Die wichtigsten Anwendungsfälle des Servers sind:

- Stelle Service zur Verfügung
- Verbindung mit Client aufbauen

Der Anwendungsfall “Stelle Service zur Verfügung” initialisiert den Server für die Aufnahme der Verbindung. Anschließend kann der Client sich mit dem Server verbinden. Der Anwendungsfall “Verbindung mit Client aufbauen” beinhaltet alle Aspekte der Kommunikation zwischen Bluetooth Client und Bluetooth Server, so dass der Datenaustausch zwischen ihnen problemlos verlaufen kann. Die folgenden wichtigsten Anwendungsfälle beziehen sich auf den Client:

- Suche nach Geräten
- Suche nach Services
- Verbindung mit dem Server aufbauen

Ein Bluetooth Client kann mit einem Bluetooth Server nur dann kommunizieren, wenn letzterer auf einem bluetoothfähigen Gerät läuft. Also muss der Client zunächst einmal ein bluetoothfähiges Gerät auswählen auf welchem der Server läuft. Dies wird in dem Anwendungsfall “Suche nach Geräten” absolviert. Der Anwendungsfall “Suche nach Services” listet alle Services auf, die auf einem bluetoothfähigen Gerät laufen können. Anschließend kann sich der Client mit dem Server verbinden. Das beschreibt der Anwendungsfall “Verbindung mit dem Server aufbauen”.

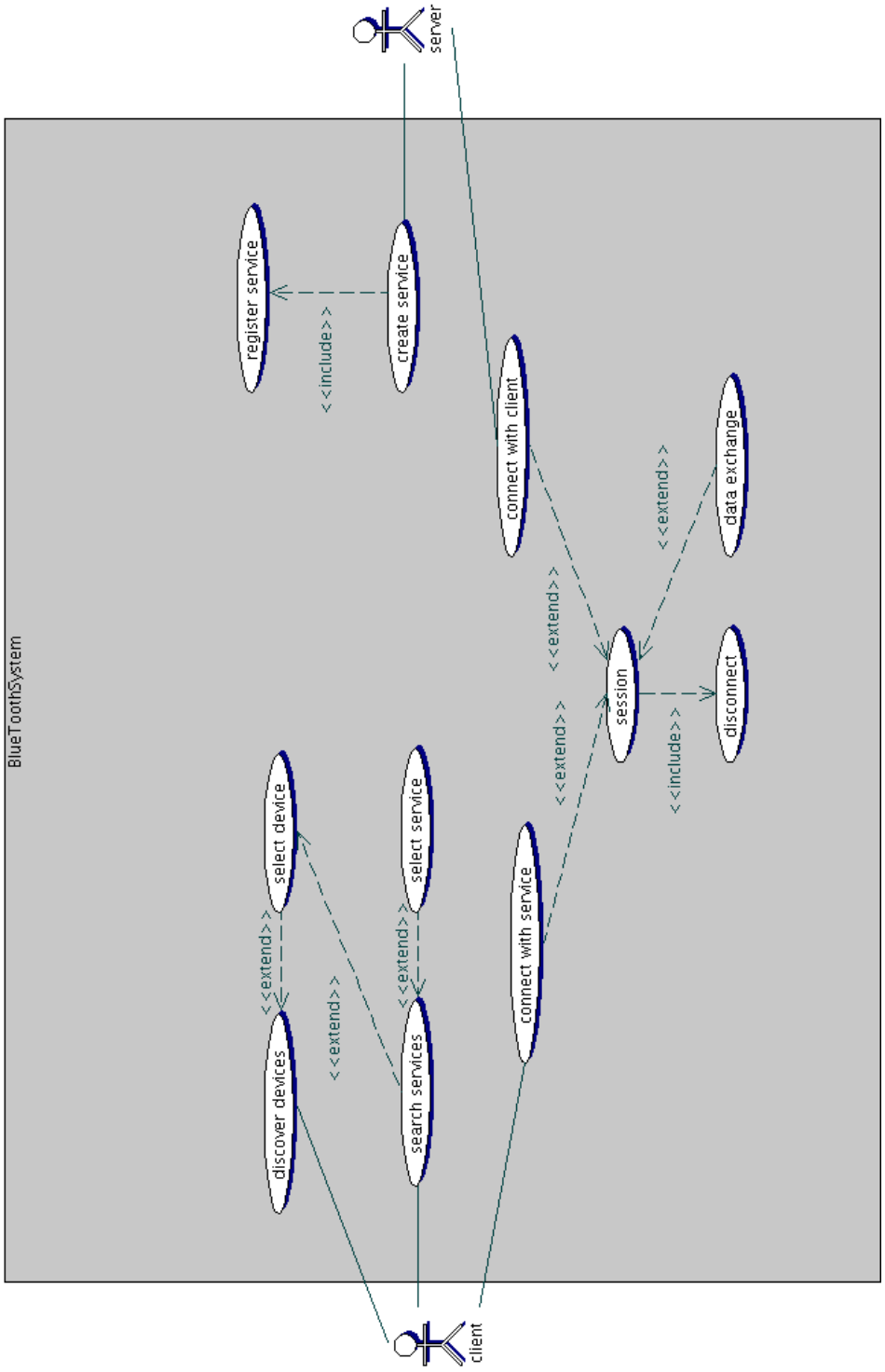

Abbildung 4.1: Bluetooth, Usecase-Diagram

4.1.2.2 Aktivitäts-Diagramm

Das Aktivitätsdiagramm für den Bluetooth Client (siehe Abbildung 4.2) zeigt den Ablauf der einzelnen Aktivitäten in richtiger Reihenfolge, so dass eine erfolgreiche Kommunikation zwischen Bluetooth Client und Server zustande kommen kann.

Zu Beginn muss der Client initialisiert werden. Danach kann der Client alle bluetoothfähigen Geräte finden und eins davon auswählen. Bevor die Verbindung aufgebaut wurde, können auch andere Geräte ausgewählt werden. Nach der Wahl eines geeigneten Services verbindet sich der Client mit dem Server dieses Services, also mit dem Gerät, welches der Bluetooth Server ist. Nun öffnet der Client seine Data Streams und kann im nächsten Schritt mit dem Datenaustausch beginnen. Nach dem Datenaustausch schliesst der Client seine Data Streams und im nächsten Schritt die Verbindung zum Server und dessen Service. Das Diagramm betrachtet außerdem mögliche Fehler, die während einer Verbindung auftreten können. In allen diesen Fällen muss die Verbindung neu initialisiert werden.

Das Aktivitätsdiagramm für den Bluetooth Server(siehe Abbildung 4.3) unterscheidet sich vom Diagramm des Clients nur am Anfang des Ablaufs. Zuerst muss der Server geeignet initialisiert werden. Danach werden Services initialisiert und registriert, so dass der Client auf diese zugreifen kann. In nächstem Schritt können alternativ neue Services hinzugefügt werden. Anschließend wartet der Server auf die Verbindung mit dem Client. Wenn der Bluetooth Client und der Server eine gemeinsame Verbindung aufbauen wollen, muss der Server seinerseits die Verbindung zu dem Client herstellen. Die restlichen Aktivitäten im Hinblick auf Data Streams, Datenaustausch und Schliessen der Verbindung sind praktisch identisch mit dem Anwendungsfalldiagramm des Client. Das gilt auch für mögliche Fehler, die im Diagramm mit angegeben sind.

4.1.2.3 Sequenz-Diagramm

Das Sequenzdiagramm für den Bluetooth Client (siehe Abbildung 4.4) und für den Bluetooth Server (siehe Abbildung 4.5) zeigen wie die Objekte der beiden Implementierungen miteinander interagieren und kommunizieren. Im Sequenzdiagramm des Client können wir ein “client” Objekt identifizieren, welches die Applikation startet und andere Objekte erstellt, die für die Kommunikation zwischen dem Server und dem Client benötigt werden. Das Objekt “localDevice” generiert das Objekt “discoveryDevice” welches später die Objekte “remoteDevice” und “ServiceRecord” generiert. Dank dieser Objekte ist der Client in der Lage eine Verbindung zu dem Server herzustellen, welcher auf einem bluetoothfähigen Gerät laufen soll, und auf diesem die Services aufrufen soll. Im nächsten Schritt generiert der Client eine Instanz von “StreamConnection”. Diese Instanz generiert “InputStream” und “OutputStream” Objekte, mit deren Hilfe der Datenaustausch durchgeführt werden kann.
Im Sequenzdiagramm von Server generiert das Objekt “Server” ein “serverURL” Objekt,

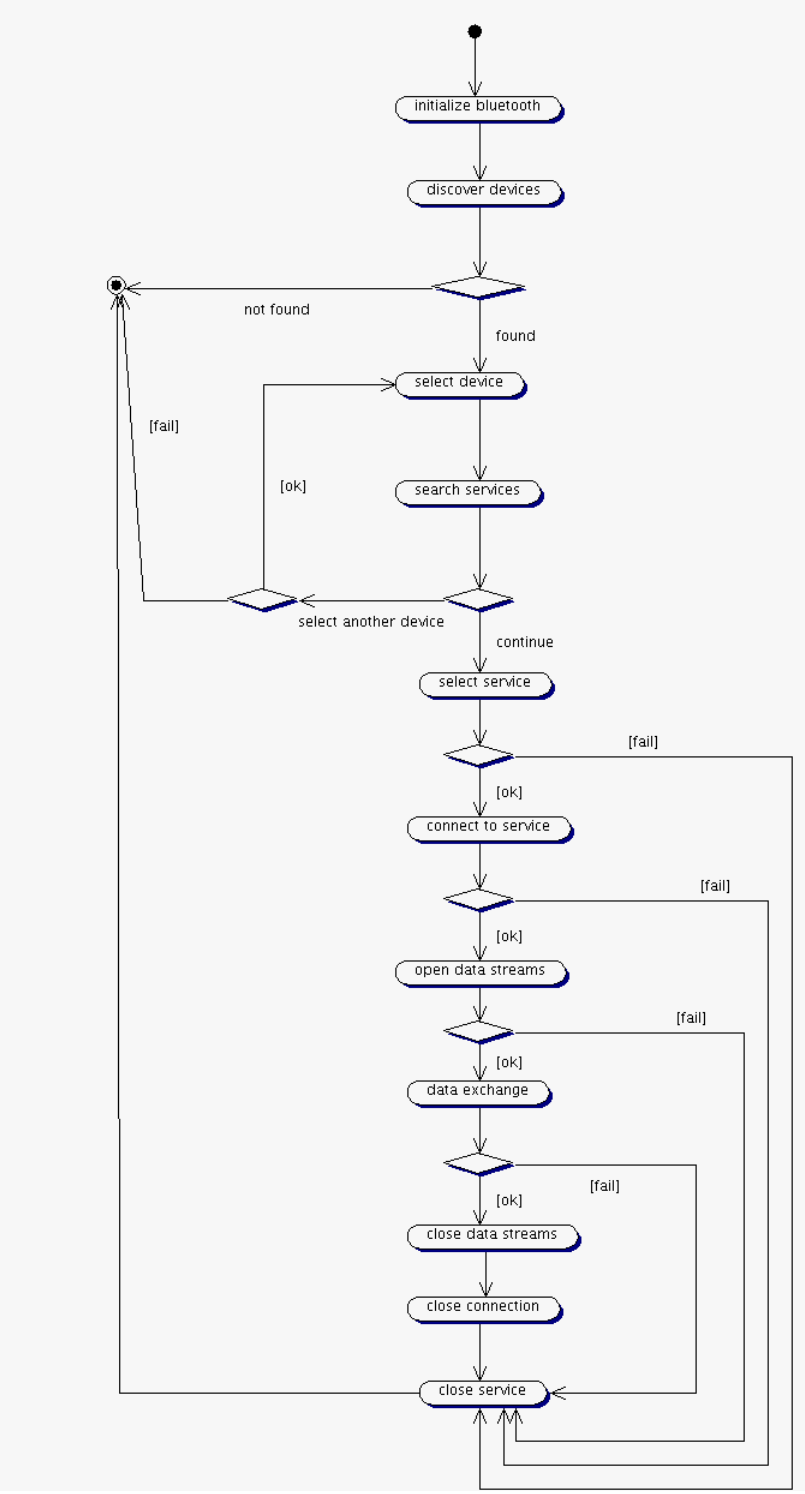

Abbildung 4.2: Bluetooth, ActivityDiagramClient

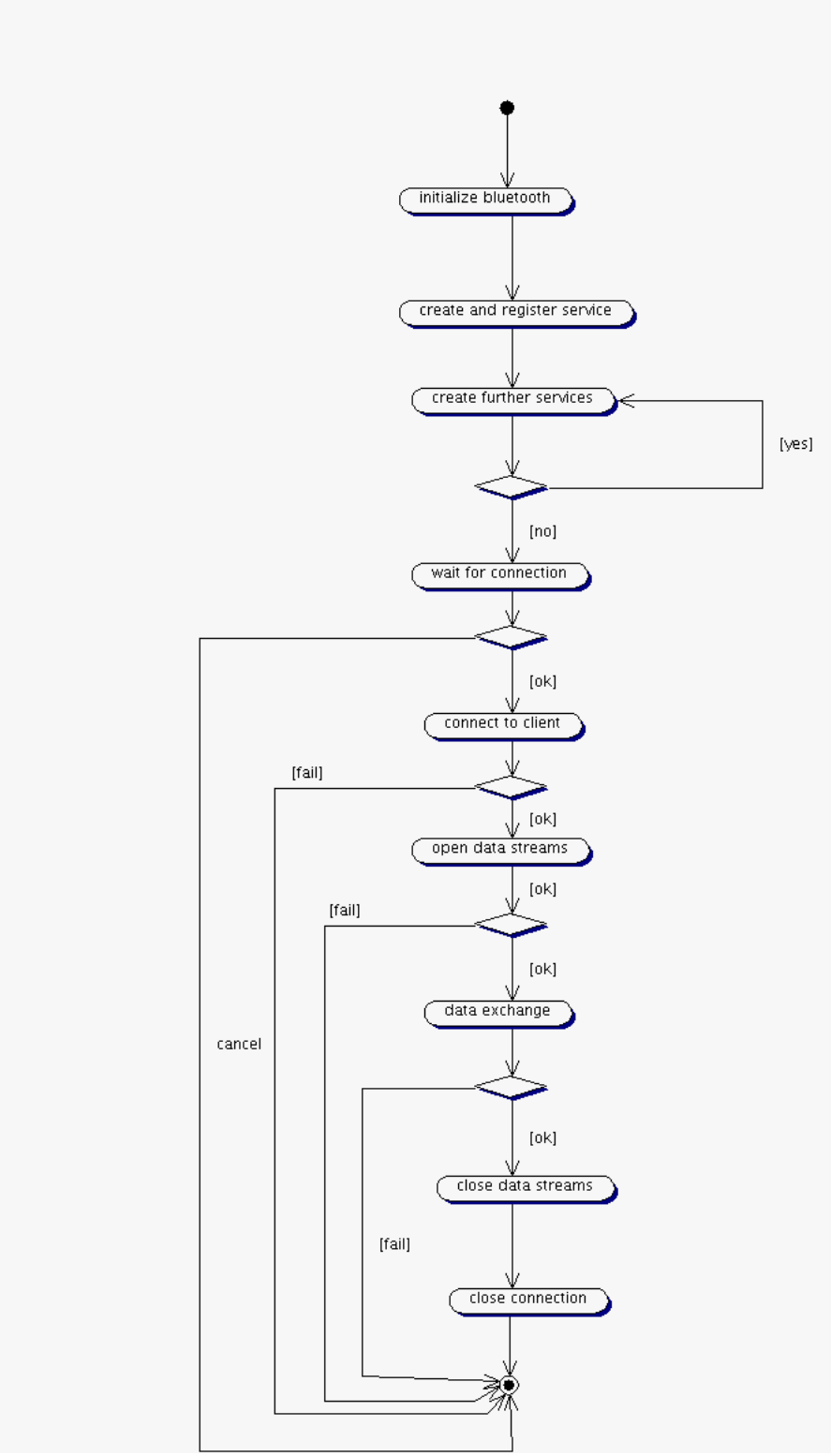

Abbildung 4.3: Bluetooth, ActivityDiagramServer

mit welchem der Server von den Clients identifiziert werden kann und ein “notifier” Objekt, welches alle erforderlichen Streams für die Kommunikation mit dem Client bereitstellt.

4.1.2.4 Klassen-Diagramm

Das Klassendiagramm verweist auf den Zusammenhang zwischen den von uns erstellten SIBs und avetanaBluetooth JSR-82 Implementierung.

Das SIB BluetoothInit erhält eine Referenz auf das `LocalDevice` Objekt, indem die statische Methode `LocalDevice.getLocalDevice()` aufgerufen wird. Dieses SIB erhält auch eine Referenz auf “local DiscoveryAgent” durch Aufruf von der Methode `getDiscoveryAgent()` auf dem “LocalDevice” Objekt. BluetoothDiscoverDevices implementiert das Interface `DiscoveryListener` und benutzt `DiscoveryAgent`, um bluetoothfähige Geräte in der Umgebung zu finden. Die SIBs BluetoothSelectDevicebyAddress, BluetoothSelectDeviceByName, BluetoothGetRemoteDeviceProperties und BluetoothGetRemoteDevicesListProperties ermöglichen dem Benutzer das Anzeigen aller gefundenen Geräte auf der Webpage um von diesen ein bestimmtes auszuwählen. BluetoothSearchServices und BluetoothSearchServicesByUUID implementieren ebenso `DiscoveryListener` und benutzen ein `DiscoveryAgent` und ein `RemoteDevice` Objekt, um Services auf einem konkreten Bluetoothgerät zu finden. Man kann die SIBs BluetoothGetServiceProperties und BluetoothGetServicesListProperties benutzen, um Services auf der Webpage anzuzeigen. Das Interface `ServiceRecord` spezifiziert ein ServiceRecord von einem Service. `UUID` und `DataElement` werden benötigt, um Services zu initialisieren und zu finden. Außerdem helfen sie dabei, die Attribute von einzelnen Services zu extrahieren und aufzulisten.

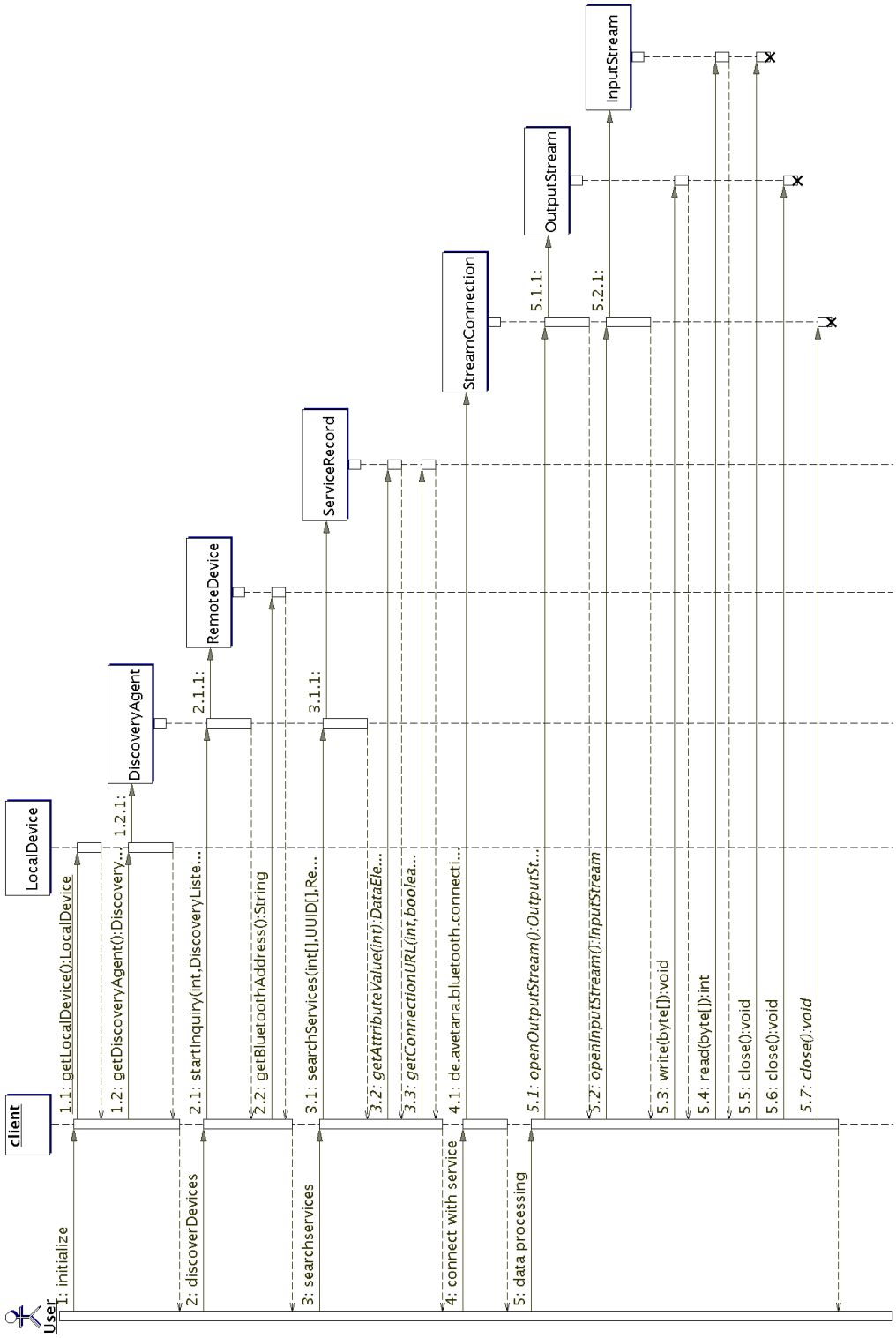

Abbildung 4.4: Bluetooth, SequenceDiagramClient

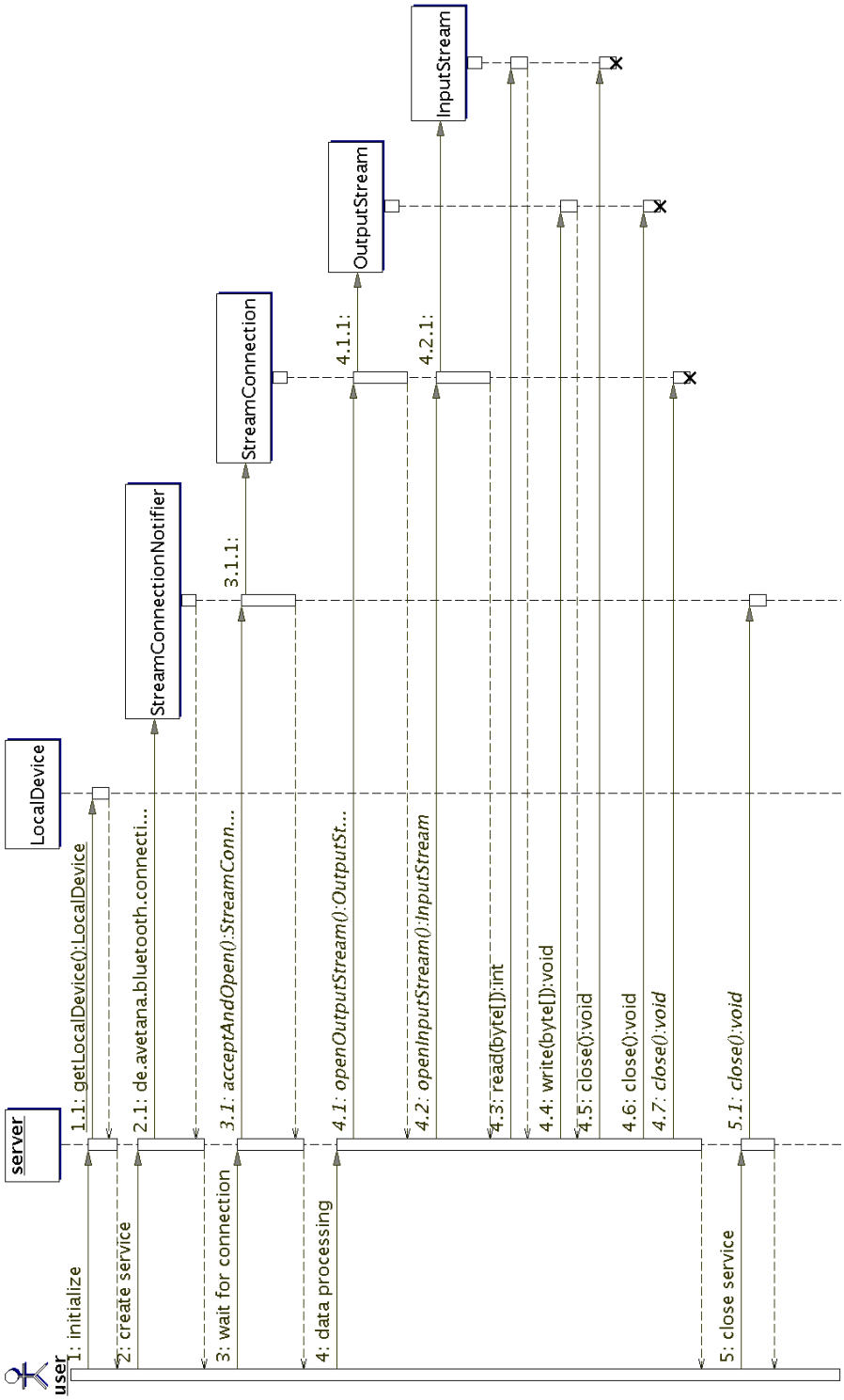

Abbildung 4.5: Bluetooth, SequenceDiagramServer

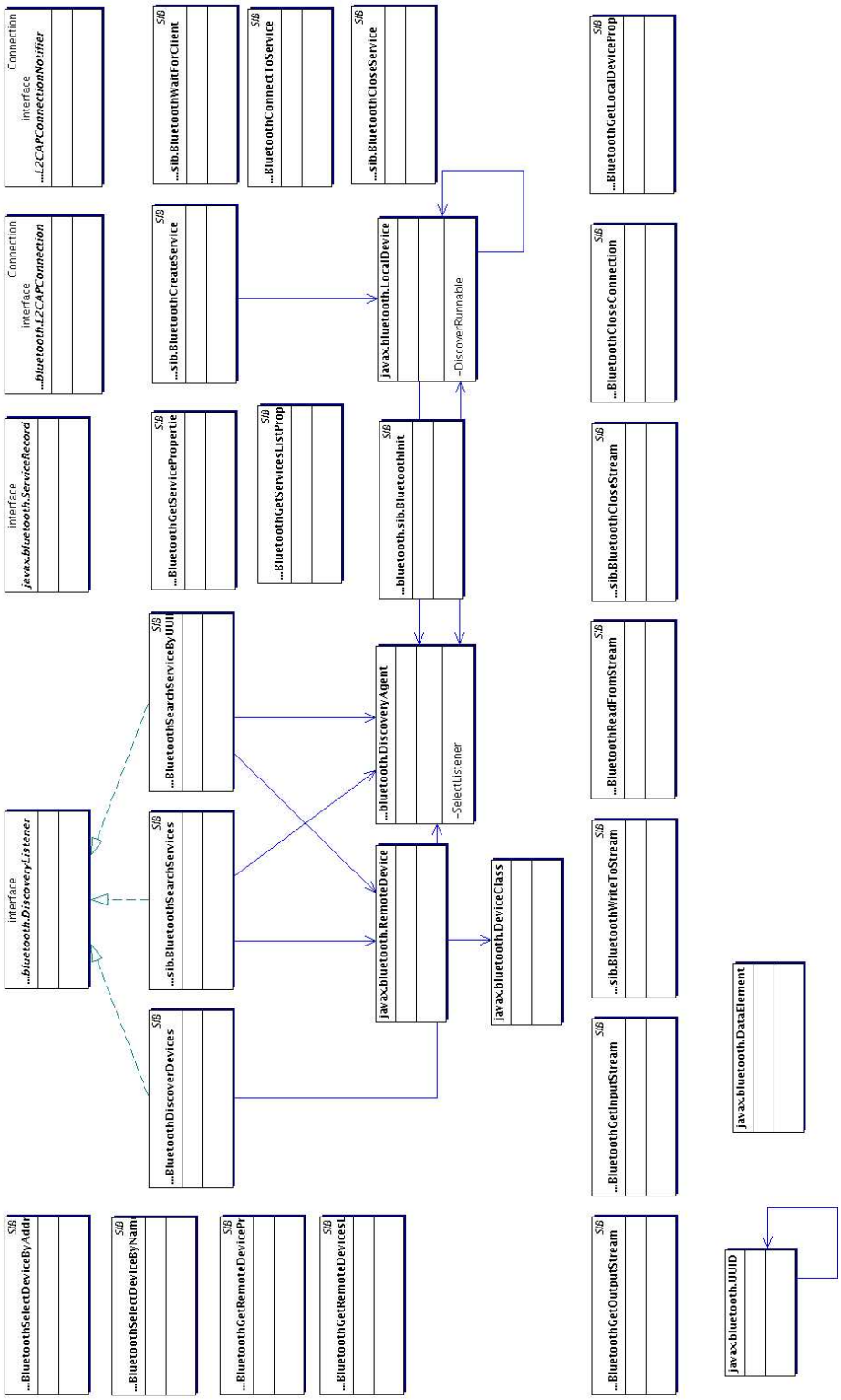

Abbildung 4.6: Bluetooth, Class-Diagram

4.1.3 Implementierung

Aufgrund des UML-Designs wurden folgende SIBs implementiert:

- `BluetoothInit`
- `BluetoothDiscoverDevices`, `BluetoothSearchServices`, `BluetoothSearchServiceByUUID`
- `BluetoothConnectToService`, `BluetoothWaitForClient` `BluetoothCreateService`,
- `BluetoothCloseConnection`, `BluetoothCloseService`
- `BluetoothSelectDeviceByAddress`, `BluetoothGetServicesListProperties`, `BluetoothGetRemoteDevicesListProperties`, `BluetoothGetRemoteDeviceProperties`, `BluetoothSelectDeviceByName`, `BluetoothGetServiceProperties`, `BluetoothGetLocalDeviceProperties`
- `BluetoothWriteToStream`, `BluetoothReadFromStream`,`BluetoothGetOutputStream`, `BluetoothGetInputStream`, `BluetoothCloseStream`

Bevor irgendein Bluetooth SIB verwendet wird, muss der Bluetooth Stack mit dem SIB `BluetoothInit` initialisiert werden. Auf Client Seite kann mit den SIBs `BluetoothDiscoverDevices`, `BluetoothSearchServices`, `BluetoothSearchServiceByUUID` und `BluetoothConnectToService` ein Bluetooth Server gefunden werden und die Verbindung zu dem angebotenen Dienst aufgebaut werden. Auf Server Seite kann mit den SIBs `BluetoothCreateService` und `BluetoothWaitForClient` ein Bluetooth Dienst angeboten werden. Nach dem Datenaustausch müssen Client und Server die verwendeten Resourcen mit den SIBs `BluetoothCloseConnection` und `BluetoothCloseService` wieder freigeben. Mit den SIBs `BluetoothSelectDeviceByAddress` und `BluetoothSelectDeviceByName` kann der Client aus der Liste der gefundenen Dienste einen Dienst auswählen. Mit den SIBs `BluetoothGetLocalDeviceProperties`, `BluetoothGetServiceProperties`, `BluetoothGetRemoteDeviceProperties`, `BluetoothGetRemoteDevicesListProperties` and `BluetoothGetServicesListProperties` können die Eigenschaften der gefundenen Bluetooth Geräte und Dienste abgefragt werden. Für den Datenaustausch wird von den SIBs `BluetoothGetInputStream`, `BluetoothGetOutputStream`, `BluetoothReadFromStream`, `BluetoothWriteToStream` und `BluetoothCloseStream` der Zugriff auf den Stream zur Verfügung gestellt.

4.1.4 Beispielapplikation BluChat

Mit Hilfe der entstandenen Bluetooth SIB-Palette entwickelten wir eine kleine Webapplikation namens BluChat. BluChat bietet Nutzern die Möglichkeit eines unidirektionalen

Datenaustauschs. Der Benutzer kann in der BluChat-Applikation sowohl als Server als auch als Client agieren; startet der Benutzer BluChat als Server, so wartet die Applikation darauf, dass sich ein Client einloggt, von dem der Server daraufhin Nachrichten empfangen kann. Als Client sucht BluChat zunächst nach Bluetooth-Geräten in Reichweite und listet diese auf. Anschließend kann der Benutzer eines dieser Geräte auswählen, woraufhin der BluChat-Service gesucht wird; wird dieser gefunden, so kann der Nachrichtenaustausch beginnen, wird der Service nicht gefunden, so muss der Nutzer von Neuem beginnen. Abbildung 4.7 zeigt die Startseite des BluChat-Webservices. Die Entwicklung von

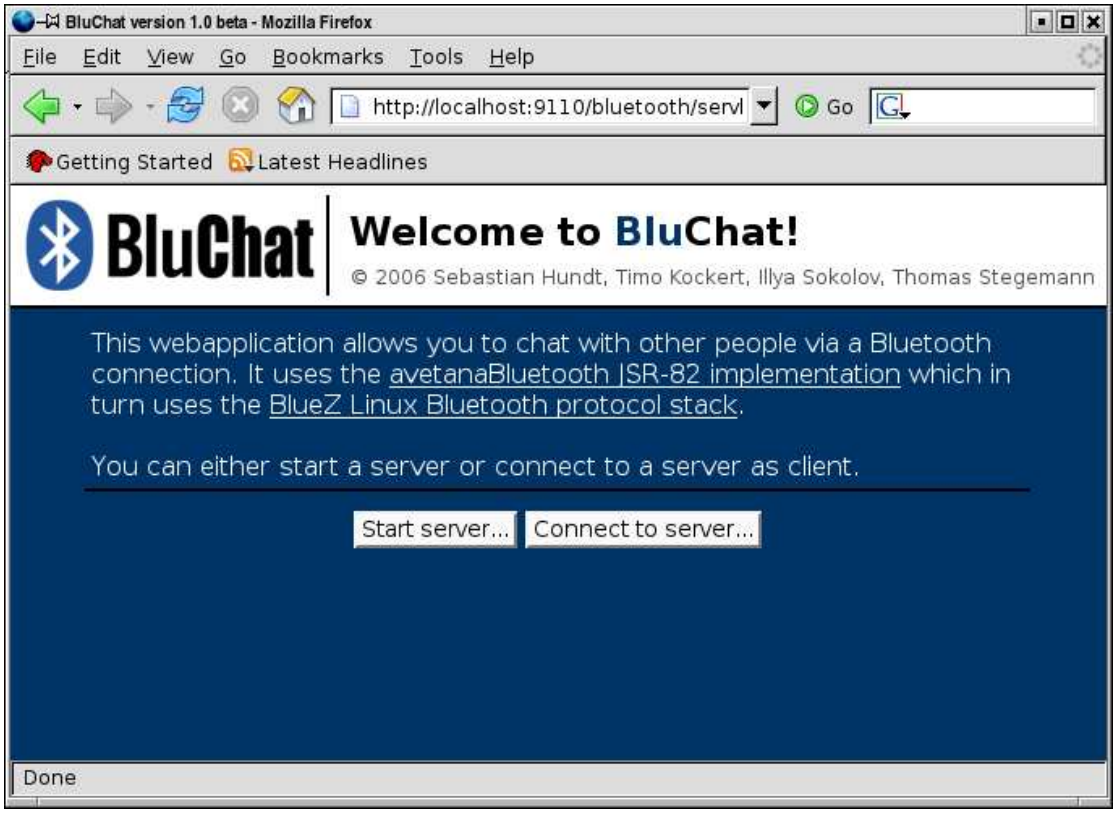

Abbildung 4.7: Bluetooth, BluChat Startseite

BluChat fand mit Hilfe des JavaABC statt. Abbildung 4.8 zeigt den enstandenen BluChat JavaABC-Graph. In diesem wird zunächst im BluetoothInit SIB der Bluetooth Stack initialisiert und der Zugriff auf das lokale Bluetooth-Gerät ermöglicht. Daraufhin teilt sich der Graph in Client- und Serverseite. Auf der Serverseite wird zunächst der BluChat-Service erstellt und veröffentlicht. Danach wartet der Server darauf, dass sich ein Client mit ihm verbindet. Sobald dies geschieht, bekommt der Server einen InputStream und wartet darauf von diesem Stream lesen zu können. Wenn auf Clientseite eine Textnachricht abgeschickt wird, liest der Server diesen aus und zeigt den Inhalt im Browser an. Anschließend wartet er sofort wieder auf neue Nachrichten.
Auf Clientseite wird zunächst mit Hilfe des BluetoothDiscoverDevices SIB nach Geräten

in Reichweite gesucht, welche anschließend im Browser angezeigt werden. Der Benutzer kann hier nun ein Gerät auswählen, auf dem nach dem BluChat Service gesucht werden soll. Bei Erfolg können nun beliebige Textmeldungen eingegeben und gesendet werden. Dies geschieht mit Hilfe von OutputStreams, in die die Textnachricht geschrieben wird. Der Nutzer hat hierbei zudem zu jeder Zeit die Möglichkeit die Verbindung zu trennen und den Nachrichtenaustausch damit zu stoppen.
Jeder Bluetooth SIB besitzt eine ”Fail” Kante, welche beim Auftreten eines Fehlers betreten wird. Der Nutzer hat dabei bei jedem SIB die Möglichkeit sich die Fehlermeldung ausgeben und gegebenenfalls auf der Webseite anzeigen zu lassen.

4.1.5 Benötigte Bibliothek

Die Implementierung verwendet die Bibliothek AvetanaBT.jar [1]. Sie verwendet im wesentlichen die von Sun definierten Schnittstellen javax.microedition, java.io und javax.bluetooth, nur die Klasse `de.avetana.bluetooth.connection.Connector` ist spezifisch für AvetanaBT.jar. Unter Linux verwendet die AvetanaBT.jar Bibliothek den offiziellen BlueZ [3] Bluetooth Stack.

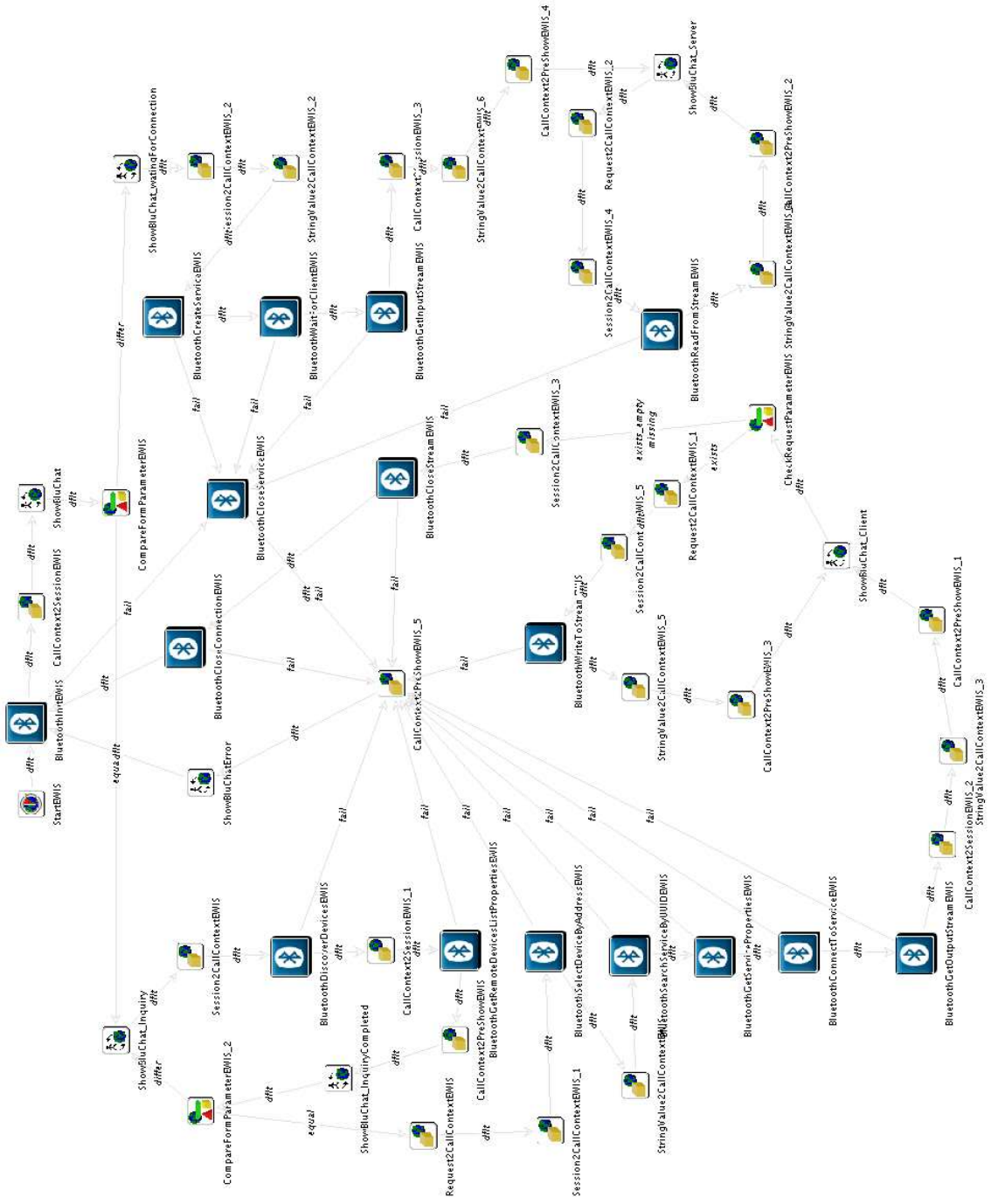

Abbildung 4.8: Bluetooth, BluChat jABC- Graph

4.2 Design und Implementierung einer SIB Bibliothek für die Ausführung von BASH-Befehlen

4.2.1 Einleitung

Das Ziel unserer Aufgabe war das Design und die Implementierung einer SIB-Bibliothek zur Erzeugung von Bash-Skripten, welche in eine Skript-Datei geschrieben bzw. direkt ausgeführt werden sollten. Unsere Hauptidee war es, für jeden Bash-Befehl und jedes Bash-Konstrukt ein eigenes SIB zu schreiben, um Skripte mit Hilfe eines Service Logic Graphs erstellen zu können. Das Skript wird während des Graph-Durchlaufs erstellt. Am Ende wird dem Benutzer die Wahl überlassen, ob das Skript sofort ausgeführt werden soll oder nicht. Dies wurde durch ein ”Exec”-SIB realisiert, welches die soeben erzeugte Skript-Datei mit Hilfe von „Runtime.exec()“ ausführt. Die möglichen Ausgaben des Skripts (stdout und stderr) werden hier mit Hilfe von OutputStreams in Dateien umgelenkt, so dass der Benutzer nach der Ausführung des Skripts die Möglichkeit hat, alle Ausgaben und Fehlermeldungen in Ruhe durchzusehen.

4.2.2 Design

Wie bereits erwähnt war unsere Designidee, dass jedes SIB seine Funktion (Aufruf und Parameter) in die Skriptdatei so reinschreibt, dass diese Datei direkt ausgeführt werden kann. Zuerst haben wir uns die benötigten Use-Cases überlegt und diese nach Befehlsgruppen sortiert. D.h., dass alle Befehle die eine ähnliche Funktion haben, in die gleiche Kategorie zusammengefasst wurden. Als Kategorien wählten wir folgende aus:

- Dateimanagement
- Zugriffskontrolle
- Prozessmanagement
- Spezielle Konstrukte (if, for...)
- Dateisystem

Mit den einzelnen Bash-Befehlen können Skripte für beliebige Abläufe erstellt werden. Die einzelnen Use-Cases in Abbildung 4.9 geben nicht die möglichen Anwendungen von Skripten wieder. Sie sind eine kleine Auswahl der wichtigsten Bashbefehle und -konstrukte und entsprechen je einem realen Befehl bzw. Konstrukt.

Als Activity-Diagramm (siehe Abbildung 4.10) haben wir uns ein Beispielskript erstellt und entsprechend dargestellt. Das Beispielskript überprüft zwei Dateien auf Gleichheit und löscht eine der beiden in diesem Falle. Wenn die Dateien unterschiedlich sind, wird

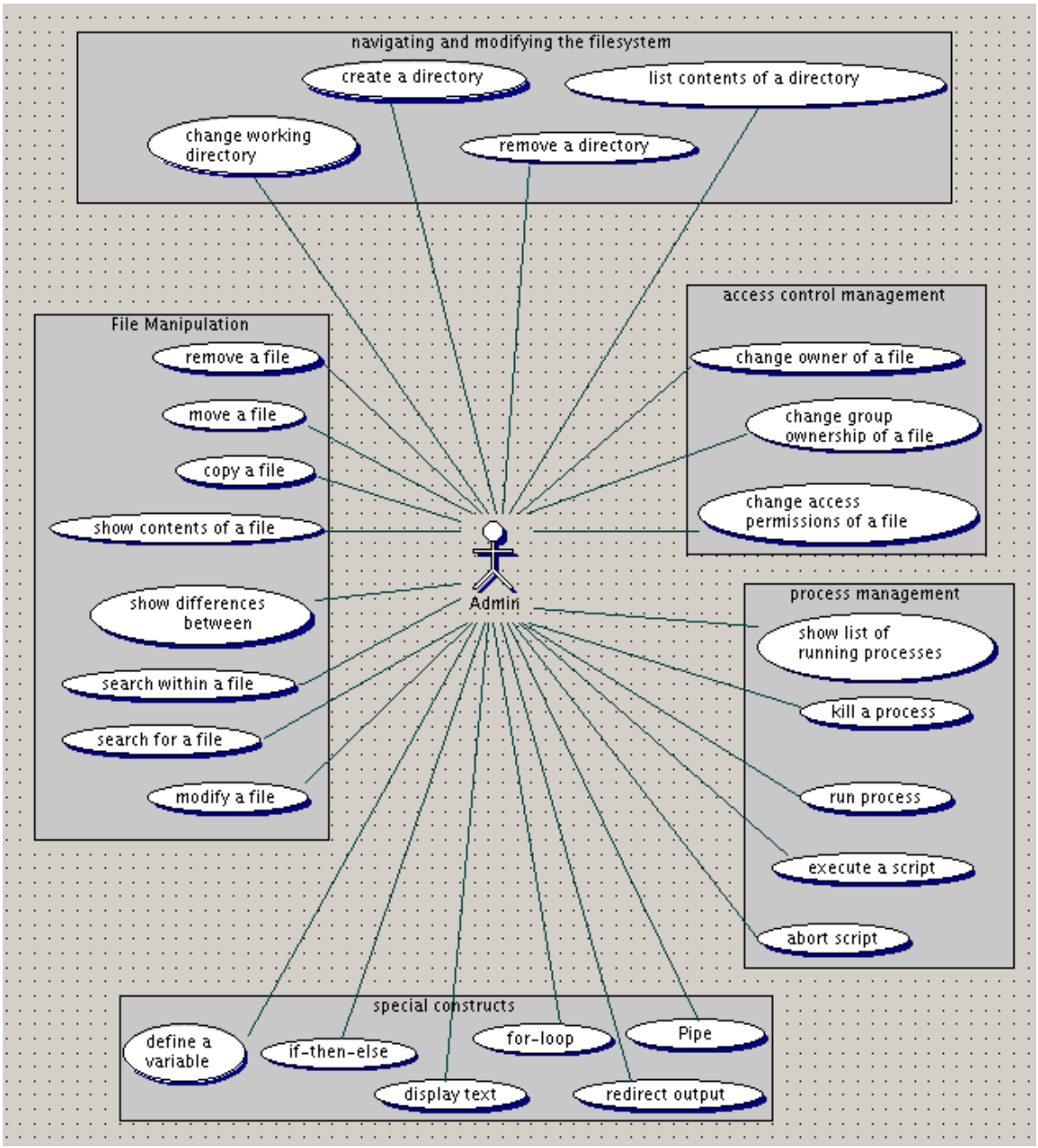

Abbildung 4.9: Bash, Usecase-Diagramm

ein Backup erzeugt und eine Datei ersetzt die andere. Das Beispielskript umfasst viele gängige Befehle und soll somit einen ”normalen” Skriptablauf simulieren. Zusätzlich zu den normalen Bashbefehlen haben wir hier noch spezielle Konstrukte wie eine if-Anweisung miteinbezogen. Jede Aktivität wird nachher in einem eigenen SIB dargestellt. Diese werden vom logischen Aufbau später in einem Service-Logic-Graph fast genauso miteinander verknüpft wie im Activity-Diagramm.

4.2.3 Implementierung und Test

Nachdem wir eine Liste von Befehlen und Konstrukten hatten, die wir in unserer SIB-Palette bereit stellen wollten, haben wir die jeweiligen SIBs implementiert. Unsere Liste umfasst folgende SIBs:

Dateimanagement:

- `Copy`
- `Cd`
- `Cut`
- `Diff`
- `Find`
- `Grep`

Zugriffskontrolle:

- `Chgrp`
- `Chmod`
- `Chown`

Prozessmanagement:

- `Command`
- `Exit`
- `Exec`
- `Kill`
- `Ps`

Spezielle Konstrukte:

- `Cat`
- `Done`
- `Else`
- `Fi`
- `For`
- `If`
- `Then`
- `Variable`
- `Pipe`
- `Redirection`

Dateisystem:

- `Ls`
- `Mkdir`
- `Move`
- `Remove`
- `Rmdir`

spezielle SIB:

- `CloseFile`
- `OpenFileEWIS`

Die Namen der SIBs, die Bashbefehle darstellen, sind jeweils identisch mit dem entsprechenden Bashbefehl. Die OpenFileEWIS, CloseEWIS und ExecEWIS-SIBs stellen keine Bashbefehle dar. Diese SIBs sind zum Öffnen bzw. Schließen der Skript- und Ausgabedateien, sowie zum Ausführen des fertigen Skripts. Das OpenFileEWIS-SIB erstellt die Skriptdatei und stellt dann ein FileWriter zur Verfügung. Das CloseEWIS-SIB schließt am Ende den FileWriter und damit die Skriptdatei. Das ExecEWIS öffnet die zwei Ausgabedateien - eine für die Standardausgabe und eine für die Fehlermeldungen - führt das Skript aus, lenkt die Ausgabe sowie die Fehlermeldungen in die vorbereiteten Dateien

und schließt die Dateien anschließend wieder. Die Implementierung der anderen SIBs war relativ einfach. Jedes SIB bekommt einen FileWriter übergeben, der direkt mit der zu schreibenden Skriptdatei verbunden ist. So kann jedes SIB direkt seinen ”Befehl” mit allen Parametern in das Skript eintragen. Die Parameter werden in der jABC-Umgebung direkt als Parameter des SIBs eingetragen und in die Skriptdatei mit übernommen.
Da am Ende der Aufgabe noch ein bisschen Zeit übrig war, haben wir uns noch mit der Erstellung von Icons für die SIBs beschäftigt und ausgewählten SIBs ein Icon erstellt und implementiert.

4.2.4 Beispielanwendung

Um unser oben erwähntes Beispielskript etwas mit Leben zu füllen, wollen wir an dieser Stelle etwas näher darauf eingehen. Wir haben uns zuerst überlegt, was das Skript realisieren soll und haben uns für einen simplen Backup-Vorgang entschieden. Danach haben wir das ”Zielskript” als Vorlage geschrieben, wie es nachher von dem jABC-Graphen erstellt werden sollte. Dann wurde mit Hilfe der geschriebenen SIB-Palette der Graph zusammengebaut, der in Abbildung 4.11 zu sehen ist:

Wir führen den fertigen Graphen im jABC aus. Entsprechend der SIBs wurde dann das Skript nach unseren Vorstellungen zusammengesetzt und mit Hilfe des Exec-SIBs direkt ausgeführt. Die Parameter, die die einzelnen SIBs benötigen, werden in die einzelnen SIBs eingetragen. Beginnend mit einem Startknoten und dem Erstellen der Ausführungsdatei werden verschiedene Befehle hintereinander eingetragen. Am Ende wird die Datei geschlossen und danach ausgeführt. Die korrekte oder fehlerhafte Ausführung der Skriptdatei wird am Ende des Graphen wieder durch HTML-Seiten angezeigt.

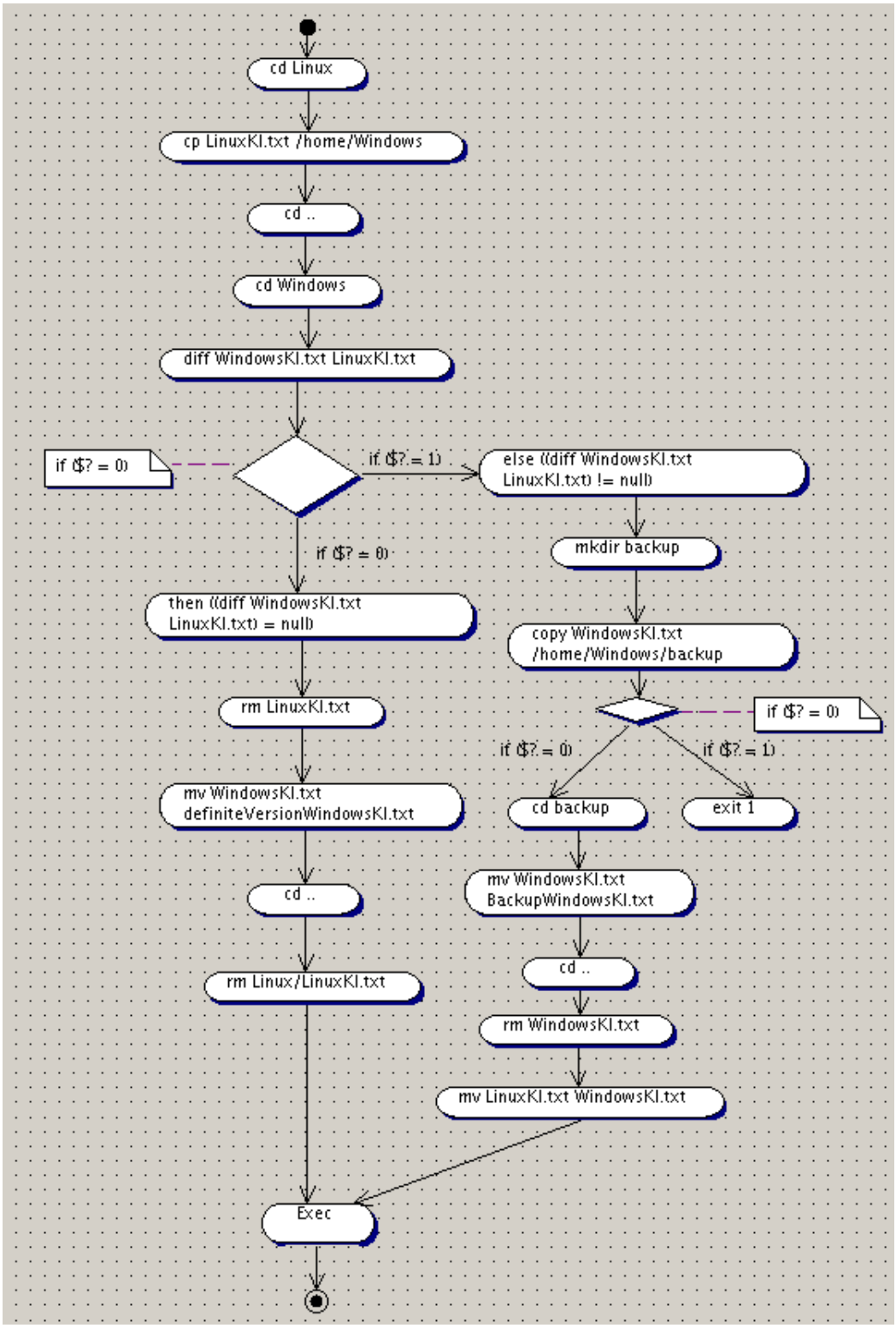

Abbildung 4.10: Bash, Aktivitätsdiagramm

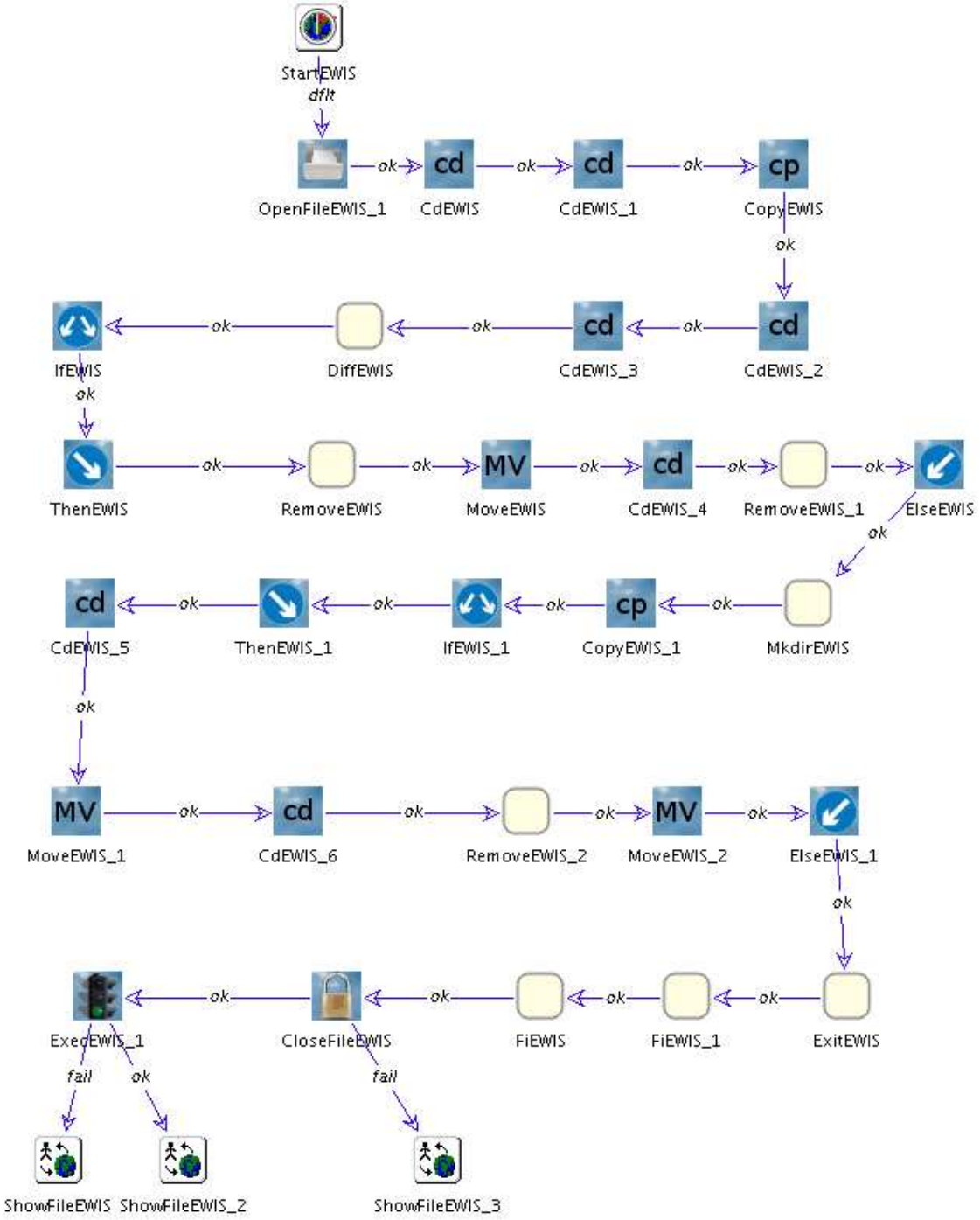

Abbildung 4.11: Bash, Beispiel jABC-Graph

4.3 Design und Implementierung einer SIB-Bibliothek zum Testen und zur Diagnose von Netzwerken basierend auf Network-Sockets

4.3.1 Einleitung

Die Aufgabe bestand in der Entwicklung einer SIB-Bibliothek für die jABC-Entwicklungsumgebung. Diese Bibliothek erlaubt es beispielsweise einem Netzwerk-Administrator die Rechner in seinem Arbeitsbereich und die Services, die auf ihnen laufen, auf ihre Verfügbarkeit und korrekte Arbeitsweise mit Hilfe von Network-Sockets zu überprüfen. Die SIBs sollten dabei so designt sein, dass sie wiederverwendbar und leicht erweiterbar sind. Dadurch können leicht neue Workflows durch einfaches Zusammenklicken eines Service-Logic-Graphs erzeugt werden.
Die entwickelten SIBs wurden in drei größere Gruppen zusammengefasst:

- Verbindungskontrolle und Ein-/Ausgabe
  - GetSocketAddress
  - CreateSocket
  - ConnectSocket
  - GetSocketInputStream
  - GetSocketOutputStream
  - ReadInputStream
  - WriteOutputStream
  - ReceiveDatagram
  - SendDatagram
  - CloseStreams
  - CloseSockets
  - StringCompareIgnoreCase
- Generierung von protokollspezifischen Anfragen
  - GenCVSRequest
  - GenDNSRequest
  - GenFTPRequest
  - GenHTTPRequest
  - GenIMAPRequest

  - GenNEWSRequest
  - GenPOP3Request
  - GenSMTPRequest

- Ausführung externer Befehle
  - ExecuteNetstat
  - ExecutePing
  - ExecuteTraceroute

Die SIBs sind flexibel, wiederverwendbar und leicht erweiterbar. Dadurch enstanden mehr SIBs, die aber weniger komplex und dadurch nicht an einen zu speziellen Kontext gebunden sind. Die Handhabung wurde dadurch stark vereinfacht und ist praktisch selbsterklärend.
Die SIBs wurden in UML mit ***Together*** designt, in Java mit ***Eclipse 3.1*** implementiert und die Service-Logic-Graphen, die die SIBs nutzen, wurden mit dem ***jABC*** erstellt.

4.3.2 Design

Das Projekt wurde mit Hilfe von UML designt. Das Usecase-Diagramm (siehe Abb. 4.12) zeigt die Möglichkeiten auf, die ein Administrator hat, mit dem System zu interagieren. Das Activity-Diagramm (siehe Abb. 4.13) zeigt ein Beispielworkflow eines Service-Logic-Graphs und das Sequence-Diagramm (siehe Abb. 4.14) erklärt den exemplarischen Ablauf beim Ausführen eines externen Befehls.

4.3.2.1 Usecase-Diagramm

Es gibt mehrere Möglichkeiten zur Interaktion, die von der Bibliothek bereitgestellt werden. Um beispielsweise eine protokollspezifische Anfrage an einen Dienst zu stellen, muss zuerst ein Socket erstellt werden. Dieser wird dann mit dem Rechner, auf dem der Dienst läuft, verbunden. Die mit diesem Socket assoziierten Streams (Ein-/Ausgabe) werden besorgt. Danach kann in den Ausgabe-Stream eine der vordefinierten Anfragen oder eine selbast definierte Folge von Zeichen geschrieben und somit zum anderen Ende des Sockets geschickt werden 4.13. Aus dem Eingabe-Stream kann dann die Antwort gelesen werden. Eine andere Möglichkeit ist das Ausführen externer Befehle. Diese werden mit Hilfe der JRE ausgeführt und die Antworten können über einen der Ausführungsschicht assoziierten Stream gelesen werden. Im Moment stehen die Befehle ***ping***4.14, ***traceroute*** und ***netstat*** zur Verfügung. Diese werden über die JRE auf dem jeweiligen System ausgeführt, indem die bereits vorhandenen Programme aufgerufen werden. Dies ist einfacher und sicherer. Würde man diese Funktionalität in Java nachbilden wollen, wären dafür Sockets vom Typ RAW nötig, welche aber nicht vom SDK angeboten werden. Man hätte also eine

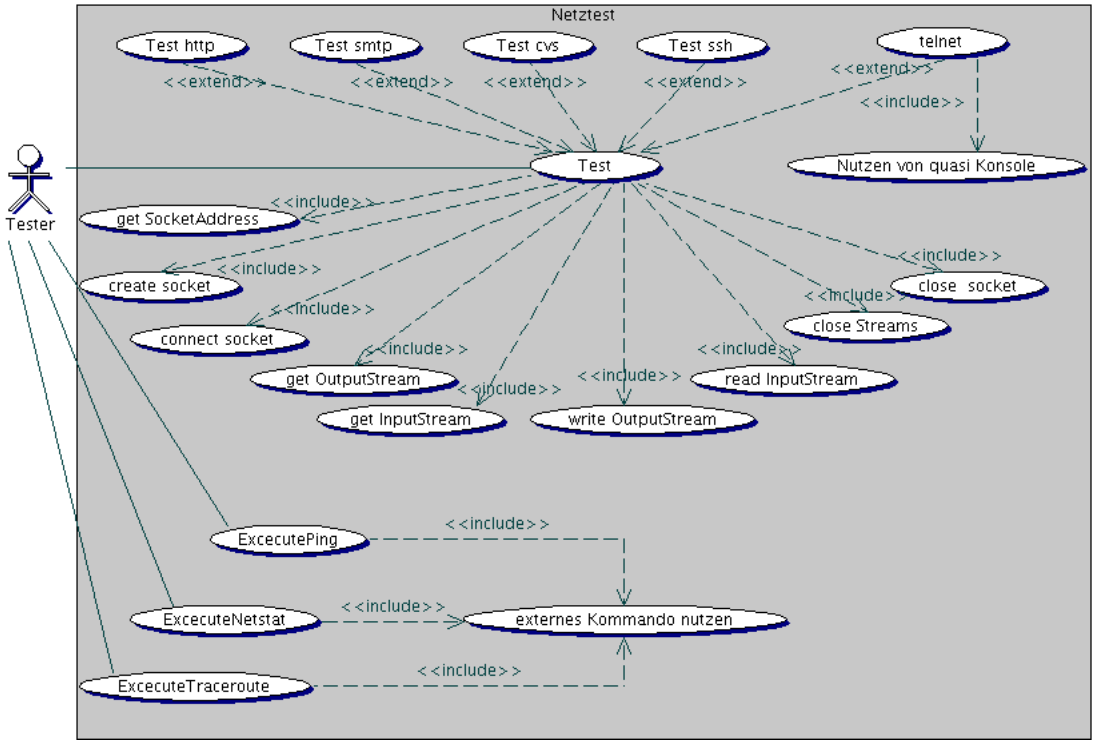

Abbildung 4.12: Sockets, Usecase-Diagramm

extra Bibliothek schreiben und per JNI anbinden müssen. Zusätzlich müsste die JVM mit root-Rechten laufen, da nur mit diesen Rechten RAW-Sockets geöffnet werden können. Dieses Risiko konnte so umgangen werden.

4.3.2.2 Activity-Diagramm

Das Activity-Diagramm beschreibt den Workflow, wie er eigentlich bei jedem Protokoll-Test durchlaufen wird. Zuerst wird ein Socket erstellt und mit dem gewünschten Endpunkt verbunden (das gilt nur für TCP, bei UDP wird die Empfängeradresse im Paket übergeben, der Socket braucht nicht verbundenzuwerden). Bei TCP werden dann die Ein-/Ausgabe-Streams besorgt. Nun kann in den Ausgabe-Stream geschrieben und vom Eingabe-Stream gelesen werden. Bei UDP werden die Pakete anhand der gewünschten Zieladresse und des zu versendenden Inhalts gebaut und über den Socket verschickt. Genauso werden über den Socket dann Pakete empfangen, die dann in Inhalt und Absenderadresse zerlegt werden können. Nachdem der gewünschte Datenaustausch beendet wurde, werden die Streams (nur TCP) und danach die Sockets geschlossen. Die empfangenen Daten oder evtl. aufgetretenen Fehler können nun beispielsweise auf einer Webseite zur Verügung gestellt werden.

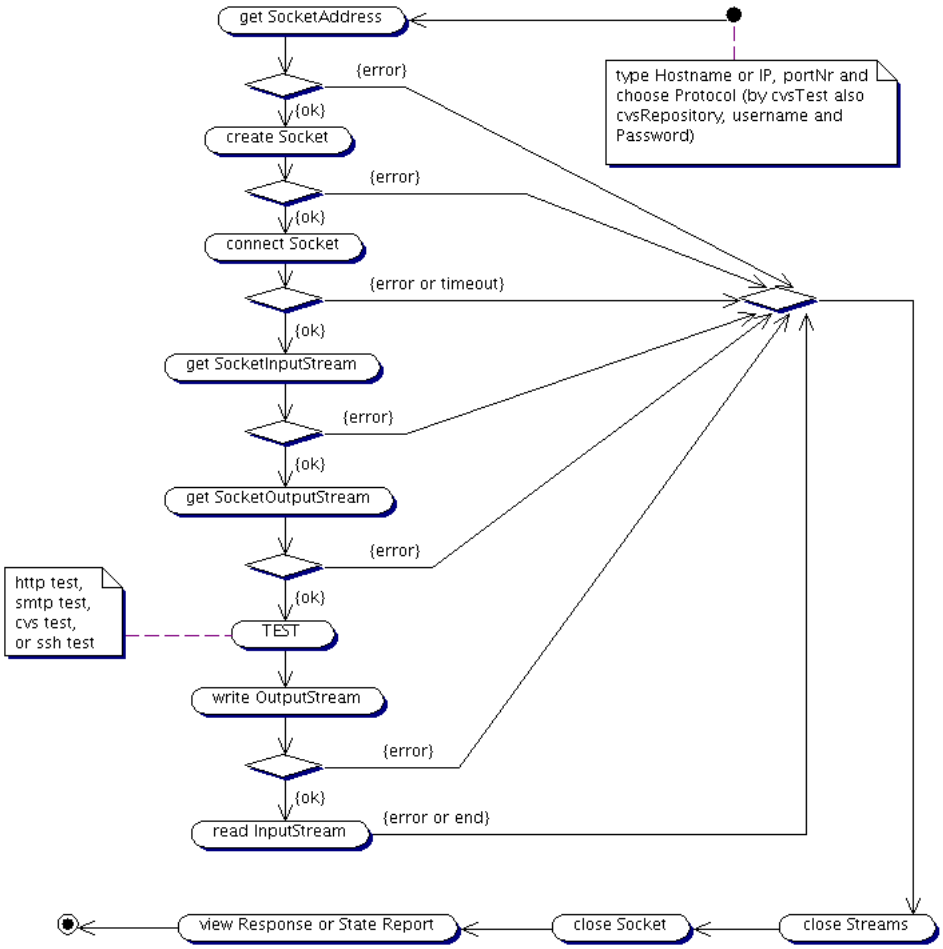

Abbildung 4.13: Sockets, Activity-Chart

4.3.2.3 Sequence-Diagramm

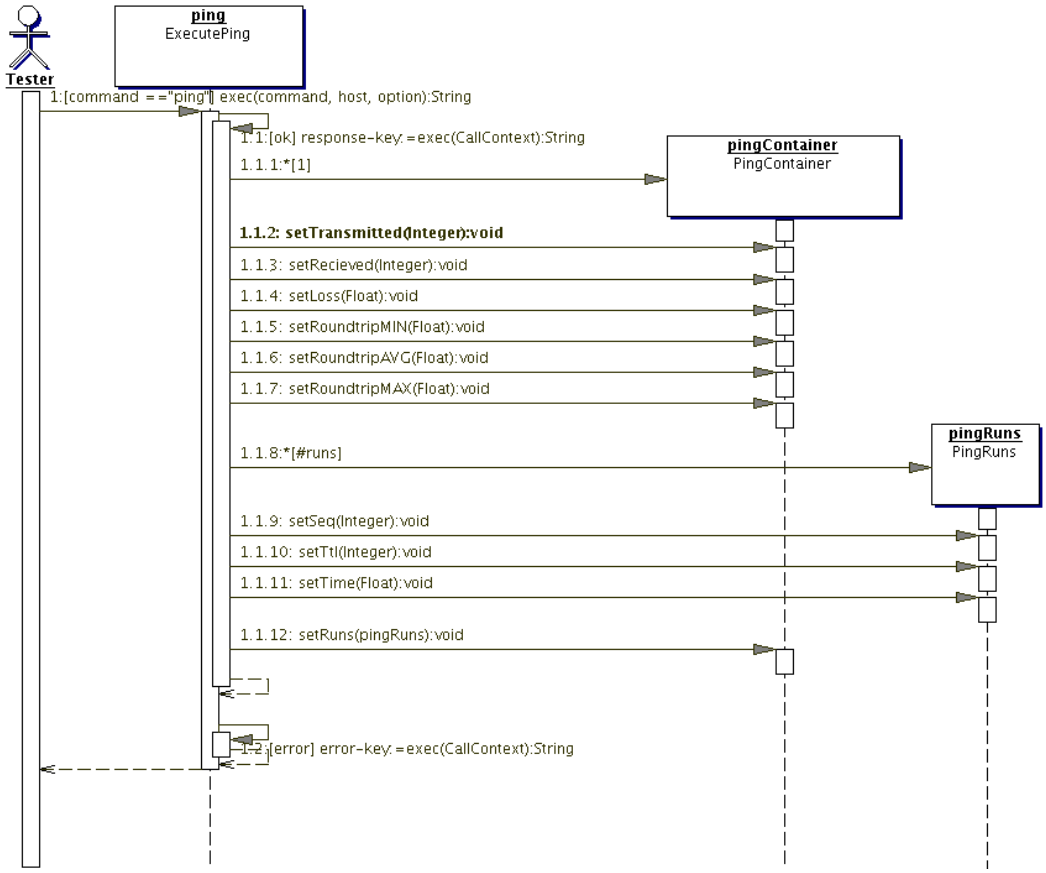

Abbildung 4.14: Sockets, Sequence-Chart

Die Ausführung der externen Befehle folgt eigentlich immer demselben Schema. Man lässt über die JRE den Befehl ausführen und liest aus dem assoziierten Eingabe-Stream die Ausgabe des Programms. Diese wird dann auseinandergenommen und der besseren Präsentationsmöglichkeit wegen in Container-Klassen gepackt.

4.3.3 Implementierung und Test

4.3.3.1 Übersicht

Die Service-Logic-Graphen wurden mit Hilfe des jABC gebaut. Die verwendeten SIBs sollten einfach zu verstehen und anzuwenden sein. Es wurde so wenig Funktionalität wie möglich in einzelne SIBs gepackt. Dadurch entstanden zwar mehr SIBs die aber weitaus weniger komplex sind und daher auch einfacher erweitert oder wiederverwendet werden

können. Man erhält einen hohen Grat an Modularität, da die SIBs nicht hochgradig spezialisiert und somit nicht auf wenige Einsatzmöglichkeiten beschränkt sind. Mit Hilfe der GraphSIBs, auch Makros genannt, die vom jABC zur Verfügung gestellt werden, kann man z.B. das Connection-Handling in einem eigenen Graphen erstellen und über einen Graph-SIB dann als einzelnen SIB in anderen Graphen zur Verfügung stellen. Das ist immer dann sinnvoll, wenn man Standardworkflows hat, die man nur einmal modellieren will, aber dann öfters verwendendet werden sollen. Ein weiterer Vorteil schlanker SIBs liegt in der einfachen Änderbarkeit. Ein hochspezialisierter SIB wäre viel schwerer neuen Gegebenheiten anzupassen.

4.3.3.2 Benutzung

Um die SIBs zu verwenden und Tests zu bauen, kann man sich einfach mit den SIBs einen entsprechenden Workflow aufbauen. Zusammengefasst entspricht das z.B. bei TCP folgendem: Adresse und Port besorgen und ein SocketAddress-Objekt erstellen (GetSocketAddress), Socket öffnen (CreateSocket), Verbindung aufbauen (ConnectSocket), Streams besorgen (GetSocketInputStream und GetSocketOutputStream), Anfrage generieren (z.B. GenCVSRequest), Anfrage schicken (WriteOutputStream) , Antwort lesen (ReadInputStream), Streams schließen (CloseStreams), Socket schließen (CloseSockets), Antwort auswerten. So kann man schnell neue Tests entwickeln, indem man neue SIBs für Protokoll-Anfragen implementiert, die noch nicht vorhanden sind, oder vorhandene erweitert. Die Antworten werden sowohl bei den Protokoll-Tests (in einem Vector oder als byte[]) als auch bei den externen Tests in Container-Klassen abgelegt, so dass bei einer späteren Auswertung gezielt Informationen herausgepickt werden können. Die SIBs wurden gezielt so implementiert, dass sie auch außerhalb des Test-Kontextes verwendet werden können. Man hat z.B. SIBs für die grundlegende Netzwerk-Kommunikation. Eine rudimentäre Telnet-Anwendung ließe sich z.B. damit verwirklichen oder auch jede andere Anwendung, die Sockets zur Kommunikation nutzt. Notwendigerweise müssten noch SIBs gebaut werden welche alle für die Anwendung benötigten Funktionen bieten.

4.3.4 Beispiel

Hier wird nun kurz die Anwendung der SIBs an zwei Beispielen gezeigt, einmal ein Protokoll-Test und dann der Aufruf eines externen Befehls.

4.3.4.1 Test des CVS-Protokolls

Betrachten wir den Service-Logic-Graph etwas näher. Zuerst wird ein SocketAddress-Objekt mit den übergegebenen Parametern ***hostname*** und ***port*** erstellt. Dieses wird für die Kommunikation benötigt. Danach wird versucht, den Socket zu diesem durch das SocketAddress-Objekt beschriebenen Endpunkt zu verbinden. Darauf werden die benötigten Streams besorgt. GenCVSRequest generiert eine CVS-spezifische Anfrage, welche

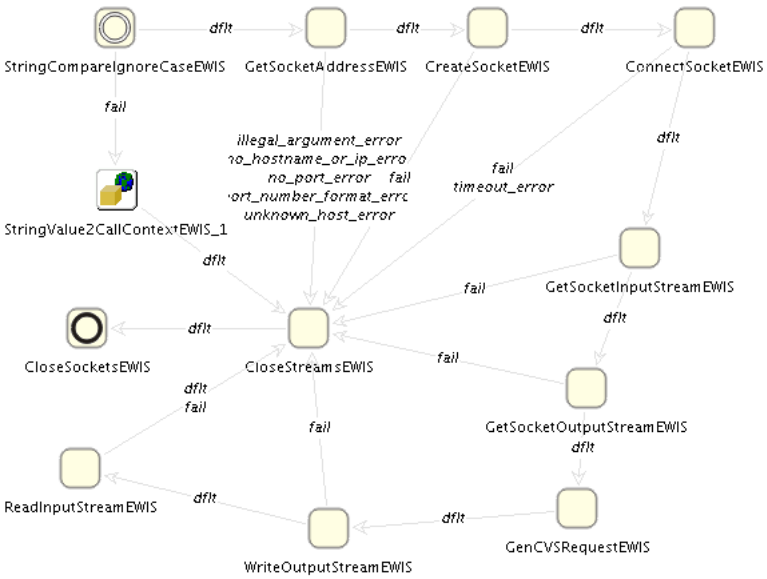

Abbildung 4.15: Sockets, jABC-Graph CVS-Test

dann als Parameter an WriteOutputStream übergeben wird. In diesem SIB wird dann die Anfrage zur anderen Seite geschickt, indem sie in den OuputStream geschrieben wird. Danach schickt die andere Seite eine Antwort, welche mit Hilfe von ReadInputStream gelesen und in einem Vector oder byte[] gespeichert wird. Schlussendlich werden die Streams und der Socket geschlossen. Sollten innerhalb des Workflows Fehler auftreten, wird eine entsprechende Fehlermeldung erzeugt und die Streams als auch der Socket geschlossen. Alle Ausgaben, sprich Fehlermeldung und Antwort, werden als Parameter zurückgegeben und können dann in einer Webseite dargestellt werden.

4.3.4.2 Aufruf eines externen Befehls

Zum Aufruf externer Befehle stehen im Moment folgende dieser Befehle zur Verfügung:

- ping vom System aus, wo die Anwendung läuft, zu einem anderen System
- traceroute vom System aus, wo die Anwendung läuft zu einem anderen System
- netstat auf dem auf dem benutzten System

Nachdem der gewünschte Befehl gefunden wurde, wird zum entsprechenden SIB verzweigt und der Befehl ähnlich wie auf einer Console ausgeführt. Die Antwort wird über einen InputStream, der mit dem Aufruf über die JRE verbunden ist, gelesen und bearbeitet, indem die Antwort zerlegt und in Container-Klassen verteilt wird. Diese und deren Inhalt können dann später in einer Webseite dargestellt werden.

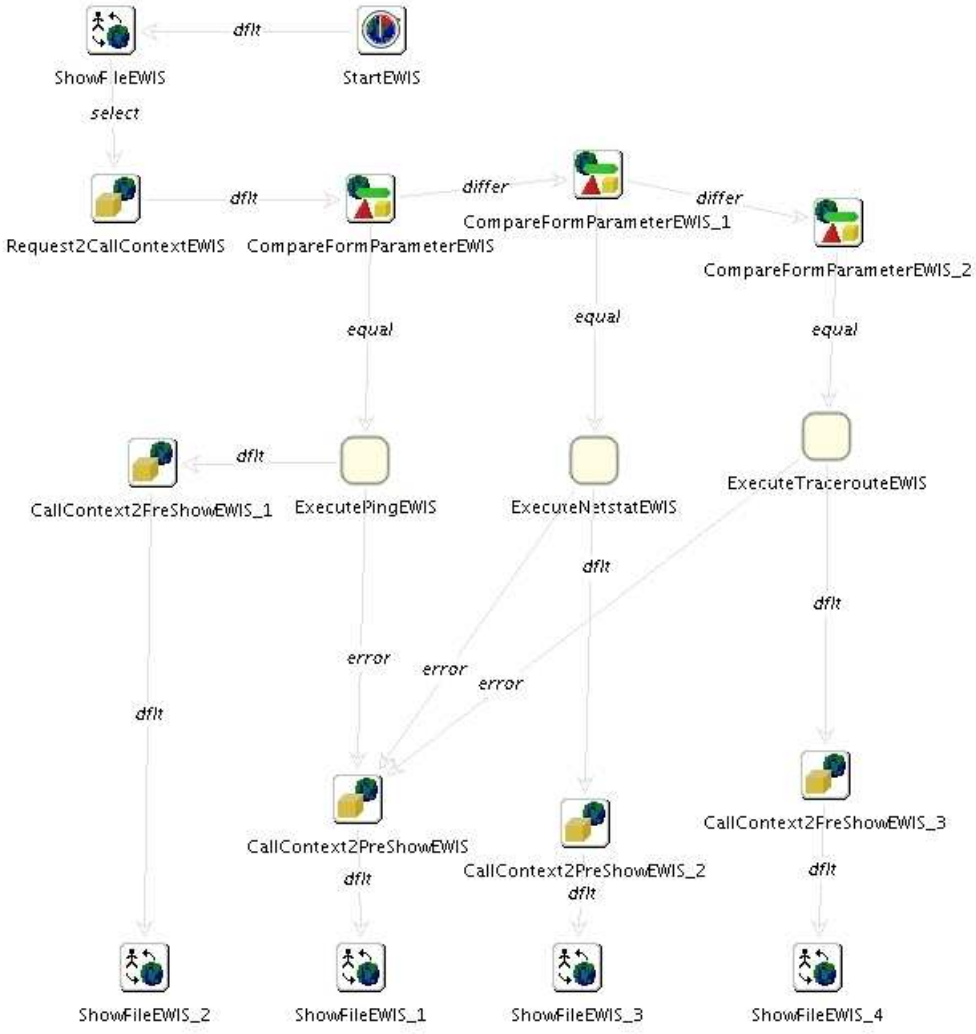

Abbildung 4.16: Sockets, jABC-Graph Aufruf Externer Befehl

4.3.4.3 Ausführung des Beispiels

Um die Beispiele auszuführen, müssen die Quellen installiert sein. Man muss einfach die ServiceTests.xml und TestCVS.xml (für den CVS-Test) oder die ExecutePing.xml (für den Aufruf des externen Befehls) in das jABC laden. Die web.xml muss generiert und die Quellen kompiliert werden. Zum Abschluss noch den ”Service” auf den Application Server der Wahl deployen und mit dem Browser die Startseite ansurfen. Nun kann das Testen beginnen.

4.4 Design und Implementierung einer SIB Bibliothek zur Kommunikation und Verwaltung von LDAP

4.4.1 Einleitung

Für die Verwaltung von großen Datenmengen eignet sich besonders LDAP. Innerhalb der MaTRICS wird eine Anbindung an einen LDAP-Server zur Benutzerauthentifizierung benötigt. Dazu sind spezielle LDAP SIBs zu erstellen, mit denen eine Kommunikation zu einem LDAP-Server möglich ist. Des Weiteren müssen auch Funktionalitäten, wie beispielsweise das Exportieren und Importieren von LDIF-Dateien unterstützt werden.

Für die Umsetzung wird auf das JNDI Framework aufgebaut, zu dem es bereits spezielle SIBs gibt. Die vorhandene SIB-Palette ist nun entsprechend zu ergänzen.

Die Aufgabe besteht im Design einer allgemeinen erweiterbaren SIB-Bibliothek zur Kommunikation und Verwaltung von LDAP.

4.4.2 Design

4.4.2.1 UseCase Diagramm

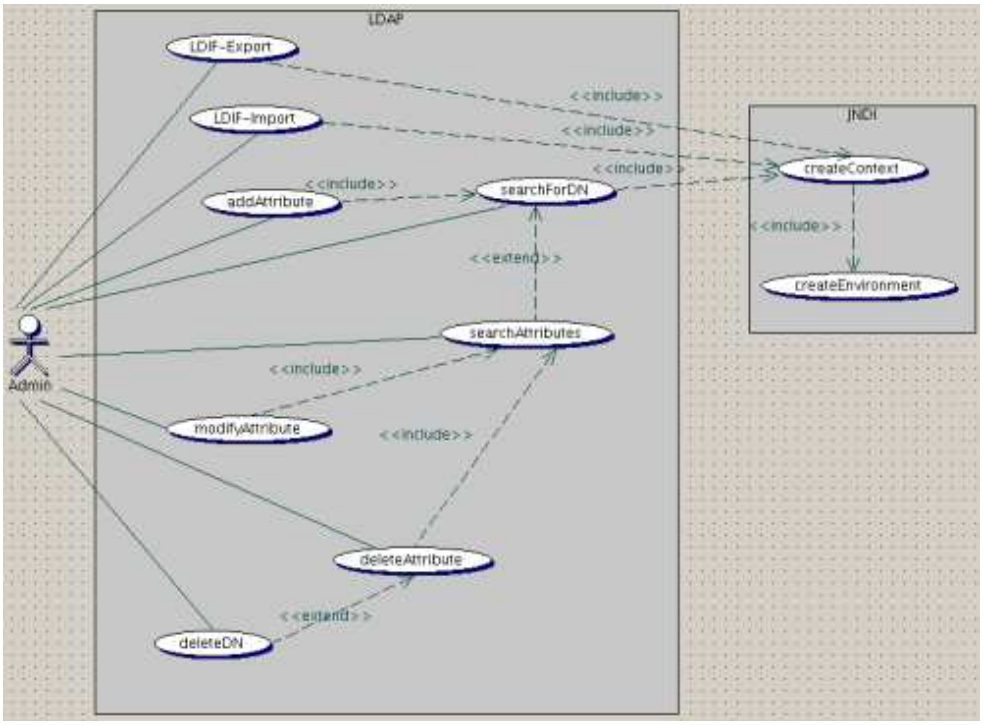

Abbildung 4.17: LDAP, Usecase-Diagramm

Das UseCase Diagramm (Abbildung 4.17) ist in zwei Teile unterteilt. Auf der linken Seite sind alle notwendigen LDAP-Funktionen abgebildet, die ein Admin tätigen kann. Zu diesen gehört:

- LDIF-Export
- LDIF-Import
- addAttribute
- searchForDN
- searchAttributes
- modifyAttribute
- deleteAttribute
- deleteDN

Alle diese LDAP-Funktionen benötigen einen DirContext, der im rechten Teil des UseCase abgebildet wird. Bevor ein DirContext erzeugt wird, benötigt man ein Environment. Das Environment beinhaltet alle nötigen Informationen, um sich an einem LDAP-Server anzumelden. Zu diesen gehören beispielsweise initial context factory, provider url, username und password. Falls keine Veränderungen am Verzeichnisbaum vorgenommen werden soll, sind die letzten beiden Parameter nicht nötig.

Zu einigen Funktionen der ”operation” in Abbildung 4.18:

addAttribute beinhaltet searchForDN, da ohne die DN, der einen „Knoten“ repräsentiert keine Attribute an diesen „Knoten“ hinzugefügt werden können. Des Weiteren erweitert searchAttributes, searchDN. Um ein Attribut zu suchen, muss man erst ein DN finden. modifyAttribute und deleteAttribute beinhaltet searchAttribute, da man erst ein Attribut finden muss, um es zu ändern oder zu löschen. deleteDN ist eine Erweiterung von deleteAttribute, weil beim Löschen eines DN gleichzeitig alle Attribute mit gelöscht werden.

4.4.2.2 Aktivitätsdiagramm

Das Aktivitätsdiagramm (Abbildung 4.18) zeigt einen Ablauf, um eine LDAP-Funktion auszuführen. Dies gilt für alle im UseCase Diagramm beschriebenen LDAP-Funktionen. Als erstes muss der Admin die benötitgen Daten eingeben, die im erzeugten Environment in einer Hashtable gespeichert werden. Im nächsten Schritt wird ein DirContext erzeugt. Alle Operationen arbeiten nun mit diesem erzeugten DirContext. Es können nun mehrere Operationen ausgeführt werden. Wenn alle Operationen ausgeführt wurden, gelangt man zum Endzustand.

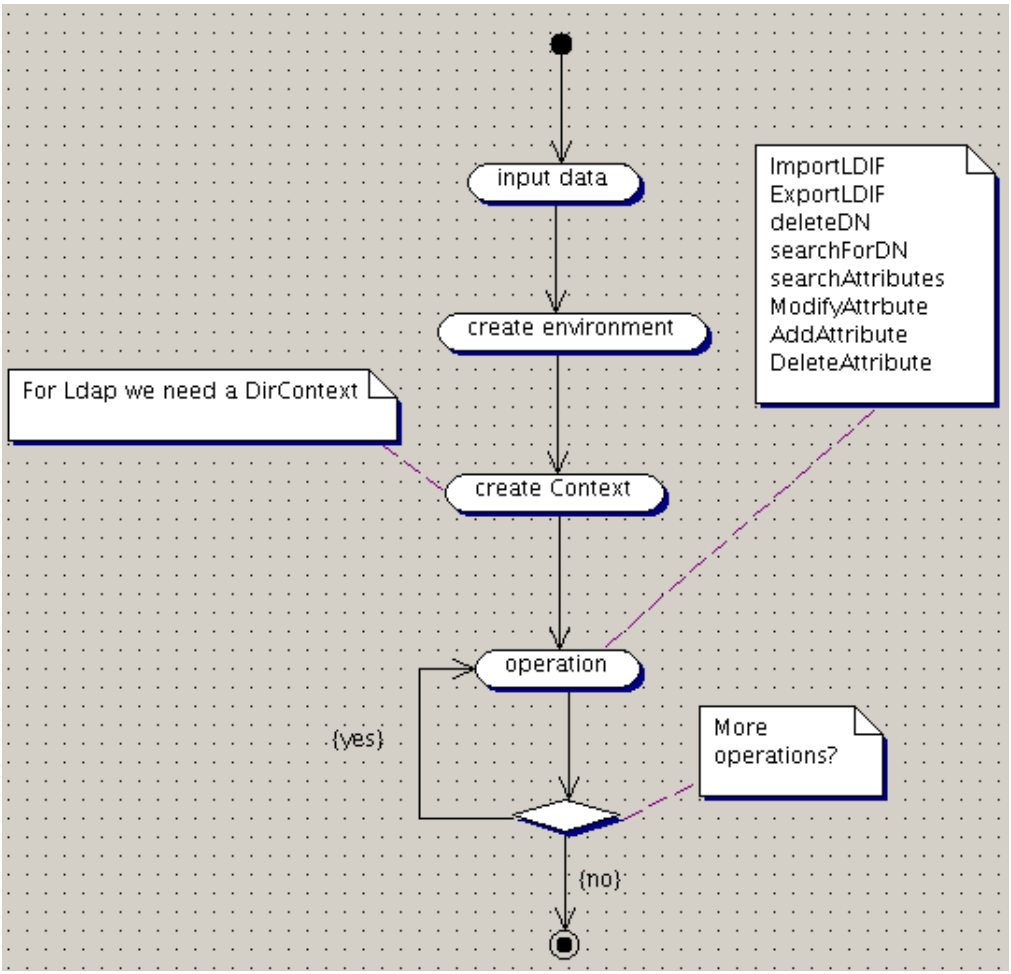

Abbildung 4.18: LDAP, Aktivitätsdiagramm

4.4.3 Implementation

4.4.3.1 Überblick

Um eine LDAP-Anwendung in jABC zu modellieren, hat der Benutzer eine Menge von SIBs zu Verfügung, die mit Hilfe von JNDI realisiert wurden. Es gibt zwei Arten von SIBs. Die erste Gruppe besteht aus zwei SIBs. Es handelt sich dabei um „JNDICreateEnvironmentLDAP“ und „JNDICreateContextLDAP“. Diese sind für das Erzeugen des Environment und den Verbindungsaufbau zu einen LDAP-Server verantwortlich. Hat man dies in seiner Anwendung vorgenommen, kann man die zweite Gruppe von SIBs benutzen, die alle nötigen LDAP-Funktionen bereitstellen.

4.4.3.2 Anwendung

Nachdem wir einen DirContext erzeugt haben und damit die Verbindung zu einem LDAP-Server hergestellt haben, können wir zwischen acht Operationen wählen. Der LDAP-Server zeigt eine Verzeichnisstruktur (directory information tree (DIT)) an, die beispielsweise ein Unternehmen mit all seinen Mitarbeitern repräsentieren könnte. Wir benötigen Operationen, womit wir solche Verzeichnisstrukturen sichern können (LDAP-Export) oder gesicherte Strukturen wieder einspielen können (LDAP-Import). Außerdem benötigen wir einen Suchmechanismus. Wir können z.B. nach einer Person im Unternehmen suchen (searchForDN) oder nach allen Informationen über diese Person (searchAttributes). Des Weiteren muss die Möglichkeit für z.B. Mitarbeiter bestehen, (eigene) Informationen im DIT zu ändern (modifyAttribute), löschen (deleteAttribute) oder neue hinzuzufügen (addAttribute). Verlässt ein Mitarbeiter das Unternehmen, kann der „Knoten“ im Verzeichnisbaum, der diesen Mitarbeiter repäsentiert, mit all seinen Informationen gelöscht werden (deleteDN).

4.4.3.3 Beispielanwendung

In diesem Beispiel soll gezeigt werden, wie der Benutzer einen Eintrag im Verzeichnisbaum löschen kann. Siehe Abbildung 4.19

Nach dem Start-SIB, sehen wir drei SIBs, die für alle Operation in dieser Reihenfolge aufgebaut werden müssen. Der erste SIB (StringValue2CallContext) schreibt alle nötigen Informationen in den CallContext. Das nächste SIB (JNDICreateEnvironmentLDAP) erzeugt aus diesen Informationen ein Environment. Mit diesem Environment können wir uns schließlich einen DirContext erzeugen (JNDICreateContextLDAP). Schließlich kann nun mit diesem DirContext die Operation ausgeführt werden. In diesem Fall wollen wir ein „Knoten“ im DIT löschen.

Um alles anschaulicher zu machen, haben wir die Verzeichnisstruktur in einem LDAP-Browser sichtbar gemacht. Wir sehen hier den markierten „Knoten“ ou=members, den wir löschen möchten. Siehe Abbildung 4.20

Nach der Ausführung des Graphen und damit der Operation, wurde ou=members mit all seinen Kindern gelöscht. Die Verzeichnisstuktur sieht nun folgendermaßen aus, siehe Abbildung 4.21

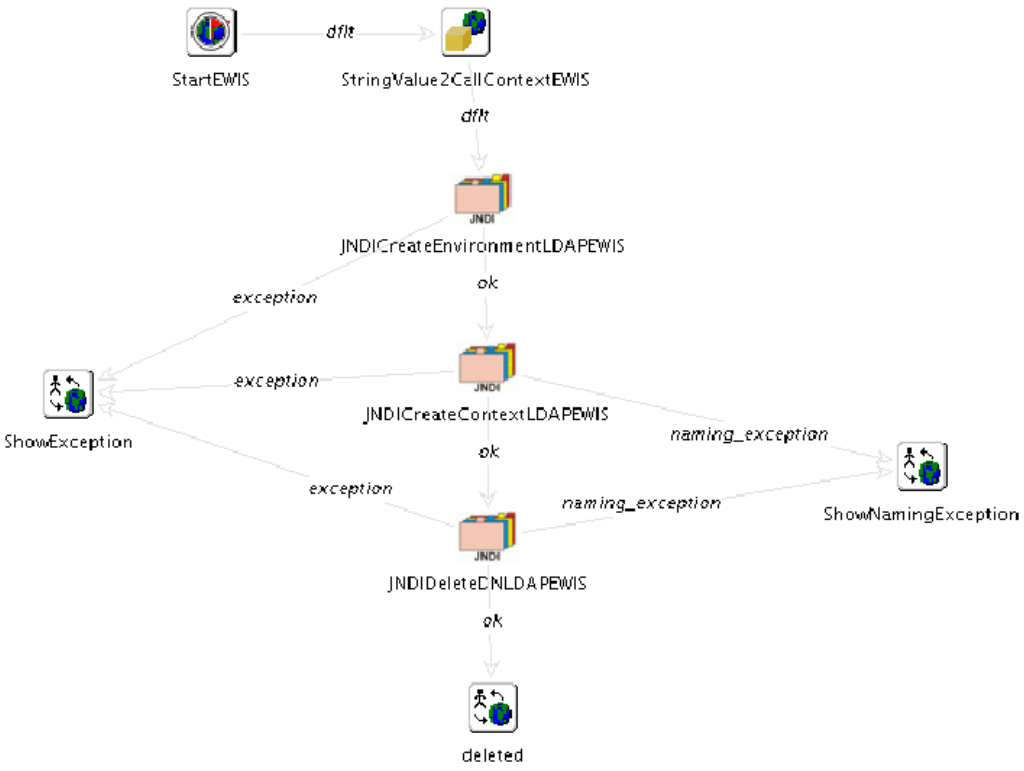

Abbildung 4.19: LDAP, jABC-Graph zum Löschen eines Eintrags

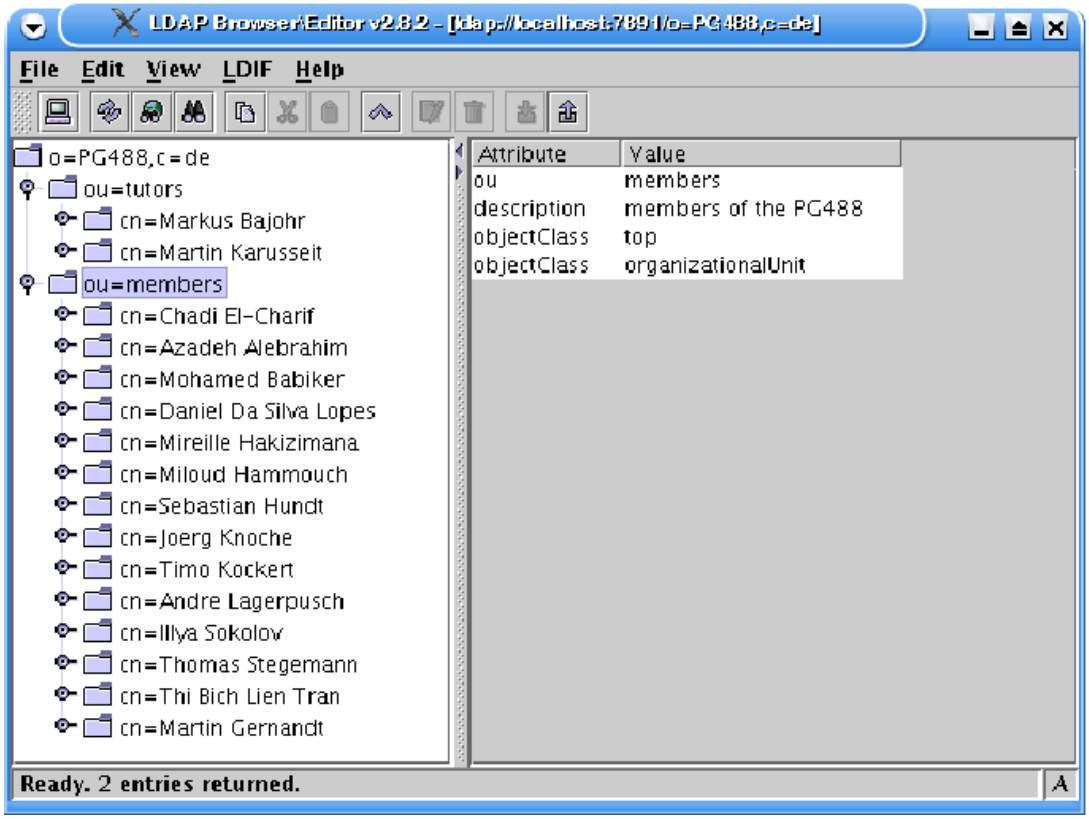

Abbildung 4.20: LDAP, Verzeichnisstruktur vor dem Löschen eines Eintrags

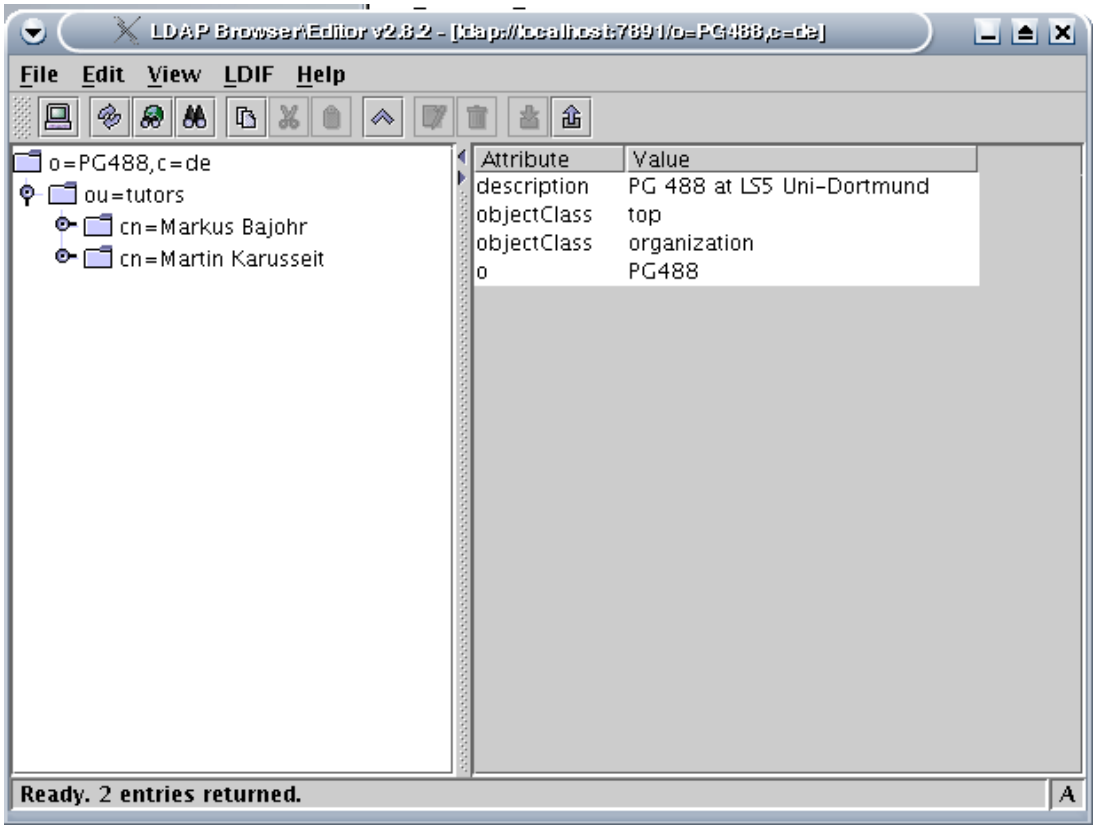

Abbildung 4.21: LDAP, Verzeichnisstruktur nach dem Löschen eines Eintrags

Kapitel 5

Zwischenthemen

Bei den Zwischenthemen sollen die Teilnehmer sich in die Grundkomponente der MaTRICS einarbeiten. Dazu werden die Themen

- Integration einer Notification Komponente
- Erweiterung der ConfigAgents um eine allgemeine GUI-Beschreibungssprache
- Entwicklung einer autonomen Ausführungskomponente für ConfigClients
- Design einer Monitoringbibliothek auf Basis von autonomen ConfigClients

in dreier bzw. vierer Gruppen bearbeitet, wobei die Gruppenaufteilung der Kurzthemen beibehalten wurde. Die Themen beinhalten sowohl die Entwicklung neuer Komponenten und neuer Protokolle für die MaTRICS als auch die Integration vorhandener Komponenten in die MaTRICS. Hier werden auch Komponenten verwendet, die im Rahmen der Kurzthemen entwickelt wurden. Zuerst wird das Design mit Hilfe von UML erstellt. Darauf aufbauend werden dann die Komponenten entwickelt und in die MaTRICS integriert, und schliesslich muss noch eine englische Dokumentation erstellt werden. Dafür stehen insgesamt 12 Wochen zur Verfügung. Im Folgendem ist der Stand der Entwicklung nach den ersten fünf Wochen festgehalten, in dieser Zeit haben die Gruppen sich auf das Design und die Einarbeitung in die Komponenten der MaTRICS konzentriert.

5.1 Integration einer Notification Komponente

5.1.1 Einleitung

Ziel dieser Aufgabe ist es, den Diensten die Möglichkeit zu bieten, bei Fehlern oder wichtigen Ereignissen den Administratoren Mitteilungen zu schicken. Da der Bearbeitungszeitpunkt der Jobs in der MaTRICS nicht vorhergesagt werden kann und die Administratoren

bei Fehlern trotzdem schnell eingreifen müssen, sollen die Dienste der MaTRICS Mitteilungen per E-Mail oder SMS versenden können. Diese Mitteilungen enthalten neben einer reinen Textnachricht auch die Priorität der Mitteilung - von Information bis fataler Fehler - den Absender, also den Namen des Dienstes und den ConfigClient, bei dem das Ereignis aufgetreten ist. Der Dienst schreibt seine Mitteilung in eine Message-Queue. Für die Weiterverarbeitung und den Versand ist ein eigener Dienst zuständig. Die Administratoren können die Dienste, die ConfigClients und die Priorität der Mitteilungen, die sie erhalten wollen, einstellen. Sie können auch später noch alle Mitteilungen anschauen. Zusätzlich soll der ConfigManager über das ConfigClientProtocol die Möglichkeit erhalten, mit ConfigClients über Bluetooth zu kommunizieren.

Es wird also

- ein Dienst zum Versenden der Mitteilungen
- ein Dienst zum Einstellen der Optionen zum Versand und zum Anzeigen aller Mitteilungen
- eine Änderung im ConfigClientProtocol zur Verwendung von Bluetooth

benötigt.

5.1.2 Design

5.1.2.1 Anwendungsfalldiagramm: Notification Dienst

Mit dem Notification Dienst können die Benutzer der MaTRICS sich für Benachrichtigungen anmelden, die nach der Abarbeiten eines Jobs in der JobFlowEngine verschickt werden. Ein Benutzer kann durch Wahl des ConfigClients, des Services und der Priorität der Benachrichtigung bestimmen, welche Benachrichtigungen er erhält. Er kann sich die Meldungen anschauen und die Abonnements der Benachrichtigung verwalten, also neue Abonnements hinzufügen, sich alle vorhandenen anschauen und einzeln löschen oder ändern. Ein weiterer Mitspieler ist der Notification Manager, ein Dienst in der MaTRICS. Er erzeugt die Meldungen - schreibt sie in eine Queue - und verarbeitet sie später weiter. Dafür liest er sie aus der Queue, speichert sie in der Datenbank und schickt sie per SMS oder EMail an die Benutzer, die sie abonniert haben.

5.1.2.2 Aktivitätsdiagramm: Send Notification

In diesem Kapitel wird die Bearbeitung beim Versand von `Notification`s beschrieben. Dabei gibt es drei nebenläufige Vorgänge. Sie beginnen alle drei beim Hochfahren der Plattform und enden, wenn die Plattform gestoppt wird.

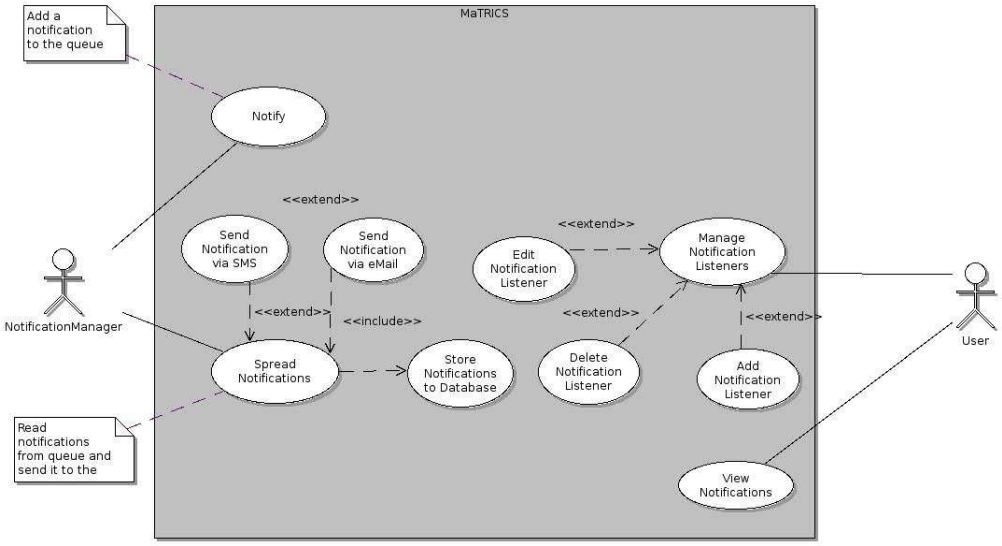

Abbildung 5.1: Notification, Usecase-Diagramm

Beim Vorgang ”Notification Generation” werden die `Notification`s erzeugt. Wenn innerhalb der Plattform ein Fehler auftritt oder ein anderes Ereignis, das dem Administrator angezeigt werden soll, wird eine `Notification` erzeugt und in die `NotificationQueue` geschrieben. Danach kann der Prozess seine normale Verarbeitung fortsetzen.

Beim Vorgang ”Process Notification” werden `Notification`s empfangen und weitergeleitet. Bevor die `Notification`s gelesen werden können muss der `NotificationManager` initialisiert werden - die `NotificationQueue` wird erzeugt und die Datenbankverbindung wird geöffnet. Sobald eine `Notification` in die `NotificationQueue` geschrieben wird, wird sie bei diesem Vorgang aus der `NotificationQueue` geholt und in die Datenbank geschrieben. Dann wird für diese `Notification` eine Liste von `NotificationReceivern` aus der Datenbank gelesen und für jeden `NotificationReceiver` wird eine `NotificationMessage` mit der `Notification` erzeugt und in die entsprechende `NotificationMessageQueue` geschrieben.

Beim Vorgang ”Process Message” werden die `Notification`s an die `NotificationReceiver` gesendet. Bevor die `Notification`s verschickt werden können, muss der `MessageManager` initialisiert werden - die `MessageQueue` wird erzeugt und die Schnittstellen zum Versenden der `Message`s werden geöffnet. Sobald eine `Message` in die `NotificationMessageQueue` geschrieben wird, wird sie bei diesem Vorgang aus der `NotificationMessageQueue` geholt und an die `NotificationReceiver` per SMS oder E-Mail verschickt.

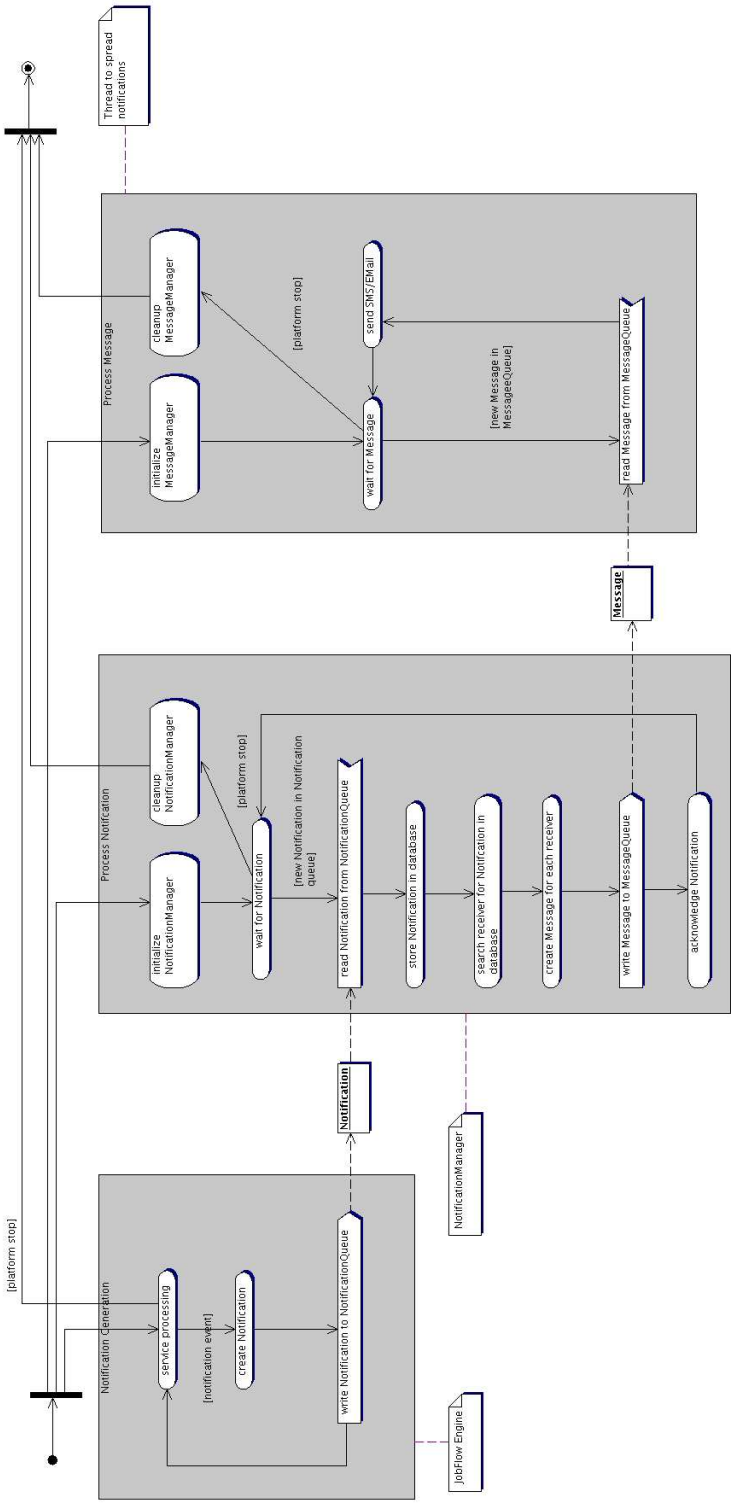

Abbildung 5.2: Notification, Aktivitätsdiagramm: Send Notification

5.1.3 Aktivitätsdiagramm: Notification Management

Das Aktivitätsdiagramm Notification Management beschreibt die Verwaltung von Abonnements. Zu Beginn muss der Benutzer sich einloggen. Dann kann er sich die Meldungen anschauen, neue Abonnements erzeugen, oder die vorhandene Abonnements löschen oder ändern. Beim Anschauen der Meldungen wählt der Benutzer einen Filter und erhält eine gefilterte Liste von Meldungen. Beim Erzeugen eines Abonnements wählt der Benutzer einen ConfigClient, einen Service, einen Notificationlevel und die Art der Benachrichtigung. Diese Werte werden im neuen Abonnement gespeichert. Beim Löschen wählt der Benutzer ein Abbonnement aus, das gelöscht werden soll. Beim Ändern kann der Benutzer den ConfigClient, den Service den Notificationlevel und die Art der Benachrichtigung ändern.

5.1.4 Implementierung

5.1.4.1 Notification Komponente

In diesem Kapitel werden alle SIBs aufgelistet, die verändert oder neu hinzugefügt wurden. Für eine ausführliche Beschreibung wird das HOWTO des Themas empfohlen.

SIBs zum Verwalten der Abonnements der Benutzer:

- `NotificationMGR_CreateNotificationTask`,
- `NotificationMGR_DeleteNotification`
- `NotificationMGR_DeleteNotificationTask`
- `NotificationMGR_GetNotifications`
- `NotificationMGR_GetNotificationTasks`
- `NotificationMGR_ModifyNotificationTask`
- `NotificationMGR_StoreNotification`.

SIBs zum Versenden der Meldungen:

- `NotificationMGR_CompareNotificationWithSubscriptions`
- `NotificationMGR_EnqueueNotification`
- `NotificationMGR_GetNotificationFromQueue`
- `NotificationMGR_GetNotificationTaskByID`
- `NotificationMGR_GetUserFromNotificationTask` and

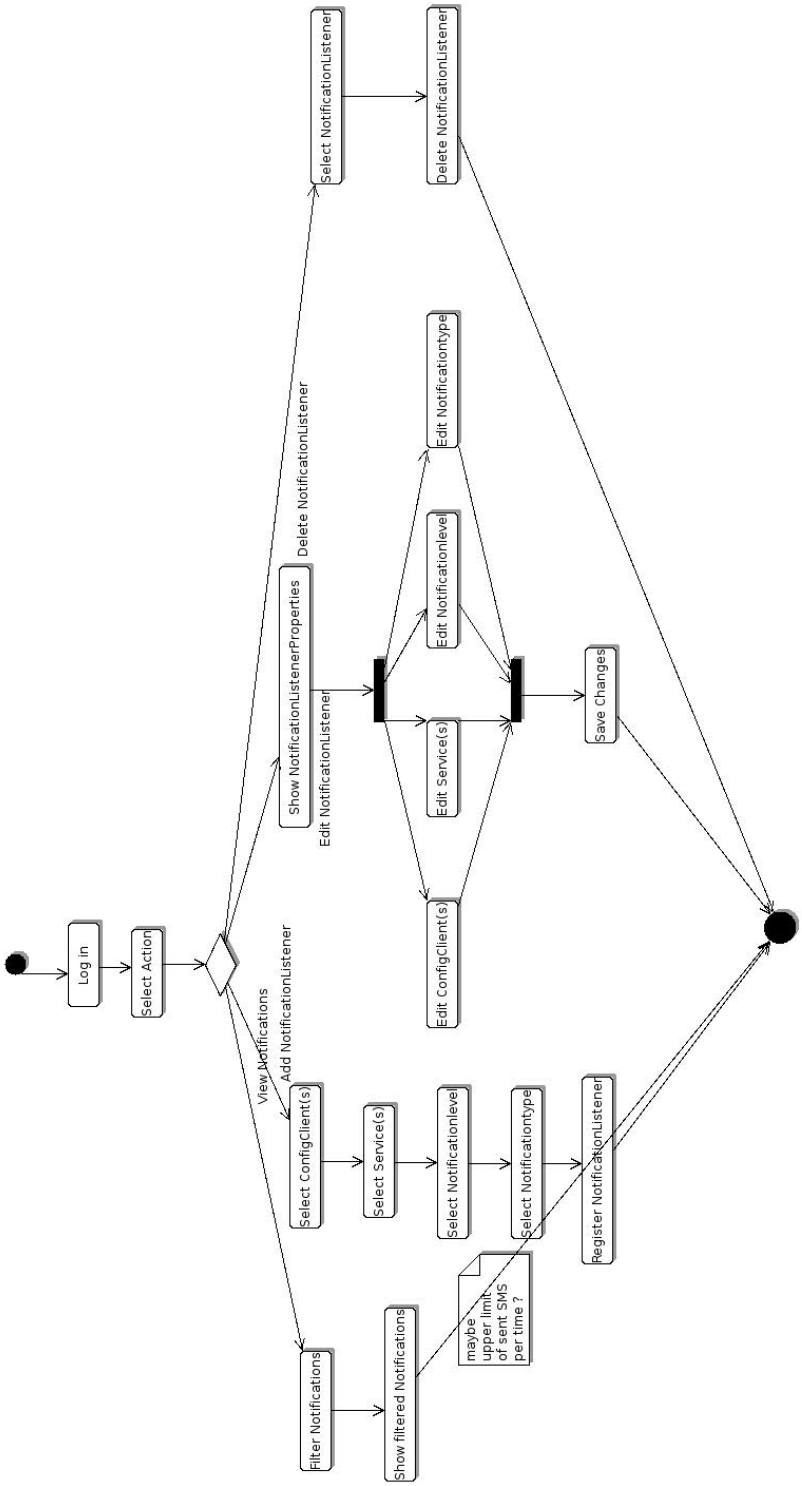

Abbildung 5.3: Notification, Aktivitätsdiagramm: Notification Management

- NotificationMGR_Status2Notification.

Wichtige Klassen:

- Notification, entspricht einer Meldung
- NotificationTask, entspricht einem Abonnement
- NotificationMessage, entspricht einer Nachricht
- NotificationManager, zentrale Implementierung

Für die Verwaltung der Notifications werden zwei zusätzliche Tabellen in der Datenbank benötigt. In der Tabelle **notification** werden alle Notifications gespeichert, die aufgetreten sind.

- id: Eindeutige Id der Notification.
- date: Wann ist das Ereignis aufgetreten.
- configclient: ConfigClient, bei dem das Ereignis aufgetreten ist.
- service: Dienst, bei dem das Ereignis aufgetreten ist.
- level: Prioritat des Ereignisses - DEBUG, INFO, WARNING, ERROR oder FATAL.
- message: Die Beschreibung des Events.

In der Tabelle **notificationtask** wird gespeichert, für welche Notifications sich die Benutzer registriert haben.

- id: Eindeutige Id der Registrierung
- matricsuser: Account des Benutzers
- service: Dienste, für die dem Benutzer Notifications geschickt werden.
- configclient: ConfigClients, für die dem Benutzer Notifications geschickt werden.
- level: Art der Ereignisse, die dem Benutzer Notifications geschickt werden - DEBUG, INFO, WARNING, ERROR oder FATAL.
- contacts: Adresse, an die die Notifications geschickt werden.
- type: Bestimmt, wie die Notifications geschickt werden - per SMS oder E-Mail.

5.1.4.2 Notification senden

Das ConvertAndStore-Makro, Abbildung 5.4, verschickt zu dem Rückgabewert eines Jobs eine Nachricht. Zuerst wird aus dem Status eine `Notification` erzeugt. Diese wird über die JMS-Queue an den `NotificationManager` gesendet und dann in die Datenbank gespeichert.

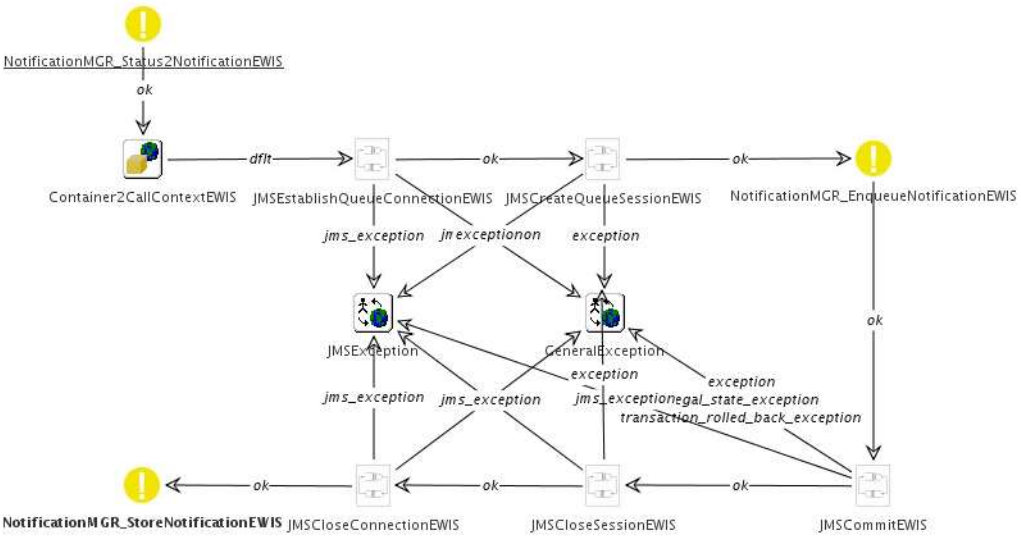

Abbildung 5.4: Notification, jABC Graph - Macro ConvertAndStore

5.1.4.3 ClientProtocolManager

Der `ClientProtocolManager` wird um die Protokolle Bluetooth und OBEX erweitert. Dazu werden die Klassen `ClientProtocolManagerBluetooth` und `ClientProtocolManagerOBEX`, die die Schnittstelle vom `ClientProtocolManager` implementieren, hinzugefügt und beim `ClientProtocolManager` eingetragen. Das HostManagement wird um den Datensatz `BluetoothInfo` erweitert. Dieser enthält eine Bluetooth- bzw. OBEX-Adresse. Jedem `ConfigClient` kann im HostManagement eine Bluetooth-Adresse und eine OBEX-Adresse zugeordnet werden und das Bluetooth oder OBEX-Protokoll kann als Standard Protokoll für diesen Client eingetragen werden. Ausserdem kann im HostManagement nach Bluetooth-Geräten und ihren Diensten gesucht werden, hier wird auch die Adresse der Dienste ermittelt, die im BluetoothInfo eingetragen werden muss.

Zum Testen des Bluetooth-Protokolls kann auf dem ConfigClient die Beispielapplikation zur Bluetooth SIB Bibliothek BluChat gestartet werden. Zum Testen des OBEX-Protokolls kann auf dem ConfigClient ein obexserver gestartet werden. Genaueres steht in dem HOWTO zu diesem Thema.

5.1.5 Ausblick

Damit die Notification Komponente auch verwendet wird, müssen `Notification`s in der MaTRICS erzeugt werden. Die wichtigste Anwendung ist die Information der Administratoren bei Fehlern bei der Konfiguration der ConfigClients. Also wird der JobManager so erweitert, dass er bei Fehlern auch `Notification`s erzeugt. Aber auch andere Dienste können die Notifcation Komponente verwenden.

5.2 Erweiterung der ConfigAgents um eine allgemeine GUI Beschreibungssprache

5.2.1 Einleitung

Die Kommunikation der Agents mit dem ConfigManager findet über das MaTRICS Manager Protokoll (MMP) statt. Da das MMP keine Beschreibungsinformationen für die späteren Darstellungen beim ConfigAgent enthält, werden separate Templates benötigt. Somit müssen bei einer Änderung eines Konfigurationsdienstes auch sämtliche Templates angepasst werden. Dabei haben die verschiedenen Ausgabekanäle auch unterschiedliche Darstellungseigenschaften, die der Serviceentwickler beachten muss. Beispielsweise lässt sich eine Tabelle in HTML relativ einfach darstellen, wohingegen dies bei einer SMS nicht ohne weiteres möglich ist.
Eine am Lehrstuhl entwickelte Visualisierungskomponente auf Basis von XML erlaubt es, die Darstellung unabhängig von einem Ausgabeformat zu beschreiben. Dazu wird eine spezielle Beschreibungssprache, die GUI-Description Language eingesetzt, mit der eine GUI für unterschiedliche Ausgabekanäle optimal gerendert werden kann.
Interaktive Kanäle, wie beispielsweise HTML oder Swing, erlauben es eine Benutzereingabe entgegenzunehmen, die dann von der Visualisierungskomponente geparst wird, so dass die entsprechenden Eingaben als Key/Value-Paar von der Dienst-Logik gelesen werden kann.

5.2.2 Aufgabenbeschreibung

- Die Aufgabe besteht zum einen darin, die Visualisierungs-Komponente in den ConfigManager und den ConfigAgents zu integrieren. Dabei sind die Frame-Beschreibungen (XML) via MMP zu kapseln und an die ConfigAgents zu übertragen. Das Rendering eines Frames delegiert der ConfigAgent an seine lokale Visualisierungs-Instanz, so dass die gerenderte Ausgabe von der Dienst-Logik des Agents weiterverarbeitet werden kann.
- Der zweite Teil der Aufgabe besteht in der Entwicklung eines EmailAgents, der zur Erzeugung und Verarbeitung der Konfigurationsanfrage via Email die integrierte Visualisierungskomponente nutzt. Dazu wird der bereits bestehende WebAgent als Grundlage verwendet.

5.2.3 Design

5.2.3.1 Anwendungsfalldiagramm

Das Anwendungsfalldiagramm besteht aus zwei Teilen(ConfigAgent level und Service logic level). Damit der ConfigAgent mit jXML arbeiten kann, muss dieser erst initialisiert

werden. Die Initialisierung des ConfigAgent beinhaltet 3 Schritte. Zuerst wird ein Controller im Client-Modus erzeugt und die Stylesheet angegeben. Die Stylesheets enthalten Templates für die Erzeugung des Ausgabeformats. Zum Schluß wird ein Client hinzugefügt. Der nächste Anwendungsfall auf der ConfigAgent Ebene ist die Generierung der Ausgabe. Dafür wird ein Format ausgewählt. Man kann zwischen HTML und Mail wählen, entsprechend den eingegebenen Stylesheets. Außerdem kann der ConfigAgent den Request eines Clients parsen und in eine XML-Datei schreiben. Diese Datei (XML-DL) ist nach der GUI-Description-Language aufgebaut. Die XML-DL wird schließlich zur ConfigManager Seite gesendet.
Auf der Serverseite(Service logic level) werden die Variablen aus dem XML-DL extrahiert. Eine neue XML-DL für die HTML Seite oder Text für Mail, mit eventuellen Variablen von der ConfigAgent Seite, wird erzeugt und an den ConfigAgent geschickt. Die Initialisierung der Serverseite beinhaltet nur das erzeugen eines Controller im Server-Modus. (Siehe Abb. 5.5)

5.2.3.2 Sequenzdiagramm

InitializeGUIengine-server Bevor die GUI-Description-Engine auf der Serverseite verwendet werden kann, muss diese initialisiert werden. Dazu muss ein Controller im Server-Modus erzeugt werden. (Siehe Abb. 5.6)

InitializeGUIengine-client Ähnlich sieht es auch auf der Clientseite aus. Zuerst muss ein Controller erzeugt werden und die zu verwendenden Stylesheets und die Clients angemeldet werden. (Siehe Abb. 5.7)

GenerateOutput Die Abbildung 5.8 zeigt die Generierung des Zielformats. Hier existieren zwei GUI-Description-Engines, zwischen denen XML-basierte Dokumente ausgetauscht werden, die der GUI-DL entsprechen. Die GUI-Description-Engine auf der Serverseite ruft die Methode *generateXML* im Controller auf. Der Controller der Engine liefert die benötigte Informationen zurück. Die Engine sendet das XML-Dokument über MMP an die Engine auf der Clientseite(WebAgent). Diese Engine ruft die Methode *generateOutput* des Controllers auf. Die Informationen werden aus dem XML-Dokument extrahiert und die Ausgabe wird in Form eines Objekts an die Engine zurückgesendet, welche die Ausgabe an den User sendet.

HandleRequest Die Abbildung 5.9 zeigt die Verarbeitung eines Requests. Der User sendet einen Request an den Agent. Dieser ruft die Methode *parseToXML* im Controller des Clients auf. Das zurückgelieferte XML-Dokument wird über MMP an den ConfigManager geschickt, der die Methode *parseToVars* im Controller-Server aufruft. Diese Methode extrahiert aus dem XML-Dokument die Variablen und liefert ein Objekt mit den Variablen und weiteren Informationen zurück.

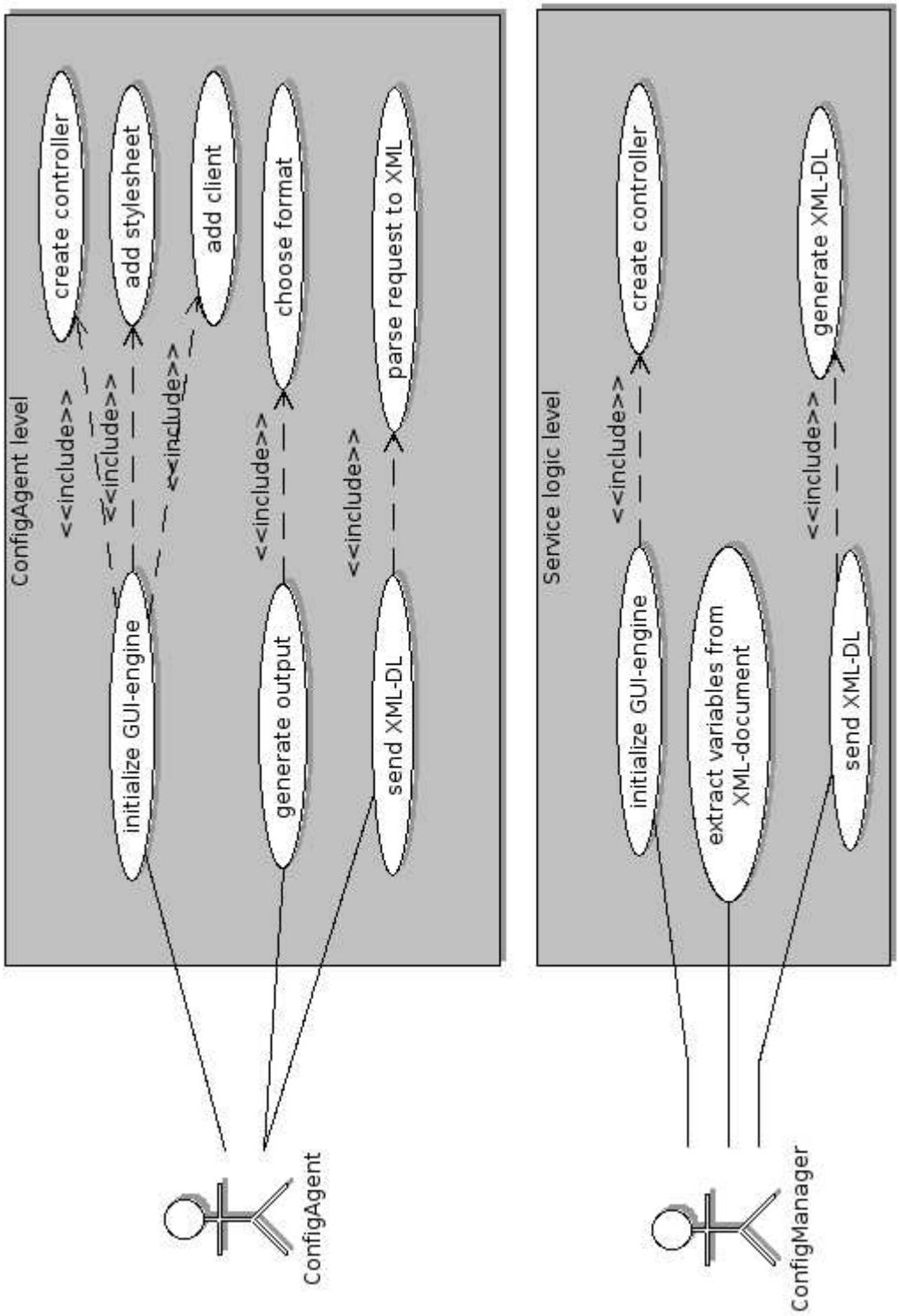

Abbildung 5.5: GUI, Usecase-Diagramm

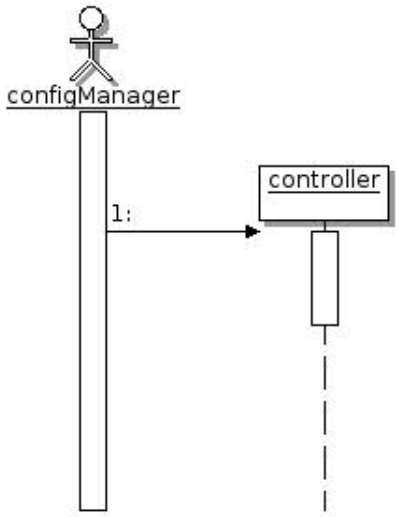

Abbildung 5.6: GUI, initializeGUIengine-server

5.2.4 Implementierung und Test

In diesem Kapitel werden alle SIBs aufgelistet, die verändert oder neu hinzugefügt wurden. Für eine ausführliche Beschreibung wird das HOWTO des Themas empfohlen.

SIBs innerhalb von jXMLGui:

- `InitjXMLGuiControllerEWIS`
- `InitjXMLGui_AddClientEWIS`
- `InitjXMLGui_AddStyleSheetEWIS`
- `StringInArrayEWIS`
- `jXMLGui_GetKeyOfValueFromMapEWIS`
- `jXMLGui_GetAllKeysOfValueFromMapEWIS`
- `XMLFile2XMLStringEWIS`
- `XMLString2XMLFileEWIS`
- `jXMLGui_ParseToXMLEWIS`

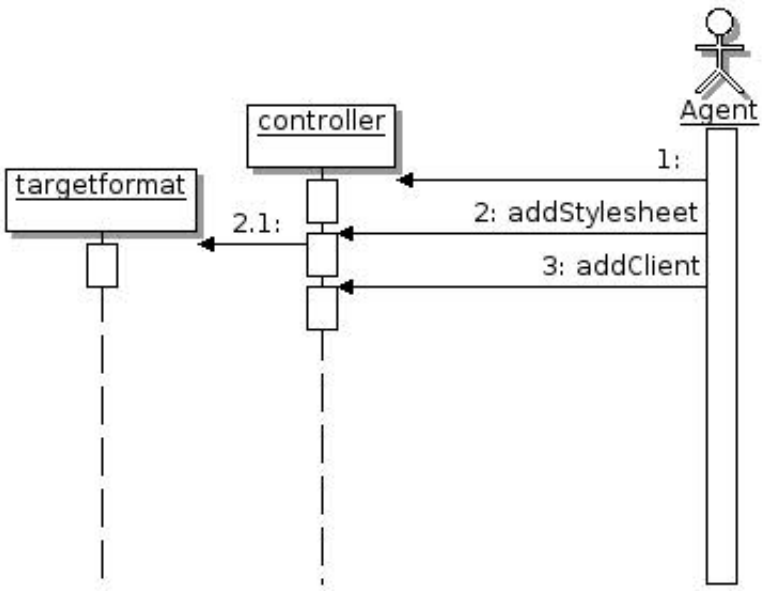

Abbildung 5.7: GUI, initializeGUIengine-client

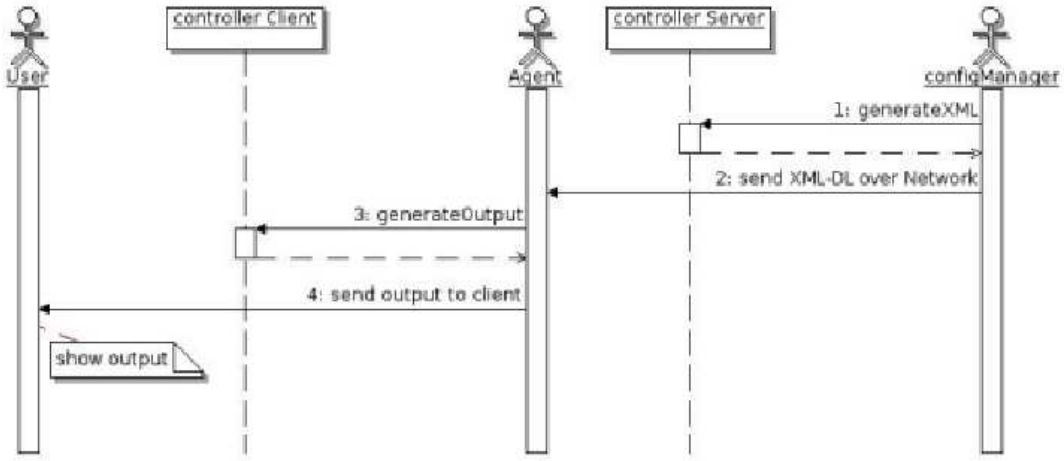

Abbildung 5.8: GUI, generateOutput

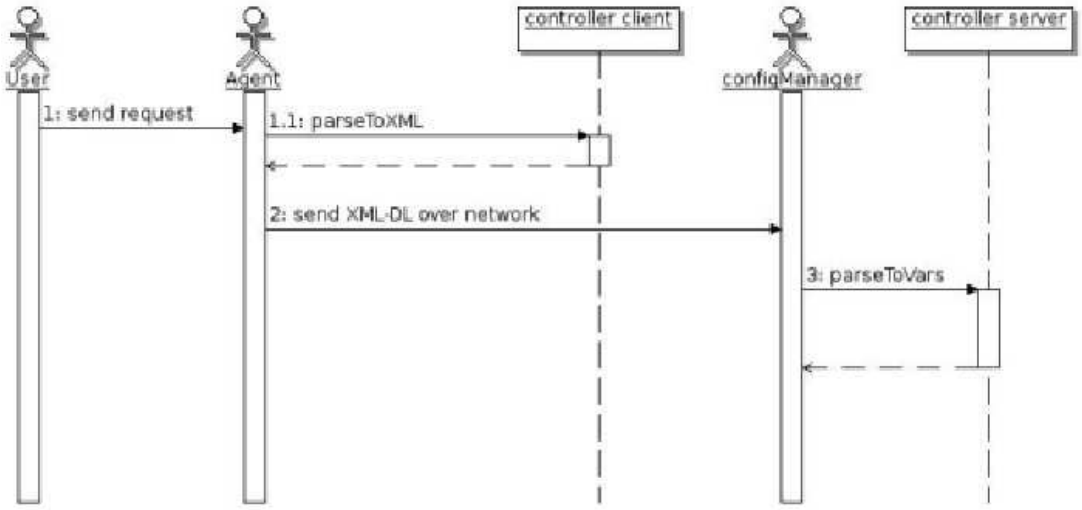

Abbildung 5.9: GUI, handleRequest

- `jXMLGui_CreateIOContainerEWIS`
- `jXMLGui_CreateIOContainer2EWIS`
- `jXMLGui_CreateSessionEWIS`
- `jXMLGui_GenerateOutputEWIS`
- `jXMLGui_GenerateOutput2EWIS`
- `jXMLGui_GenerateXMLEWIS`
- `jXMLGui_HashMap2CallContextEWIS`
- `jXMLGui_Parameters2HashMapEWIS`
- `jXMLGui_ParseToVarsEWIS`
- `jXMLGui_getMessageContentEWIS`
- `jXMLGui_getParametersEWIS`
- `jXMLGui_parseRequestEWIS`
- `selectBranchEWIS`

SIBs innerhalb der MaTRICS:

- `MMPVersionEWIS`

- `xmlStringFromMMP2CallContextEWIS`
- `TransmitMMPVersion2EWIS`

5.2.5 WebAgent

Hier wird die Erweiterung des WebAgent um eine allgemeine GUI-Beschreibungssprache erläutert. Es wird angenommen, dass die Funktionsweise vom WebAgent bekannt ist, daher wird nur auf die Unterschiede zu der Version ohne GUI-Beschreibungssprache eingegangen. Mehr Information über den WebAgent findet man in der HowTo ”WebAgent”.
Im WebAgent-Graph wird mit der Kante init_service zusätzlich noch die jXMLGui-Engine initialisiert. Es wird ein Controller erzeugt, ein Stylesheet ausgewählt und ein Client hinzugefügt. (Siehe Abb.5.10). Durch den Controller im Container erzeugt der SIB jXMLGui_CreateSessionEwis eine Session, die von der jXMLGui-Engine verwendet wird. Dieser Vorgang wird in der Abb.5.11 gezeigt. Bei jedem Aufruf von RootEntry wird diese Session an den ConfigManager gesendet.
Nachdem der WebAgent ein MMP-Objekt zurückbekommt, schaut er nach der Version (MMPVersionEWIS). Es gibt hier 2 Möglichkeiten. Wenn es sich um die Version eins handelt, wird der Branch ”html” ausgeführt. Wenn es Version 2 ist bedeutet dies, dass der WebAgent ein Template im XML-Format zurückbekommen hat. In diesem Fall wird dann der Branch ”xml” ausgeführt. Der nächste SIB XmlStringFromMMP2CallContextEwis nimmt den XML-String und andere Parameter vom MMP-Objekt und legt diese in den CallContext. Im SIB jXMLGui_GenerateOutput2EWIS wird eine HTML-Seite erzeugt, die dem SIB ShowStringEWIS_MMP übergeben wird. Dieser SIB zeigt diese erzeugte HTML-Seite dem Benutzer an. Siehe Abb.5.12. Nach der Interaktion mit dem Benutzer wird ein IOContainer erzeugt. Im nächsten SIB jXMLGui_ParseToXMLEWIS wird ein XML-String mit all den Informationen vom Benutzer erzeugt. Siehe Abb.5.13. Der XML-String wird dem SIB RMIInvokeMethodeWithParameters übergeben und die Methode ”executeNextStep” mit dem XML-String als Parameter wird ausgeführt. Der Service wird aufgerufen und der WebAgent wartet auf ein neues MMP-Objekt.

5.2.6 EmailAgent

Der EmailAgent wurde in zwei Graphen realisiert. Der erste Graph dient der Initialisierung. Es wird die Verbindung zum senden und empfangen von Emails initialsiert, sowie die Gui-Engine (im Client-Modus), RMI für die Kommunikation zwischen Agents und ConfigManager und ein Thread, der regelmäßig läuft und den Emaileingang überprüft (Abb.5.14).

Der zweite eigentliche Graph (Service) kann in vier Teile unterteilt werden. Im ersten Teil (Abb.5.15) wird nach Aktivierung des Threads (Thread_setStateEWIS) eine POP-Verbindung aufgebaut (OpenConnectionEWIS_pop3) und alle vorhandenen Mails vom

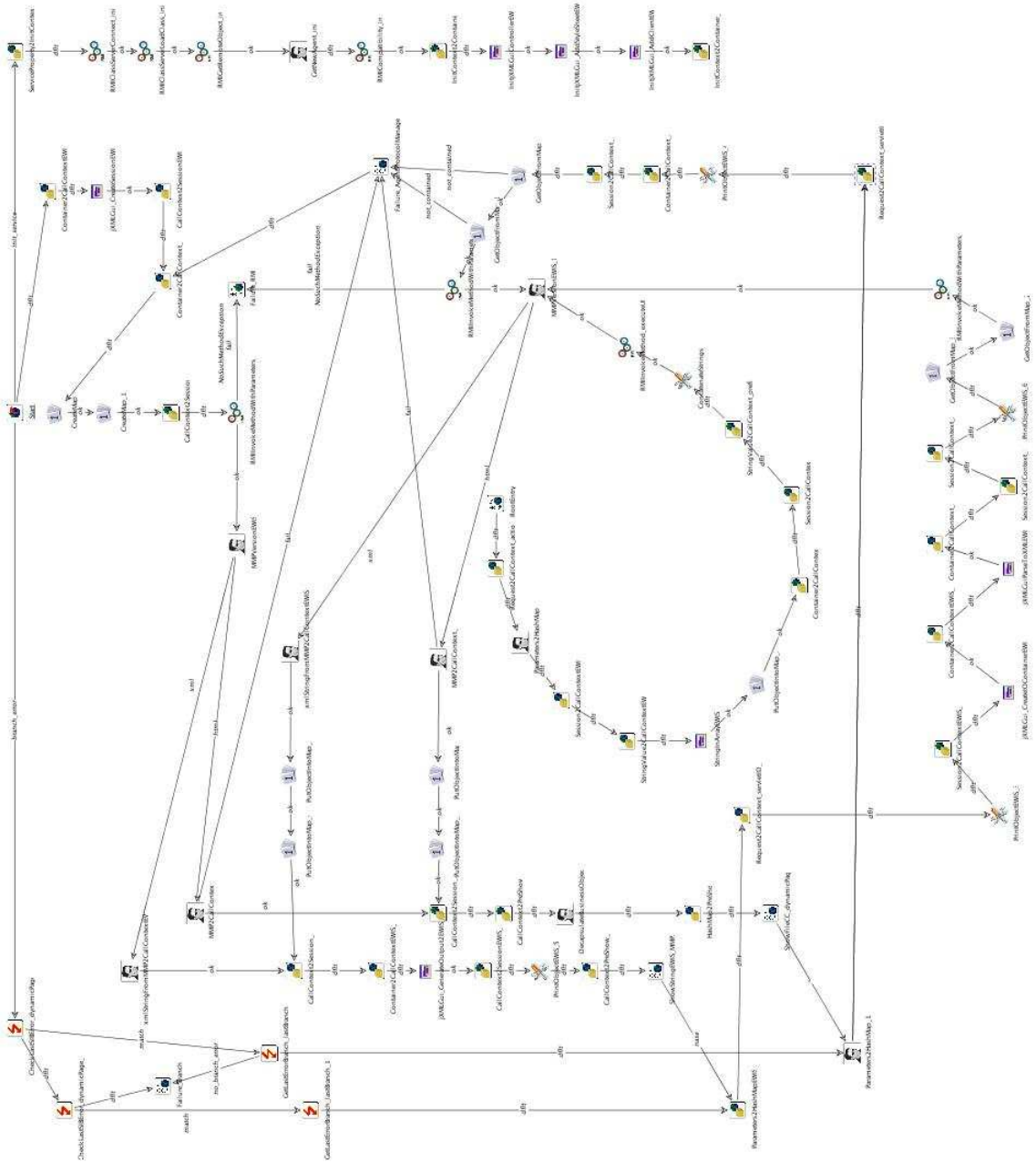

Abbildung 5.10: GUI, WebAgent

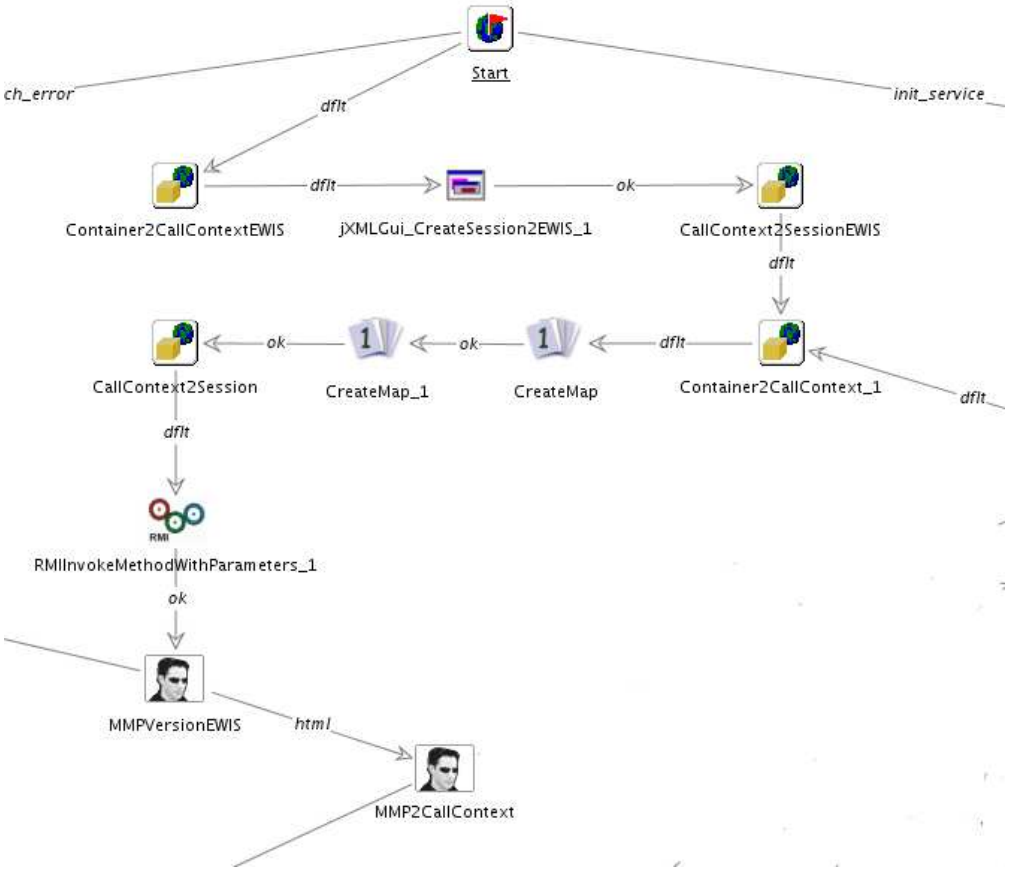

Abbildung 5.11: GUI, WebAgent:Nach der Initialisierung

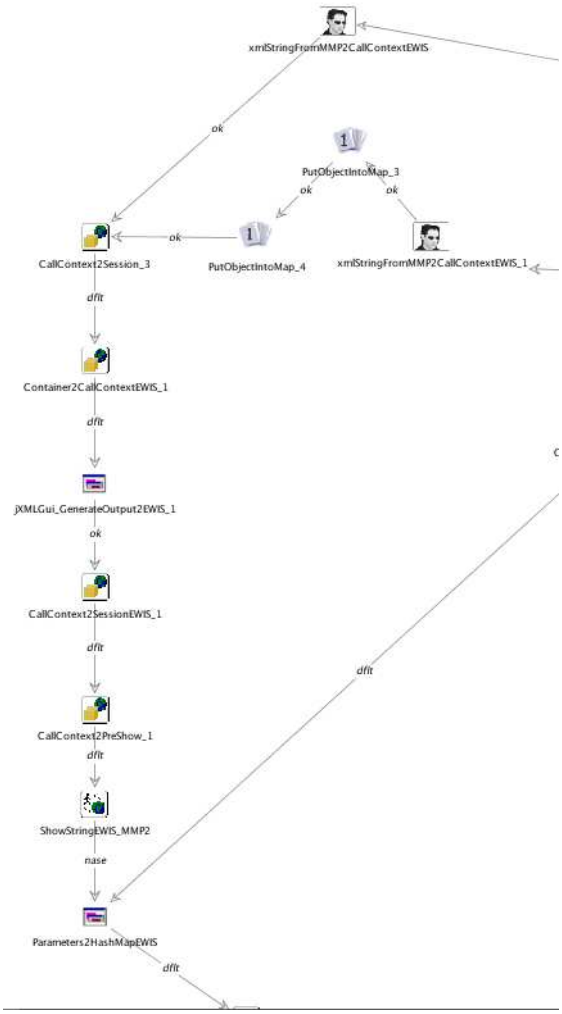

Abbildung 5.12: GUI, WebAgent:Generierung der Ausgabe

Abbildung 5.13: GUI, WebAgent:Parst den Request und übergibt ein xml-String

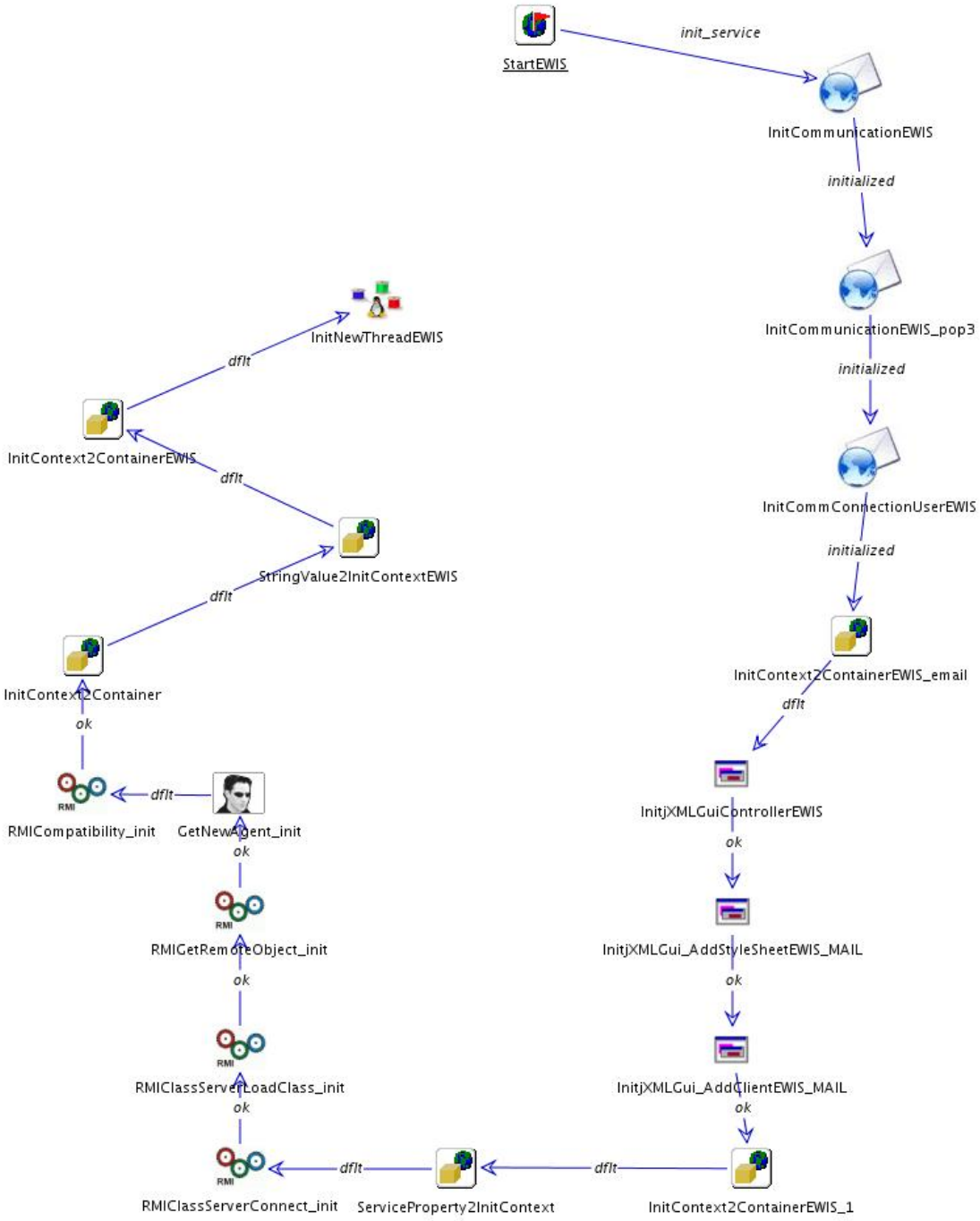

Abbildung 5.14: GUI, EmailAgent: Init service

inbox folder auf dem Server als ein message array geholt und separat entnommen (LoopOverMessageArrayEWIS). Diese werden dann in den CallContext gelegt. Gibt es keine neuen Nachrichten, wird im SIB JXMLGui_getMessageContentEWIS der fail-branch durchlaufen. Die Verbindung wird dann geschlossen und der Thread beendet. Nach einer bestimmten Zeit, die im Intervall angegeben ist, startet der Thread wieder neu.

Gibt es jedoch eine Nachricht, wird im SIB jXMLGui_getMessageContentEWIS die ok-Kante durchlaufen (Abb.5.16) und der Body der Mail und die Emailadresse vom Client zurückgegeben. Danach wird der inbox folder und die POP-Verbindung geschlossen. Aus dem Body der Email wird ein IOContainer erzeugt (jXMLGui_CreateIOContainerEWIS) und an jXMLGuiParseToXMLEWIS übergeben. Wenn dies die erste erhaltene Mail ist, wird die fail-Kante durchlaufen, weil diese Mail nicht dem jXML-Standard entspricht. In der fail-Kante wird in RMIInvokeMethodWithParameters_1 die Methode executeInitialStep aufgerufen. Dies wird benötigt, um an die Parameter vom MMP-Objekt zu kommen. Die erste Seite, die eigentlich für den WebAgent gedacht ist wird verworfen und in jXMLGui_GenerateOutputEWIS eine feste Seite, die Startseite erzeugt. Diese Datei wird in ReadTextFileContentEWIS eingelesen. Danach wird eine SMTP-Verbindung zum Senden von Mails hergestellt (OpenConnectionEWIS), eine Mail zusammengebaut, gesendet und die SMTP-Verbindung wieder geschlossen. Der Thread wird wieder beendet und legt sich für eine kurze Zeit wieder schlafen. Der EmailAgent schickt die erste Mail an den Client und bietet seine Dienste an, die der EmailAgent unterstützt. Somit kann der Client nun diesen Dienst nutzen. Wenn der Client auf diese Mail antwortet sowie einen Dienst gewählt hat, wird beim nächsten parsen in jXMLGuiParseToXMLEWIS die ok-Kante durchlaufen (Abb.5.17) und der Eingabewert in eine Hashmap gelegt (jXMLGui_ParseToVarsEWIS). Nach einem Check, ob dies eine Antwort ist (in CompareStringsEWIS_5) wird die equals-Kante durchlaufen. Auf die erste Mail (mit Auswahl des Dienstes) wird ähnlich wie beim WebAgent executeURL ausgeführt und der Client bekommt die erste Mail von der ConfigManager Seite zurück. Ab da an wird nun beim Antworten der Mail bei CompareStringsEWIS_5 nur noch die not_equal-Kante ausgeführt (Abb.5.18). In RMIInvokeMethodWithParameters_2 wird die Methode executeNextStep ausgeführt und der Client bekommt vom ConfigManager die nächste Mail zu der ausgewählten Auswahl im Dienst zurück. Schickt der Client eine falsche oder neue Mail, so bekommt er wieder die Startseite zurück.

5.2.7 Anwendungsbeispiel

Das Anwendungsbeispiel ist ein einfacher Dienst namens ”GUI_DL_Test”. Dieser Dienst bietet dem Benutzer ein Textfeld und einen Button an. Gibt der Benutzer z.B. seinen Namen ein und klickt auf den Button, bekommt der Benutzer eine Welcome-Seite mit seiner zuvor eingegebenen Eingabe zu sehen. Die Abb. 5.19 zeigt den jABC-Graph zu diesem Dienst. Am Anfang wird der Controller für die Gui-Engine initialisiert und in einem Container gespeichert. Nach der Initialisierung wird der vom WebAgent gesendete xml-String durch das SIB Request2CallContext in den CallContext gelegt (WebAgent erzeugt am An-

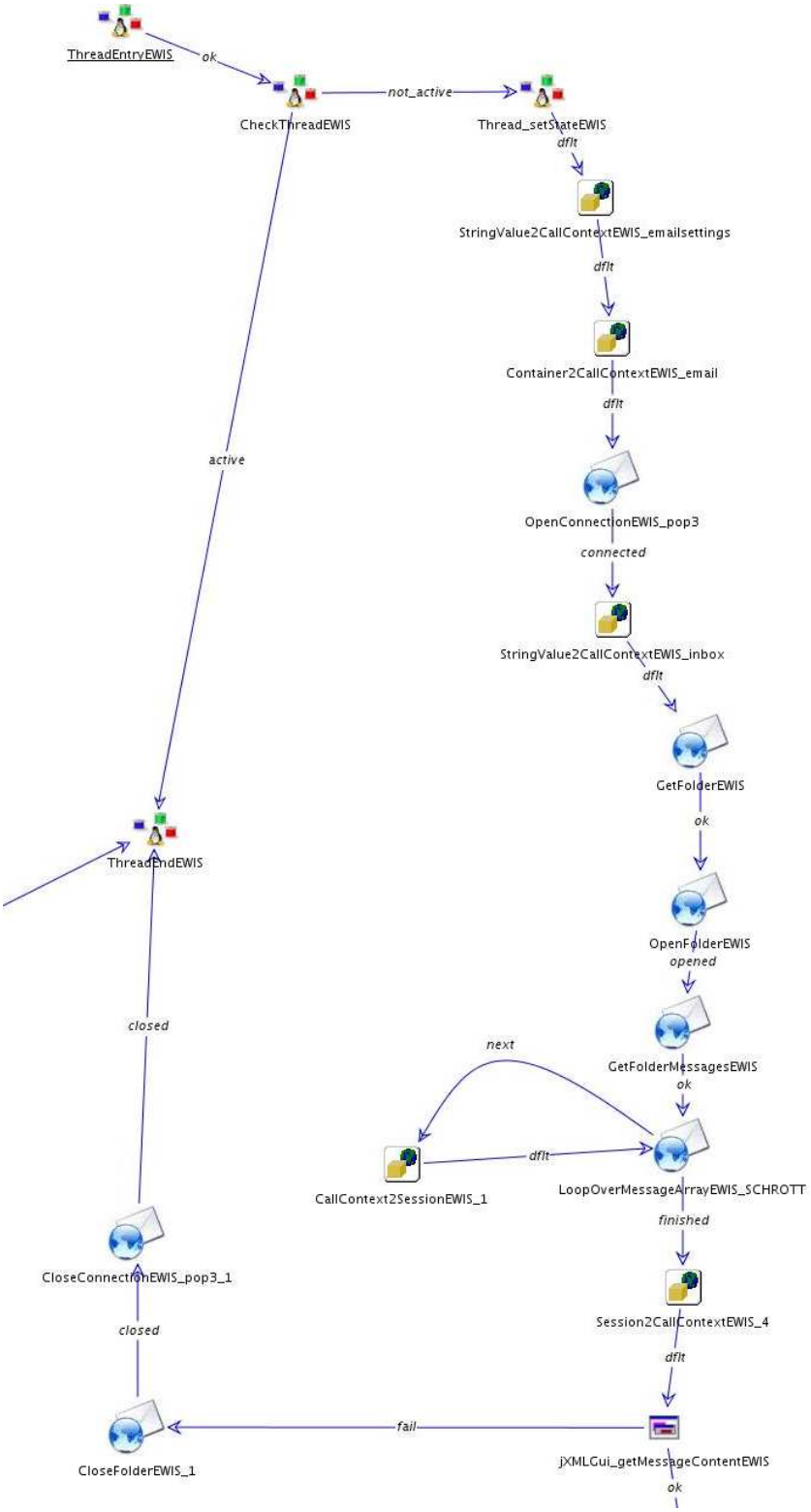

Abbildung 5.15: GUI, EmailAgent: Kreislauf zum schauen nach neuen Mails

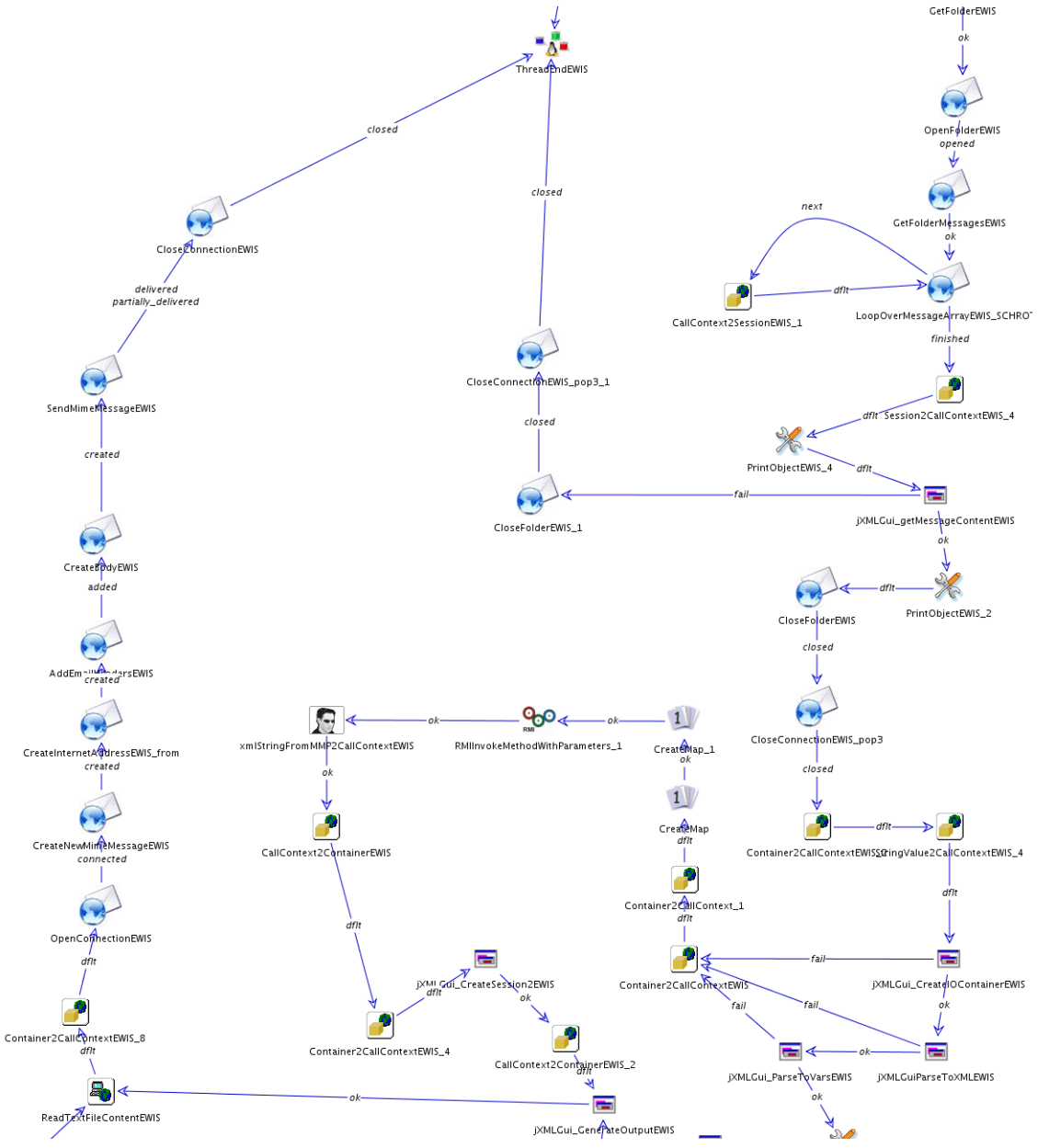

Abbildung 5.16: GUI, EmailAgent: Erste Mail

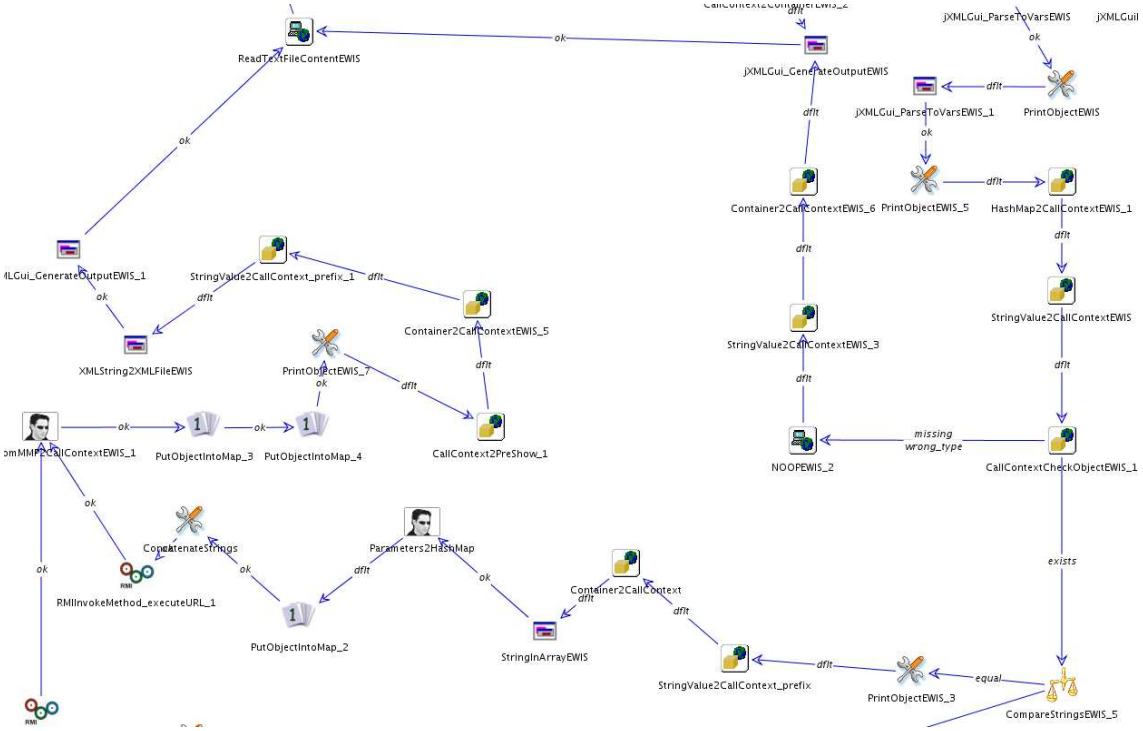

Abbildung 5.17: GUI, EmailAgent: Antworten auf die erste Mail und Auswahl des Dienstes

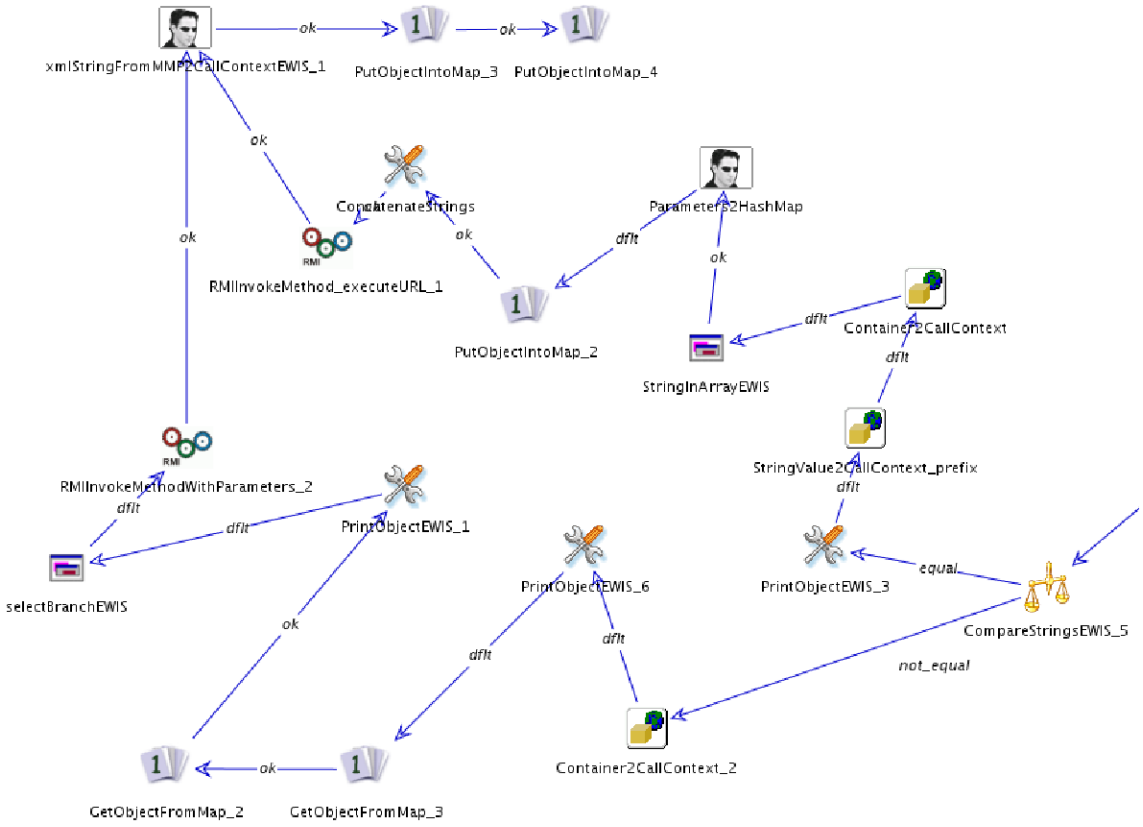

Abbildung 5.18: GUI, EmailAgent: executeNextStep

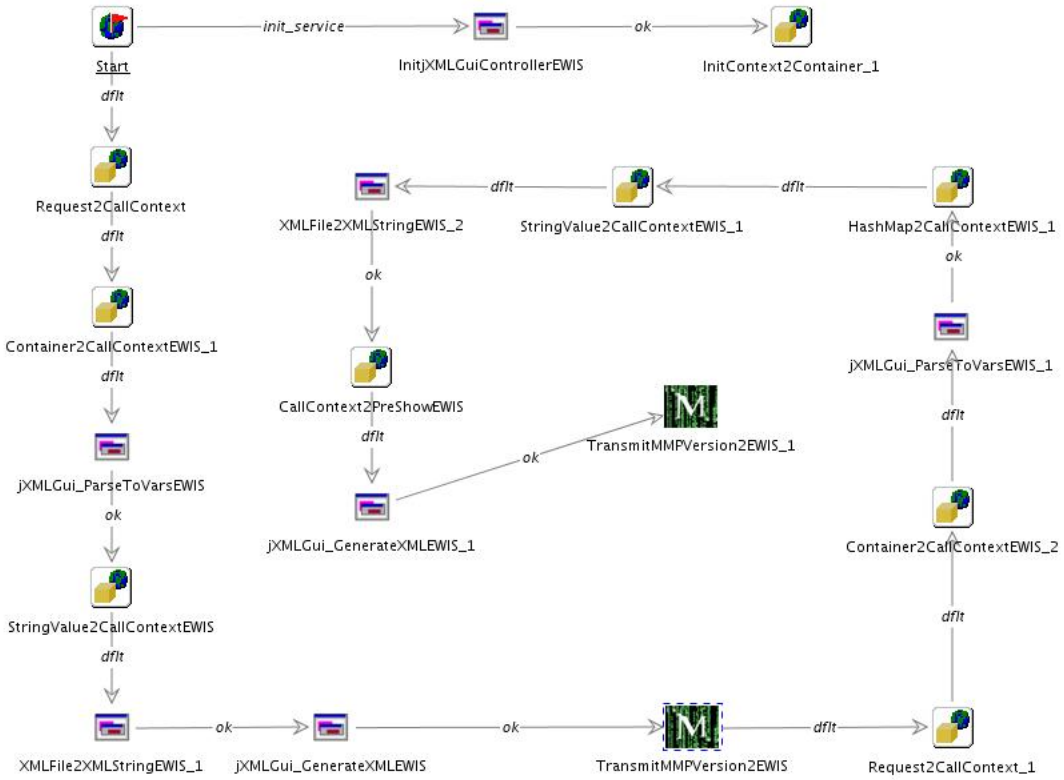

Abbildung 5.19: GUI, GUI_DL_Test Graph

fang eine Session für die Gui-Engine und sendet diese an den ConfigManager). Das SIB jXMLGUI_ParseToVarsEWIS parst den xml-String und gibt Session, Transaction, Client und HashMap mit allen Eingabedaten vom Benutzer zurück und tut diese in den CallContext. Das SIB XMLFile2XMLStringEWIS_1 wandelt ein xml-File in einen xml-String um, weil die Engine nur mit Strings arbeitet. Das nächste SIB jXMLGui_GenerateXMLEWIS generiert daraus einen neuen xml-String, dass Session, Transaction und den Client beinhaltet. Dieser xml-String wird über MMP an die Agents gesendet. Der WebAgent erzeugt mit diesem String eine HTML-Seite und der EmailAgent einen Text für Email. Nach der Interaktion mit dem Benuzer wird eine ähnliche Sequenz ausgeführt. Die Eingabedaten werden mit dem SIB HashMap2CallContextEWIS in den CallContext und dann mit dem SIB CallContext2Preshow in das Preshow gelegt. In diesem Test wird die Velocity-Variable &name durch den Namen vom Benutzer auf der ersten Seite ersetzt.

5.3 Entwicklung einer autonomen Ausführungskomponente für ConfigClients

5.3.1 Einleitung

Unsere Aufgabe besteht darin, ein neues Protokoll MCP (MaTRICS Communication Protocol) zu entwerfen und zu implementieren. Dieses Protokoll soll zur Kommunikation zwischen MaTRICS und ConfigClient eingesetzt werden. Im Zuge dessen mussten wir auch einen autonomen ConfigClient entwerfen und realisieren. Zu Beginn haben wir uns die Funktionsweise der MaTRICS näher angeschaut, und alle Teile der MaTRICS die durch unsere Aufgabe geändert werden mussten ausfindig gemacht werden. Da wir innerhalb der MaTRICS die ankommenden MCP-Pakete weiterverteilen sollen, mussten wir uns mit dem JMS (JAVA Messaging System) auseinandersetzen. Hierbei fiel unsere Wahl auf die Topics, da wir eine n-zu-m Kommunikation realisieren wollten.

5.3.2 Aufgabenbeschreibung

Die Ziele dieses Zwischenthemas sind:

- Das zu entwerfende Protokoll MCP soll synchron und paketorientiert funktionieren. Alle möglichen Objekte wie Jobs, Dateien oder Klassen sollen über dieses Protokoll zwischen MaTRICS und ConfigClient ausgetauscht werden können.
- Der autonome ConfigClient soll auf jeden Fall über MCP mit der MaTRICS kommunizieren können. Er soll selbständig nach neuen MCP-Paketen schauen, diese entpaken und entsprechend dem gesetzten Typ des Inhalts weiterverarbeiten. Gegebenenfalls soll er dann auch MCP-Pakete an die MaTRICS schicken können.
- Die MaTRICS soll so erweitert werden, dass sie auf Basis von JMS-Topic und JBoss die empfangenen MCP-Pakete für alle Services zur Verfügung stellen kann.

Die anderen Gruppen, die in ihren entsprechenden Projekten auf das MCP-Protokoll zurückgreifen und angewiesen sind, um Nachrichten zwischen ConfigClient und MaTRICS auszutauschen, werden ab hier dann in dem JMS-Topic nachschauen können, ob Nachrichten für sie vorhanden sind.

5.3.3 Design

5.3.3.1 ConfigClient/ConfigManager Use Case Diagramm

Im Use Case Diagramm in Abbildung 5.20 wird gezeigt, wie der ConfigClient und der Configmanager zusammenarbeiten. Beide Seiten können sich ein MCP-Object erstellen,

um einen bestimmten Inhalt zu verschicken und den Inhalt eines empfangenen MCP-Objectes auszupacken. Senden und empfangen von MCP-Paketen kann remote ausgeführt werden. Beiden Seiten stehen verschiedene Typklassen zur Verfügung, um den Inhalt eines MCP-Objektes genau zu spezifizieren. Dies ist notwendig für das MCP-Protokoll, da der ConfigClient selbstständig Jobs ausführen, oder den Inhalt an die entsprechende Stelle der Festplatte schreiben soll.

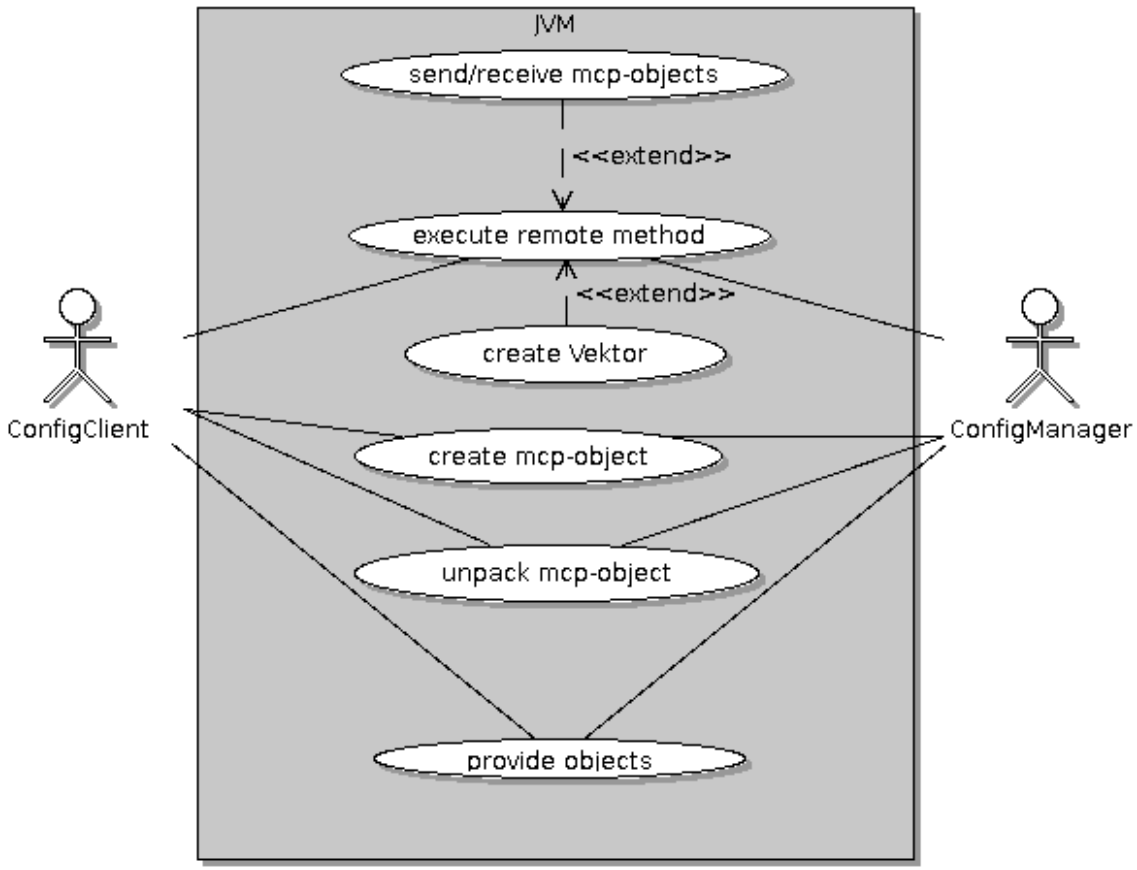

Abbildung 5.20: Ausführungskomponente, Usecase-Diagramm Configclient/Configmanager

5.3.3.2 MCPServer Use Case Diagram

Das MCPServer Use Case Diagramm in Abbildung 5.21 zeigt alle Operationen, die der ClientProtocolManagerMCP unterstützt. Dies sind: execute job, undo job, check if client alive, put configfile to client, fetch data from configfile on client und fetch configfile from client. Der ClientProtocolManagerMCP benutzt das MCP-Protokoll, um diese Operationen auszuführen.

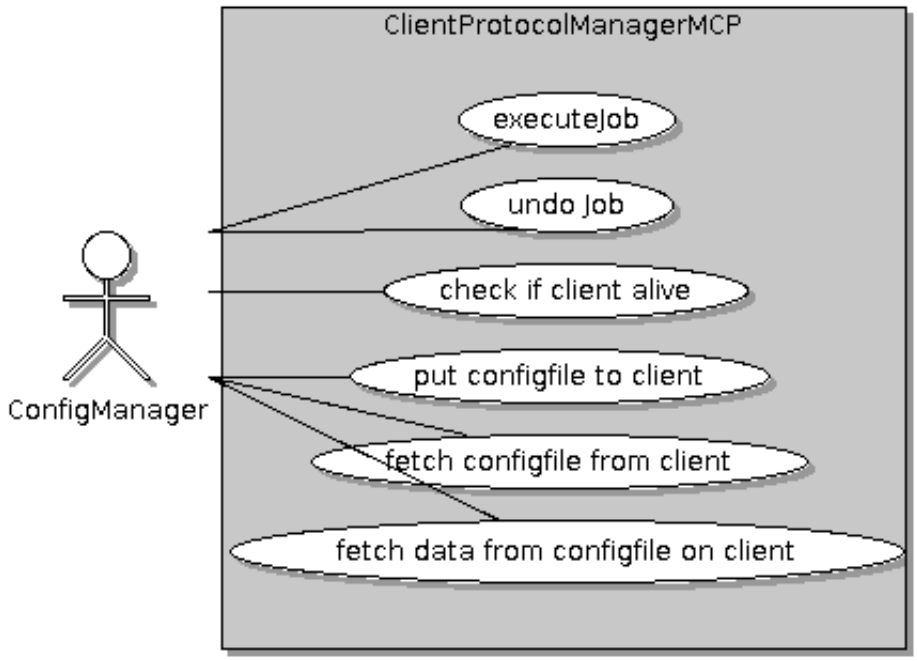

Abbildung 5.21: Ausführungskomponente, Usecase-Diagramm MCPServer

5.3.3.3 Sequenz Diagramm ExecuteJob

Das Jobmanagement in Abbildung 5.22 holt sich den nächsten Task und den ConfigClient anhand der ID und der Queue um die Jobs zu empfangen. Dann wird die Methode ExecuteJob vom ClientProtocolManagerMCP aufgerufen. Diese Methode packt die Kommandosequenz des Jobs in das MCP-Paket und überträgt es an den ConfigClient. Danach wird auf die Bestätigung des Clients gewartet, ob der Job erfolgreich ausgeführt wurde oder nicht. Das Jobmanagement sucht im Topic nach dem entsprechenden MCP-Paket, das die Bestätigung enthält. Entsprechend der empfangenen Bestätigung setzt das SIB MCP_CheckTaskTransmitStatus den Status der Jobs fest.

5.3.3.4 Klassendiagramm, MCP

Das Klassendiagramm in Abbildung 5.23 enthält alle Klassen, die aufgrund des MCP-Protokolls neu hinzugekommen sind, oder verändert wurden und wie die einzelnen Klassen untereinander zusammenhängen.

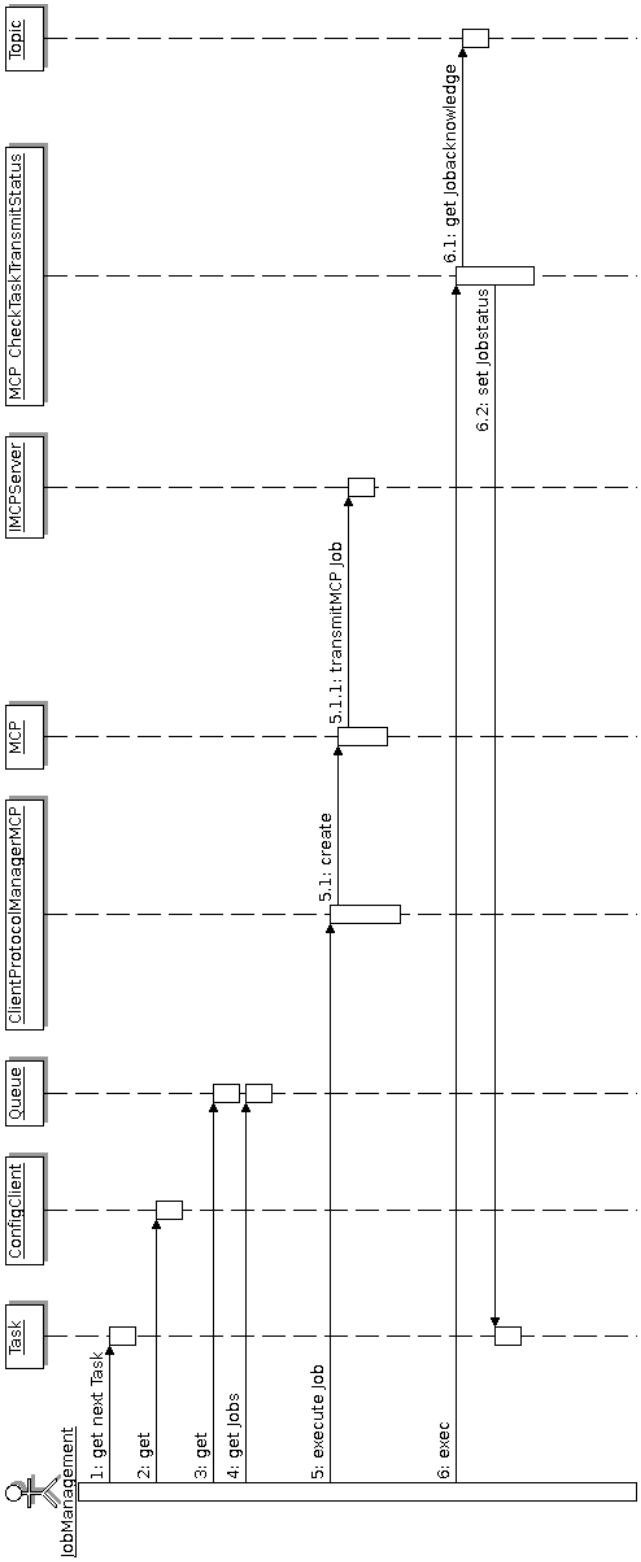

Abbildung 5.22: Ausführungskomponente, Sequenzdiagramm ExecuteJob

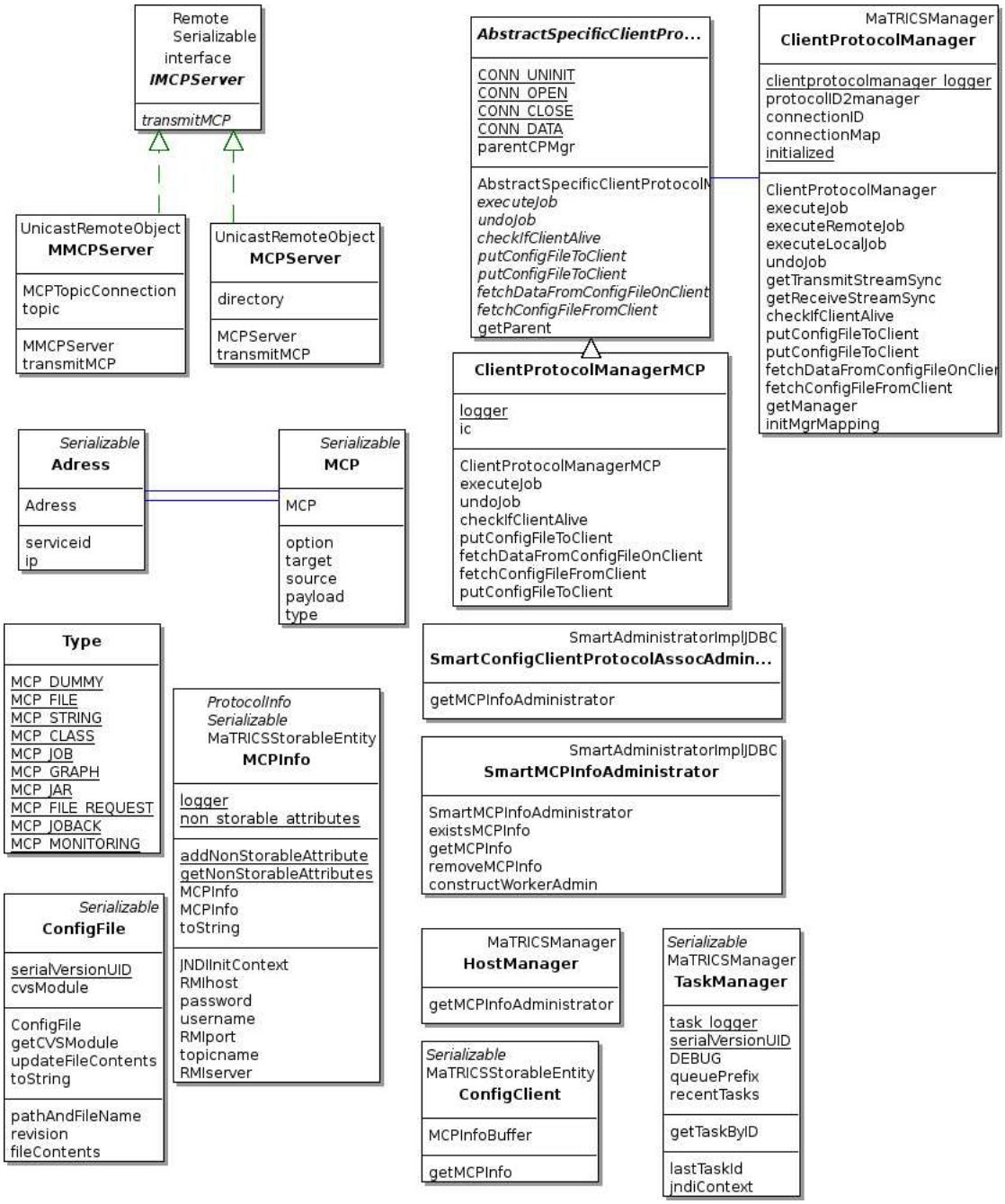

Abbildung 5.23: Ausführungskomponente, Klassendiagramm MCP

5.3.4 Implementierung und Test

In diesem Kapitel werden alle SIBs aufgelistet, die verändert oder neu hinzugefügt wurden. Für eine ausführliche Beschreibung wird das HOWTO des Themas empfohlen.

SIBs innerhalb der MaTRICS:

- `HostMGR_AddConfigClient`
- `HostMGR_EditConfigClient`
- `ClientProtocolMGR_ExecuteJobMCP`
- `InitMMCPServer`
- `JMSGet_MCPByServiceID`
- `MCP_CheckMCP`
- `MCP_CheckProtocolForClient`
- `MCP_CheckTaskTransmitStatus`
- `MCP_ErrorTaskTransmitStatus`
- `MCP_SetMCP`
- `MCP_GetMCPInfo4ConfigClient`

SIBs im ConfigClient:

- `Check4NewMCP`
- `CreateAdress`
- `CreateMCP`
- `DummyAusgabe`
- `ExecuteJar`
- `ExecuteJob`
- `File2StringBuffer`
- `GetClass2load`
- `InitBindMCPServer`
- `InitMCPStorageVector`

- `Parameters2CallContext`
- `SaveFile`
- `SaveMonitoringObject`
- `SendMCP`

SIBs im JMS-Paket:

- `InitJMSEstablishTopicConnection`
- `InitJMSGetTopic2`
- `JMSGetTopic2`
- `JMSCreateTopicDurableReceiver`

5.3.5 Beispielanwendung

Unser selbstständiger ConfigClient ist eine Beispielanwendung, die nur mit dem MCP-Protokoll funktioniert. Alle benötigten Initialisierungen werden im Hauptgraphen ausgeführt, der auch den MCP-Thread startet. Das beinhaltet auch das Erstellen des Vektors, das ”binding” des MCP-Servers zur RMI-Registry und die Initialisierung des MCP-Threads. Abbildung 5.24 zeigt den MCP-Thread, den der ConfigClient in periodschen Zeitabständen startet. Der ConfigClient überprüft den Vektor nach MCP-Paketen mit dem SIB Check4NewMCP. Wenn ein neues gefunden wird, entpackt der ConfigClient dieses mit dem SIB Parameters2CallContext und gibt den gesamten Inhalt an den CallContext weiter. Für jede Art Inhalt gibt es eine Kante, die zum entsprechenden Ziel für das Paket führt. Einige Verarbeitungen sind im Kapitel Sequenzdiagramm dargestellt.

Um das Beispiel auszuführen muss die RMI-Registry gestartet sein, ebenso der Tomcat. Dieser enthält den ConfigClient und muss auf dem Rechner entsprechend eingerichtet sein. Dazu muss die web.xml angepasst werden, später kann noch der Pfad, wo der Vektor für das MCP und das Monitoring gespeichert werden, angegeben werden.

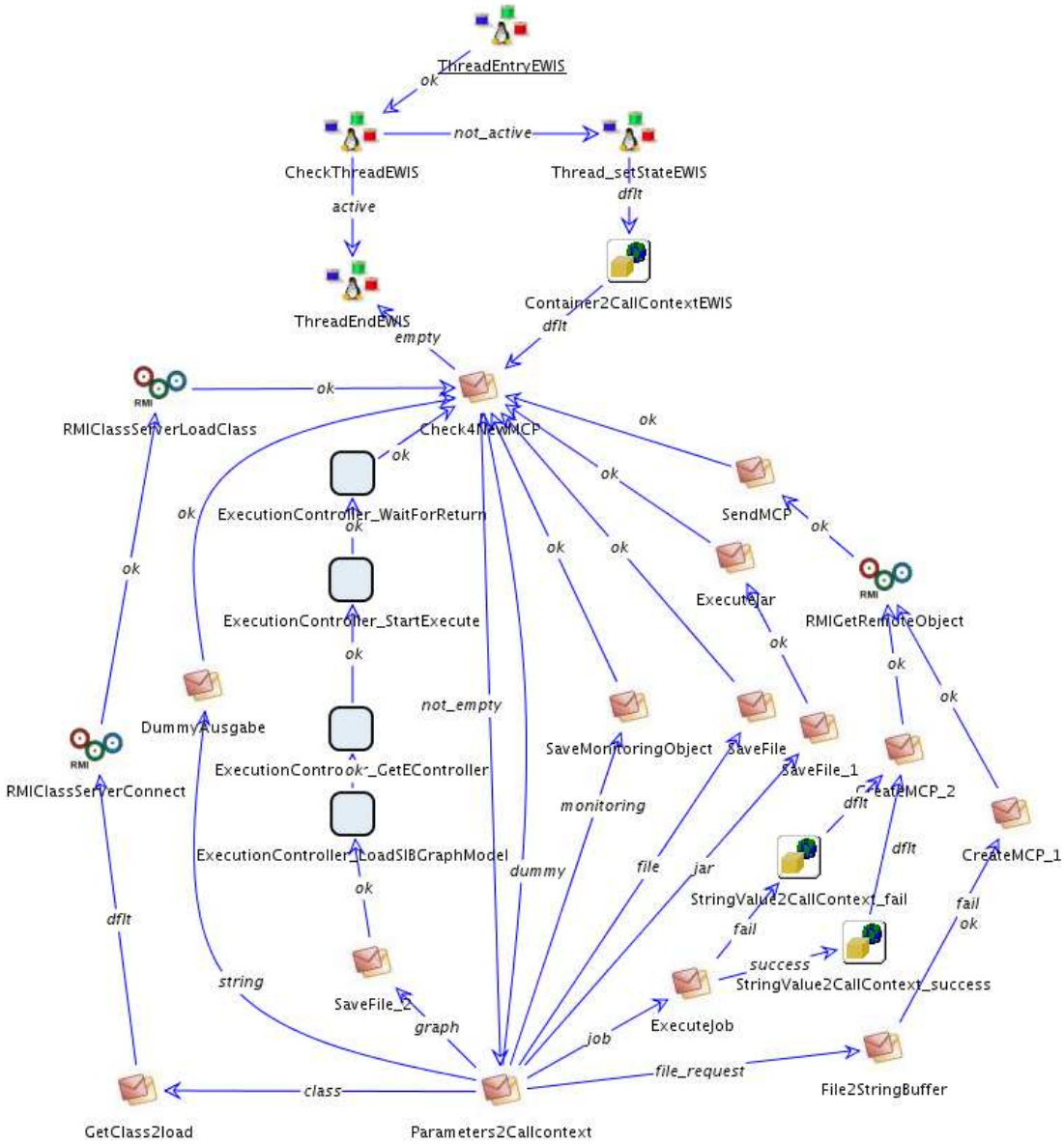

Abbildung 5.24: Ausführungskomponente, MCPThread jABC-Graph

5.4 Design einer Monitoringbibliothek auf Basis von autonomen ConfigClients

5.4.1 Einleitung

In der MaTRICS können viele ConfigClients verwaltet werden. Da diese vom ConfigManager weit entfernt stehen können oder auch weder Tastatur noch Display besitzen müssen und somit der Status der ConfigClients nicht immer zugänglich ist, besteht die Notwendigkeit diese ConfigClients online mithilfe der MaTRICS überwachen zu können. Es liegt daher nahe dies mithilfe einer in der MaTRICS integrierten Monitoring-Komponente zu erledigen.

5.4.2 Aufgabenbeschreibung

Aufgrund der Notwendigkeit zur Überwachung der ConfigClients bestand unsere Aufgabe in der Entwicklung einer solchen Monitoring-Komponente, welche auf autonomen ConfigClients zum Einsatz kommen sollte. Gefordert wurde die Möglichkeit durch die Erstellung von EWIS-Graphen Überwachungsvorschriften zu erstellen. Diese sollten dann auf dem ConfigManager verwaltet (gespeichert und parametrisiert) werden und über das in Kapitel 5 beschriebene MCP zu einem ConfigClient gesendet und dann ausgeführt werden. Die Ergebnisse dieser Messung galt es dann zurück an den ConfigManager zu senden. Dort sollen die Ergebnisse in der Datenbank gespeichert uns bei Bedarf angeschaut werden können.

5.4.3 Design

5.4.3.1 Use Case Diagramm

In unserem Use Case Diagramm, siehe Abbildung: 5.4.3.2, ist deutlich zu erkennen, dass der Administrator beim Monitoring keinen direkten Kontakt mit dem CVS, dem ConfigClient oder der Datenbank hat. Sämtliche Aktionen laufen über den Dienst der auf dem ConfigManager läuft.

5.4.3.2 Activity Diagramm

Das Activity Diagramm, siehe Abbildung: 5.4.3.2, beschreibt den allgemeinen Ablauf des Monitorings. Zuerst müssen geeignete Testgraphen mit dem jABC erstellt und durch Speicherung im CVS der MaTRICS zugänglich gemacht werden. Auf Basis dieser Graphen muss ein Measurement erstellt und mit einem Namen und einer Beschreibung versehen werden.

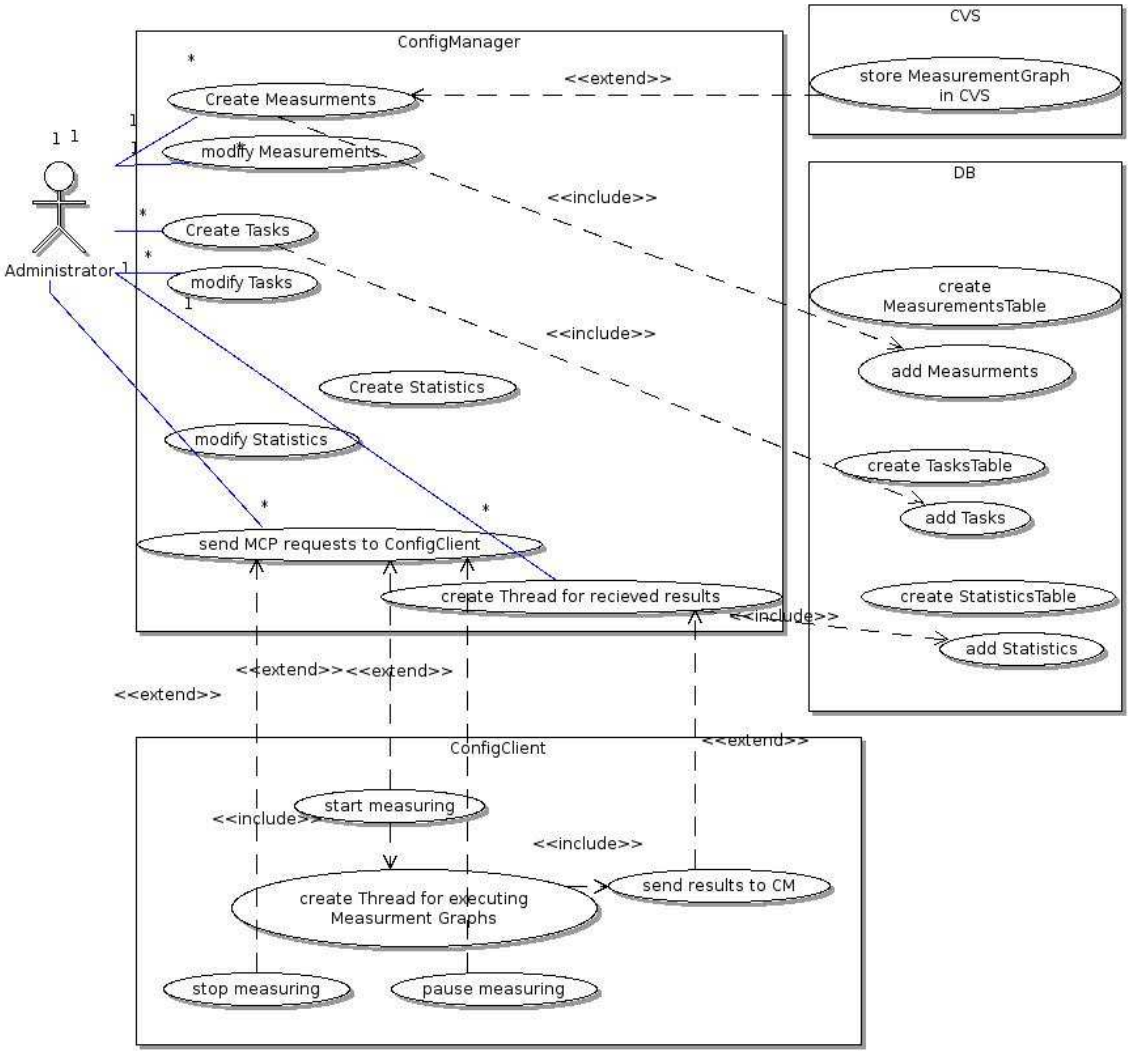

Abbildung 5.25: Monitoring, Usecase-Chart

Aus den in der Datenbank gespeicherten ConfigClients wird ein ConfigClient ausgewählt; ein Task wird erstellt, indem diesem ConfigClient ein Measurement samt Measuring-Schedule zugeordnet wird. Anschließend wird der Testgraph via MCP an den ConfigClient gesendet und dort mit dem ExecutionController ausgeführt. Die Ergebnisse werden periodisch zurück an den ConfigManager gesendet und gespeichert. Der ConfigManager nutzt den `ControllerThread` um den Thread auf dem ConfigClient gegebenfalls zu unterbrechen, neu zu starten oder zu stoppen. Daraufhin erhält der ConfigManager noch den Status vom ConfigClient.

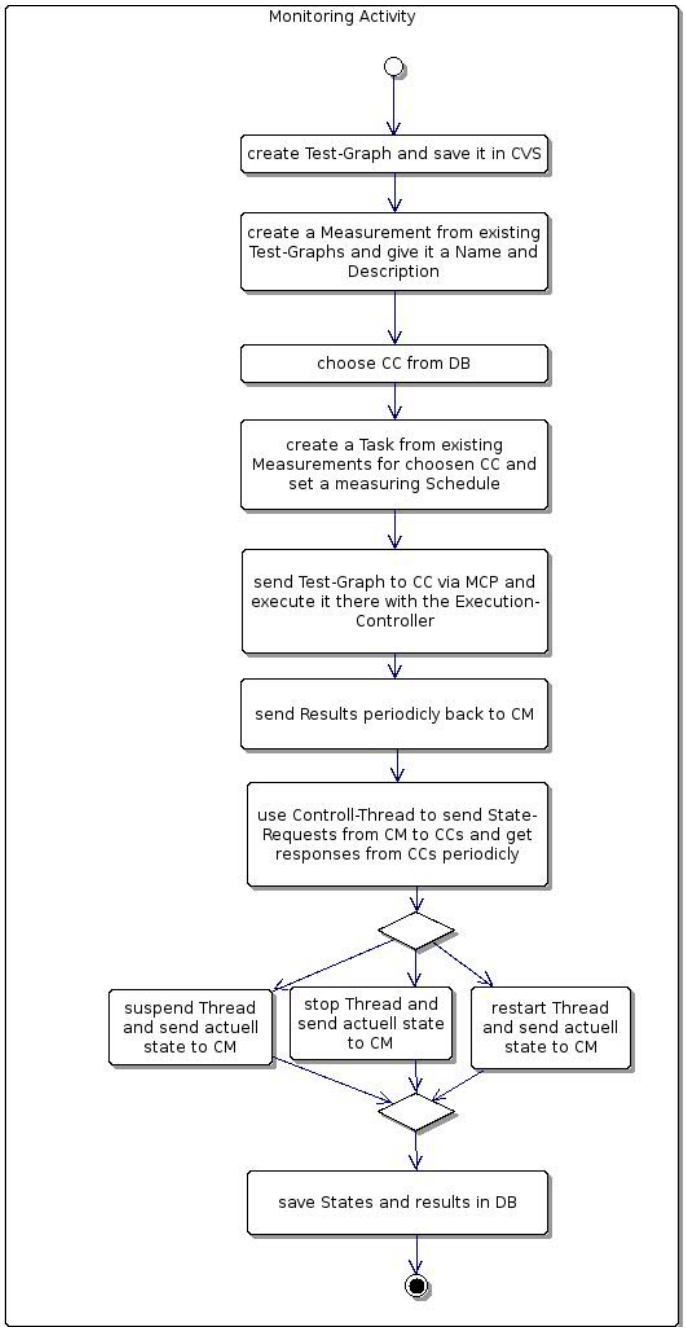

Abbildung 5.26: Monitoring, Aktivitätsdiagramm der Ausführung eines Monitorings

5.4.3.3 Sequence Diagramm

In dem hier dargestellten Diagramm 5.4.3.3 sieht man in leicht abstrahierter Form den Ablauf von der Erstellung eines MonitoringJobs bis zum Senden an den ConfigClient. Der Nutzer des Monitoring wählt beziehungsweise erzeugt die notwendigen Komponenten wie z.B. Measurement und ConfigClient für das Monitoring in der MaTRICS. Die Interaktion mit Komponenten wie dem Thread geschieht dabei nur über das Interface der Webseiten.

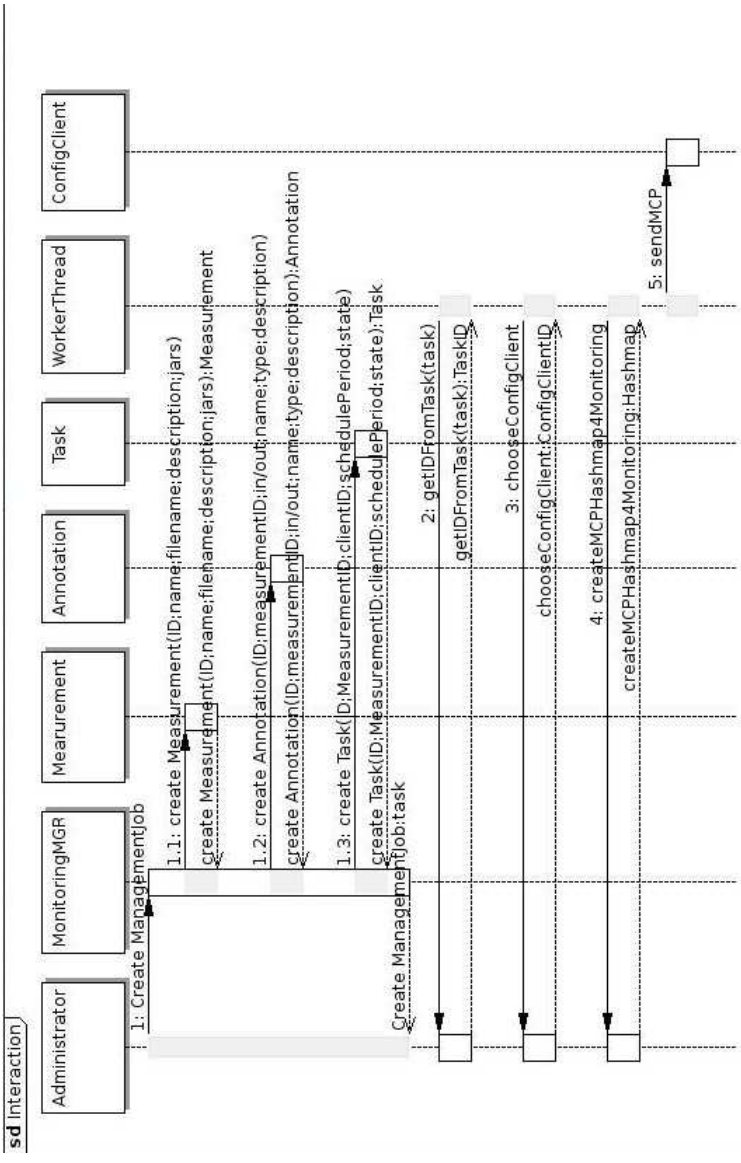

Abbildung 5.27: Monitoring, Sequence-Chart von der Erstellung eines MonitoringJobs bis zum Senden an den ConfigClient via MCP

5.4.3.4 Class Diagramm

5.4.4 Implementierung

5.4.4.1 Erstellte JAVA Klassen

Da sowohl Measurements, Statistiken und Tasks in der Datenbank gehalten werden, wurde eine Erweiterung der Datenbank notwendig. Damit ging die Erstellung neuer Administratoren einher. Eine detaillierte Beschreibung aller SIBs ist im zugehörigen HOWTO zu finden.

- `MonitoringMeasurement`, die eigentliche Messung in Form eines Graphen
- `MonitoringStatistic`, das Ergebniss einer Messung
- `MonitoringTask`, eine laufende Messung

- `JDBCMonitoringMeasurementAdministrator`
- `JDBCMonitoringTaskAdministrator`
- `JDBCMonitoringStatisticAdministrator`
- `SmartMonitoringMeasurementAdministrator`
- `SmartMonitoringTaskAdministrator`
- `SmartMonitoringStatisticAdministrator`

5.4.4.2 Dateiverwaltungs SIBs

Um mit den Dateien umzugehen wurden ebenfalls neue SIBs notwendig.

- `CreateAndStoreFile`
- `GetFileList`
- `GetUploadedFile`
- `GetFileContent`

5.4.4.3 Threading SIBs

Da unser Dienst sowohl auf ConfigClient- als auch auf ConfigManager-Seite in Threads läuft war eine Erweiterung der Threading Bibliothek in der MaTRICS durch neue SIBs notwendig. Um die Messungen z.B. anzuhalten und danach neu zu starten, mussten komplett neue SIBs geschrieben werden. Dabei wurde auch die Möglichkeit auf Semaphoren zurückzugreifen eingeführt.

- `MonitorMutexAcquire`
- `MonitorMutexNewCall`
- `MonitorMutexNewInit`
- `MonitorMutexRelease`
- `MonitorSemaphoreAcquire`
- `MonitorSemaphoreNewCall`
- `MonitorSemaphoreNewInit`
- `MonitorSemaphoreRelease`
- `MonitorThreadCheckState`
- `MonitorThreadEntry`
- `MonitorThreadGetID`
- `MonitorThreadGetName`
- `MonitorThreadGetSleepingTime`
- `MonitorThreadGetURL`
- `MonitorThreadNewCall`
- `MonitorThreadNewInit`
- `MonitorThreadPrint2Log`
- `MonitorThreadReset`
- `MonitorThreadResume`
- `MonitorThreadSetName`
- `MonitorThreadSetSleepingTime`
- `MonitorThreadStop`
- `MonitorThreadSuspend`

5.4.4.4 Application SIBs

Um Messvorschriften auf dem ConfigManager speichern zu können, stand uns nicht die Möglichkeit der Speicherung in der Datenbank zur Verfügung. Einer einfachen Speicherung im Dateisystem stand aber z.B. die fehlende Versionsverwaltung im Wege. Daher entschlossen wir uns zur Speicherung mithilfe des CVS. Obwohl es schon eine CVS-Komponente innerhalb eines Dienstes gab, wurden neue SIBs geschrieben, da die alten zu unflexibel waren.
Um die Messungen und Tasks, sowie die Statistiken verwalten zu können, wurde eine Vielzahl weiterer SIBs benötigt.

- `MonitoringMGR_CVSInet`
- `MonitoringMGR_CVSCheckFileExist`
- `MonitoringMGR_CVSAddFile`
- `MonitoringMGR_CVSRemoveFile`
- `MonitoringMGR_CVSCommit`
- `MonitoringMGR_CVSUbdateFile`

- `MonitoringMGR_AddMeasurement`
- `MonitoringMGR_RemoveAllMeasurements`
- `MonitoringMGR_RemoveMeasurements`
- `MonitoringMGR_RamoveMeasurementWithAttribute`
- `MonitoringMGR_EditMeasurement`
- `MonitoringMGR_GetAllMeasurements`
- `MonitoringMGR_GetMeasurements`
- `MonitoringMGR_GetMeasurementsWithAttribute`
- `MonitoringMGR_GetIDsFromMeasuremens`
- `MonitoringMGR_GetAttributesFromMeasurements`

- `MonitoringMGR_AddAnnotation`
- `MonitoringMGR_RemoveAllAnnotations`
- `MonitoringMGR_RemoveAnnotations`

- MonitoringMGR_RemoveAnnotationsWithMeasurementID
- MonitoringMGR_RemoveAnnotationsWithAttributes
- MonitoringMGR_EditAnnotation
- MonitoringMGR_GetAllAnnotations
- MonitoringMGR_GetAnnotations
- MonitoringMGR_GetAnnotationsWithAttributes
- MonitoringMGR_GetAnnotationsWithMeasurementID
- MonitoringMGR_GetAttributesFromAnnotations

- MonitoringMGR_CreatePoolOutput
- MonitoringMGR_CompareJar
- MonitoringMGR_TransformJarAppearance
- MonitoringMGR_GetAndCheckParametersFromRequest

- MonitoringMGR_AddTask
- MonitoringMGR_RemoveAllTasks
- MonitoringMGR_RemoveTasks
- MonitoringMGR_RemoveTasksWithAttribute
- MonitoringMGR_RemoveTasksWithAttributes
- MonitoringMGR_GetAllTasks
- MonitoringMGR_GetTasks
- MonitoringMGR_GetTasksWithAttribute
- MonitoringMGR_GetTasksWithAttributes
- MonitoringMGR_GetIDsFromTasks
- MonitoringMGR_GetAttributesFromTasks
- MonitoringMGR_ChangeTaskScheduleperiod
- MonitoringMGR_ChangeTaskState

- MonitoringMGR_Parameters2HashMap

- MonitoringMGR_AddParameter
- MonitoringMGR_RemoveAllParameters
- MonitoringMGR_RemoveParameters
- MonitoringMGR_RemoveParametersWithTaskID
- MonitoringMGR_EditParameter
- MonitoringMGR_GetAllParameters
- MonitoringMGR_GetParameters
- MonitoringMGR_GetParametersWithTaskID
- MonitoringMGR_GetAttributesFromParameters

- MonitoringMGR_CheckPeriod
- MonitoringMGR_GetSendStates
- MonitoringMGR_DifferAnnotationsAndParameters

- MonitoringMGR_AddStatistic
- MonitoringMGR_RemoveAllStatistics
- MonitoringMGR_RemoveStatistics
- MonitoringMGR_RemoveStatisticsWithTaskID
- MonitoringMGR_GetAllStatistics
- MonitoringMGR_GetStatistics
- MonitoringMGR_GetStatisticsWithTaskID
- MonitoringMGR_GetAttributesFromStatistics

- MonitoringMGR_AddResult
- MonitoringMGR_RemoveAllResults
- MonitoringMGR_RemoveResults
- MonitoringMGR_RemoveResultsWithAttributes

- MonitoringMGR_RemoveResultsWithStatisticID
- MonitoringMGR_GetAllResults
- MonitoringMGR_GetAttributesFromResult
- MonitoringMGR_GetAttributesFromResults
- MonitoringMGR_GetResults
- MonitoringMGR_GetResultsWithAttributes
- MonitoringMGR_GetResultsWithStatisticID

5.4.4.5 ConfigClient SIBs

Auch auf Seite des ConfigClients war die Erstellung von neuen SIBs notwendig. Zum einen weil wir eine Monitoringschnittstelle zum MCP brauchten, zum anderen weil die Ergebnisse, welche uns die Messungen aus dem ExecutionController liefert, weiterverarbeitet werden müssen.

- Check4NewMonitoring
- CreateMCPHashmap4Monitoring
- GetMonitoringCommand
- SaveMonitoringObject
- UnpackMCPHashmap4Monitoring

5.4.5 jABC-Graphen

Es gibt eine Vielzahl von jABC Graphen, wobei es drei Komponenten gibt, die sich in ihrer Bedeutung deutlich hervorheben. Dabei sollte nicht aus den Augen gelassen werden, dass auch Graph-SIBs bzw. Macros eine hohe Komplexität aufweisen können, auch wenn der Graph in dem sie eingebunden sind, dies auf den ersten Blick nicht vermuten lässt. Besonders ausgeprägt ist das im MonitoringManagement.
Da unser Monitoring permanent läuft, mussten wir in Form von jABC-Graphen Threads implementieren. Es wurden zwei Hauptthreads notwendig. Zum einen der MonitoringWorkerThread, siehe Abbildung: 5.29, und zum anderen der MonitoringControllerThread, siehe Abbildung: 5.30. Dies sind die Threads, die auf der ConfigManager-Seite laufen. Auf ConfigClient-Seite unterscheidet sich dies aufgrund dessen Eigenschaften.
Die dritte große Komponente ist das MonitoringManagement, siehe Abbildung: 5.28. Dieses Management enthält vier Graph-SIBs. Diese beinhalten den detaillierten Ablauf

des Managments. Die Steuerung der Threads, des Filesystems und der Datenbank sind hier untergebracht. Zudem ist es die eigentliche Kernkomponente unseres Dienstes, da von hier aus auf Programmebene die übrigen Komponenten des Monitoring angestossen werden, bzw. darauf zurückgegriffen wird.

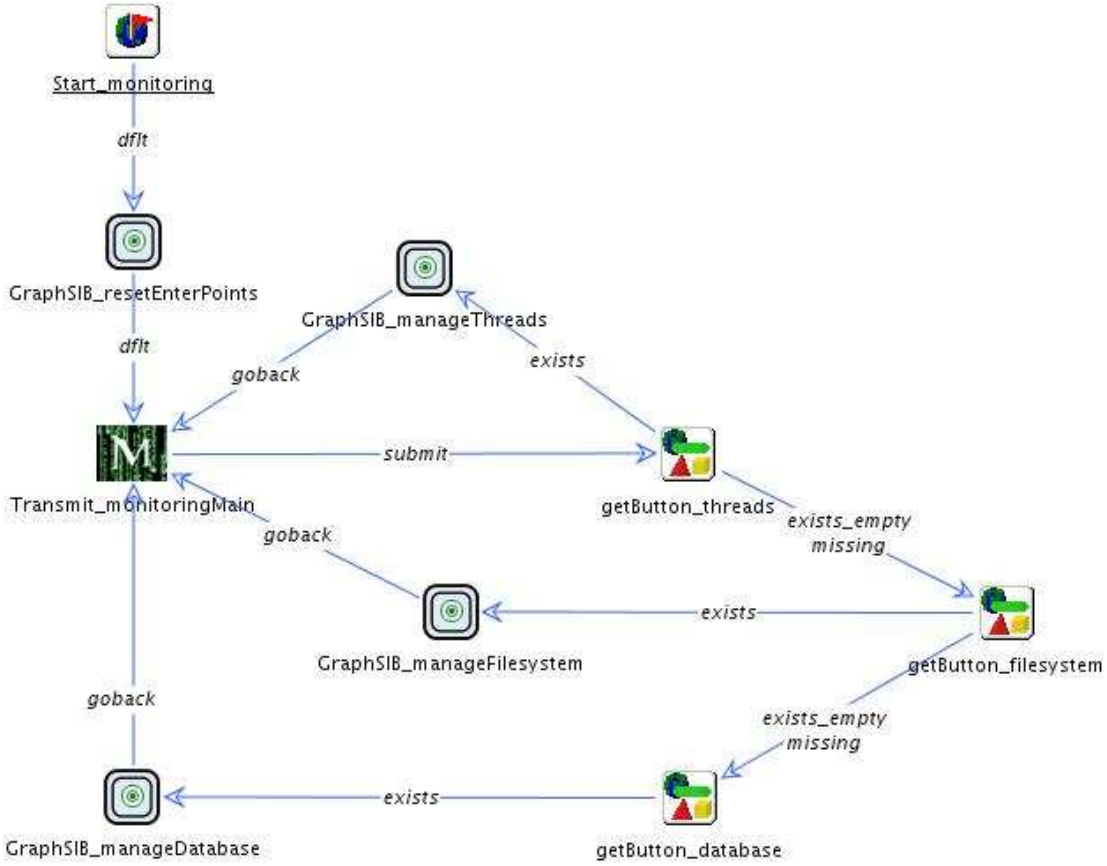

Abbildung 5.28: Monitoring, MonitoringManagement Dienst

5.4.6 Ausblick

Mithilfe dieser Komponente wurde ein Dienst in die MaTRICS eingebunden, der die gestellten Aufgaben erfüllt. Im weiteren Verlauf der Projektgruppe kam uns die Architektur des Dienstes sehr zu Gute, da durch Erweiterungen und Überarbeitungen eine Kernkomponente für unser Hauptthema geschaffen wurde. Die Überwachungen im WatchDog erfolgen durch das bereits vorhandene Monitoring.

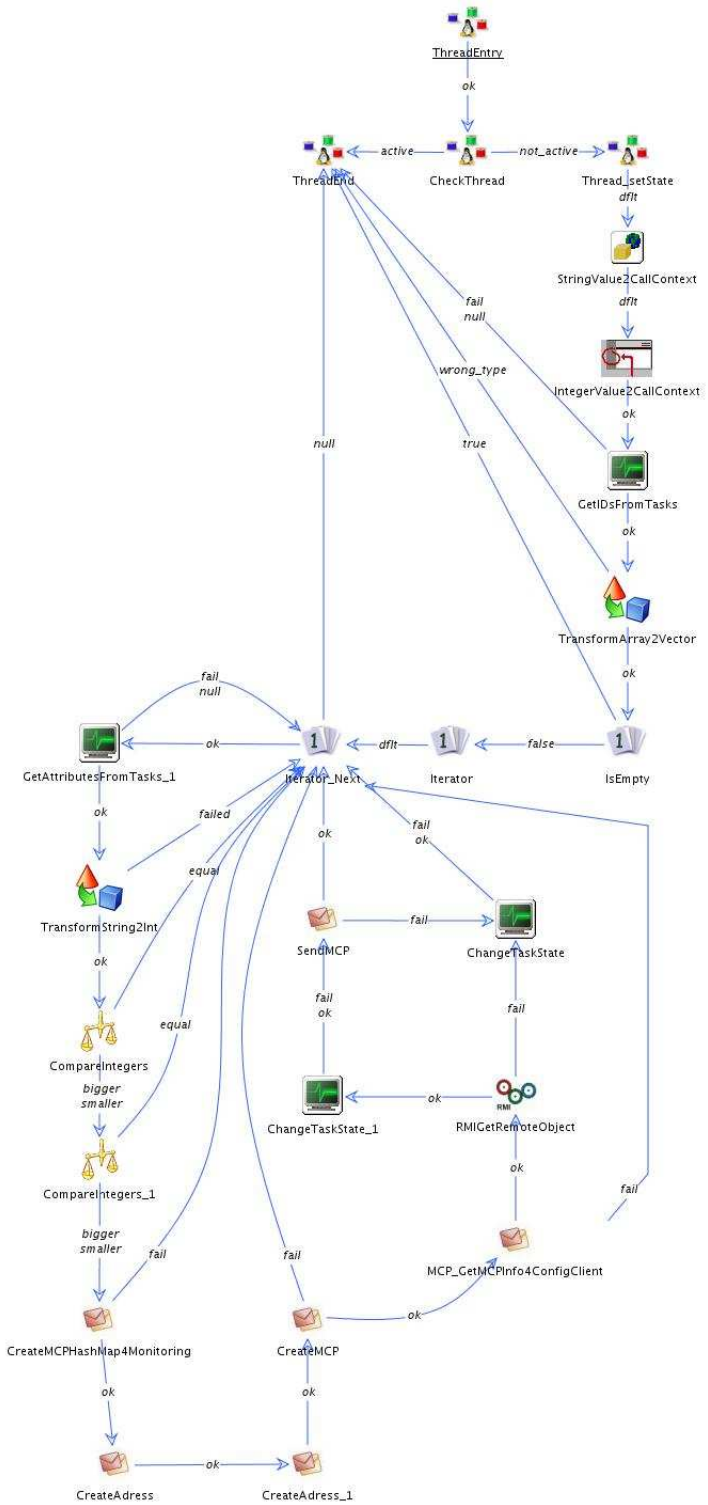

Abbildung 5.29: Monitoring, jABC-Graph des MonitoringWorkerThread auf ConfigManager-Seite

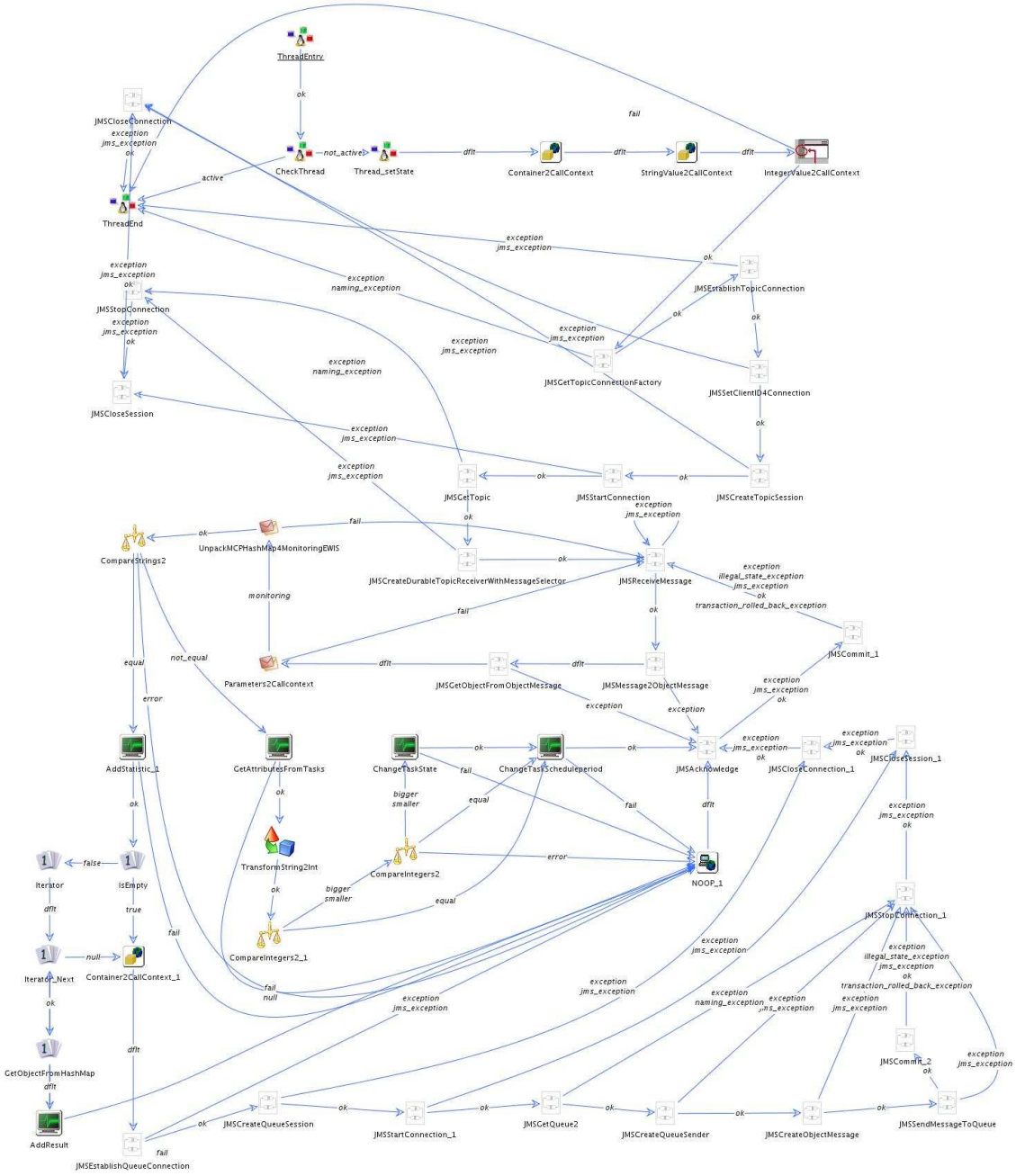

Abbildung 5.30: Monitoring, jABC-Graph des MonitoringControllerThread auf Config-Manager-Seite

Kapitel 6

Hauptthema - Entwicklung eines Watchdog-Systems zur Diagnose und Steuerung von Unix-Prozessen

6.1 Einleitung

Die letzte Aufgabe der Projektgruppe besteht in der Entwicklung eines Watchdog-Systems als Service für die MaTRICS, welches Ressourcenverbrauch auf einzelnen Configclients überwachen soll und beim Überschreiten von vorgegebenen Grenzwerten bestimmte Aktionen anstoßen soll. Diese Aufgabe ist das Hauptthema der Projektgruppe an der alle Mitglieder teilnehmen, im Unterschied zu den vorherigen Aufgaben, die in kleineren Gruppen von 3-4 Personen durchgeführt wurden.

Das Thema wurde in drei Teilaufgaben unterteilt, wobei für jede Teilaufgabe eine eigene Arbeitsgruppe gebildet wurde. Die Teilaufgaben sind:

- Watchdog-Management
- WatchdogThread und ReactionThread
- ConfigClient-Monitoring

Das Hauptthema stützt sich auf die Services, die bei den Zwischenthemen der Projektgruppe implementiert wurden. Diese sind Notification-Komponente, jXMLGui-Komponente und vor allem die Monitoring-Komponente. Zuerst wird das Design entworfen. Dazu werden Anwendungsfall-, Aktivitäts- und Sequenzdiagramme mit Hilfe von UML erstellt. Daneben wird ein ausführliches Design für die Datenbank konstruiert.

Nach dem Design sollen die Teilkomponenten des Services implementiert, getestet und in die MaTRICS integriert werden. Zum Schluss folgt eine englische Dokumentation zu Design und Implementierung des Services. Für die Aufgabe sind 5 Wochen geplant.

6.2 Aufgabenbeschreibung

Die Aufgabe des Hauptthemas besteht darin, einen intelligenten Watchdog-Mechanismus für Unix-Systeme zu entwickeln. Dieser Dienst soll innerhalb der MaTRICS laufen und automatisch Reaktionen ausführen können. Eine Reaktion kann das Ausführen eines Bash-Skript auf dem autonomen ConfigClient, das Starten und Stoppen eines WatchdogProzesses oder das Senden einer Notification an den Administrator sein. Die Durchführung der eigentlichen Tests soll mit Hilfe der Monitoring-Komponente realisiert werden. Abhängig von den Messergebnissen werden die Reaktionen ausgeführt. Dabei kann der Benutzer bei Überschreiten eines bestimmten Grenzwertes beispielsweise eine Benachrichtigung erhalten. Bei Überschreitung eines höheren Grenzwertes können dann härtere Massnahmen angewendet werden, wie z. B. alle Prozesse dieses Benutzers beenden.

Der Dienst sollte folgende Funktionalitäten besitzen:

- Watchdog-Management: Anzeigen, Einrichten, Ändern und Entfernen von Watchdog-Prozessen
- Direkte Ausführung der Aktionen auf dem Configclient
- Durchführung einer Messung und Anzeige der Ergebnisse
- Verwaltung aller Messergebnisse zur Generierung von Statistiken

Ein Watchdog-Prozess besteht aus einem Graphen, der die Messung beschreibt, einem Schedule, der den Zeitpunkt der Messung festlegt und beliebig vielen Aktionen, die bei Überschreiten der Grenzwerte ausgeführt werden sollen.

Die Kommunikation zwischen MaTRICS und autonomen ConfigClient soll auf dem MCP basieren. Die Notification sollen über den NotificationManager verschickt werden. Das Starten und Stoppen von anderen Watchdog-Prozesse soll über die Monitoring-Komponente durchgeführt werden.

Zur Ausführung von Messungen auf dem ConfigClient wird das MCP zur Übertragung und die Bibliothek für autonome ConfigClients verwendet.
Der Dienst setzt jXML-GUI als Frontend ein und benutzt damit auch das MMP V2.

6.3 Design

6.3.1 Anwendungsfalldiagramm

Im Anwendungfalldiagramm wird unterschieden zwischen Anwendungfällen auf dem ConfigClient und Anwendungsfällen in der MaTRICS. Innerhalb der MaTRICS wird noch unterschieden zwischen dem Watchdogmanagement und der Verarbeitung der Ergebnisse der Watchdog-Prozesse.

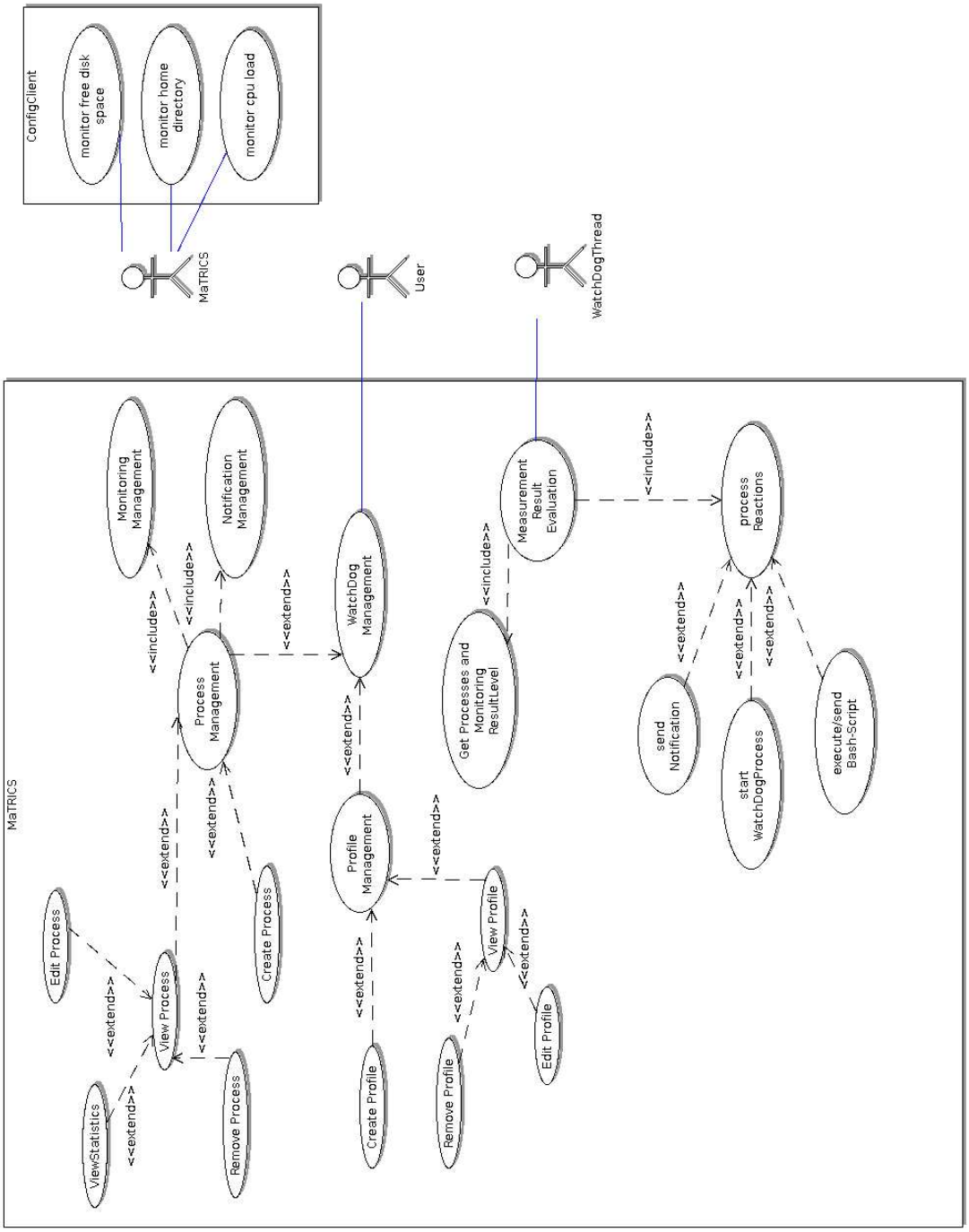

Abbildung 6.1: Watchdog, Anwendungsfalldiagramm

6.3.1.1 Watchdogmanagement

Beim Watchdogmanagement kann der Benutzer die Prozesse und die Profile verwalten. Im Profil werden eine Messung und die möglichen Aktionen definiert. Ein Prozess ordnet ein Profil einem ConfigClient zu. Im Profilmanagement kann der Benutzer ein Profile erzeugen, sich die vorhandenen Profile anschauen und diese einzeln löschen und ändern. Im Prozessmanagement kann der Benutzer einen Prozess erzeugen, sich die vorhandenen Prozesse anschauen und diese einzeln löschen und ändern. Ausserdem kann er sich die Statistiken zu einem Prozess anschauen. Über das Prozessmanagement kann der Benutzer auch das Monitoringmanagement erreichen, um die Messung eines Prozesses zu steuern.

6.3.1.2 Watchdog Thread

Bei der Auswertung der Messungsergebnisse werden die Ergebnisse aus der Datenbank gelesen, und der zugehörige Watchdog-Prozess und das Watchdog-Profil in der Datenbank gesucht. Abhängig von den im Profil definierten Reaktionen werden Notifications verschickt, Watchdog-Prozesse gestartet oder gestoppt, oder Bash-Skripte auf dem ConfigClient ausgeführt.

6.3.1.3 Messung

Auf dem ConfigClient startet die MaTRICS die Messungen, das können beispielsweise die Überwachung des freien Festplattenplatzes, die Überwachung der Grösse der Home-Verzeichnisse oder die Überwachung der CPU-Auslastung sein. Diese Messungen bestehen aus einem Graphen, der mit Hilfe des Execution Controllers ausgeführt wird.

6.3.2 Anwendungsfalldiagramm ConfigClient

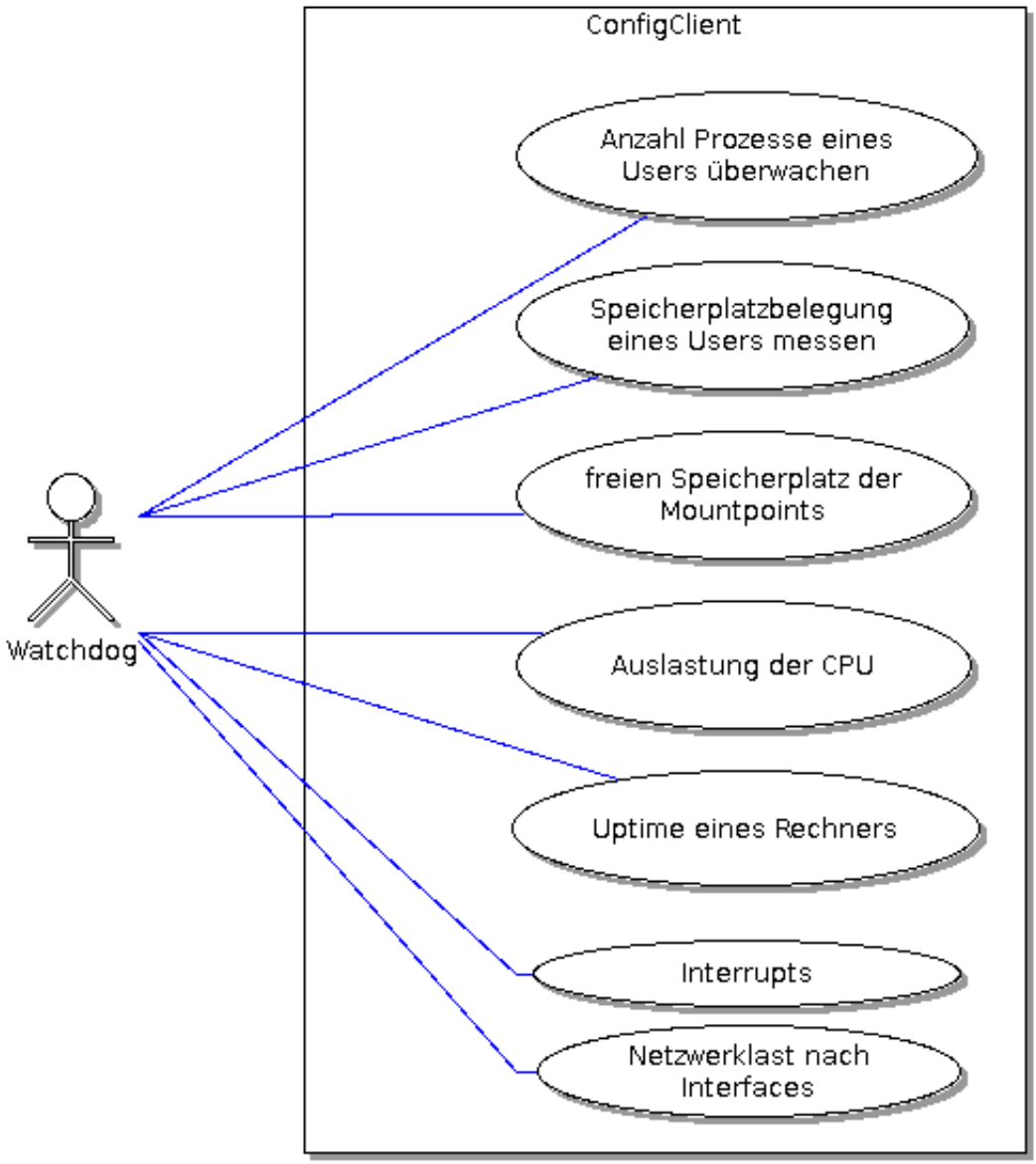

Abbildung 6.2: Watchdog, Anwendungsfalldiagramm ConfigClient

Im Anwendungsfalldiagramm in Abbildung 6.2 werden alle Befehle dargestellt, die auf dem ConfigClient ausgeführt werden können.

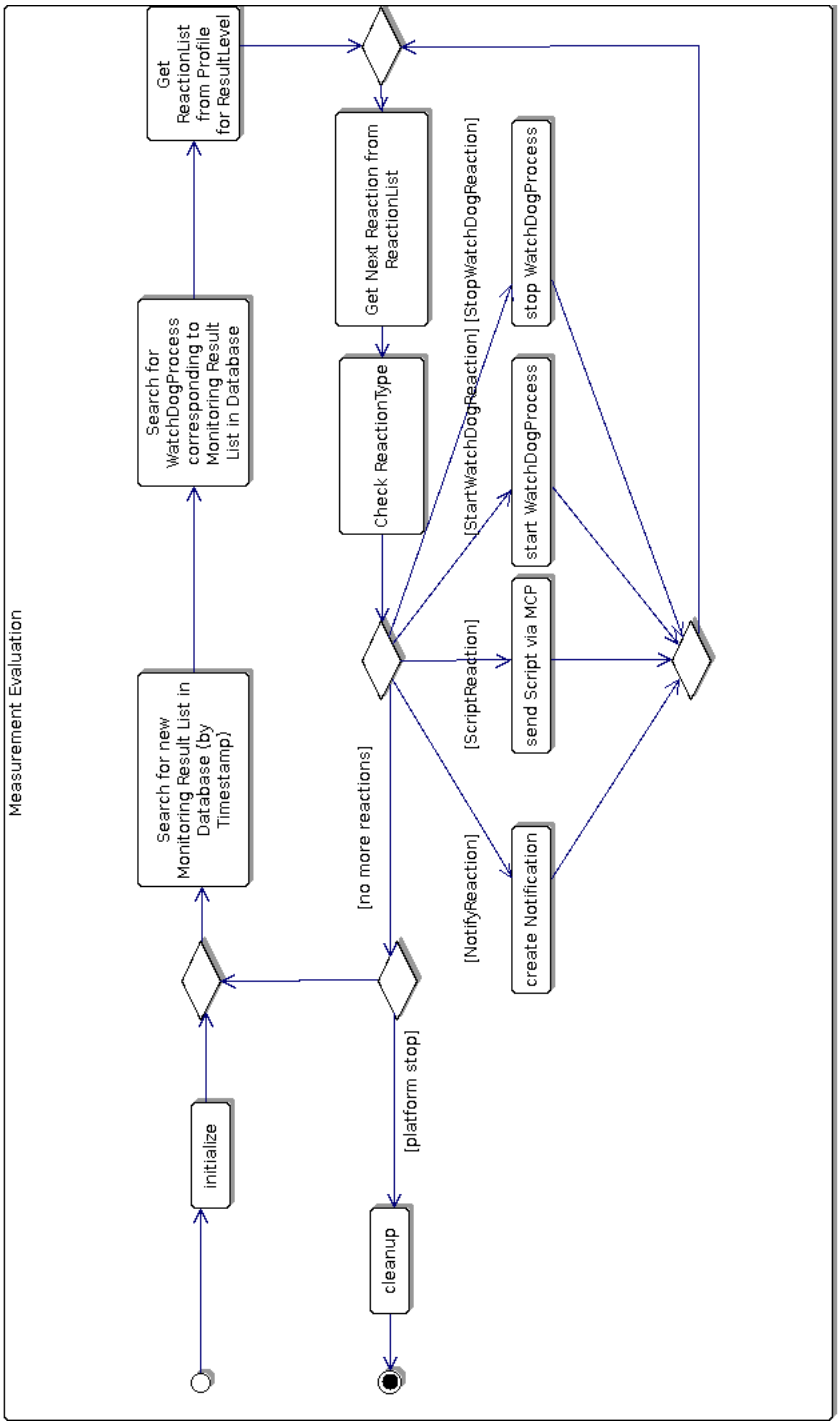

Abbildung 6.3: Watchdog, Watchdog Engine, Aktivitätsdiagramm

6.3.3 Aktivitätsdiagramm, Watchdog Engine

Die Watchdog-Engine führt abhängig von den Ergebnissen der Messungen die im Profil definierten Aktionen aus. Die Watchdog-Engine wird beim Hochfahren der Plattform gestartet und initialisiert. Dann sucht sie in der Datenbank nach neuen Messergebnissen. Hierzu sucht sie in der Datenbank den passenden Watchdog-Prozess, und das Watchdog-Profile. Für alle Aktionen aus dem Profile werden die im Aktionstyp definierten Aktionen ausgeführt:

- Notification verschicken
- Script ausführen
- Einen weiteren Watchdog-Prozess starten
- Einen Watchdog-Prozess stoppen

Danach wird wieder nach neuen Messergebnissen gesucht. Wird die Plattform heruntergefahren, gibt die Watchdog-Engine noch die belegten Resourcen frei und beendet sich.

6.3.4 Aktivitätsdiagramm, Profile Management

Im Profil Management kann der Benutzer wählen, ob er ein neues Profil anlegen oder sich die vorhanden Profile anschauen will. Legt er ein neues Profil an, gibt er den Namen des Profils an, wählt eine Messung aus, und gibt jeweils eine Liste von Aktionen, die beim Erreichen des Warning, Critical oder Error Zustand der Messung ausgeführt werden. Für diese Zustände muss er noch die Grenzwerte definieren. Daraus wird ein neues Watchdog-Profil erzeugt. Schaut der Benutzer sich die vorhandenen Watchdog-Profile an, kann er ein einzelnes auswählen, löschen oder ändern. Beim Ändern des Profils kann der Benutzer die Messung, die Aktionen und die Grenzwerte editieren.

6.3.5 Aktivitätsdiagramm, Prozess Management

Im Prozess Management kann der Benutzer wählen, ob er einen neuen Prozess anlegen will, oder sich die vorhandenen Prozesse anschauen will. Legt er einen neuen Prozess an, wählt er den ConfigClient auf dem die Messung gestartet wird, das Profil das die Messung definiert, und gibt an in welchen Zeitabständen die Messungen wiederholt wird. Daraus wird dann der Watchdog-Prozess und ein MonitoringTask erzeugt. Schaut der Benutzer sich die vorhandenen Watchdog-Prozesse an, kann er einen einzelnen auswählen und löschen, oder sich die Messergebnisse von dem Prozess anschauen, oder auch den Prozess ändern. Beim Ändern des Prozesses kann der Benutzer den ConfigClient, das Profil und das Messintervall editieren.

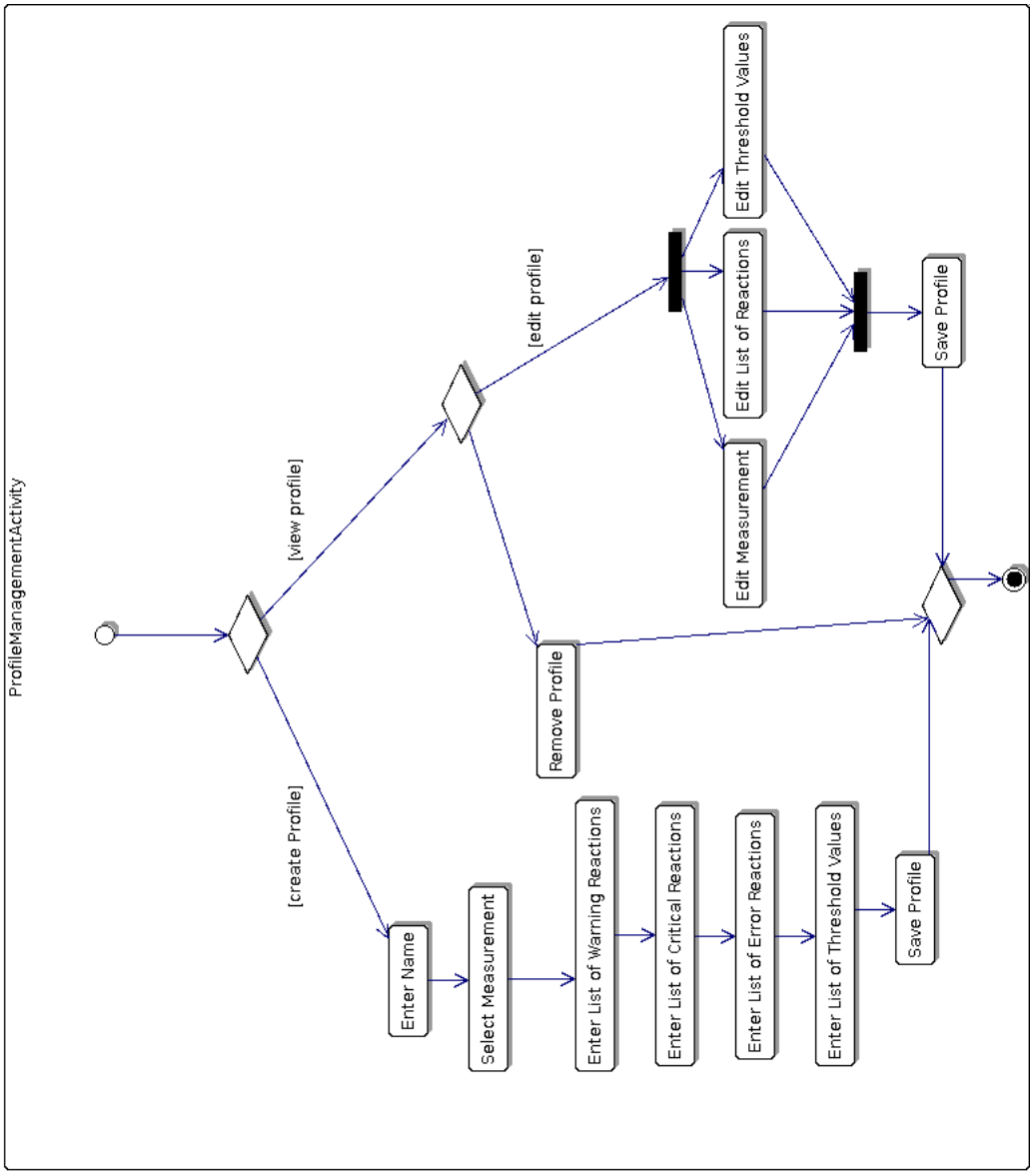

Abbildung 6.4: Watchdog, Watchdog Profile Management, Aktivitätsdiagramm

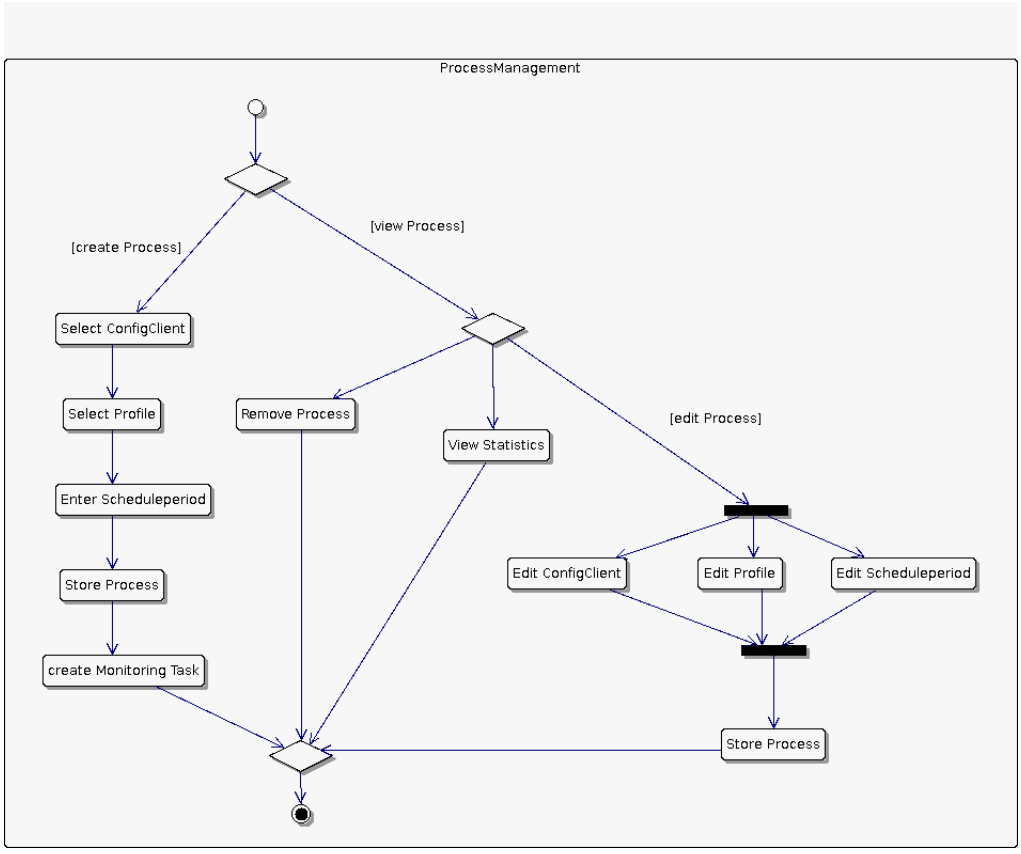

Abbildung 6.5: Watchdog, Watchdog Prozess Management, Aktivitätsdiagramm

6.3.6 Aktivitätsdiagramm, Freier Speicherplatz

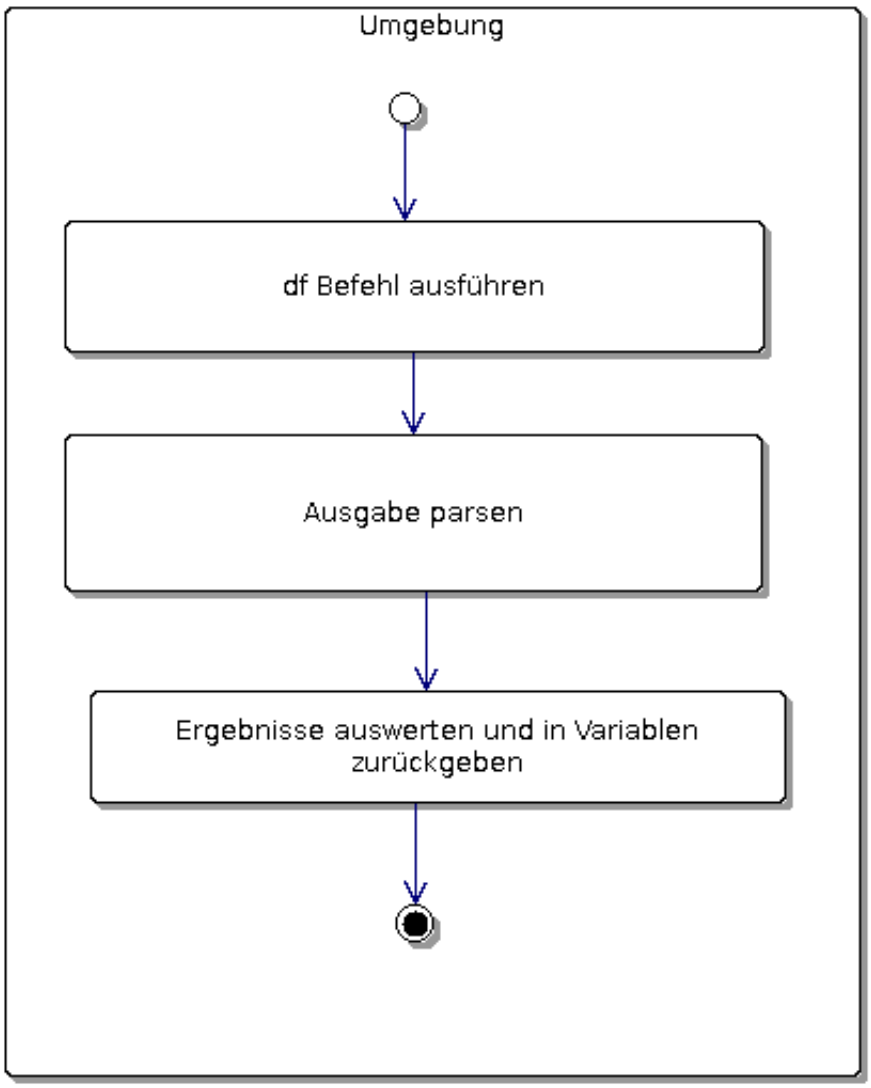

Abbildung 6.6: Watchdog, Aktivitätsdiagramm Freier Speicherplatz

Im Aktivitätsdiagramm 6.6 ist die Ausführung eines Befehls dargestellt. Dabei wird der Befehl ausgeführt, die Ausgabe entsprechend eingelesen, die Ergebnisse entsprechend ausgewertet und diese in Variablen gespeichert und zurückgegeben.

6.3.7 Sequenzdiagramm, Watchdog Engine

Bei der Auswertung eines Messergebnisses, beginnt die Watchdog-Engine mit einem MonitoringStatistic-Objekt. Aus diesem Objekt wird das Result (die Liste von ResultI-

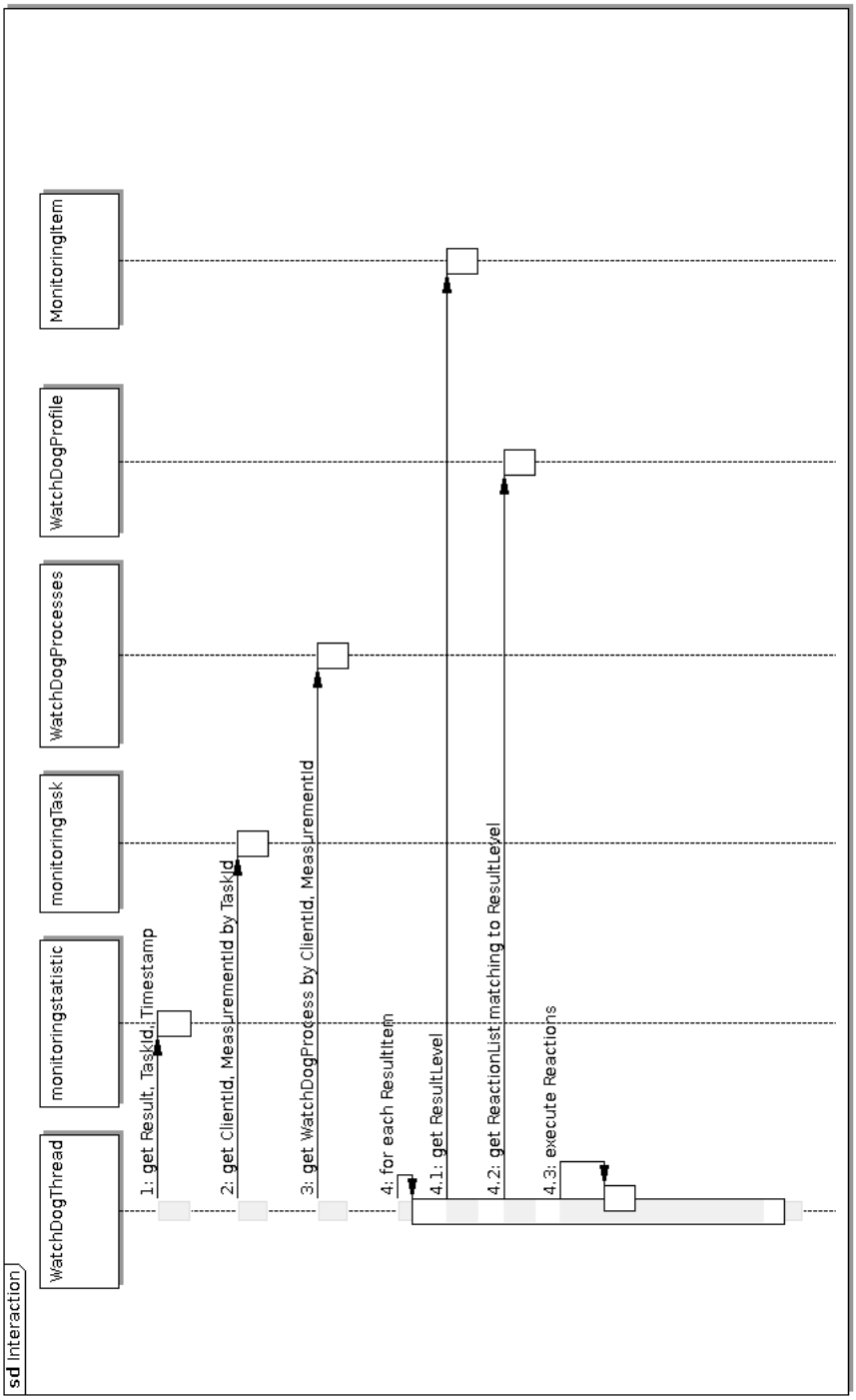

Abbildung 6.7: Watchdog, Watchdog Engine, Sequenzdiagramm

tems), die TaskId und der Timestamp gelesen. Mit der TaskId wird ein MonitoringTask gesucht, und davon werden die ClientId und die MeasurementId gelesen. Mit der ClientId und der MeasurementId wird der WatchdogProcess und das WatchdogProfile gesucht. Dann wird für jedes ResultItem aus dem Result der Level gelesen, dazu die Liste der Aktionen aus dem WatchdogProfile gesucht, und schliesslich die entsprechende Aktion ausgeführt.

6.3.8 Klassendiagramm

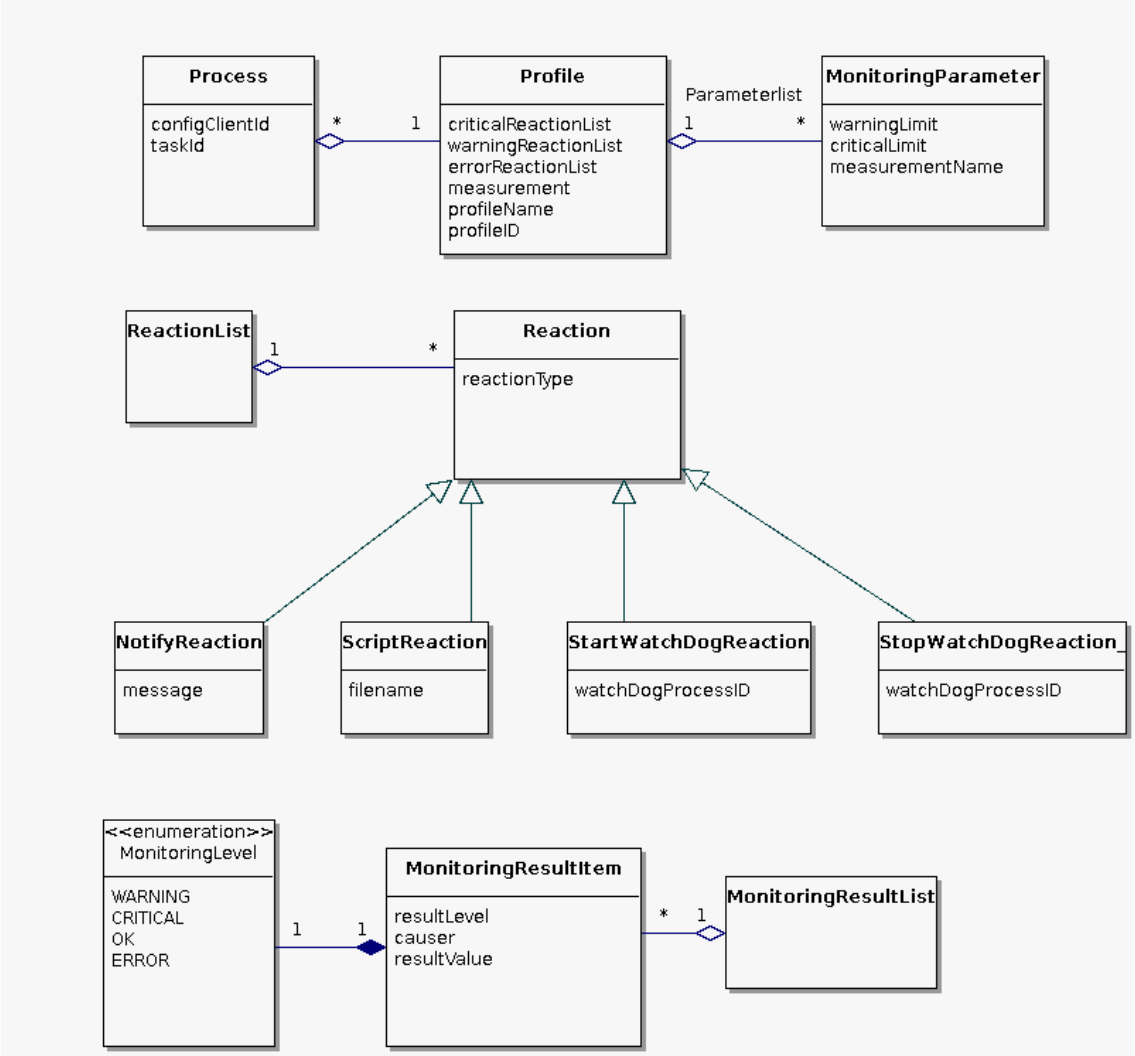

Abbildung 6.8: Watchdog, Klassendiagramm

6.3.8.1 Management

Ein Process definiert

- mit der taskId den MonitoringTask der ausgeführt wird,
- mit der configClientId, wo die Messung ausgeführt wird und
- mit der profileId wie die Messergebnisse ausgewertet werden.

Ein Profil definiert

- mit criticalReactionList die Aktionen, die im Zustand Critical ausgeführt werden,
- mit warningReactionList die Aktionen, die im Zustand Warning ausgeführt werden,
- mit errorReactionList die Aktionen, die im Zustand Error ausgeführt werden,
- mit measurement den Messgraphen, der ausgeführt wird,
- mit profileName den Namen des Profils,
- mit ParameterList die Grenzwerte für die Zustände der Messung

Ein MonitoringParameter definiert

- mit warningLimit den Grenzwert für den Zustand Warning,
- mit criticalLimit den Grenzwert für den Zustand Critical,
- mit measurementName den Namen der Teilmessung im Messgraphen, für den diese Grenzwerte gelten

6.3.8.2 Reaction

Eine Reaction definiert mit reactionType, welche Aktion ausgeführt werden soll. Es gibt insgesamt vier Aktionen: NotifyReaction, ScriptReaction, StartWatchdogReaction und StopWatchdogReaction. Eine NotifyReaction definiert mit message die Nachricht, die verschickt wird. Eine ScriptReaction definiert mit filename das Script, das ausgeführt wird. Eine StartWatchdogReaction definiert mit watchdogProcessId den WatchdogProcess der gestartet wird. Eine StopWatchdogReaction definiert mit watchdogProcessId den WatchdogProcess der gestoppt wird.

6.3.8.3 Result

Ein MonitoringResultItem definiert

- mit resultLevel den Zustand der Messung
- mit causer den Verursacher des Messergebnisses
- mit resultValue den tatsächlichen Wert der Messung

Ein MonitoringLevel legt die Zustände einer Messung fest:

- OK
- WARNING
- CRITICAL
- ERROR

6.4 Implementierung Management

6.4.1 SIB Implementierung

Für die Realisierung des WatchdogManagements wurden eine Reihe von SIBs implementiert.

- WatchdogMGR_CreateContainerProfile
- WatchdogMGR_DeleteProfile
- WatchdogMGR_HashMap2ParameterContainerList
- WatchdogMGR_ManageReactionNotificationContainerList
- WatchdogMGR_ManageReactionProcessContainerList
- WatchdogMGR_ManageReactionScriptContainerList
- WatchdogMGR_StoreContainers2DB
- TransformCollection2Array
- WatchdogMGR_GetProcessList
- WatchdogMGR_CreateProcess

- WatchdogMGR_GetAllWatchdogProcesses
- WatchdogMGR_GetTaskIDsFromProcesses
- WatchdogMGR_WatchdogParameters2MonitoringParameters

6.4.2 WatchdogManagement

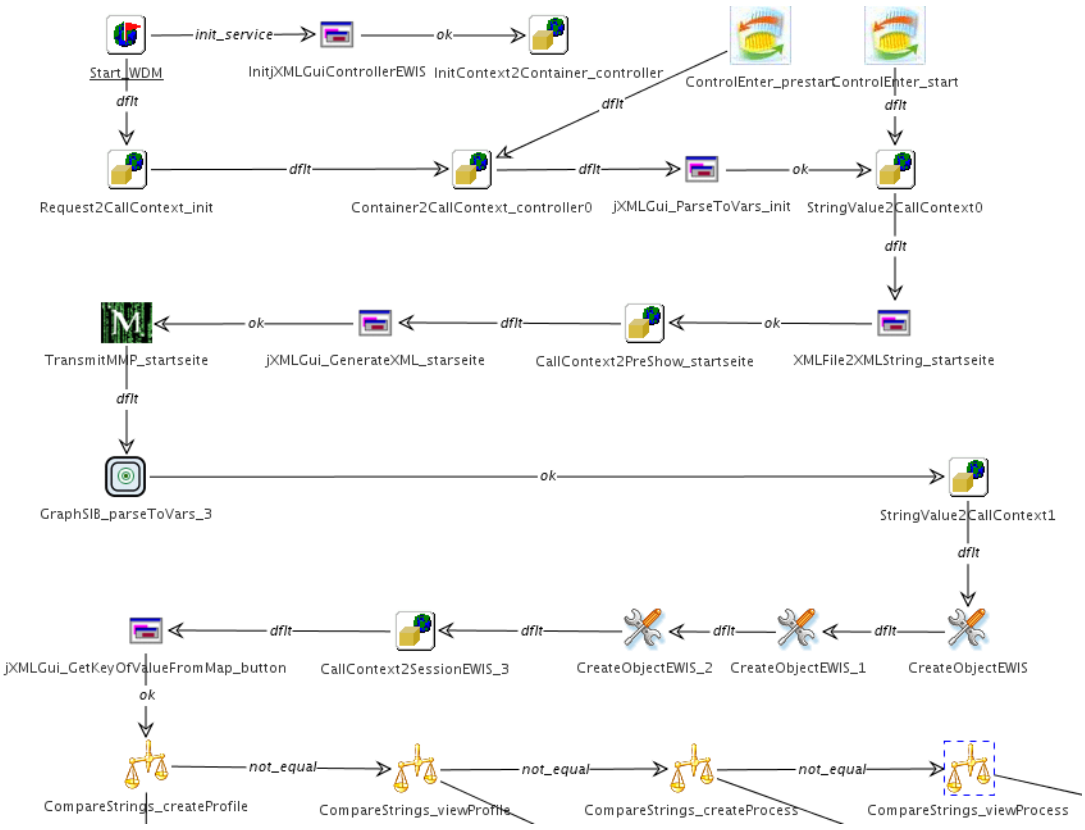

Abbildung 6.9: jABC Graph - WatchdogManagement Initial Part

Abbildung 6.9 zeigt den einleitenden Teil des WatchdogManagement-Graphs. Der ”init service” branch dient dazu die jXML-Engine zu initialisieren. Mit dem ”dflt” branch beginnt der eigentliche Service. In diesem wird zunächst dafür gesorgt, dass eine Hauptseite angezeigt wird, auf der der Benutzer zwischen den einzelnen Funktionen wählen kann. Die Entscheidung, welcher Workflow verfolgt wird, wird auf Graphebene durch die `CompareString` SIBs ermöglicht.
Wichtig sind zudem die drei `CreateObject` SIBs. Diese erstellen ArrayListen, welche später im ”Create Profile” Workflow dazu genutzt werden, verschiedene WatchdogReactions zu halten.

6.4.2.1 Create Profile

Abbildung 6.10 zeigt den ersten Teil des ”Create Profile” Workflows. In diesem werden zu Beginn alle bestehenden Measurements aus der Datenbank geholt, um sie auf der ersten Seite des Workflows anzuzeigen. Auf dieser Seite werden die Basisinformationen über das zu erstellende WatchdogProfil gesammelt und daraufhin mittels des `WatchdogMGR_StoreContainerProfile` SIBs temporär in ein ContainerProfile gespeichert. Diese ContainerProfile wird bis zum Ende des Workflows in der Session abgelegt.
Abbildung 6.11 zeigt den zweiten Teil des Workflows. Hier werden für das ausgewählte Measurement alle zugehörigen Input- und Outputannotations geholt. Diese werden nun auf einer Webseite angezeigt. Auf dieser muss der Benutzer nun Grenzwerte für jede Annotation angeben. Diese Angaben sind daraufhin in der jXML-HashMap verfügbar, welche als Eingabe für den SIB `WatchdogMGR_HashMap2ParameterContainerList` dient. In diesem werden die Informationen als WatchdogParameter in eine Containerliste abgelegt, die wiederum in der Session abgelegt wird. Abbildung 6.12 zeigt den nächsten Teil des ”Create Profile” Workflows. Hier wird dreimal das ”Watchdog_SelectReactions” Makro benutzt. In diesem steckt der Workflow, mit dem der Benutzer WatchdogReactions für jeden Level bestimmen kann.
Abbildung 6.13 zeigt dieses ”Watchdog_SelectReactions” Makro. Im ersten Schritt kann der Benutzer sich Notifications abonnieren, was im Prinzip genauso funktioniert, wie im NotificationManagement: Zunächst wählt der Benutzer einen Benutzeraccount, um dann einen dazugehörigen Kontakt auszuwählen. Der Benutzer hat zudem die Möglichkeit ein EL-Statement pro Reaktion anzugeben. Dieses wird auf Graphebene durch den SIB `ExpressionLanguageCheck` auf Validität geprüft. Alle Informationen über die Notification wird nun in einer der am Anfang erstellten ArrayListen durch den SIB `WatchdogMGR_ManageReactionNotificationContainerList` abgelegt.
Nachdem NotificationReactions definiert wurden, kann der Benutzer nun Script- und WatchdogProcessReactions erstellen. Vorhandene Skripte werden durch den SIB `GetFileListEWIS` aus dem CVS ausgelesen, vorhandene WatchdogProcesse durch den SIB `GetAllEntitiesEWIS` aus der Datenbank geholt. Die gemachten Eingaben werden getrennt in zwei der am Anfang erstellten ArrayListen durch die SIBs `WatchdogMGR_ManageReactionScriptContainerList` und `WatchdogMGR_ManageReactionProcessContainerList` abgelegt und auch in der Session abgespeichert.
Abbildung 6.14 zeigt den letzten Teil des Workflows, durch welchen eine Zusammenfassungsseite mit allen gemachten Eingaben angezeigt wird und daraufhin der Inhalt aller Container in der Datenbank abgespeichert werden. Zu Beginn dieses Teils werden die Container ArrayListen für den `PutOut2MMP` SIB in Arrays umgewandelt. Abschließend werden alle Container in der Datenbank abgespeichert. Dies darf erst ganz am Ende des Workflows geschehen, da der Benutzer den Workflow an einer beliebigen Stelle abbrechen könnte. Somit werden keine unvollständigen Eingaben in die Datenbank gespeichert.

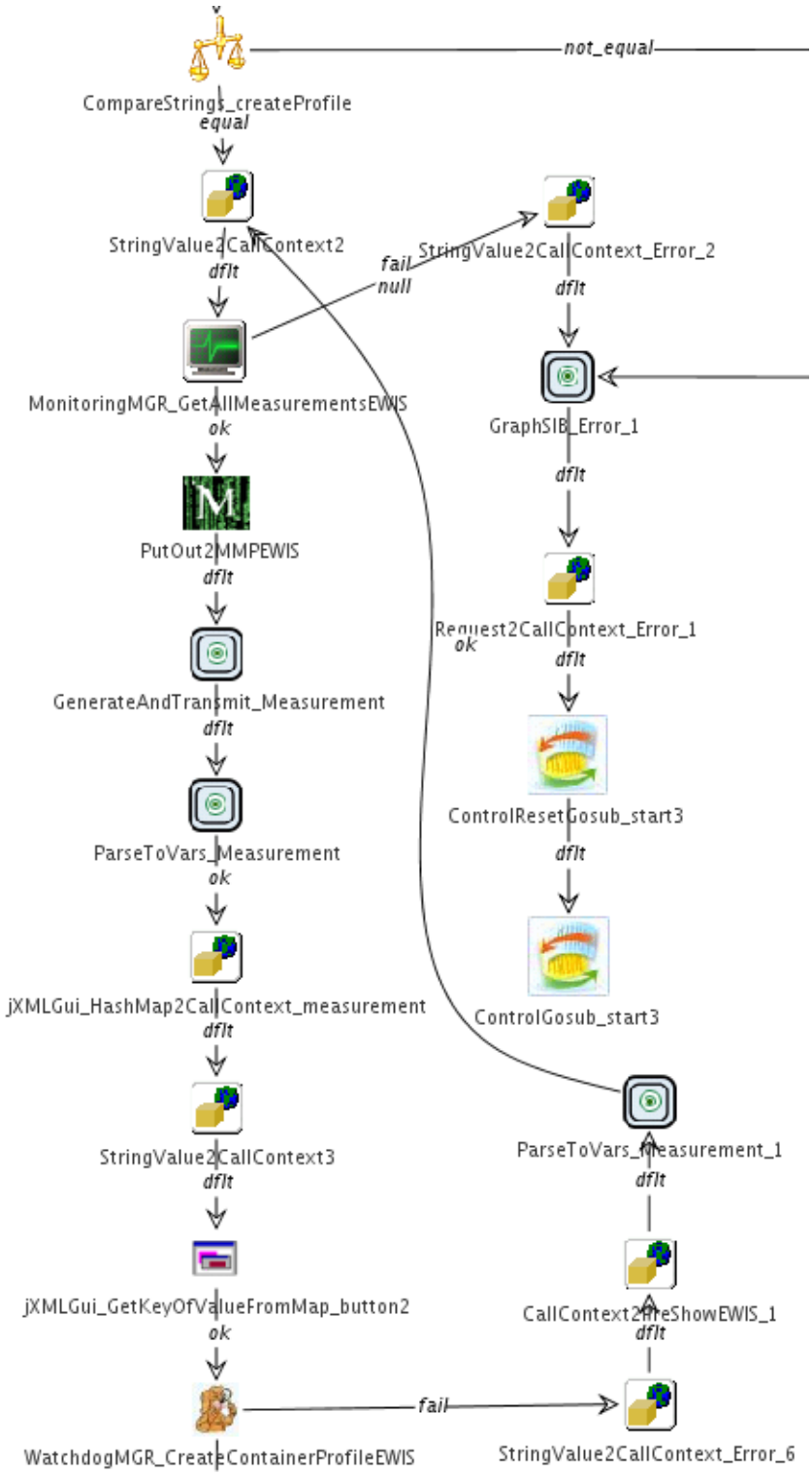

Abbildung 6.10: jABC Graph - WatchdogManagement CreateProfile - CreateContainer-Profile

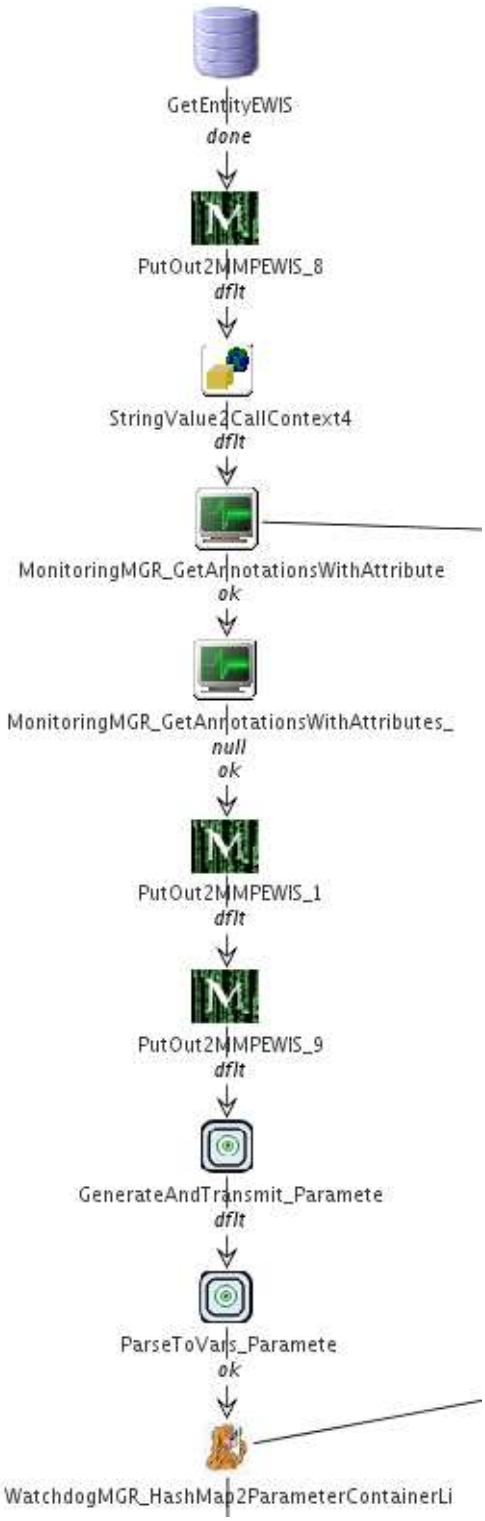

Abbildung 6.11: jABC Graph - WatchdogManagement CreateProfile - CreateParameter-ContainerList

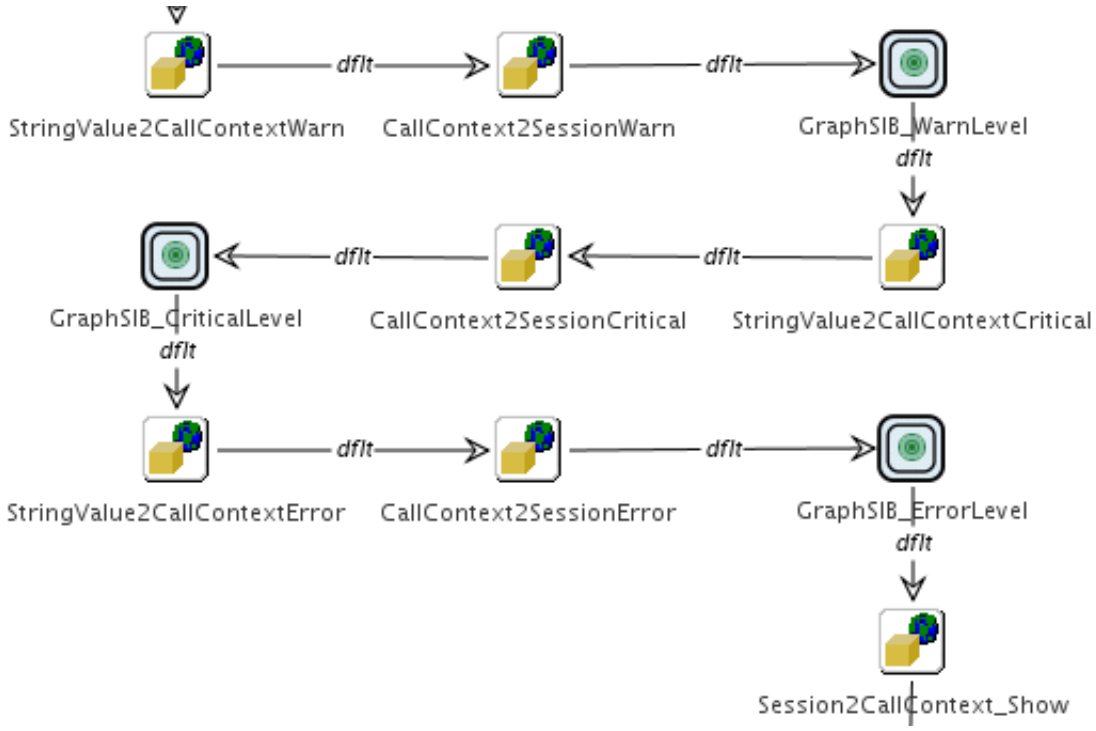

Abbildung 6.12: jABC Graph - WatchdogManagement CreateProfile - Select Reactions

6.4.2.2 View Profiles

Abbildung 6.15 zeigt den ”View Profiles” Workflow. In diesem werden zunächst alle vorhandenen WatchdogProfiles aus der Datenbank geholt und auf einer Webseite angezeigt. Auf dieser hat der Benutzer nun die Möglichkeit die Profile einzeln zu löschen. Dabei werden alle abhängigen Tabelleneinträge ebenfalls gelöscht.

6.4.2.3 Create Process

Abbildung 6.16 zeigt den ersten Teil des ”Create Process” Workflows. In diesem werden zu Beginn alle bestehenden ConfigClients aus der Datenbank geholt, um sie auf der ersten Seite anzuzeigen. Zusätzlich werden sie in der Session gespeichert (`CallContext2Session_ConfigClients`). Neben der Wahl eines ConfigClient gibt es noch zwei Buttons. Mit ”Cancel” gelangt man wieder auf die Startseite. Mit ”Next” gelangt man zum zweiten Teil des Workflows, siehe Abbildung 6.17. Hier werden als erstes die ConfigClients in den CallContext gelegt und mit `GetObjectFromArrayOrCollection_ConfigClients` wird der ausgewählte ConfigClient bestimmt und in die Session gelegt. Dieser wird am Ende für die Zusammenfassung der Auswahl benötigt. Für die nächste Seite werden alle Profile aus der Datenbank geholt,

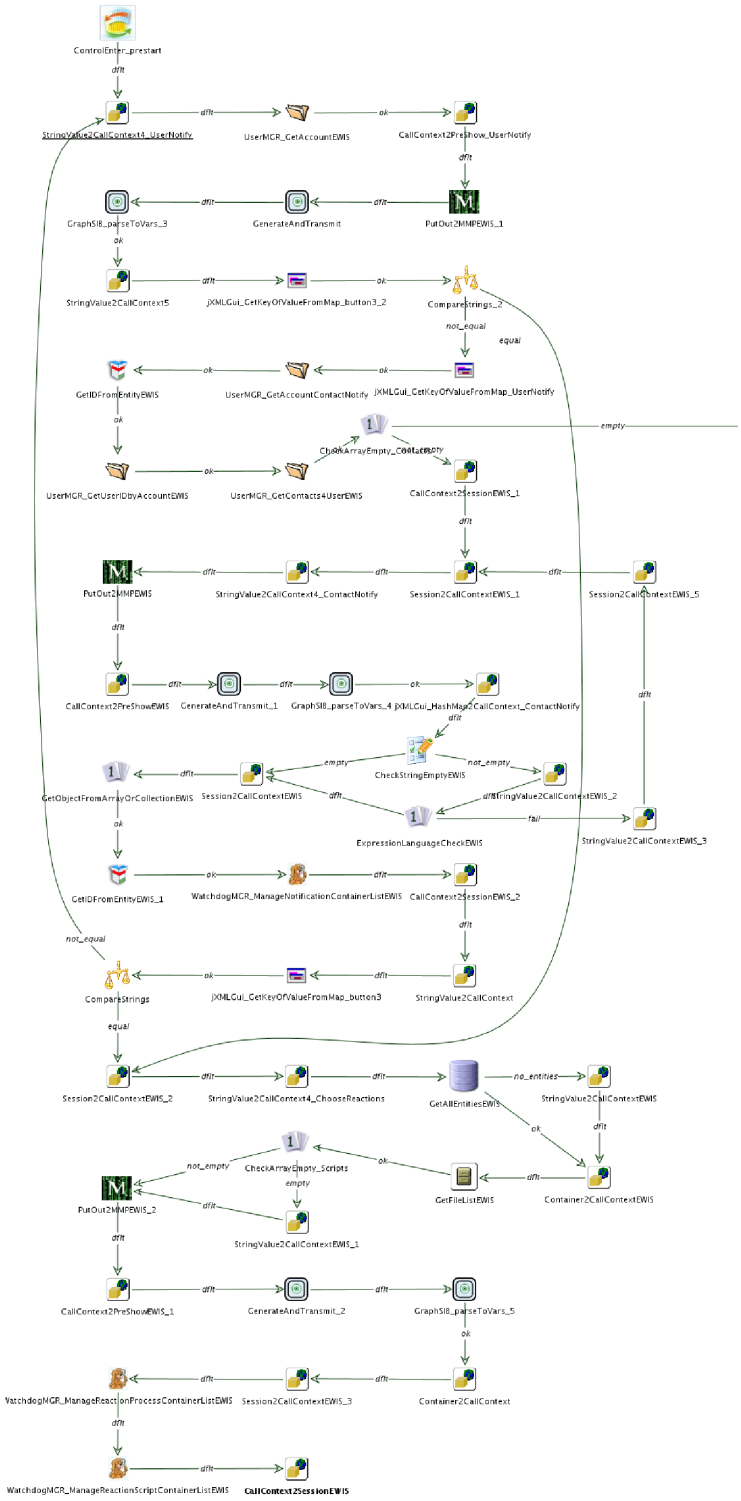

Abbildung 6.13: jABC Graph - WatchdogManagement - Macro ”Watchdog_SelectReactions”

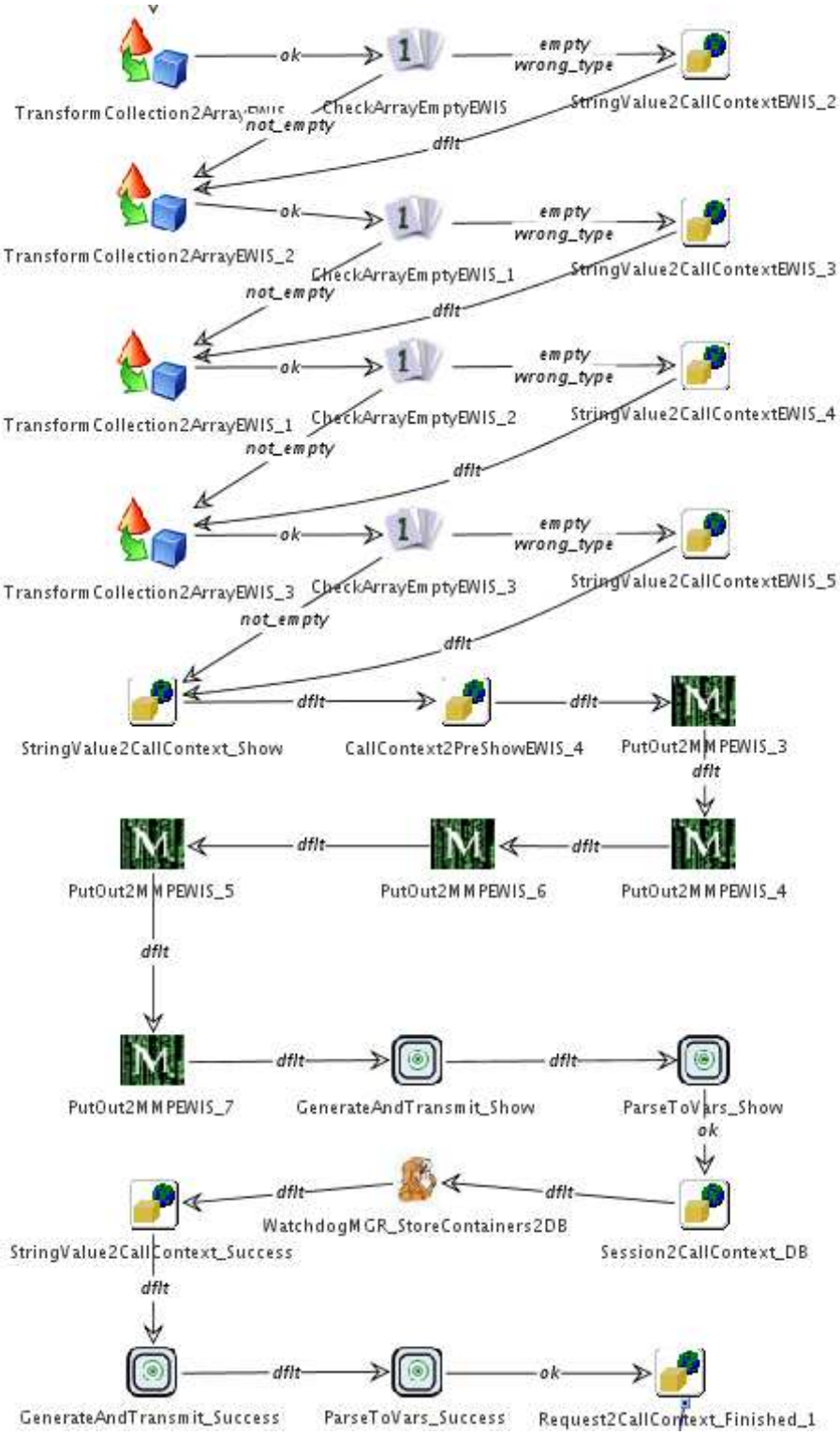

Abbildung 6.14: jABC Graph - WatchdogManagement CreateProfile - StoreContainers2DB

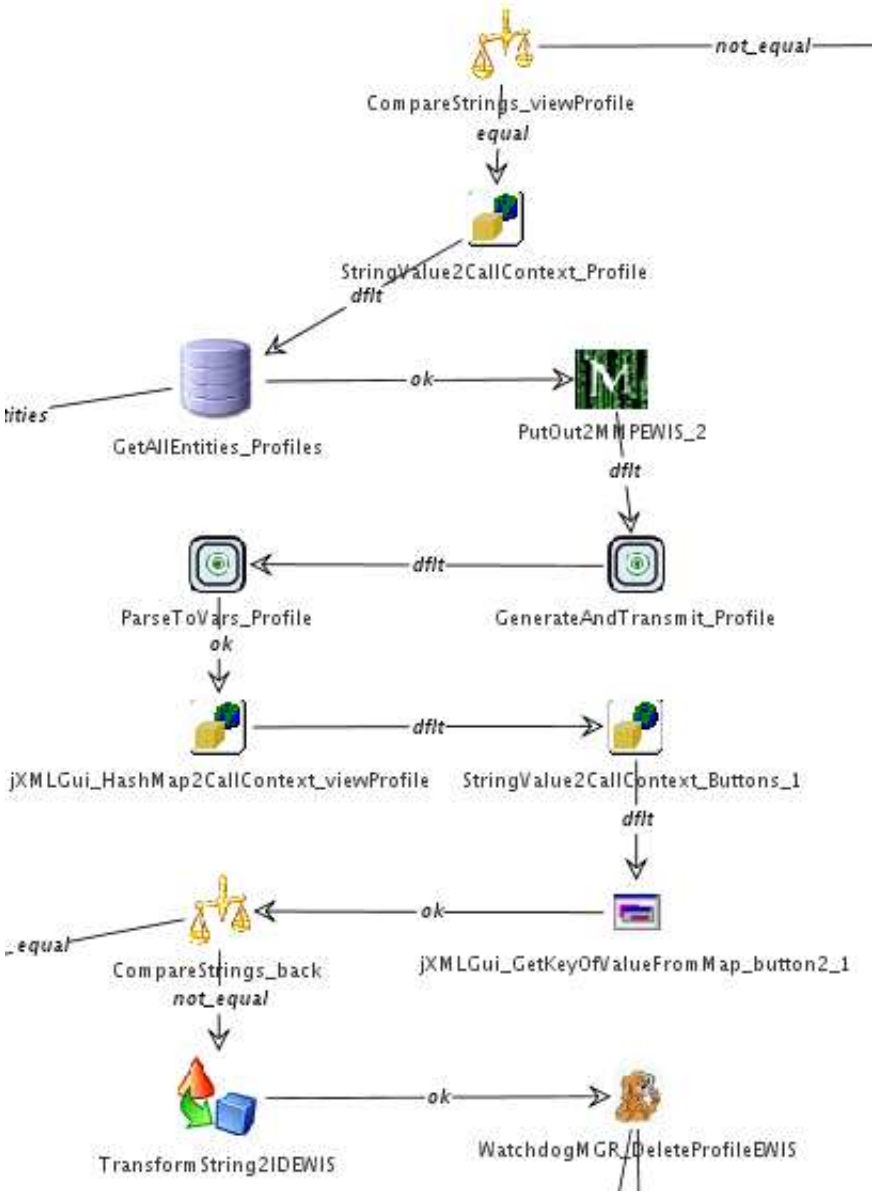

Abbildung 6.15: jABC Graph - WatchdogManagement ViewProfiles

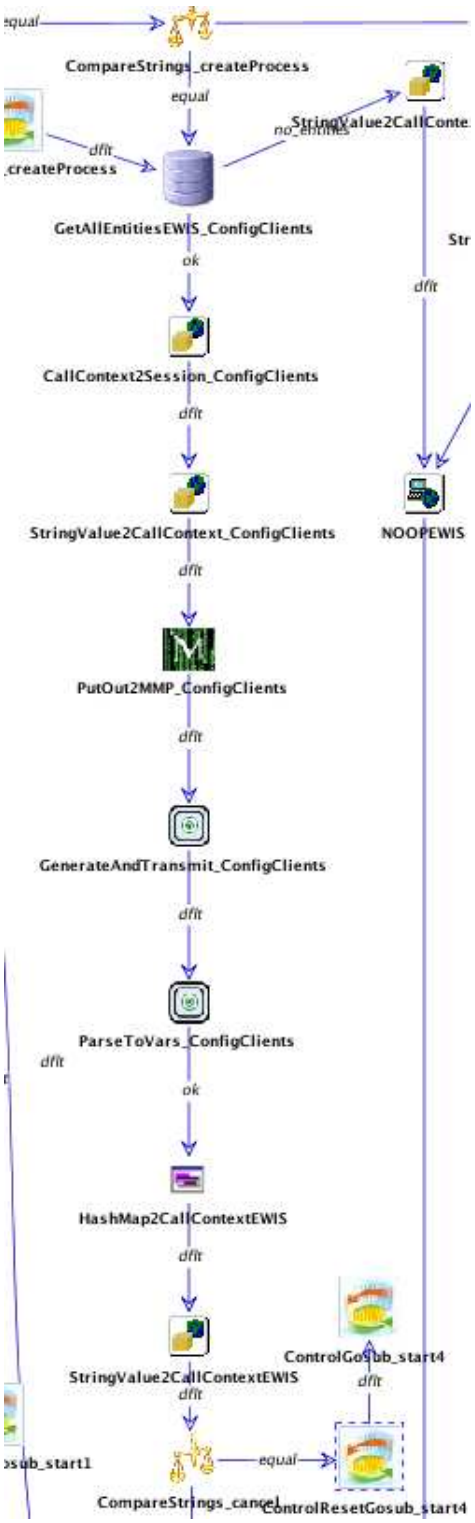

Abbildung 6.16: jABC Graph - WatchdogManagement CreateProcess - Select ConfigClient

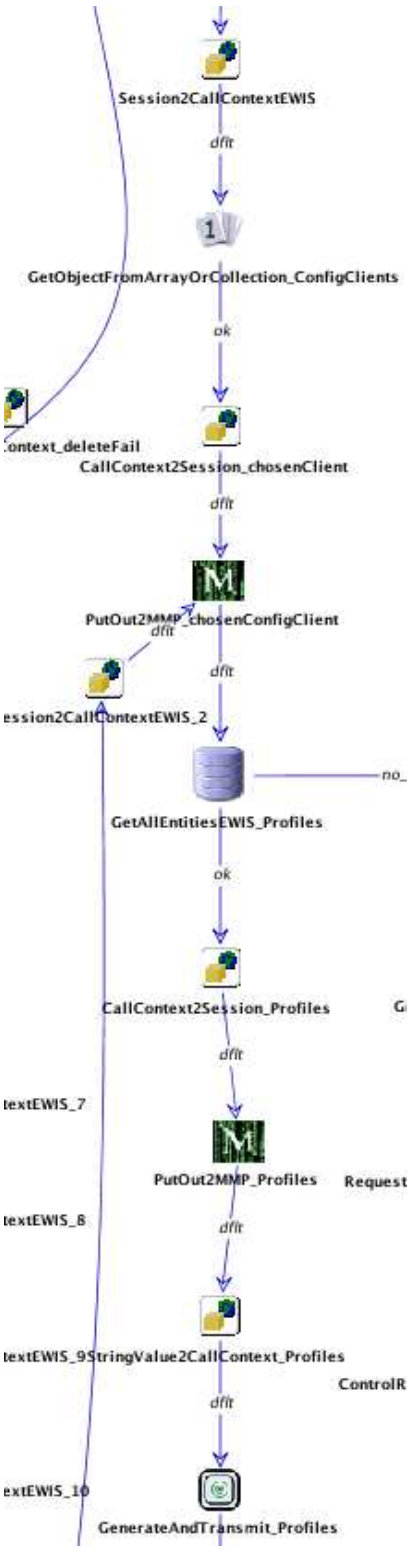

Abbildung 6.17: jABC Graph - WatchdogManagement CreateProcess - Select Profile

anschließend angezeigt und wie die ConfigClients in die Session gelegt. Nachdem man auf den ”Next” Button gedrückt hat, gelangt man zum dritten Workflow, siehe Abbildung 6.18. Auf dieser letzten Seite von ”Create Process” werden alle Werte, die in der Session zwischengespeichert wurden (ConfigClient, Process, Scheduleperiod), in den CallContext gelegt und auf der Webseite angezeigt. Der Benutzer kann die Werte einsehen, bevor er auf den Button ”Finish” drückt und den Prozess endgültig einrichtet. Neben dem Prozess wird unter anderen noch ein MonitoringTask mit allen nötigen Parametern erzeugt, die aus dem Profil und den Eingaben entnommen werden, siehe Abbildung 6.19. Desweiteren werden NotificationTasks für die im Profil ausgewählten Notifications erstellt. Nach einer Bestätigung für den erzeugten Prozess gelangt man wieder zur Startseite.

6.4.2.4 View Processes

Unter ”View Processes” kann man sich - wie der Name schon sagt - die bereits erzeugten Prozesse ansehen. Dabei werden zunächst alle WatchdogProcesse aus der Datenbank gelesen, in die Session gespeichert und auf der nächsten Webpage angezeigt. Anschließend ist es möglich, die angezeigte Webpage zu aktualisieren, den detaillierten Status der einzelnen Prozesse anzuschauen oder zur Hauptseite zurück zu gehen. Zu jedem einzelnen Prozess kann man sich außerdem die Statistiken und die Parameter anzeigen lassen (über das MonitoringManagement). Ein Prozess kann schließlich auch noch kontrolliert werden (ebenfalls über das MonitoringManagement) oder gelöscht. Letzteres erfordert, dass der gewählte Prozess in keinem WatchdogProfile benutzt wird. Dazu wird die entsprechende Tabelle des Profils nach der ID des Prozesses durchsucht. Wird ein Eintrag gefunden, wird eine Fehlermeldung auf der Webpage angezeigt und der Prozess kann nicht gelöscht werden. Ansonsten wird zunächst der Status des mit dem WatchdogProcesses erstellten MonitoringTasks auf ”removed” gesetzt und anschließend der Prozess aus der Datenbank gelöscht.

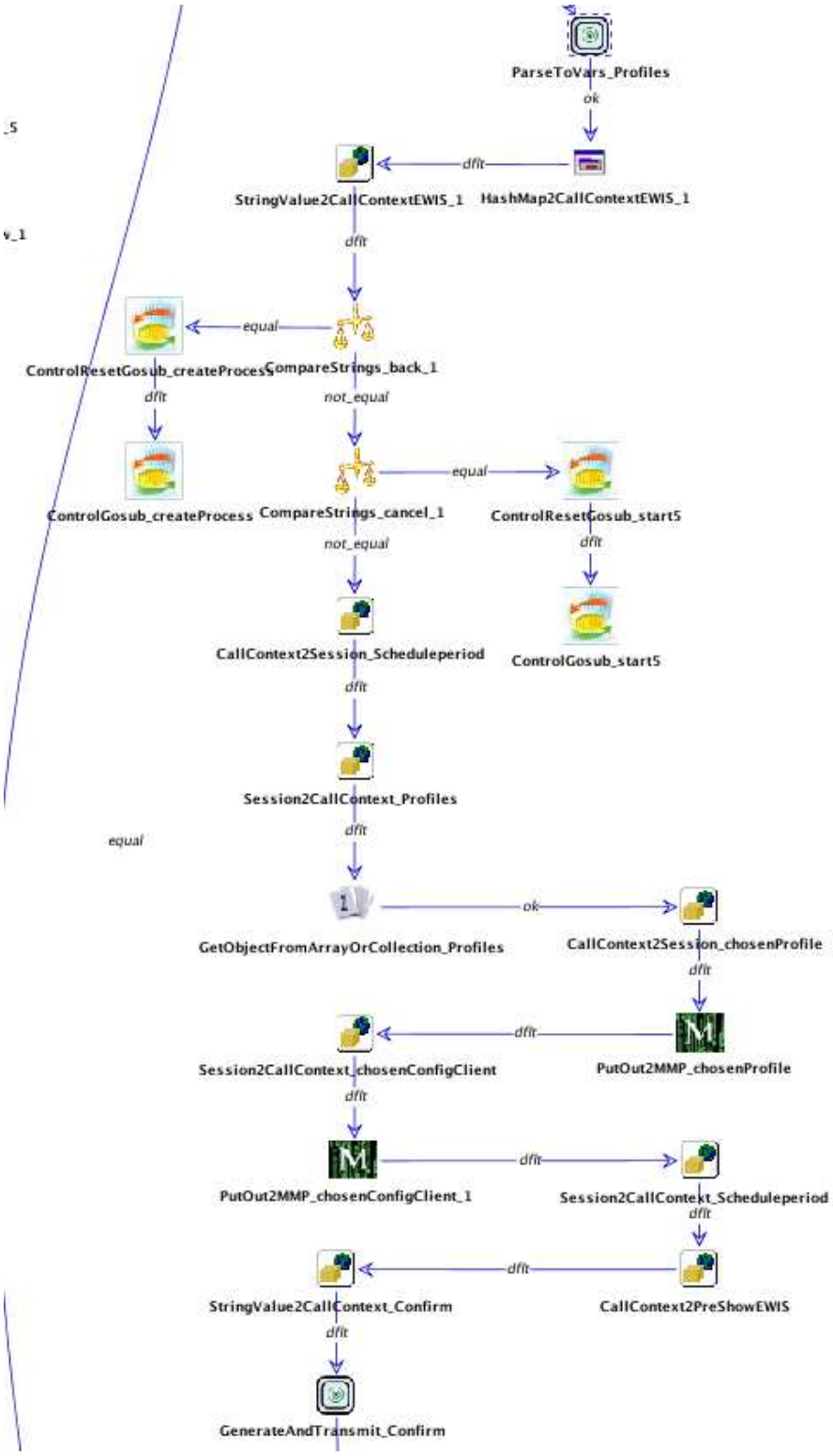

Abbildung 6.18: jABC Graph - WatchdogManagement CreateProcess - Bestätigen

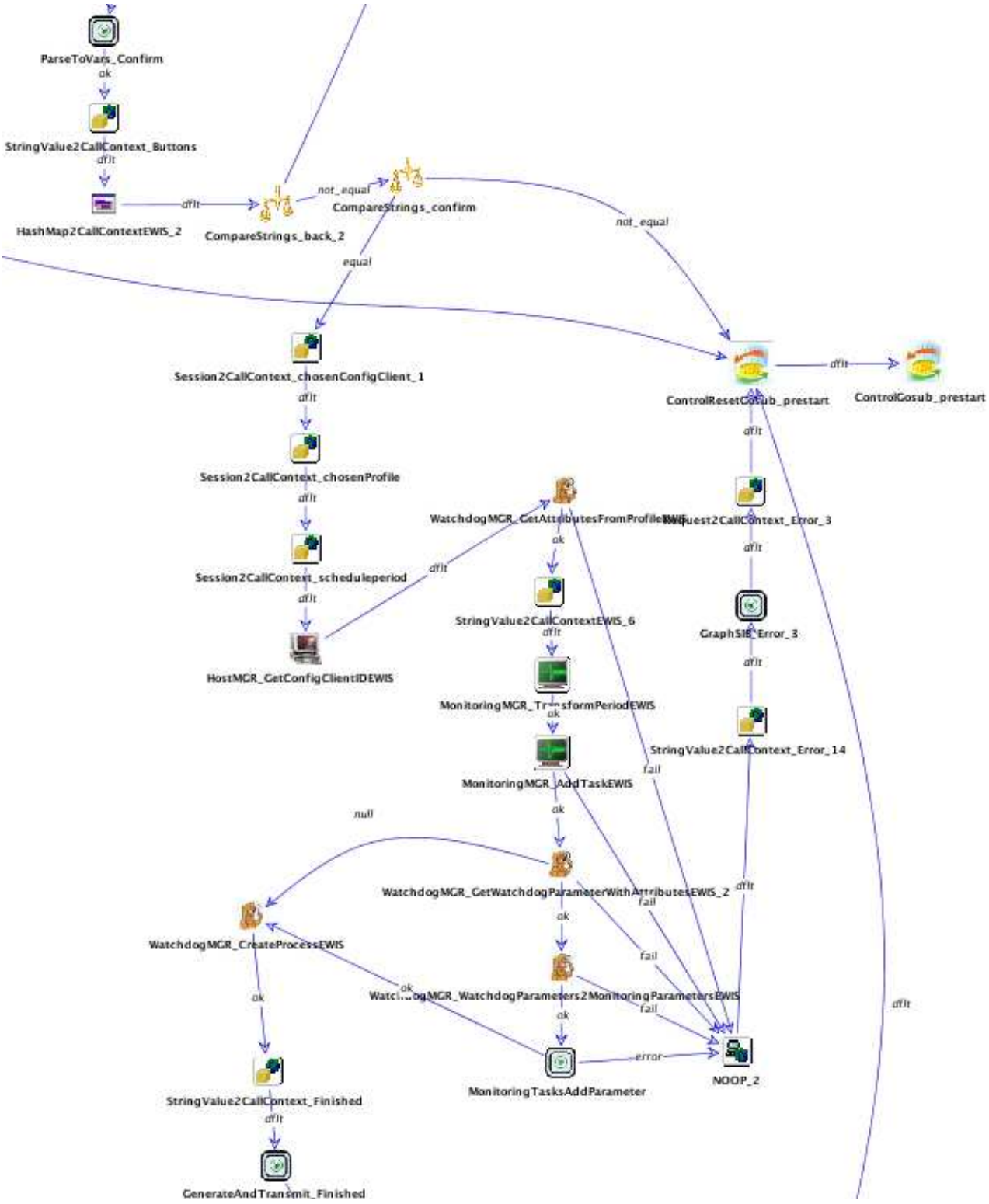

Abbildung 6.19: jABC Graph - WatchdogManagement CreateProcess - Finish

6.5 Implementierung Service

6.5.1 SIB Implementierung

Für den Service haben wir verschiedene SIBs implementiert um auf die Datenbank zu zugreifen.

Für den Zugriff auf die verschiedenen Tabellen haben wir folgende SIBs:

- WatchdogMGR_GetWatchdogActionControlInfo
- WatchdogMGR_GetWatchdogActionNotificationInfo
- WatchdogMGR_GetWatchdogActionScriptInfo
- WatchdogMGR_GetWatchdogParameterByProfileID
- WatchdogMGR_GetWatchdogProcess
- WatchdogMGR_GetWatchdogProcessByTaskID
- WatchdogMGR_GetWatchdogProfile
- WatchdogMGR_GetWatchdogReactionByParameterID
- WatchdogMGR_GetWatchdogReactionListByParameterID
- WatchdogMGR_GetWatchdogParameterWithAttributes

Für den Zugriff auf die einzelnen Spalteninhalte gibt es folgende SIBs:

- WatchdogMGR_GetAttributesFromActionControlInfo
- WatchdogMGR_GetAttributesFromActionNotificationInfo
- WatchdogMGR_GetAttributesFromActionScriptInfo
- WatchdogMGR_GetAttributesFromProcess
- WatchdogMGR_GetAttributesFromReaction

Für die Bearbeitung von StatisticResult und dessen EL-Ausdruck verwenden wir folgende SIBs:

- WatchdogMGR_CreateMapFromStatisticResult
- ExpressionLanguageEvaluate

- ExpressionLanguageCheck

Weitere SIBs, die wir verwendet haben:

- WatchdogMGR_WatchdogParameters2HashMapArrayList, packt jeden Watchdogparameter in eine HashMap für den Start eines Watchdogprozesses.
- WatchdogMGR_CheckAction, verzweigt abhängig vom Typ der Aktion
- NotificationMGR_WatchdogResult2Notification, erzeugt eine Nachricht für den Administrator
- WatchdogMGR_CheckTaskStatus verzweigt abhängig davon, ob ein WatchdogProcess gestartet bzw. gestoppt werden muss.

Sonstige Änderungen:

- ConfigClient: Wir haben den MCPThread um einen weiteren Workflow erweitert, der Bash-Skripte ausführen kann. Hierzu wurden folgende SIBs und Klassen geändert bzw. neu implementiert.
  - Parameter2Callcontext, entpackt zusätzlich aus dem option_key den input_key, der Optionen zur Ausführung des Skript enthalten kann.
  - ExecuteScript, speichert das auszuführende Skript zum Ausführen in das Verzeichnis /tmp und löscht es anschließend wieder.
  - CreateMCP, der Wert von causer_key wird im option_key hinzugefügt und durch ein Trennzeichen voneinander getrennt.
  - Type
- MonitoringControllerThread: Der Thread schreibt nun für den WatchdogThread jedes Mal, wenn er MonitoringResults in die Datenbank einträgt, die zugehörige Statistic_id in die StatisticQueue.
- GeneralControl und Web.xml: GeneralControl initialisiert den WatchdogThread und die StatisticQueue. Das SIB `InitTaskMGR_CreateQueueForWatchdog` ist so implementiert, dass es mit Hilfe des Taskmanager die StatisticQueue anlegt, falls die Queue noch nicht exitiert. Die Web.xml ist für den WatchdogThread und StatisticQueue angepasst.

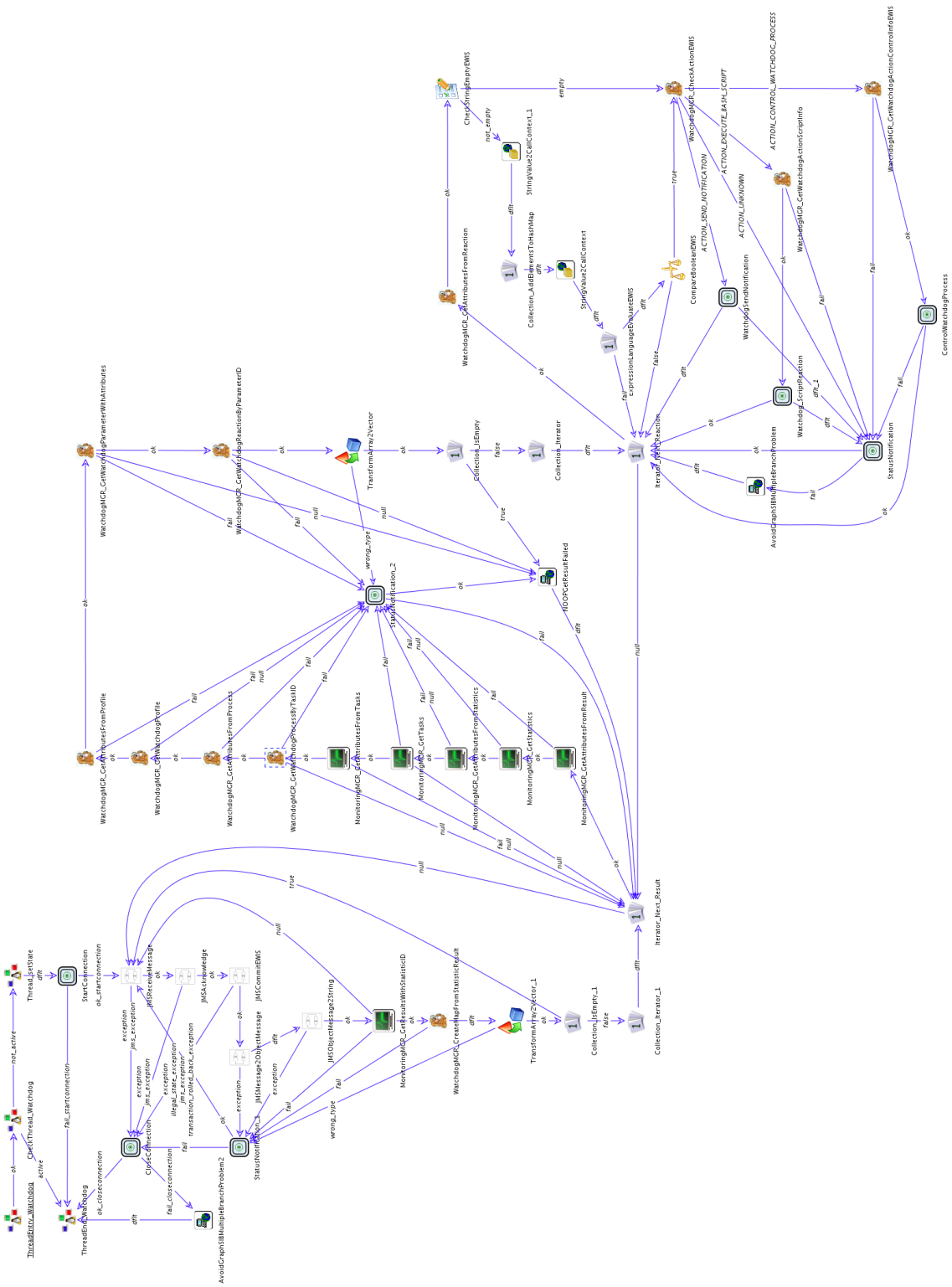

Abbildung 6.20: Watchdog, jABC Graph - WatchdogThread

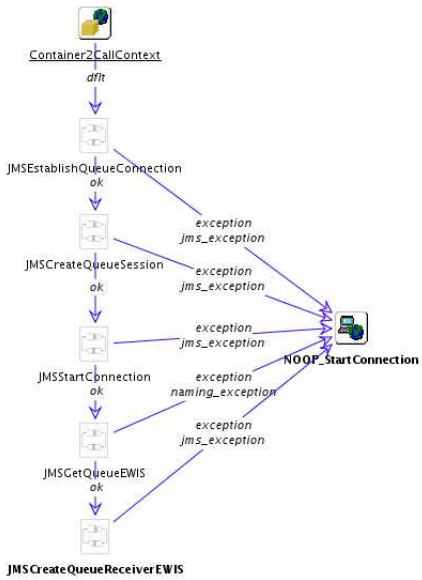

Abbildung 6.21: Watchdog, jABC Graph - StatisticQueue öffnen

6.5.2 Service Graph

Abbildung 6.20 zeigt den Graphen des WatchdogThreads. Der WatchdogThread wertet die Ergebnisse einer Watchdog-Messung aus und führt die Reaktionen aus.

Zu Beginn wird einmal die StatisticQueue geöffnet, siehe Abbildung 6.21. In jedem Durchlauf wird aus dieser StatisticQueue eine Statistic-ID gelesen; der MonitoringControllerThread schreibt die Statistic-ID in die StatisticQueue, wenn er neue Messergebnisse in die Datenbank schreibt. Falls keine Statistic-ID in der Queue ist, wird auf eine neue gewartet.
Zu der ID wird ein Array von Ergebniswerten aus der Datenbank geholt. Aus diesem Array werden eine Map für die Auswertung mit `ExpressionLanguage` und ein Vektor für den Durchlauf durch alle Ergebniswerte erzeugt, siehe Abbildung 6.22. Tritt beim Lesen der ID ein Fehler auf, wird die StatisticQueue geschlossen, und der WatchdogThread wird beendet. Er wird dann automatisch neu gestartet. Tritt beim Lesen der Ergebniswerte ein Fehler auf, wird eine Notification geschickt, und die nächste Statistic-ID geholt.

Wenn die Ergebniswerte gelesen worden sind, werden sie einzeln bearbeitet und die zugehörigen Daten werden aus der Datenbank gelesen, siehe Abbildung 6.23.
Zu jedem Ergebniswert werden die zugehörigen `MonitoringStatisticResult`, `MonitoringTask`, `WatchdogProcess` und `WatchdogProfile` aus der Datenbank gelesen. Abhängig vom `WatchdogProfile`, Level und Namen des Ergebnisses wird der `WatchdogParameter` ausgelesen. Zu dem `WatchdogParameter` wird ein Array von `WatchdogReactions` gelesen. Aus diesem Array wird ein Vektor für den Durchlauf durch alle `WatchdogReactions` erzeugt. Tritt ein Fehler beim Lesen der Daten zu einem Ergebniswert auf,

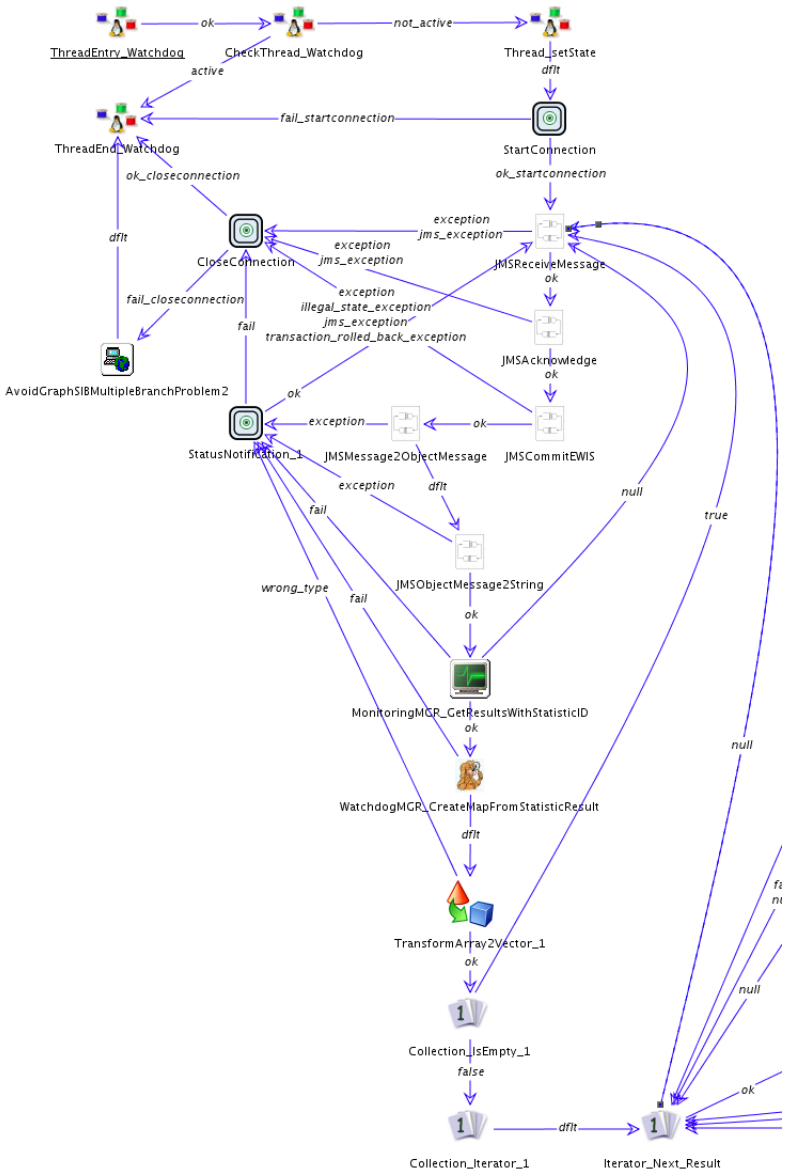

Abbildung 6.22: Watchdog, jABC Graph - Statistik-ID aus der Queue lesen

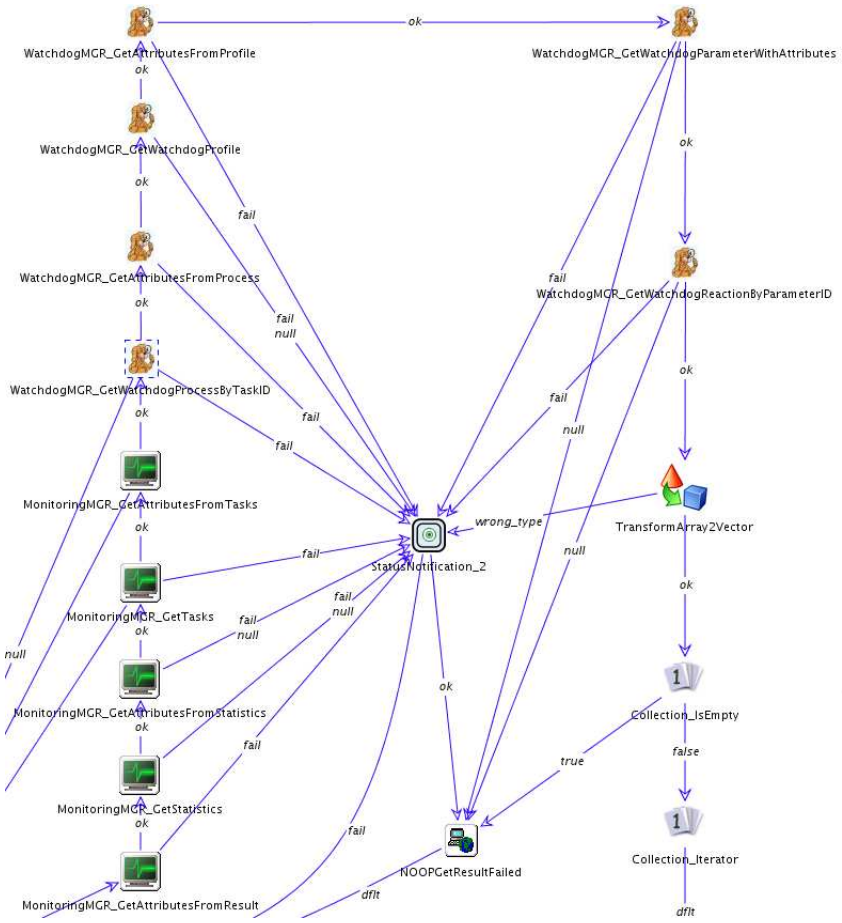

Abbildung 6.23: Watchdog, jABC Graph - Statistikdaten lesen

wird eine Notification verschickt und der nächste Ergebniswert bearbeitet.

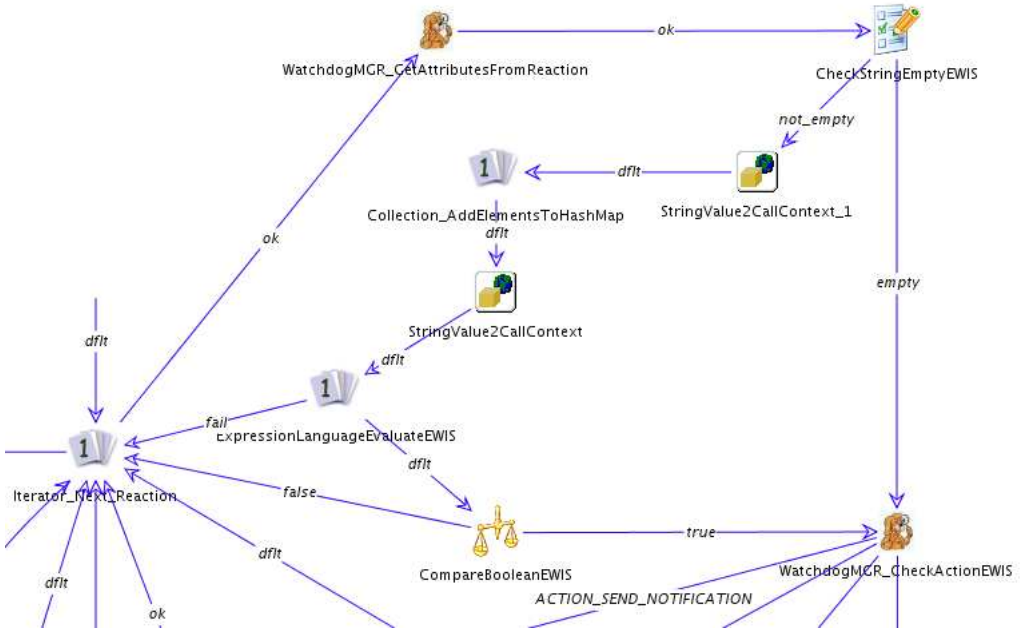

Abbildung 6.24: Watchdog, jABC Graph - Reaktion lesen

Wenn die `WatchdogReaction` gelesen worden sind, werden sie einzeln bearbeitet und für jede Reaktion wird geprüft, ob sie ausgeführt werden muss, siehe Abbildung 6.24.
Wenn einer Reaktion ein `ExpressionLanguage` Ausdruck zugeordnet ist, werden der Wert, der Level und der Name des aktuellen Ergebniswertes zu der zuvor erzeugten Map von Ergebniswerten hinzugefügt, und der Ausdruck wird ausgewertet. Ist kein Ausdruck vorhanden, oder der Ausdruck liefert true, wird die Reaktion ausgeführt. Tritt ein Fehler auf, oder der Ausdruck liefert false, wird die nächste Reaktion bearbeitet.

Die Ausführung der Reaktionen hängt vom Typ der Reaktion ab, siehe Abbildung 6.24. Der WatchdogThread verzweigt abhängig vom Typ Reaktion, liest die entsprechende `ActionInfo` aus, und führt dann das entsprechende Makro aus. Tritt hier ein Fehler auf, wird eine Notification geschickt, und die nächste Reaktion wird bearbeitet.

Für Reaktionen vom Typ `ACTION_SEND_NOTIFICATION` wird eine Benachrichtigung aus den zuvor gelesenen Daten erzeugt und in die NotificationQueue und in die Datenbank geschrieben, siehe Abbildung 6.26.

Für Reaktionen vom Typ `ACTION_EXECUTE_BASH_SCRIPT`, wird das konfigurierte Skript eingelesen und mit den nötigen Daten wie Verursacher oder Zielpfad über MCP an den ConfigClient geschickt, siehe Abbildung 6.27.

Für Reaktionen vom Typ `ACTION_CONTROL_WATCHDOG_PROCESS`, werden der Watchdog-Process, der gestartet/gestoppt werden soll, die WatchdogParameter und der Monitoring-Task zu dem Process gelesen. Die WatchdogParameter werden in eine Liste von HashMap

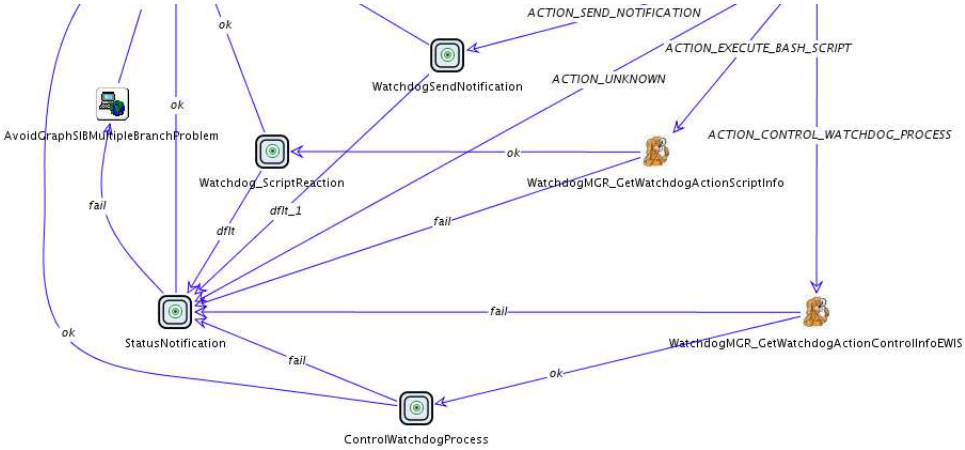

Abbildung 6.25: Watchdog, jABC Graph - Reaktion ausführen

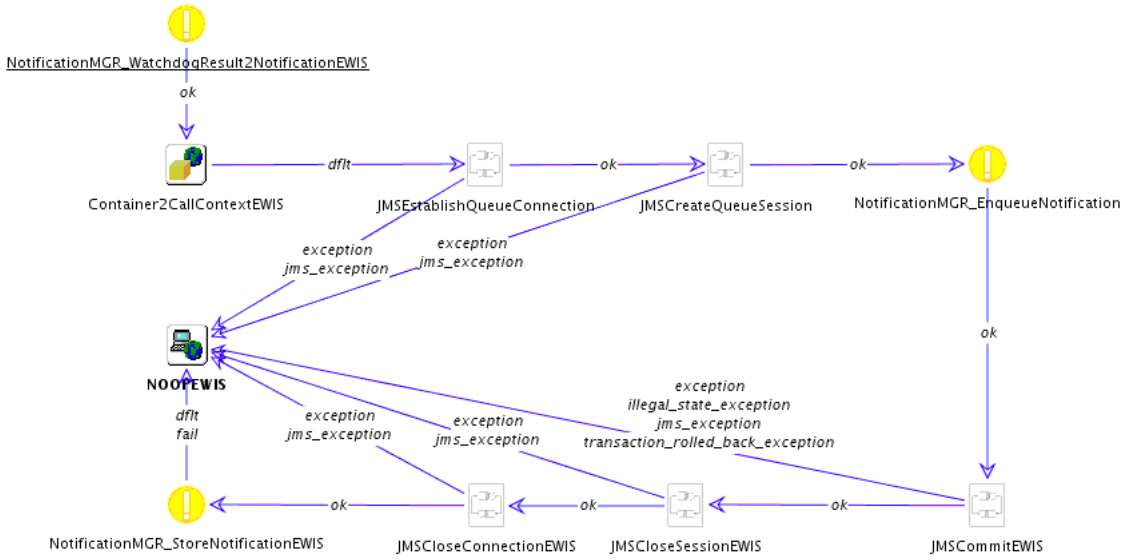

Abbildung 6.26: Watchdog, jABC Graph - Watchdog Benachrichtigung verschicken

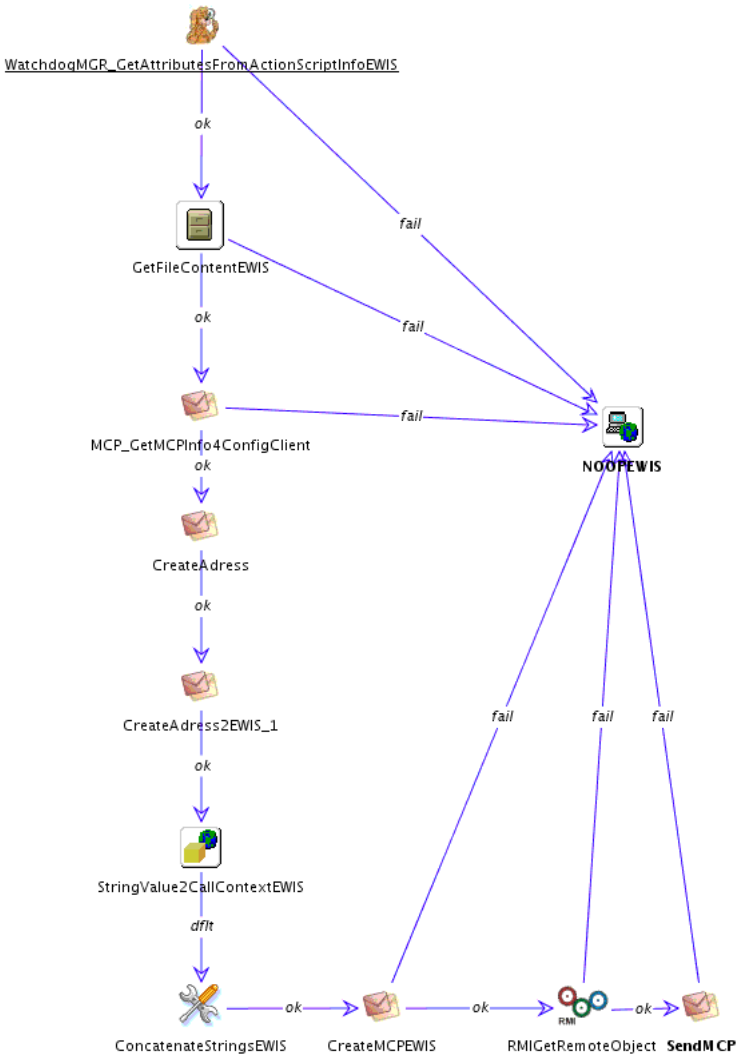

Abbildung 6.27: Watchdog, jABC Graph - Watchdog Script ausführen

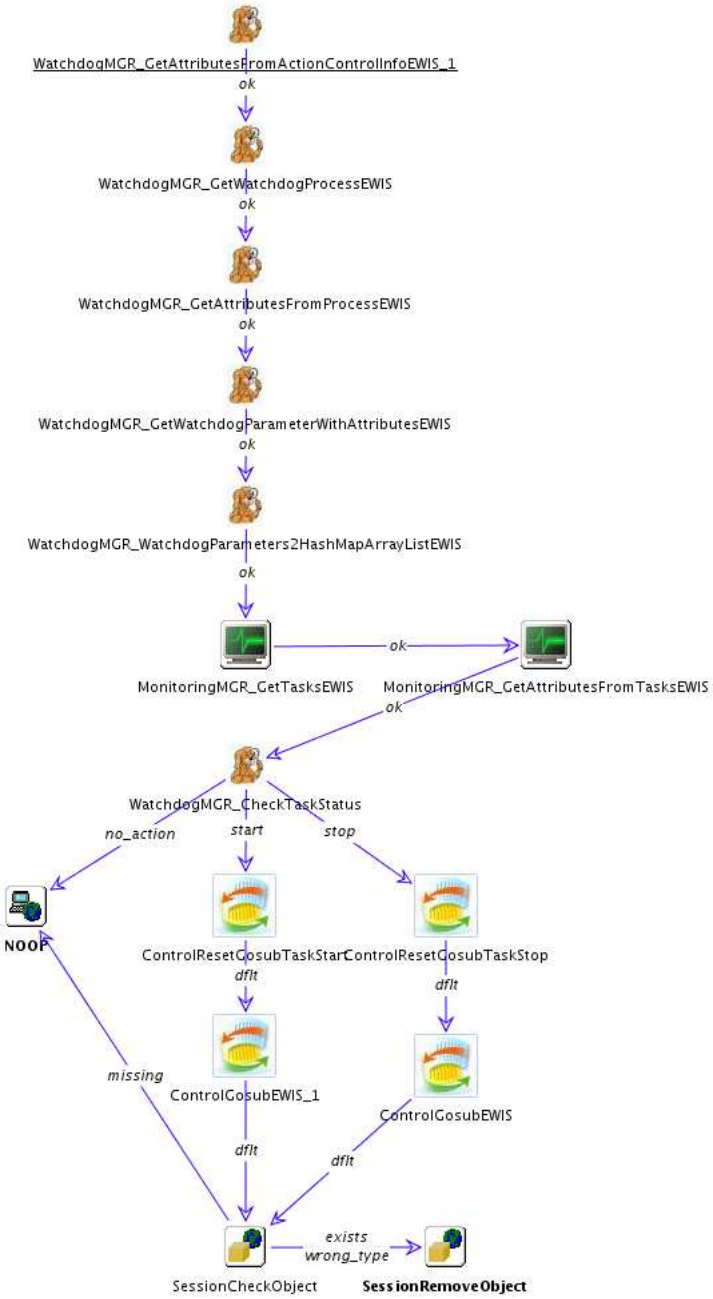

Abbildung 6.28: Watchdog, jABC Graph - Watchdog Process starten/stoppen

gespeichert. Abhängig von der konfigurierten Aktion wird der Task über das Monitoring-Management gestartet oder gestoppt. Hat der MonitoringTask schon den gewünschten Zustand, wird nichts weiter getan, siehe Abbildung 6.28.

6.6 Implementierung ConfigClient

Um auf ConfigClient-Seite Monitoring-/WatchDog Graphen ausführen zu können wird eine SIB-Bibliothek benötigt, welche mit Hilfe des ExecutionControllers ausgeführt werden kann. Diese SIB-Bibliothek sowie einige Beispielgraphen wurden von uns implementiert und hier beschrieben. Innerhalb dieser SIB-Bibliothek befinden sich zum einen SIBs welche speziell für das Monitoring geschrieben wurden als auch SIBs zur generellen Steuerung von Graphen (Schleifen, Abfragen etc.).
Im folgenden Abschnitt werden alle SIBs aufgelistet, die verändert oder neu hinzugefügt wurden. Für eine ausführliche Beschreibung der einzelnen SIBs wird das englische HOWTO des Themas empfohlen.

SIBs für den ConfigClient:

- CPU
- CPUParser
- CounterInit
- CounterIsGreater
- CounterModify
- CreateHashmap
- CreateHashmapWithValues
- DF
- DFInode
- DFParser
- DU
- DUParser
- GenerateHashMapCPU
- GenerateHashMapDF
- GenerateHashMapDU

- GenerateHashMapInterrupts
- GenerateHashMapMemory
- GenerateHashMapNetz
- GenerateHashMapPS
- GenerateHashMapUptime
- GetParameterFromHashMap
- If
- Interrupts
- InterruptsParser
- Memstat
- MemstatParser
- NOOP
- Netzwerk
- NetzwerkDiagnose
- NetzwerkParser
- ObjectIsNull
- PS
- PSbyUser
- PSParser
- PrintObject
- PutObjectsToList
- SetLevelsAscending
- SetLevelsAscendingWithHashMapSelection
- SetLevelsDescending
- SetLevelsDescendingWithHashMapSelection
- StringValue2ExecutionContext

- Thread_Sleep
- UnpackWatchdogHashMap
- Uptime
- UptimeParser

Abbildung 6.29 zeigt einen Graphen, mit dem auf dem ConfigClient der Speicherverbrauch überwacht werden kann. Neben diesem Graphen existieren weitere Graphen, welche die Überwachung der Netzwerk-Interfaces, der Interrupts, des freien Speicherplatzes auf den Festplatten, die Uptime sowie die laufenden Prozesse realisieren. Diese Graphen haben aber alle große Ähnlichkeit mit dem hier gezeigten Graph und werden daher nicht mehr separat dargestellt. In jedem Graphen sind SIBs zu finden, hier z.B. das SIB Memory, welche die Daten, die abgefragt werden sollen beschaffen (indem im /proc-Dateisystem die entsprechende Datei gelesen wird), danach die Daten parsen, z.B. mit dem SIB MemoryParser, um an die einzelnen Werte zu kommen und diese Werte dann Parametergesteuert als result zurückgeben. Dies geschieht dann in den SIBs GetParameterFromHashMap und GenerateHashMapMemory. Danach erfolgt standardmäßig eine Überprüfung, ob eine Watchdogmessung vorliegt. Sollte keine Hashmap vorliegen, so handelt es sich nur um einen Monitoringgraphen und die ”Levelzuweisung” im nächsten Teil übersprungen. Handelt es sich um einen Watchdoggraphen wird ein weiteres SIB benötigt, welches die Werte mit vorher definierten Grenzwerten vergleicht und dementsprechend ein ”Level” setzt, welches diesem Messwert entspricht. In Abbildung 6.29 wird das durch das SIB SetLevelsDescendingWithHashMapSelection erledigt.

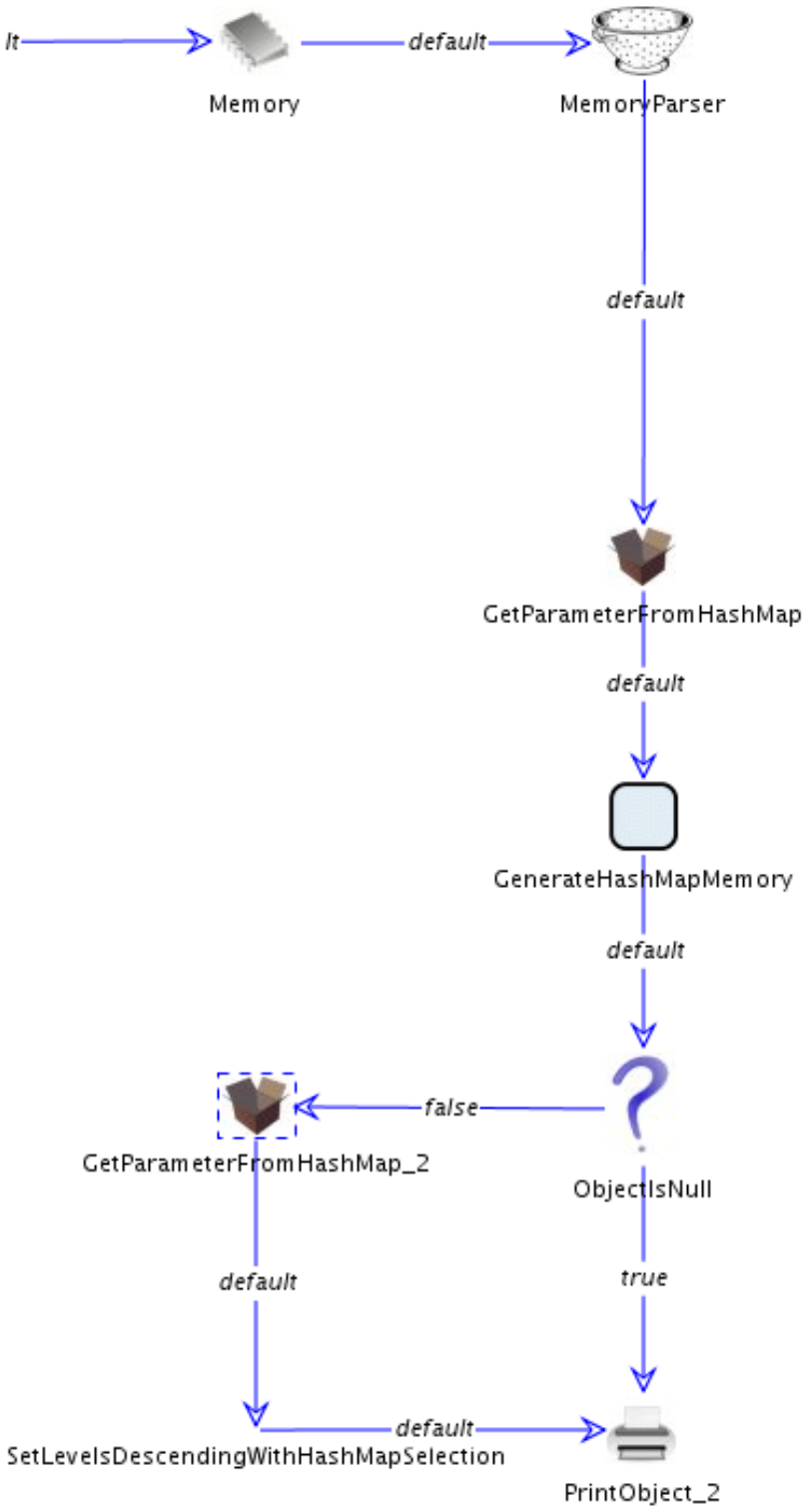

Abbildung 6.29: jABC Graph - Measurementgraph(Memory)

6.7 Beispielanwendung

6.7.1 WatchdogManagement

Abbildung 6.30: Example Application - WatchdogManagement - Main

Abbildung 6.30 zeigt die Hauptseite des WatchdogManagements. Der Benutzer kann hier neue Profile und Prozesse anlegen und sich vorhandene ansehen und eventuell löschen.

6.7.1.1 Create Profile

Abbildung 6.31 zeigt die erste Seite des ”Create Profile” Workflows. Hier muss der Benutzer Grundinformationen über das zu erstellende Profil angeben und ein Measurement auswählen. Mit einem Klick auf dem entsprechenden ”Select” Button gelangt der Benutzer auf die nächste Seite (Abb. 6.32).

Hier hat der Benutzer Grenzwerte für jede Annotation des gewählten Measurements und jeden Level anzugeben. Mit einem Klick auf ”Next” kommt der Benutzer auf die nächste Seite.

Auf der in Abb. 6.33 gezeigten Seite kann der Benutzer sich Notifications als Reaktion abonnieren. Dazu muss er einen User auswählen, wodurch er auf die nächste Seite gelangt. Natürlich muss der Benutzer keine Notification abonnieren. In diesem Fall kann er auf den Button ”No Notifications for xxx level” klicken.

Abbildung 6.34 zeigt die Auswahlseite für einen Kontakt des ausgewählten Benutzers. Hier muss sich der Benutzer für einen Kontakt entscheiden. Mit dem ”Next” Button gelangt der Benutzer auf die folgende Seite. Will er sich für eine weitere Notification registrieren, muss der Benutzer auf ”Subscribe to another Notification”.

Auf der in Abb. 6.35 gezeigten Seite kann der Benutzer Skripte oder Prozesse als Reaktion hinzufügen. Dazu muss er lediglich die Checkboxen an den entsprechenden Stellen setzen.

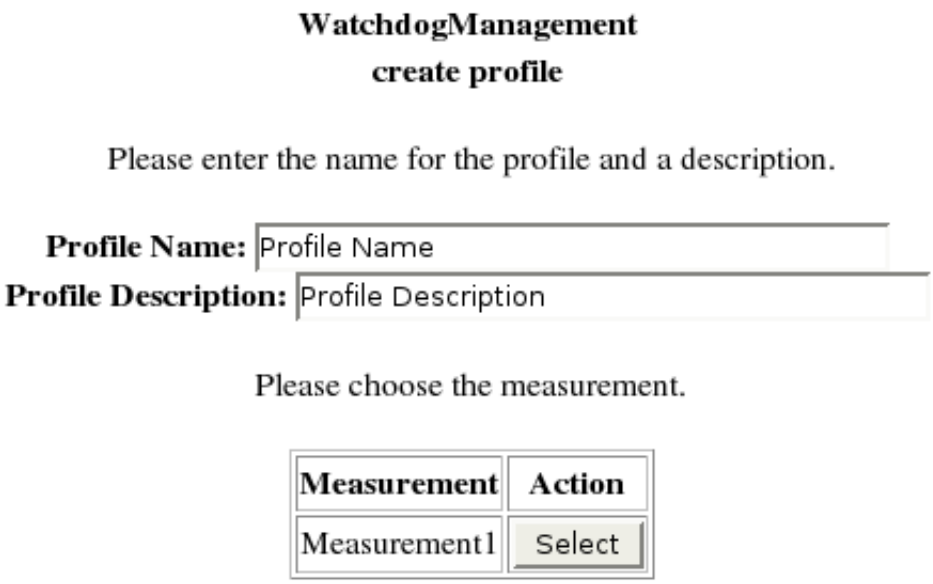

Abbildung 6.31: Example Application - WatchdogManagement - Create Profile

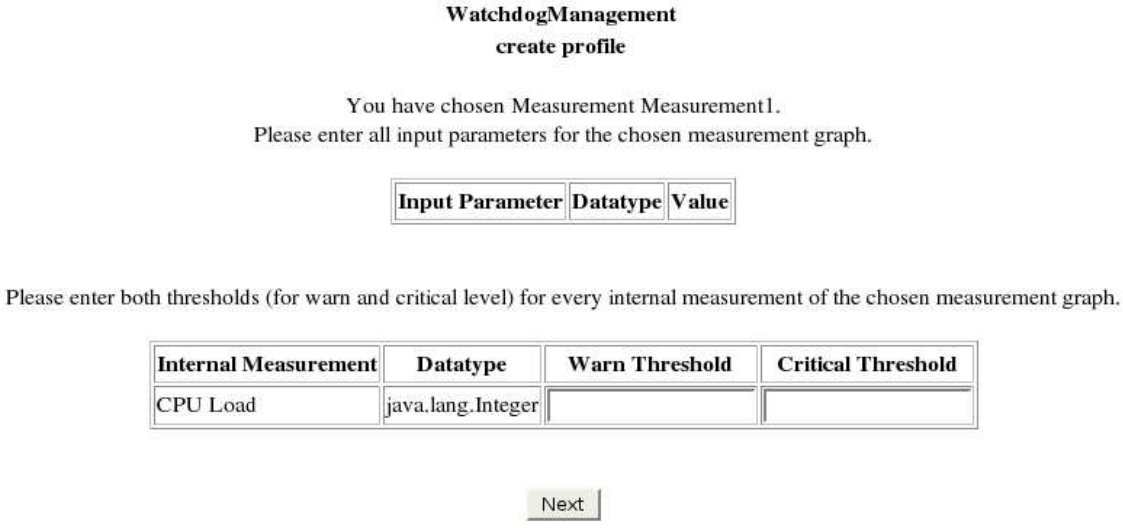

Abbildung 6.32: Example Application - WatchdogManagement - Create Profile

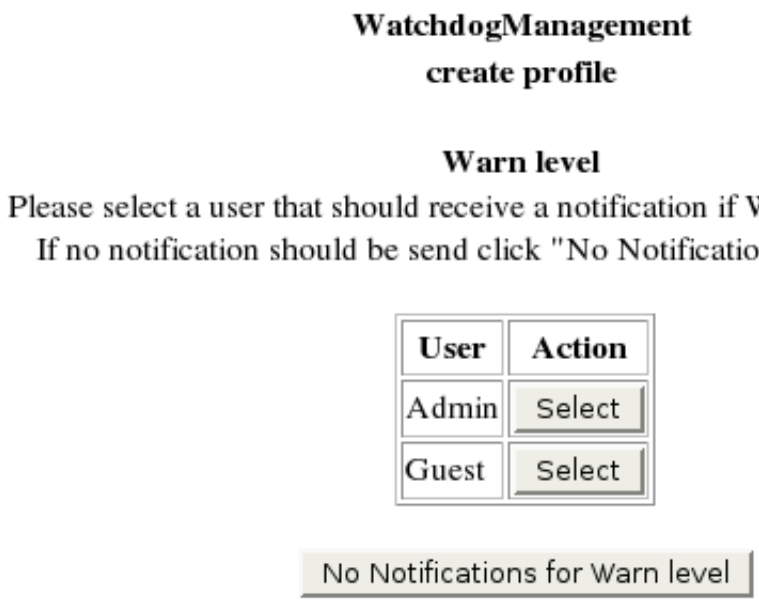

Abbildung 6.33: Example Application - WatchdogManagement - Create Profile

Abbildung 6.34: Example Application - WatchdogManagement - Create Profile

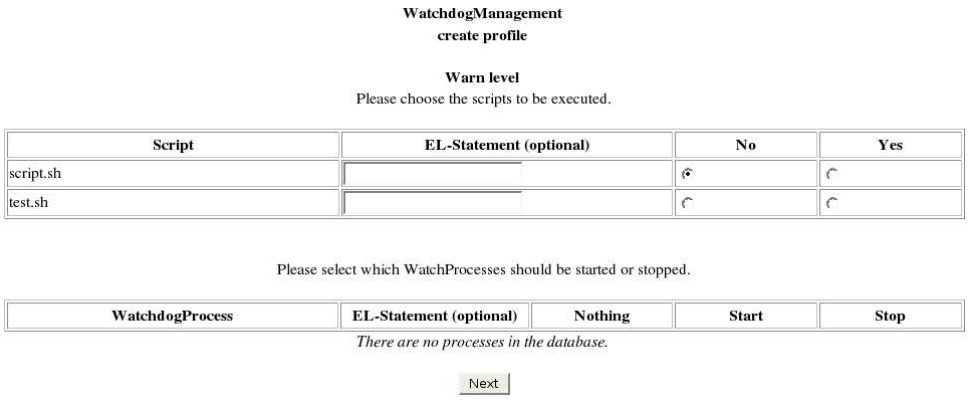

Abbildung 6.35: Example Application - WatchdogManagement - Create Profile

Mit einem Klick auf ”Next” setzt der Benutzer die Erstellung des Profils fort.
Der Reaktionsauswahl-Workflow wird insgesamt dreimal durchgeführt; jeweils einmal für jeden Level (Warn, Critical, Error).

Abschließend wird eine Zusammenfassungsseite aller Eingaben angezeigt (6.36). Mit einem Klick auf ”Submit” wird das Profil angelegt und in der Datenbank abgespeichert. Es kann nun genutzt werden, um einen WatchdogProcess anzulegen.

WatchdogManagement
create profile

Now it is time to review the settings you entered and press "Submit" if everything is correct.

Profile Name: Profile Name
Profile Description: Profile Description
Measurement: Measurement1

You entered the following parameters.

Parameters	Type	Direction	Value
CPU Load	Critical	OUT	45
CPU Load	Warn	OUT	23
CPU Load	Error	OUT	n/a

You chose the following notifications to be send.

User	Contact	Type	EL-Statement	Level
Admin	0130-12345678	SMS	n/a	Warn

You selected the following processes to be started or stopped.

Process	Action	EL-Statement	Level

No processes chosen.

You chose the following scripts to be executed.

Name	Path	EL-Statement	Level
script.sh	/tmp/MaTRICS_CVS_Sandbox_hundt/monitoring__hundt_dir	n/a	Warn

Submit

Abbildung 6.36: Example Application - WatchdogManagement - Create Profile

6.7.1.2 View Profiles

Abbildung 6.37: Example Application - WatchdogManagement - View Profiles

Abbildung 6.37 zeigt die ”View Profiles” Seite. Hier hat der Benutzer die Möglichkeit sich existierende WatchdogProfile in einer Übersicht anzusehen. Mit einem Klick auf ein ”Delete” löscht er das entsprechende dazugehörige Profil und alle abhängigen Tabelleneinträge. Falls ein WatchdogProfil aktuell in einem WatchdogProcess benutzt wird, so kann der Benutzer das Profil nicht löschen. Der Benutzer müsste zunächst den WatchdogProcess entfernen.

6.7.1.3 Create Process

Abbildung 6.38 zeigt die erste Seite des ”Create Process” Workflows. Hier muss der Benutzer einen ConfigClient auswählen. Mit einem Klick auf den ”Next” Buttons gelangt der Benutzer auf die nächste Seite (Abb. 6.39). Nachdem der Benutzer die Scheduleperiod angegeben hat, kann dieser zu dem Prozess ein zuvor erzeugtes Profil auswählen. Mit dem ”Next” Button gelangt der Benutzer zur Übersicht der Eingaben, siehe Abbildung 6.40. Auf dieser Seite hat der Benutzer noch die Möglichkeit, die Angaben zu überprüfen, bevor er dann mit den ”Finish” Button den Prozess endgültig anlegt. Hat er den Prozess angelegt, erscheint auf der nachsten Seite eine Bestätigung dazu (Abb. 6.41). Danach gelangt der Benutzer mit den ”Ok” Button wieder zurück zur Startseite.

6.7.1.4 View Processes

Abbildung 6.42 zeigt die ”View Processes” Seite. Hier hat der Benutzer die Möglichkeit sich existierende Watchdog-Prozesse in einer Übersicht anzeigen zu lassen. Zu einem per Radiobutton ausgewähltem Prozess kann man sich über die Buttons auf der rechten Seite

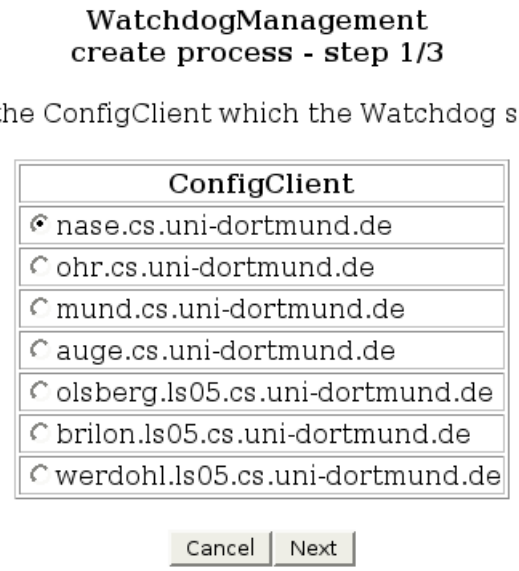

Abbildung 6.38: Example Application - WatchdogManagement - Create Process

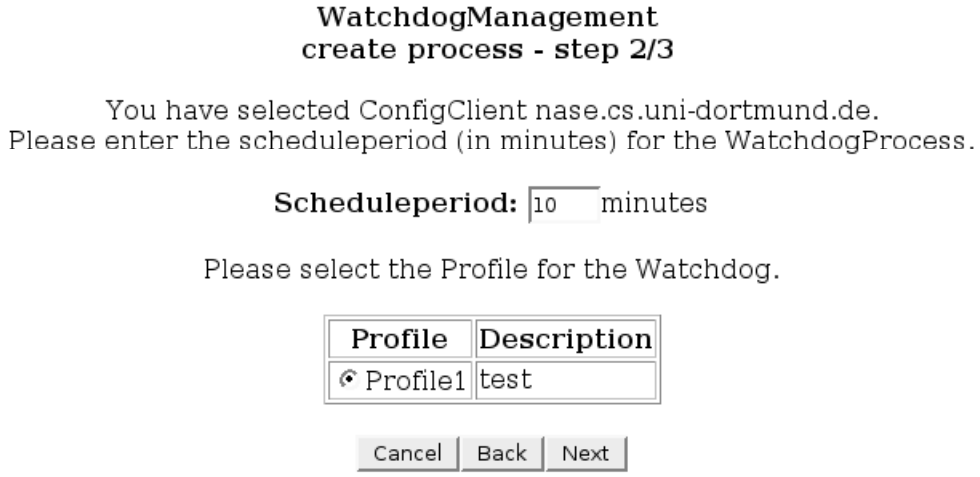

Abbildung 6.39: Example Application - WatchdogManagement - Create Process

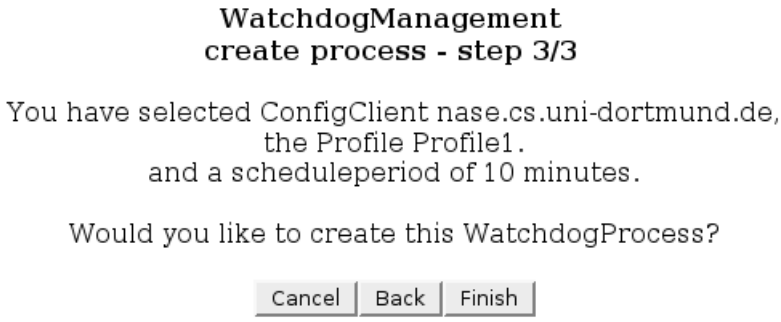

Abbildung 6.40: Example Application - WatchdogManagement - Create Process

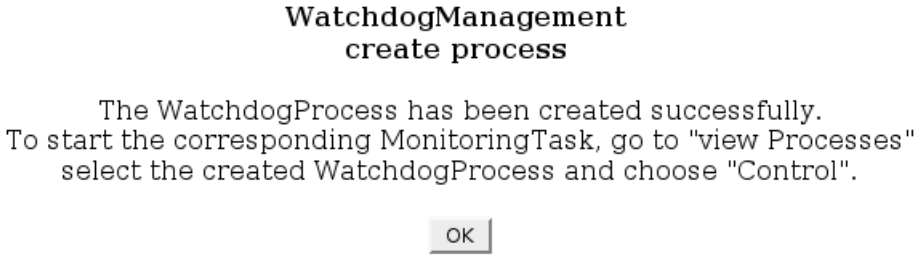

Abbildung 6.41: Example Application - WatchdogManagement - Create Process

Abbildung 6.42: Example Application - WatchdogManagement - View Process

die Statistiken und die Parameter anzeigen lassen, mit dem Button ”Control” gelangt man zur Steuerung des Prozesses. Dies ist ein Teil des MonitoringManagments, es ermöglicht den Prozess zu starten und zu stoppen. Außerdem ist es möglich, den Prozess per ”Delete”-Button zu löschen. Jedoch kann ein Prozess nicht gelöscht werden, wenn dieser in einem Profil verwendet wird.

6.7.2 Test-Graphen

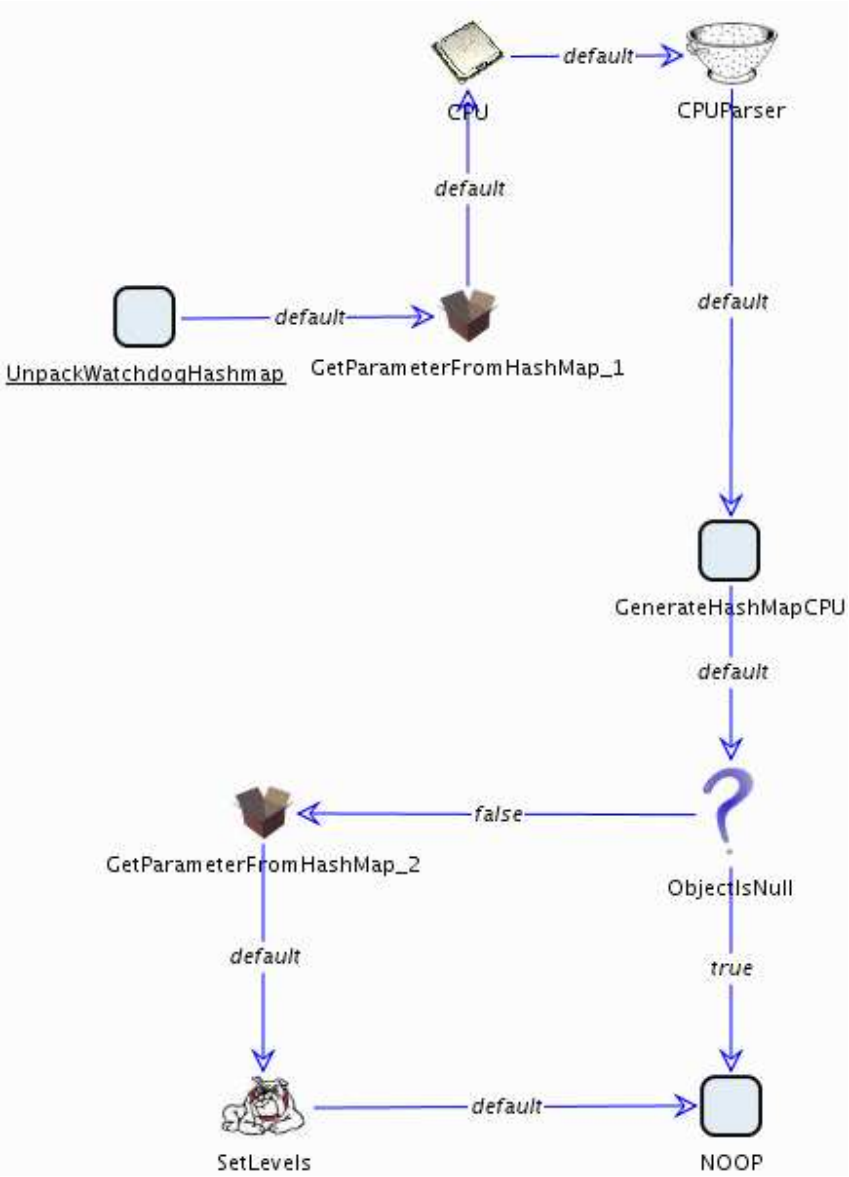

Abbildung 6.43: Watchdog, jABC Graph - CPU-Measurement

Um einen Test auf einem ConfigClient durchzuführen muss zuerst ein Graph erstellt werden, der diese Messung vornimmt, z.B. wie in Abbildung 6.43 zu sehen. In diesem Graphen werden Parameter definiert, welche den Input- und Output-Annotations im Watchdog Profil entsprechen. Der Graph beginnt mit einem ”UnpackWatchdogHashMap”-SIB

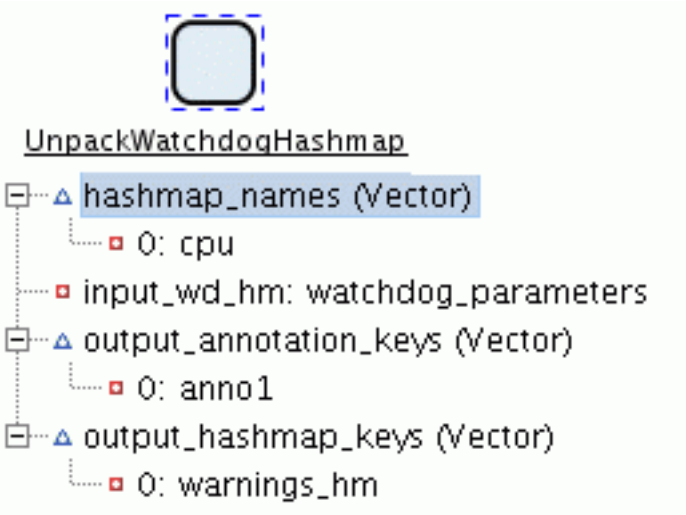

Abbildung 6.44: Watchdog, jABC Graph - UnpackHashMap

(s. Abb. 6.44), welches die ”watchdog_parameters”-HashMap auspackt und die Warn und Crit-Grenzen in separate HashMaps packt. Die Namen dieser HashMaps entsprechen den **Output Annotations** welche während der Erstellung eines Watchdog Profils angegeben werden müssen. Jede Output Annotation entspricht einem separaten Satz von Warn- und Crit-Grenzen, so dass für verschiedene Messungen innerhalb eines Graphen auch verschiedene Grenzwerte benutzt werden können.
Als nächstes kommt ein ”GetParametersFromHashMap”-SIB (s. Abb. 6.45). Dieses SIB holt alle Parameter aus der ”monitoring_hashmap”, welche füer den weiteren Ablauf des Graphen benötigt werden. Diese Parameter entsprechen den **Input Annotations**, welche während der Erstellung eines Watchdog Profils angegeben werden müssen.
Nach diesen Vorbereitungen kann die Messung nun stattfinden. Hierfür werden die entsprechenden Messungs-SIBs (z.B. CPU und CPUParser für eine Messung der CPU Last) eingefügt.
Nach dem Parser-SIB folgt das ”CreateHashMap...”-SIB (s. Abb. 6.46), welches die Ergebnisse für diese Messung in eine ”Result”-HashMap packt. Für einen Grossteil der Messungen ist es von Nöten Parameter anzugeben, welche die genaue Messung beschreiben (einzig die ”DF”-Messung benötigt keine Parameter). Wenn der Graph sowohl als Monitoringgraph als auch als Watchdoggraph benutzt werden soll muss nun die ”watchdog_hashmap” überprüft werden, ob sie nicht vielleicht ”null” ist.

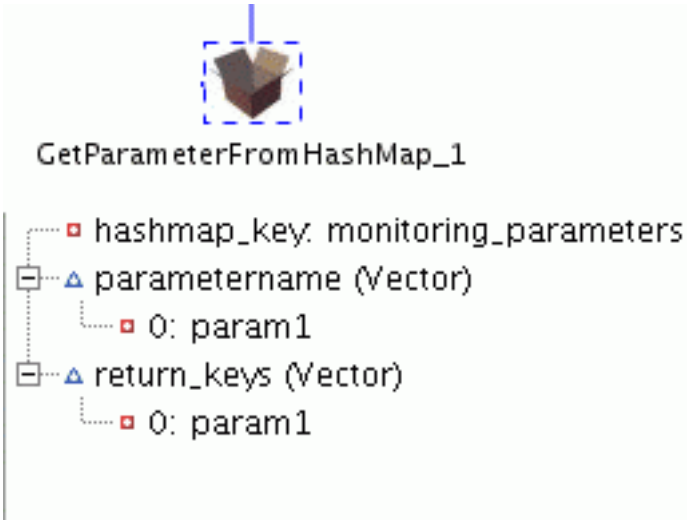

Abbildung 6.45: Watchdog, jABC Graph - GetParameterFromHashMap

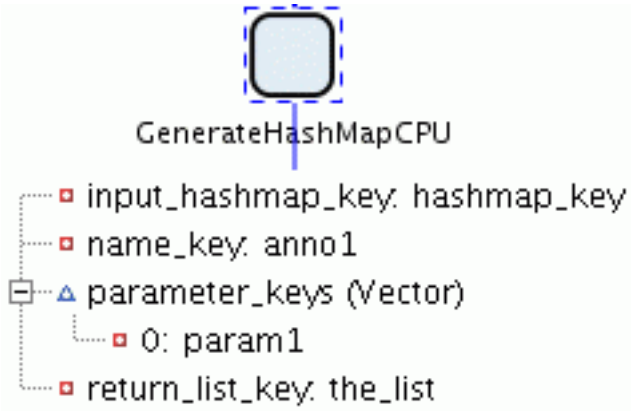

Abbildung 6.46: Watchdog, jABC Graph - CreateHashMapCPU

Wenn wir einen Graphen mit nur einem Satz von Warn/Crit Werten haben, müssen wir nun mit einem weiteren ”GetParametersFromHashMap”-SIB (s. Abb. 6.47) die Warn und

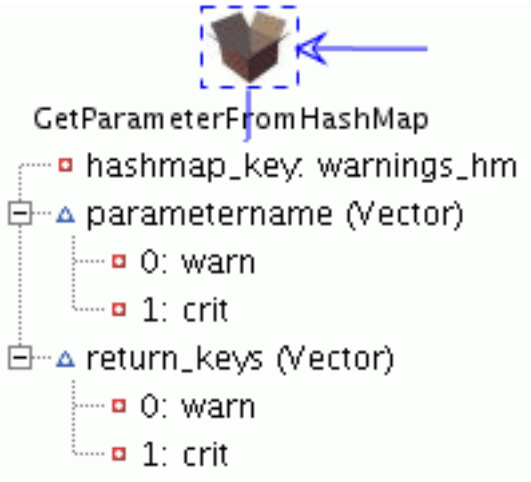

Abbildung 6.47: Watchdog, jABC Graph - Get Warn/Crit Parameter

Crit Werte aus der HashMap, welche am Anfang erzeugt wurde, auslesen.
Danach überprüft das ”SetLevels”-SIB (s. Abb. 6.48) die Resultate anhand der angegebenen Grenzwerte und setzt die Levels im Result entsprechend. Wenn mehre-

Abbildung 6.48: Watchdog, jABC Graph - Set Levels

re Messungen mit verschiedenen Warn/Crit Grenzen durchgeführt werden, muss stattdessen das ”SetLevels...withHashMapSelection”-SIB benutzt werden, welches direkt die Warn/Crit-HashMaps benutzt, so dass hier die Parameter nicht erst ausgepackt werden müssen.

Kapitel 7

Fazit

Die einjährige Arbeit an der Projektgruppe war so gestaltet, dass das gesamt Ziel in Teilaufgaben geschafft wurde, die zum Ende hin immer größer wurden. Die Teilnehmer erlernten neben dem Umgang mit dem jABC viele weitere Softwareanwendungen. Programmiert wurde in Java, und in der PG bekam man gut zu sehen, was alles mit einer Programmiersprache möglich. Über dies verschaffte sie Leuten, die bis dahin wenig praktische Erfahrung hatten einen guten Einblick in das Gebiet der Softwareentwicklung. Die Teilnehmer konnten eine solide Basis von Wissen und Fähigkeiten aufbauen, die sie nach der PG schnell ausbauen können. Auch die Teilnehmer mit mehr Erfahrung hatten die Möglichkeiten sich neue Dinge anzueignen oder sich in vorhandene Bereiche zu vertiefen.

Die Arbeitsbereiche waren nicht nur modern ausgestattet, sondern auch reichlich vorhanden. In den insgesamt vier Pools am Lehrstuhl 5 gab es weder Platzprobleme noch ein Mangel an Arbeitsplätzen.

Trotz der Vielzahl an verschiedenen Studenten aus unterschiedlichen Ländern gab es keine sprachlichen Barrieren. Auch gab es weder Streit noch große Kommunikationsproblem.

Die meisten Probleme gab es am Anfang mit dem jABC. Dieses war eine Erneuerung vom alten ABC und wir waren einer der ersten, die damit arbeiten durften. Wie man es von neuen Programmen kennt, gibt es oft viele Bugs. Das Programm und vor allem die MaTRICS selbst waren so komplex, dass oft nur noch die Betreuer die letzte Rettung waren um wieder alles zum laufen zu bringen. Diese jedoch hatte nicht immer die Zeit für 13 Teilnehmer die Fehler zu beseitigen was besonders am Anfang dazu führte, dass viele mit der Hälfte der Zeit damit beschäftigt waren, ihre MaTRICS wieder zum laufen zu bringen. Dieses führte oft zu großem Frust.

Im Großen und Ganzen war jedoch die PG sehr positiv. Wir haben viele neue praktische Dinge gelernt. Unsere Betreuer wirkten immer sehr motiviert auf uns und glänzten mit viel wissen.

Literaturverzeichnis

[1] Avetana GmbH *Avetana Bluetooth API - http://www.avetana-gmbh.de/avetana-gmbh/produkte/jsr82.eng.xml*

[2] Sourceforge - AvetanaBT *http://sourceforge.net/projects/avetanabt*

[3] BlueZ *Official Linux Bluetooth protocol stack - www.bluez.org*

[4] JavaABC Framework *www.jabc.de*

[5] METAFrame Technologies GmbH *1999 The Agent Building Center: User s Guide*

[6] METAFrame Technologies GmbH (Kai Plociennik, Susanne Bollin, Andreas Holzmann, Markus Nagelmann, Volker Braun) *The Agent Building Center: Bringing a web application into operation*

[7] T. Ermer, M. Meyer *2004 Die Linuxfibel*, http://www.linuxfibel.de

[8] K. Gerhardt *2005 Bash-Befehle und Bash-Programmierung*, http://www.kg-it.de/-linuxseiten/index.php?index=themes_bash

[9] OpenLDAP *www.openldap.org*

[10] Java Naming and Directory Interface *java.sun.com/products/jndi*

[11] Howto and Tutorial (Hakim Adhari, Bastian Reining, Hasan Tasdemir) *2005 Design of a communication component based on RMI*

[12] Howto and Tutorial () *2004 Design of a communication component based on JMS|JMX|SOAP|RMI*

[13] Termpaper (Thomas Wilk) *2004 Communication via JMS*

[14] Getting startet with JBoss (Luke Taylor and The JBoss Group) *2004 J2EE applications on the JBoss 3.2.x Server*