Endbericht

SOS for MaTRICS

Service Orchestration System for Management Tool for
Remote Intelligent Configuration of Systems

Teilnehmer:
Rumen Angov, Christian Bergmann, Björn Beulshausen, Marcel Krüger, Michael Lybecait, Gergana Nalbantova, Eugen Reinke, Christopher Schaumann, Marina Schwacke, Ahmet Tosun, Marina Velikova, Mark Zieten, Timoteus Ziminski, Isam Mohamed Fadol

PG 519
am Fachbereich Informatik
der Universität Dortmund
Lehrstuhl V

25. August 2008

Betreuer:

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

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Einleitung / Übersicht

1.1 Einleitung

1.2 Was ist die MaTRICS?

MaTRICS besteht aus 3 verteilten Komponenten, siehe Abb. 1.1: dem ConfigManager, den ConfigClients und den ConfigAgents. Die Akteure der MaTRICS sind die zu konfigurierenden Server.

- Kern der MaTRICS ist der ConfigManager. Dieser realisiert die einzelnen Workflows zu den Administrations- und Konfigurationsvorgängen eines Servers in Form von Features und Diensten. Konflikte und Abhängigkeiten bei der Konfiguration eines Servers werden dabei vom ConfigManager intelligent erkannt und diagnostiziert.
- ConfigAgents realisieren die verschiedenen Kommunikationswege: sie stellen die Schnittstelle zwischen Benutzer und dem ConfigManager dar.
- Instanzen der ConfigClients realisieren die Schnittstelle zwischen Server und ConfigManager. Dabei steuert der zentrale ConfigManager die Akteure fern (remote management).

1.2.1 Arbeitsweise der MaTRICS

Im Folgenden wird die Arbeitsweise der MaTRICS anhand der webbasierten Konfiguration eines Apache Servers veranschaulicht:

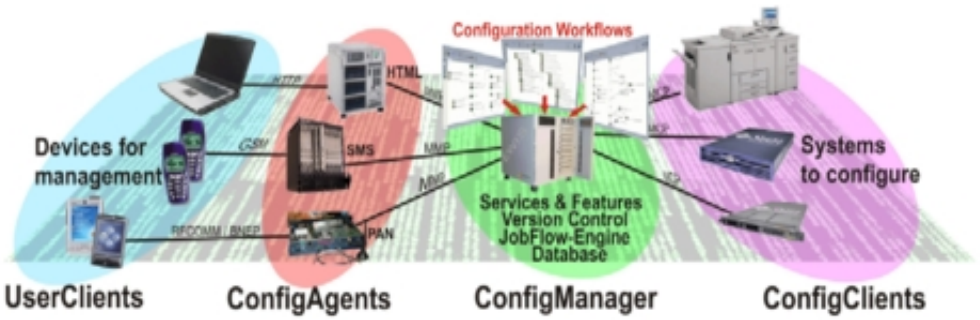

Abbildung 1.1: Schematischer Aufbau der MaTRICS

1. Benutzer authentifiziert sich mittels Web Browser beim WebAgent, welcher zur Klasse der ConfigAgents gehört und den Zugang für das Medium HTTP realisiert.
2. Der WebAgent kodiert die Authentifizierungsdaten in ein einheitliches Format und leitet sie via MMP zur weiteren Verarbeitung zum ConfigManager.
3. Die Schaltzentrale der MaTRICS, der ConfigManager, stellt mittels Nutzer/Rollen Management die jeweiligen Features für den Benutzer bereit und kapselt diese via MMP.
4. Die Generierung einer HTML Seite zur Auswahl eines Features erfolgt mit Hilfe der Daten vom ConfigManager beim WebAgent . Der Benutzer wählt ein Feature, beispielsweise C̈onfigure Apache.
5. Der Request wird vom WebAgent angenommen und via MMP an den ConfigManager geschickt.
6. Der Configuration Workflow C̈onfigure Apachewïrd vom ConfigManager gestartet. Im ersten Schritt werden aus der Datenbank die ConfigClients gelesen, auf denen ein Apache installiert ist.
7. Übertragung der ConfigClients via MMP zum WebAgent und Generierung einer HTML Seite.
8. Der Benutzer selektiert die zu konfigurierende Maschine.
9. Der Request wird vom WebAgent via MMP zum ConfigManager gereicht. Damit beginnt der zweite Schritt des Configuration Workflow. Mit Hilfe der Auswahl des ConfigClients wird die aktuelle Konfigurationsdatei ḧttpd.conf̈für diese Maschine vom CVS-Server ausgecheckt.
10. Im dritten Schritt wird die ḧttpd.conf̈validiert und aus der aktuellen Konfiguration werden vom WebAgent die entsprechenden HTML-Seiten für die webbasierte Konfiguration erstellt.

11. Der Benutzer führt die Konfiguration im Browser durch. Dabei wird jeder Request vom WebAgent via MMP zum ConfigManager geschickt und vom Configuration Feature verarbeitet und validiert. Sobald der Benutzer die Konfiguration abgeschlossen hat, wird aus dieser im ConfigManager eine entsprechende ẗhttpd.conf̈erstellt und im CVS-Server eingecheckt.

12. Die neue Konfiguration wird in die JobFlow-Engine des ConfigManagers eingereiht.

13. Die JobFlow-Engine vearbeitet den Apache-Konfigurations-Auftrag und überträgt die ẗhttpd.conf̈via MCP zum entsprechenden ConfigClient. Wegen der sich ständig ändernden Anforderungen an die Konfigurationsumgebung, kommt es zu folgenden Anforderungen an ein adäquates Design:

    - Service Oriented Architecture (SOA)
    - Modularität
    - objektorientiertes Softwaredesign
    - Rollen- und Feature-basierte Personalisierung
    - flexible Anpassung der Ablaufsteuerung
    - erweiterbare offene Protokolle

    Das in Kooperation mit dem Lehrstuhl V entstandene Agent Building Center (jABC) ist für die Umsetzung derartiger Anforderungen konzipiert: eine graphische Entwicklungsumgebung unterstützt die Entwickler durchgängig in ihrem prozessorientierten, komponentenbasierten Softwaredesign.

1.3 Ziele der PG

Mittels Service Orchestrierung möchten wir grundlegende eigenständige Systeme, wie beispielsweise Network/VoIP Switches mit anderen Web-Services koppeln, um daraus Geschäftsprozesse höherer Ordnung zu modellieren. Dabei betrachten wir solche Systeme als BlackBox, die ausschließlich über ihre WSDL-Spezifikation angesprochen werden können.

MaTRICS stellt dabei ein Meta-Konfigurationstool dar, das komplexe Konfigurationen an die orchestrierten Teilsysteme weiterdelegiert. Die Funktionalität der MaTRICS wird durch die Menge der Dienste des ConfigManagers festgelegt. Dazu baut der ConfigManager auf Technologien von Java Standard, wie CVS, JDBC (PostgreSQL) und Java Enterprise Services für JMX und JMS auf.

Im Rahmen der Projektgruppe möchten wir einen Dienst entwickeln, der mittels Orchestrierung eine VoIPPBX (Siemens HiPath, Asterisk) ansteuert und beispielsweise die Funktionalitäten einer Groupware eines Fremdanbieters ausnutzt um eine Telefonkonferenz mit den beteiligten Teilnehmern zu schalten. Dazu soll ein allgemeines Framework entstehen, das beliebige Web-Services miteinander kombiniert:

Orchestrierung von Web-Services (Framework): Die Orchestrierung einzelner Web-Services lässt sich mit Hilfe von gerichteten Graphen modellieren. Dabei realisieren die Knoten den Zugriff auf die jeweiligen Adapter der Web-Services und die Kanten repräsentieren die Ablaufsteuerung zwischen den einzelnen Diensten. Die Kommunikation zwischen MaTRICS und dem Framework soll auf Basis von Inter Process Communication (IPC) erfolgen. Dieses Framework ist mittels des jABC umzusetzen, das bereits alle benötigten Komponenten zur Entwicklung von serviceorientierten Diensten bereitstellt.

Adapterschnittstellen: Hier soll eine Softwarekomponente entstehen, die mittels WSDL-Spezifikation den Zugriff auf externe Dienste über MaTRICS ermöglicht. Dabei sollen die Adapter auf Basis des serviceorientierten Designs entwickelt werden. Somit lassen sich die Workflows verschiedener Dienste zu einer Einheit miteinander verschmelzen, die sich in der MaTRICS als geschlossenes System repräsentieren.

Kapitel 2

Seminarphase

Die Seminarphase der Projektgruppe 519 fand in der zeit zwischen dem 01.10.07 und 02.10.07 im Haus Bommerholz in Witten-Bommerholz statt. Ziel war es die Teilnehmer mittels Vorträgen auf die kommenden Monate vorzubereiten. Diese Vortäge wurden jeweils einzeln von einem Studenten vorbereitet und innerhalb einer halben Stunde mit Hilfe von Präsentationsfolien vorgetragen. Dazu wurden Kopien der zuvor angefertigten Ausarbeitung ausgehändigt. Im Anschluss eines jeden Themas wurde die Zeit wahrgenommen über das eben gehöhrte zu diskutieren.
Ein weiterer nicht zu vernachlässigender Aspekt der Seminarfahrt war das Kennenlernen der einzelnen Gruppenmitglieder, das dazu diente die Zusammenarbeit zu verbessern.
Der folgende Abschnitt enthält eine Zusammenfassung der einzelnen Seminarthemen in der Reihenfolge wie sie abgearbeitet wurden.

2.1 Thema 1: Objektorientiertes Design auf Basis der UML

Im Bereich der Softwareentwicklung ist die Unified Modeling Language (UML) eine standardisierte Spezifikationssprache für Objektmodellierung. Diese ist eine allgemein anwendbare Modellierungssprache, die benutzt wird, um ein abstraktes Modell eines Systems zu erzeugen, das so genannte UML Modell. Zur Erzeugung dieses Modells definiert UML verschiedene Diagrammtypen, die sich in drei Gruppen bzgl. der Sichtweisen auf das System verteilen.

- Funktionale Anforderungsdiagramme
- Statische strukturelle Diagramme
- Dynamische Verhaltensdiagramme

Während aber diese Diagramme mehr einer graphischen Darstellung des Modells entsprechen, beinhaltet das komplette UML Modell zusätzlich noch semantische Ergänzungen, z.B. schriftliche Beschreibungen der Use Cases usw.

2.2 Thema 2: Konzepte von SOA und MDA

Dieses Seminarthema behandelt die Gebiete Service-oriented architecture (SOA) sowie Model-driven architecture (MDA).

Das Stichwort SOA gehört zu einem der am meist „gehypten“Begriffe der Informatik in den letzten Jahren, nicht zuletzt weil es mit einer Reihe von großen Versprechen aufwartet: Kosten- und Zeitvorteile, hohe Flexibilität und Anpassbarkeit, kurze Reaktionszeiten auf sich ändernde Geschäftsbedingungen, gesteigerte Produktivität und vieles mehr. Der erste Teil der Ausarbeitung zu diesem Seminarthema versucht zu beleuchten, wie diese Versprechen von einer nach SOA strukturierten IT eingelöst werden sollen.

Service-oriented architecture ist ein Architekturmuster, das den Aufbau eines Systems aus einzelnen, abgegrenzten Komponenten beschreibt. Diese Komponenten haben bestimmte Funktionalitäten, die sie in Form von Services nach außen hin bereitstellen. Ein Service ist hierbei eine fest definierte, tendenziell grobgranulare Leistung, deren technischer Hintergrund der Außenwelt nach dem Black Box Prinzip verborgen bleibt. Ein sich nach SOA richtendes System bietet für die genannten Komponenten einen geeigneten Rahmen, der es ihnen ermöglicht zu kommunizieren und die angebotenen Services zu nutzen.

Im Rahmen der Ausarbeitung werden die wichtigsten Komponenten einer SOA Architektur näher vorgestellt, wie z.B. der Begriff des Service, der Enterprise Service Bus, die Orchestrierung von einzelnen Services und die SOA-Governance. Des weiteren wird auf die wichtigsten Hürden eingegangen, die ein Unternehmen auf dem Weg zu einer SOA nehmen muss, sowie auf SOA-Produkte der bekanntesten Hersteller.

Der zweite Teil des Seminarthemas widmet sich der Model-driven architecture, die einen Ansatz zur generativen Herstellung von Hard- und Software aus Modellen beschreibt. Eine MDA Spezifikation besteht hierbei aus einer Reihe von Modellen: dem Platform-independent Model (PIM), mindestens einem plattformspezifischen Modell (PSM) mit zugehöriger Schnittstellenbeschreibung und mindestens einem konkreten Code Modell, das die tatsächliche Anwendung auf einer Plattform darstellt. Ziel der MDA ist es, die Tranformationen von den abstrakten Modellen, über die konkreten hinweg, bis hin zur fertigen Anwendung so weit wie möglich zu automatisieren.

Zum Abschluss der Ausarbeitung werden die möglichen Vorteile von einem parallelen Einsatz von SOA und MDA betrachtet.

2.3 Thema 3: BPEL - eine Sprache zur Modellierung von Geschäftsprozessen

Diese Seminararbeit behandelt die Business Process Execution Language (BPEL), einer auf XML basierenden Sprache. Diese erweiterbare und ablaufbasierende Sprache vereinigt Services in dem sie die Abfolge der Service-Interaktionen koordiniert.

Diese Seminararbeit behandelt die Business Process Execution Language (BPEL), einer auf XML basierenden Sprache. Diese erweiterbare und ablaufbasierende Sprache vereinigt Services in dem sie die Abfolge der Service-Interaktionen koordiniert. Innerhalb des Beitrags wird zu Beginn geklärt worum es sich bei BPEL handelt und auch ein kurzen Einblick in die Geschichte von BPEL wird gegeben. Anschließend folgt ein kurzer Exkurs zum Thema Service Orchestrierung, um sich dann ausgiebig der Aufbau der Sprache zu widmen, sowie der Modellierung von Prozessen. Ein Beispiel zur BPEL Modellierung mit dem Eclipse BPEL Plugin zeigt den Einsatz und den Zusammenhang der einzelnen Sprachkonstrukten von BPEL. Abschließend werden die Schwächen, sowie die Möglichkeiten von BPEL genauer betrachtet.

Innerhalb des Beitrags wird zu Beginn geklärt worum es sich bei BPEL handelt und auch ein kurzen Einblick in die Geschichte von BPEL wird gegeben. Anschließend folgt ein kurzer Exkurs zum Thema Service Orchestrierung, um sich dann ausgiebig der Aufbau der Sprache zu widmen, sowie der Modellierung von Prozessen. Ein Beispiel zur BPEL Modellierung mit dem Eclipse BPEL Plugin zeigt den Einsatz und den Zusammenhang der einzelnen Sprachkonstrukten von BPEL. Abschließend werde die Schwächen, sowie die Möglichkeiten von BPEL genauer betrachtet.

2.4 Thema 4: WSDL und UDDI

Die zentralen Themen der vorliegenden Arbeit sind WSDL und UDDI.

WSDL steht für Web Service Description Language und ist die durch das World Wide Web Consortium (W3C) ratifizierte Beschreibungssprache für Web Services. Web Services sind lose gekoppelte, wieder verwendbare Software Komponenten, die Funktionalitäten zu logischen Einheiten kapseln. Kommuniziert wird dabei über standardisierte Internet-Protokolle.

Die WSDL Architektur setzt sich nach Weerawarana et al.[7] aus ein paar zentralen Elementen zusammen.

- Erweiterbarkeit
- Unterstützung verschiedener Schemensprachen
- Einheitlicher Datentransfer und Remote Procedure Calls

- Trennung von ’Was’,’Wie’ und ’Wo’
- Unterstützung verschiedener Transportprotokolle
- Keine Ausführungsreihenfolge
- Keine Semantik

Danach befasst sich diese Arbeit mit dem Thema Universal Description, Discovery and Integration (UDDI). Auch hier wird mit einer Kurzvorstellung der Einstieg in das Thema vorbereitet. Der Schwerpunkt dieses Kapitels liegt neben der Vorstellung der UDDI Datenmodelle auch auf der Vorstellung der umfangreichen UDDI Version 3 API, die für jeden Entwickler, der mit Web Services im allgemeinen und UDDI im speziellen zu tun hat, Basisliteratur darstellt. Ein Unterkapitel beschäftigt sich mit dem Mapping, der Transformation von Daten von einem Format ins andere Format, von WSDL nach UDDI und schlägt so die Brücke zwischen beiden Themen.

UDDI liegt zum Zeitpunkt dieser Ausarbeitung in Version 3 (3.0.2) vor. Diese Version wurde im Juli 2002 auf den Weg gebracht und ist seit Februar 2005 ratifiziert. Besonderes Augenmerk wurde bei dieser Version auf eine erweitertes Zusammenspiel zwischen privaten und öffentlichen Implementierungen gelegt. Technische Neuerungen in dieser Version sind die folgenden Features.

- Unterstützung digitaler Signaturen
- vom Verfasser zugewiesene Schlüssel
- definiertes Verhalten (Policy)
- Subscription API
- Verbesserung im Hinblick auf Kategorisierungen
- Änderungen in der Authentifizierung

2.5 Thema 5: Orchestrierung von Webservices mit Hilfe von SOAP

Dieses Seminarthema betrachtet SOAP, ein auf XML basierendes Protokol, hier in der Version 1.2, das bei der Übertragung von strukturierten Daten zwischen Anwendungen, und bei der Ausführung von Remote Procedure Calls in einer dezentralisierten Umgebung, wie dem Internet, Verwendung findet.
Zunächst wird auf den Vorgänger XML-RPC eingegangen, da dieser durch seine einfachere Struktur, besser dafür geeignet ist, in die Welt der XML basierten Nachrichten einzuführen.

Später wird der Aufbau und die Struktur der zu versendenden Nachrichten, die für diese Aufgaben nötig sind, beschrieben. Hierzu gehört die Aufteilung in einzelne Unterstrukturen, wie Header und Body, sowie der Aufbau von Antwortnachrichten insbesondere der Fehlermeldungen. In dem darauf folgenden Kapitel wird die Bindung der Nachrichten an verschiedene Datenübertragungsprotokolle auf der Anwendungsschicht, wie zum Beispiel HTTP und SMTP, sowie auch auf der Transportschicht, üblicherweise mittels TCP, des TCP/IP-Protokollstapels,erläutert. Anhand von Axis2 wird ein Beispiel für eine Implementierung des SOAP Protokolls geliefert und anhand eines Kurzen Beispiels in die Verwendung eingeführt. Abschließend wird dieses Beispiel erweitert um einen einen Eindruck zu geben, wie Serviceorchestrierung mittels SOAP erreicht werden kann.

2.6 Thema 6: J2EE und der Applikationsserver JBOSS

Mit J2EE steht eine gute Plattform zur Verfügung um sehr große und professionelle Unternehmens-Anwendungen zu erstellen und mit dem JBoss Applikation Server steht ein Container zur Verfügung um diese Anwendungen lauffähig zu machen.Es gibt auch andere Applikation Server,doch JBoss AS ist frei erhältlich wird von einem Vollzeit arbeitenden Team entwickelt,hält sich an die J2EE-Spezifikationen und hat eine gute Clustering-Funktion.

2.7 Thema 7: Servlets und Portlets

Diese Ausarbeitung beschäftigt sich mit dem Thema Servlets und Portlets. Dabei wird die Historie zur Entwicklung von dynamischen Web-Applicationen betrachtet und ein Überblick über die Grundlagen von Servlets gegeben. Zudem bietet diese Ausarbeitung eine kurze Einführung in die Portlet Technologie.

Mit der zunehmenden Beliebtheit des Internet wuchsen auch die Ansprüche daran. Ansprüche sind z.B. das Daten eines Benutzers für die Web-Seite benötigt werden, wie etwa bei Suchmaschinen. Mit Hilfe von Servlets lassen sich Web-Applikationen mit dynamischen Inhalten erzeugen. Mit Servlets bietet Java eine umfangreiche API für die Erstellung von Web-Application mit dynamischen Inhalten. Es werden Fragen behandelt, wie etwa: Was ist ein Servlet oder auch ein Servlet-Container? Wie sieht der Lebenszyklus von Servlets aus? Wie werden Informationen abgerufen und versendet?, Wie funktioniert die Sitzungsverfolgung und welche Sicherheitsaspekte sollten beachtet werden? Oder mit der Zusammenarbeiten von Servlets? Ausserdem werden die Neuerungen der Servlet Spezifikation V 2.5 vorgesetllt. Abschliessend wird an einem Beispiel ein Servlet genauer untersucht und eine Erläuterung zu Apache Tomcat gegeben.

Durch die immer grösser werdende Informationsflut im Internet bestand die Notwendigkeit, die Informationen für verschiedene Anwender übersichtlich, personenbezogen und organisiert zur Verfügung zu stellen. Ein Portal bietet solch eine Möglichkeit. Es bildet die Präsentationsschicht eines web-basierten Informationssystem und integriert die Inhalte verschiedener Applikationen und Applikations-Komponenten. Durch die wachsende Popularität der Portale, wurde eine Spezifikation für Portlets wünschenswert. Als Portlets werden die einzelnen Komponenten eines Portals bezeichnet. Von der Java Community Process wurde die Java Specification Request 168, als Standard für Portlets auf der Grundlage von Java-Technologien erstellt. Sie ist eine Erweiterung der Java Servlet Technologie. In diesem Teil der Ausarbeitung beschäftigen wir uns mit dem Grundlagen von Portlets, ihrem Aufbau und ihrem Konzept. Weiter soll die Spezifikation JSR-168 und JSR-268 und Orchestrierung mittels Portlets vorgestellt werden, was mit einem Beispiel und einer kurzen Vorstellung des Portletservers Pluto veranschaulicht wird.

2.8 Thema 8: Entwicklung von Webapplikationen mit Ajax

AJAX steht für Asynchronous JavaScript and XML und stellt die asynchrone Datenübertragung über das HTTP-Protokoll zwischen dem Browser und einem Webserver dar. Dadurch wird ermöglicht, dass nur HTML-Seitenteile aktualisiert werden ohne dass die ganze Webseite neu geladen werden muss.

Ajax basiert auf dem JavaScript-Objekt XMLHttpRequest. Die Daten, die mit dem abgerufen werden, können als normaler Text (bei HTML) oder als Baumstruktur (bei XML-Dateien) dargestellt werden. Dieses Objekt wurde am Anfang als ActiveX-Komponente im Internet Explorer 5 eingesetzt und später auch in allen anderen bekannten Browsern.

Die Enwicklung von Webapplikationen mit Ajax erweist sowie Vorteile, als auch einige Nachteile und Programmierschwierigkeiten.

Die meist geschätzte Vorteile der Ajax-basierten Webanwendungen bestehen darin, dass der Datenverkehr zwischen dem Server und dem Browser geringer ist und die Performanz der Applikation dadurch gesteigert wird. Ein weiterer Vorteil ist es, dass die Datenabfrage automatisiert werden kann und damit im Hintergrund ablaufen kann, ohne einen Eingriff seitens des Nutzers. Mittels Ajax können weiterhin sowohl HTML-, als auch XML-Daten abgefragt werden.

Die Vorteilhaftigkeit davon, dass eine eigenständige Installation des Programms nicht notwendig ist, da das XMLHttpRequest-Objekt in der JavaScript-Implemetierung jedes Browssers enthalten ist, ist umstritten. An der Punkt erweist sich als problematisch, dass im Browser JavaScript aktiv sein muss, welches oft als Sicherheitsrisiko eingestuft wird. Einige weitere Punkte, die problembehaftet sind:

- Umfangreiches (Browser-)Testen notwendig
- Zeichenkodierung (UTF-8)
- Rücksprünge und Lesezeichen setzen
- Polling

Ajax stellt eine Technologie dar, die zusammen mit der Entwicklung der Internet-Technik zu dem Sprung zum Web 2.0 verholfen hat. Es existieren bereits viele Tools und Frameworks, die die Entwicklung von Ajax-Webapplikationen unterstützen.

2.9 Thema 9: Anwendungsentwicklung mit Eclipse

Das Eclipse Projekt ist ein Open Source Projekt, das sich der Entwicklung einer robusten und umfassenden Entwicklungsplattform verschrieben hat, die für den kommerziellen Einsatz tauglich ist. Das Grundgerüst Projektes bildet die Eclipse Plattform, die eine erweiterbare auf Java basierende IDE darstellt. Sie zeichnet sich neben ihren eigenen Features durch einen starken Integrationscharakter aus, der das Einbinden externer Tools ermöglicht.

Was Eclipse interessant macht:

- Eclipse als IDE (Integrated Developement Environment):
  - Eclipse + JDT = Java IDE
  - Eclipse + CDT = C/C++ IDE
- Eclipse als Tool-Framework:
  - Plugins passen Eclipse an zahlreiche Aufgaben an
  - Business Intelligence and Reporting Tools (BIRT)
  - Web Tools Project (WTP)
  - Test and Performance Tooling Project (TPTP)
  - SOA Tools Platform
- Eclipse als Application-Framework:
  - Multi-Plattformfähig (Linux, UNIX, Mac OSX, Windows)
- Open Source Community / Eclipse Foundation
- Mitglieder und Entwickler aus vielen Bereichen: BEA, Borland, JBoss, IBM,SAP, RedHat, Novell, Monta Vista, Wind River, Mentor, ENEA, QNX

- Zahlreiche Plugins
- Architektur, Perspectives, Leistungsstarker Editor, Fehleranzeige, Try-catch Blöcke, Code Templates, Navigation, Suchfunktionalit, CVS Unterstützung, Debugger, JUnit Unterstützung, ...

2.10 Thema 10: Die Entwicklungsumgebung jABC mit dem jEWIS Plugin

Die Entwicklungsumgebung „Java Application Building Center“, kurz jABC, ist eine Entwicklung des Lehrstuhls 5 und bietet einen Editor zur Entwicklung von neuer Software durch die Entwicklungsmethode „Lightweight Process Coordination“. Diese Entwicklungsmethode setzt auf die bestehenden Methoden auf und besteht in der Grundidee daraus, bestehende Prozesse miteinader zu kombinieren und so neue Prozesse, Applicationen zu erzeugen.
Das jABC als Entwicklungsumgebung bietet eine graphische Möglichkeit, diese Entwicklungsmethode in Form von einem Flussgraphen zu praktizieren. Prozesse werden dabei als SIBs (Service Independent Building Blocks) bezeichnet und als Knoten, dargestellt. Diese SIBs können miteinander in Beziehung gesetzt werden. Dargestellt wird dies im jABC repräsentiert durch Kanten in dem erzeugten Graphen. Durch die Kanten die Beziehungen darstellen und eindeutig mit Hilfe eines für die Kante zu vergebenen Namens identifiert werden können, entsteht die Möglichkeit, einen bestimmten Weg durch den Graphen zu wählen.
Im jABC werden alle nötigen Werkzeuge bereitgestellt, um einen solchen Graphen zu erzeugen. Im wesentlichen sind diese Werkzeuge der Browserbereich, der Inspectorbereich und der Canvas. Im Browserbereich werden die Projekte sowie die im aktuellen Projekt vorhandenen Service Logic Graphen (SLGs) dargstellt. Ausserdem gibt es einen Bereich, in dem die dem jABC bekannten SIBs aufgelistet werden, wahlweise nach Ordung im Filesystem oder nach einer durch den Benutzer editierbaren Taxonomie. Der Inspectorbereich stellt eine GUI dar, mit dem SIBs und SLGs parametriesiert werden können. Gezeichnet werden die SLGs im grösten Bereich der jABC GUI, der Canvas.
Die Architektur des jABC sieht eine einfache Erweiterbarkeit durch ein Pluginsystem vor. Durch dieses System kann der sehr allgemein gehaltene Ansatz des jABC (die Art der Anwendung und der Einsatzbereich sind völlig unbekannt und für das jABC auch nicht relevant) auf die persönlichen Bedürfnisse angepasst werden. Eine Reihe von Plugins existiert bereits.
Hierzu gehört auch das jEWIS. Dieses Plugin bietet eine Erweiterung des jABCs, um Webservices durch Lightweight Process Coordinationßu entwickeln. Es liefert einige GUI-Erweiterungen des jABCs, die die Steuerung des eingesetzten Applicationservers Tomcat ermöglichen und die Projektverwaltung für jEWIS-Projekte verbessern. Ausserdem werden eine Vielzahl von bereits implementierten SIBS mitgeliefert, sowohl für die Entwicklung von generellen Webservices als auch für MaTRICS-spezifische Webservices.

2.11 Thema 11: Datenbanken und deren Anbindung mittels Hibernate

In dieser Ausarbeitung geht es um Datenbanken und deren Anbindung mittels Hibernate. Zuerst werden grundlegende Datenbankmodelle vorgestellt, wie z.B. relationale und objektorientierte, sowie wichtige Datenbank-Konzepte (Stichwort ACID). Im darauffolgenden Kapitel geht es um die Datenbanksprache SQL, deren Sprachkonstrukte anhand von Beispielen demonstriert werden. Desweiteren gibt es eine kurze Einführung für PostgreSQL-Datenbanken samt nüzlicher Tools. Geschlossen wird der Rahmen im letzten Kapitel mit der Zusammenf̈u)hrung aller bisher vorgestellten Themen und endet mit der einer Beispiel-Implementierung für Hibernate, die SQL und PostgreSQL benutzt.

2.12 Thema 12: Grundlagen von VoIP auf Basis von SIP

Einführend wird in dieser Ausarbeitung die geschichtliche Entwicklung der Telekommunikationsnetze vorgestellt. Es werden die zwei Typen von Telekommunikationsnetzen(Telefonnetzen): analogen(POTS) und digitalen(ISDN/PSTN) und dazugehörigen Funktionsweisen analysiert. Anschließend wird die Digitalisierung der öffentlichen Telefonnetze und die Entstehung der so genannten Intelligente Netze erklärt.
Voice over IP (kurz VoIP) wird als eine Alternative zur herkömmlichen, leitungsdebundenen Sprachkommunikation vorgestellt. Dazu werden die Grundlagen und die Funktionsweise von VoIP vorgestellt und die Vorteile und Nachteile bei der Nutzung von VoIP diskutiert.
Zum besseren Verständnis wird das 7-schichtige ISO-OSI-Referenzmodell erläutert, wobei es genauer das IPv4 (Internet Protocol Version 4), das IPv6 (Internet Protocol Version 6)und die Transportschicshtprotokolle - TCP und UDP betrachtet werden. Da bei der Realisierung von einer VoIP Sprachkommunikation mehrere Protokolle angewendet werden müssen, werden diese in drei Klassen gruppiert und detailliert vorgestellt.

1. Sprachübermittlungsprotokolle: RTP, RTCP
2. Signalisierungsprotokolle: H.323, SIP
3. Media Gateway Protokolle: MGCP, Megaco

Als Signalisierungsprotokoll, das den Aufbau einer Session zwischen kommunizierenden Rechnern über ein IP-Netz ermöglicht, um die Echtzeitmedien zu übermitteln wird das Session Initiation Protocol (SIP) vorgestellt. Es wird die SIP-Entwicklung, die SIP-Adressierung und der SIP-Nachrichtenaufbau umfangreich beschrieben, wobei auch die Funktionsweise und die Aufgaben von SIP als Signalisierungsprotokoll bei VoIP Sprachkommunikationen mit berücksichtigt wurden.
Zum Schluss werden einige verbreitete, nicht-zeitkritische SIP-Anwendungen (Instant

Messaging, Presence) und auch zeitkritische SIP-Anwendungen (IP-Telefonie (VoIP), IP-Videotelefonie, IP-Konferenz, Push-to-Talk(PTT), etc.) vorgestellt.

2.13 Thema 13: Asterisk unter Linux

Die Open-Source Software Asterisk hat sich als Allzweckwaffe in der modernen Telefonie-Welt etabliert und ist als Bindeglied zwischen multimedialer Kommunikation und IT-Infrastruktur nicht wegzudenken.

Dieses Seminarthema gab einen Überblick über den Umfang einer Asterisk Standardkonfiguration. Besonders intensiv wurde der Dialplan - die Asterisk eigene Vermittlungsstelle - thematisiert.

```
[internal]
exten=>1,1,Answer()
exten=>1,2,SayAlpha(Hallo)
exten=>1,3,Goto(1)

exten=>2,1,Answer()
exten=>2,2,SetMusicOnHold(native-random)
exten=>2,3,MusicOnHold()
exten=>2,4,Hangup()
```

Anhand von Beispielen wurde die syntakische und logische Struktur des Dialplan erläutert; die Begriffe *Kontext*, *Extension* und *Priorität* erklärt und eine Reihe Applikationen und deren Funktionen vorgestellt.

Zusätzlich wurden mit AMI(Asterisk Management Interface) und AGI(Asterisk Gateway Interface) Ansatzpunke zur Entwicklung von funktionalen Erweiterungen vorgestellt.

Eine beispielhafte Konfiguration eines Asterisk und zweier Softphones (Kphone und xLite) schloss dieses Seminarthema ab.

2.14 Thema 14: Modellierung von Telekomunikationsdienste auf Basis von JAIN, OSA und Parlay

Ein Telekommunikationsdienst ist eine Funktion eines Kommunikationsnetzes, die ein Mensch oder Programm in Anspruch nehmen kann, um einen Nutzen daraus zu ziehen. Dies reicht vom grundlegensten Service der Gesprächsverbindung zwischen zwei Teilnehmern, über Gesprächsdienste wie Halten, Umleiten oder Konferenzschaltung, bis hin

zu indiviualisierten Klingeltönen, SMS, Sprachwahl oder Remote-Steuerung von Telefonanlagen. Ein API (Application Programming Interface) ist eine Schnittstelle zur Anwendungsprogrammierung, definiert also Methoden zur Steuerung eines Systems. In dieser Seminararbeit wurde die Enstehung des Telefon- bzw. Signalnetzes, sowie deren Struktur und Konzepte dargestellt, welche die Programmierung von Diensten ermöglichen. Darauf aufbauend wurden die JAIN, OSA und Parlay APIs, die nach den Gruppierungen benannt sind von denen sie entworfen wurden vorgestellt. Dies sind die am weitesten verbreiteten APIs zur Steuerung von Telekommunikationsservices. Im Einzelnen erklärt wurden:

- die OSA Parlay Architektur
- die OSA Parlay APIs: Call Control, User Interaction, Mobiltiy, Data Session Control, Generic Messaging Service, Charging, Account Management
- die JAIN Architektur
- das JCC Objekt Modell
- das JCAT Objekt Modell
- die JAIN SIP API

Kapitel 3

Aufwärmphase

Während der ersten zwei Wochen der Projektgruppe sollten sich die Teilnehmer mit den verwendeten Werkzeugen wie jABC inklusive des jEWIS Plugins, Eclipse und Apache Tomcat vertraut machen. Hierzu wurden sieben verschiedene Aufgaben gestellt die von jeweils zwei Teilnehmern der Projektgruppe zu bearbeiten waren.
Im Folgenden werden Aufgabenstellungen und Ergebnisse skizziert, um einen Einblick in die Arbeit der Projektgruppe zu ermöglichen.

3.1 Addressmanager

Aufgabe war es, einen Webservice zur Verwaltung von Adressen und Kommunikationsdaten, wie z.B. Telefon, Fax, E-Mail, ICQ oder auch Skype zu entwickeln, um einen möglichst effizienten Kommunikationsfluss der Pojektgruppenmitglieder zu ermöglichen. Dazu benutzt wurde das am Lehrstuhl 5 entwickelte jABC, welches die in Graphen beschrieben Abläufe einfach und komfortabel mit Java Sourcecode verbindet. Nach kurzer Paper-Prototyping-Phase und Planung des Programmablaufes, wurden die erarbeiteten Lösungen in Html-Seiten umgesetzt. Dabei wurden Funktionen für das Neu-Anlegen, Editieren, Löschen und Suchen von Einträgen genauso berücksichtigt, wie eine angenehme Bedienung des Adressbuches selbst.'
Anschließend erfolgte die Implementierung der SIBs in Java. Die Abbildung 3.1 zeigt den Graphen des Adressmanagers und repräsentiert dabei alle möglichen Abläufe. Zentraler Ausgangspunkt ist der Knoten ShowFileBuch, der mit Hilfe der Velocity-Engine erst eine Übersicht aller Einträge zeigt und via Hyperlink auf die zugeörige Detailansicht eines Eintrages verweist. Vor hier aus bietet die Html-Seite noch die
Möglichkeit über Knöpfe neue Datensätze anzulegen, zu editieren oder zu löschen. Bei neu anzulegenden Datensätzen wurde darauf geachtet, dass alle relevanten Daten (z.B. Nachname) mit angegeben werden mussten. Falls nicht, informiert eine extra Hinweisseite über das Fehlen von Daten. Zur persistenten Datenspeicherung wurden *property*-Dateien verwendet, die einen einfachen und schnellen Zugriff auf die Daten ermöglichten.

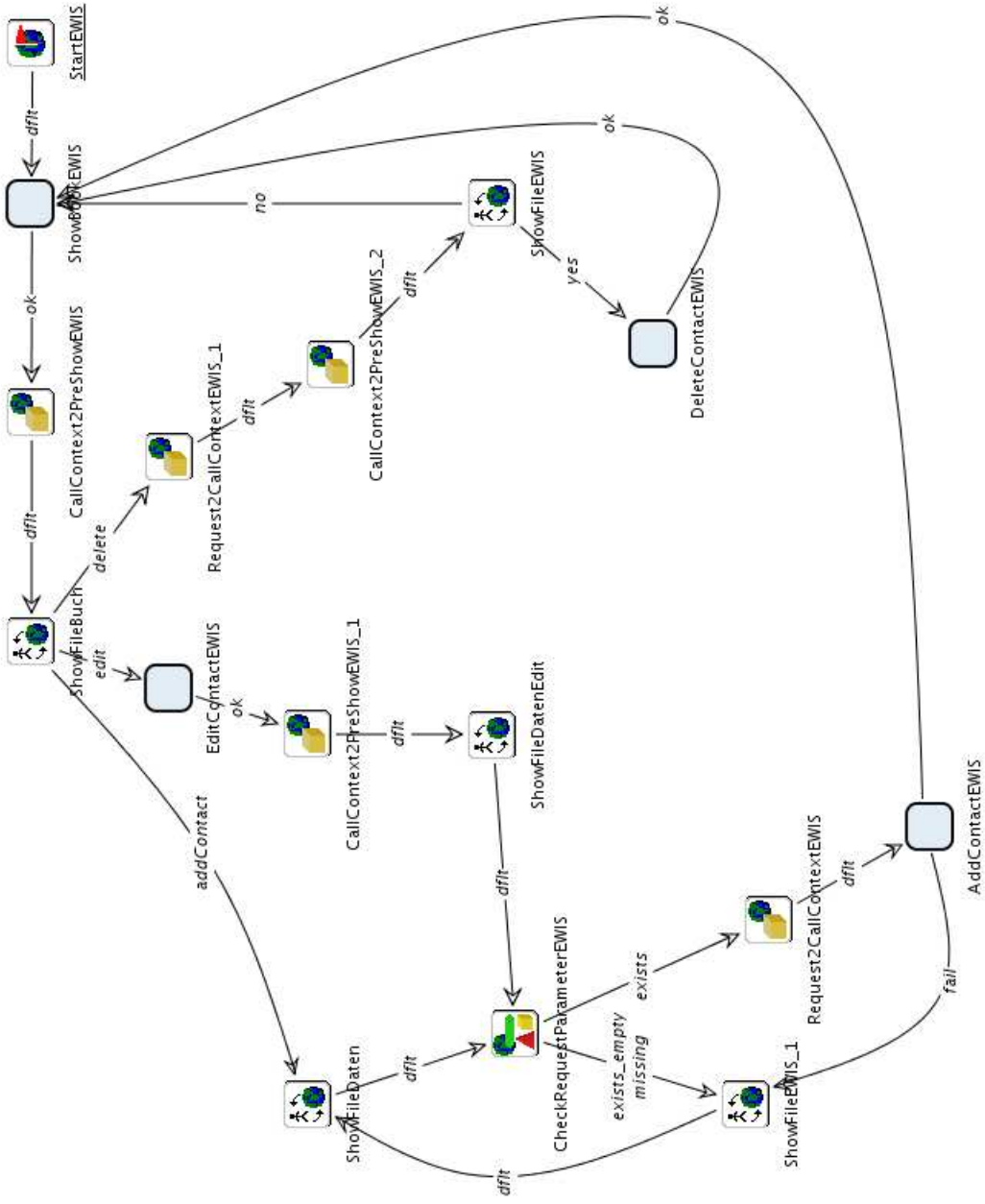

Abbildung 3.1: Service Logic Graph: Addressmanager

3.2 Deadlinemanager

Die Aufgabe war es mit Hilfe des jABC einen DeadlineManager zu erstellen, mit dem Deadlines verwaltet werden können. Dabei sollen neue Deadlines hinzugefügt werden, vorhandene Deadlines bearbeitet und gelöscht werden.

Der DeadlineManager besitzt zudem eine einfache Benutzerverwaltung. Als erstes muss man sich beim DeadlineManager anmelden. Erst nach der erfolgreichen Anmeldung stehen die Funktionalitäten zur Verwaltung der Deadlines zur Verfügung und auch die Funktionalität neue Benutzer anzulegen. Nach dem Login gelangt man auf die Hauptseite des DeadlineManagers. Abbildung 3.2 zeigt einen Screenshot von der Hauptseite des DeadlineManagers

Auf der Hauptseite werden alle Deadlines angezeigt. Sie sind in zwei Gruppen eingeteilt, in abgeschlossene Deadlines und noch offene Deadlines. Die Deadlines werden dem Status entsprechend in einer der beiden Tabellen eingeteilt. Die Reihenfolge der Auflistung innerhalb der Tabellen werden durch das Deadlinedatum der jeweiligen Deadline bestimmt. Zu jeder Deadline kann eine Detailseite aufgerufen werden und die Deadlines können gelöscht werden. Abgeschlossene Deadlines können nicht mehr bearbeitet werden. Es gibt eine Historyfunktion, die für jede Deadline anzeigt von wem und wann sie erstellt und bearbeitet wurde. Die Historydaten sind auf der Detailseite ersichtlich. Natürlich ist es auch möglich neue Deadlines anzulegen. Bei den Deadlines muss der Betreff, die Beschreibung, die Kategorie, das Deadlinedatum, die Priorität und der Status angegeben werden. Die Kategorien von Deadlines werden seperat gepflegt. Auch hier können Kategorien angelegt, bearbeitet oder gelöscht werden.

In Abbildung 3.3 ist der entsprechende Service Logic Graph für den Webservice DeadlineManagers zu sehen.

3.3 Kickerstatistics

Ziel der Aufgabe war die Erstellung einer kleinen Webapplikation zur Verwaltung der Spielergebnisse bei einem Kickerturnier mit Hilfe des jABC-Tools. Zu diesem Zweck wurde eine Startseite erstellt, von der man sich entweder die aktuelle Spieltabelle anzeigen lassen oder neue Spielergebnisse eingeben kann. Bei der Kickerstatistik Übersicht hat man die Möglichkeit mehrere Einträge gleichzeitig aus der Tabelle zu löschen und sich die neue Tabelle anzeigen zu lassen. Von dieser Seite aus kann man entweder neue Spielangaben machen, oder zurück zur Startseite gehen. Die neuen Eingaben werden durch zwei Eingabefelder für die Namen der Spieler und zwei RadioGroups jeweils für das Team und das Spielergebnis realisiert. Nach der Eingabe des Ergebnisses werden die bereits gespeicherten Daten von einer Datei mittels des LeseSIBs geholt und zusammen mit den neuen Daten in einem anderen SIB, MySIB, sortiert. Nach der Sortierung der Daten werden

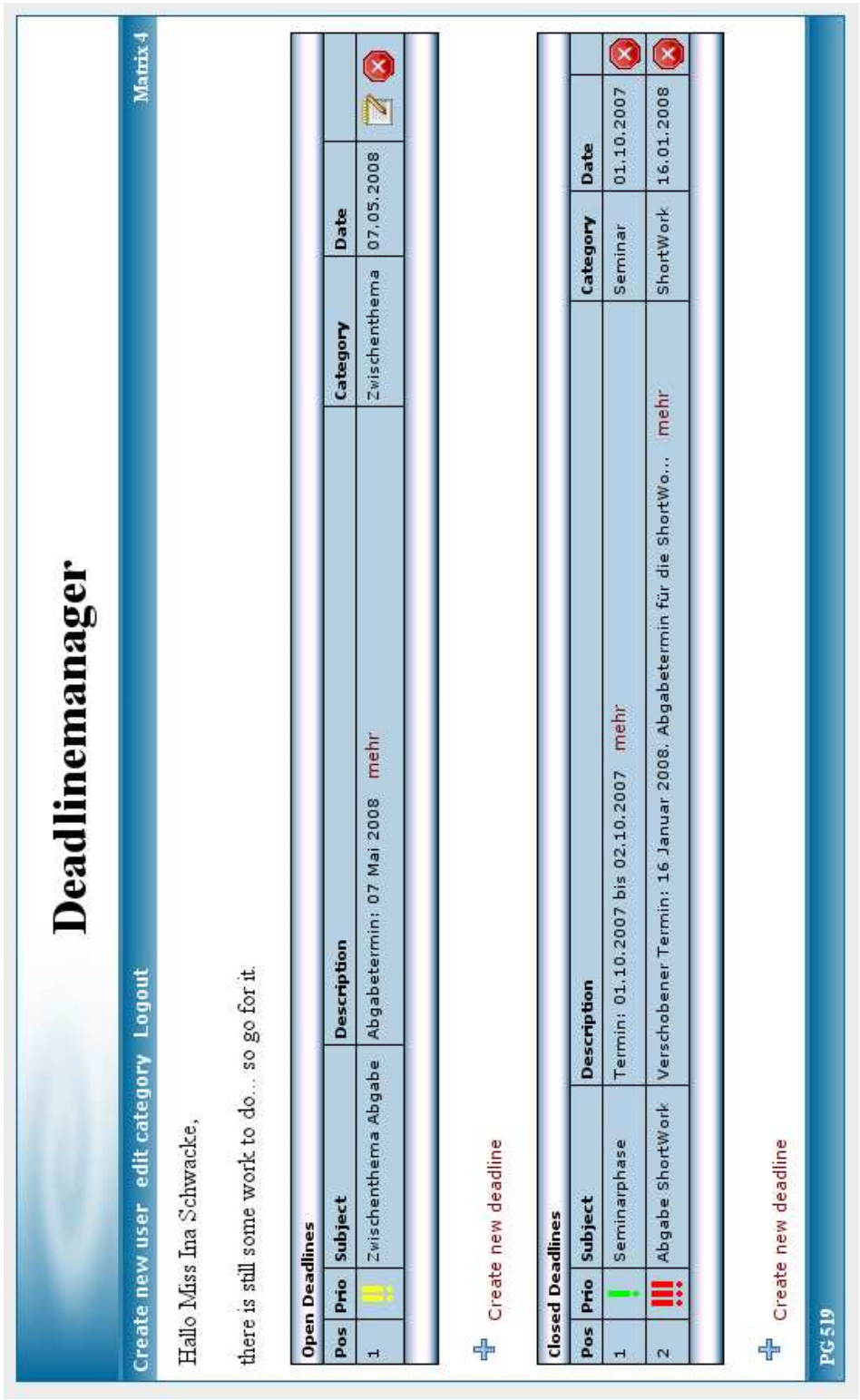

Abbildung 3.2: zentrale Seite DeadlineManager

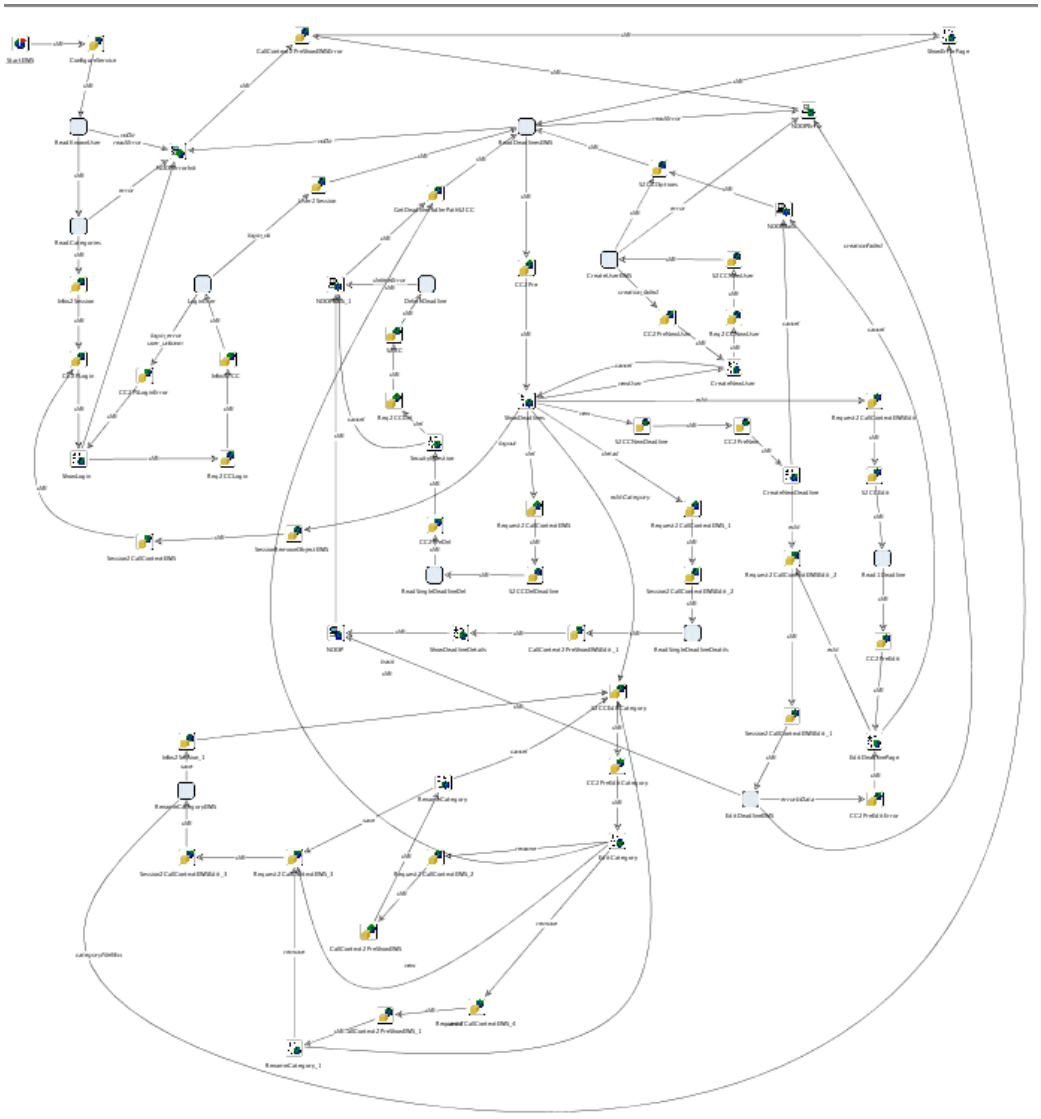

Abbildung 3.3: Service Logic Graph: DeadlineManager

die Ergebnisse mittels des SchreibeSIB in der bereits bestehenden Datei gespeichert und schließlich wird die neue Tabelle angezeigt. Beim Löschen der Einträge aus der Tabelle werden die ausgewählten Datensätze in einem SIB gelöscht und die neu entstandene Statistik wird wieder mittels des SchreibeSIBs in der Datei gespeichert. sieh Abbildung 3.4

3.4 Lotto

Diese Aufwärmaufgabe bestand darin, mit Hilfe der jABC-Umgebung das bekannte 6 aus 49 Lotto als Webservice für die PG-Mitglieder umzusetzen. Die Anwendung besitzt eine grundlegende Nutzerverwaltung, so wird über einen Login-Screen sichergestellt, dass nur eingetragene Spieler den Service nutzen können. Zudem wird zu jedem Spieler ein Tipp verwaltet, den er nach dem erfolgreichen Login anpassen kann um anschließend eine Ziehung zu starten. Während einer Ziehung werden 6 Zufallszahlen zwischen 1 und 49 generiert und sowohl mit dem Tipp des aktuellen Spieler als auch den abgespeicherten Tipps der anderen Spieler verglichen. Bei einem Treffer wird die Liste der entsprechenden Gewinner angezeigt Zusätzlich steht auch eine Statistik zur Verfügung, die die absoluten Häufigkeiten der bisher gezogenen Zahlen für die Spieler bereithält. Abbildung 3.6

3.5 Tictactoe

Die Aufwärmaufgabe dieser Gruppe bestand darin, das Spiel Tictactoe[1] als Webservice zu implementieren. Das Ergebnis lässt Einspieler-Spiele gegen den Computer, sowie Zweispieler-Spiele zwischen zwei menschlichen Spielern zu. Weiterhin wird zur späteren Einsicht gespeichert, wie oft die einzelnen Spieler Spiele gewonnen oder verloren haben oder an welcher Anzahl unentschiedenen Spielen der Spieler beteiligt war. Zu Vergleichszwecken enthält die Tabelle die Daten des Computerspielers.

3.6 Treasurer

Ziel des Warm-Up-Projektes die Erstellung eines Schatzmeisterprogramms zur Verwaltung der Anwesenheits- und Abwesenheitsliste der Projektgruppeteilnehmer. Zentral sollte dabei zum einen ein Algorithmus entwickelt werden, der die ausgehandelten Zahlungsmodalitäten widerspiegeln sollte und zum anderen eine rudimentäre Benutzerverwaltung erstellt werden. Jeder Nutzer bekam ein Konto, dass seinen aktuellen Kontostand abgefragen lässt sowie wann welche Buchungen darauf getätigt wurden, sprich, wann wurden Verspätungen eingetragen und wann wurden diese Schulden beglichen. Zur einfacheren

[1] http://en.wikipedia.org/wiki/Tictactoe

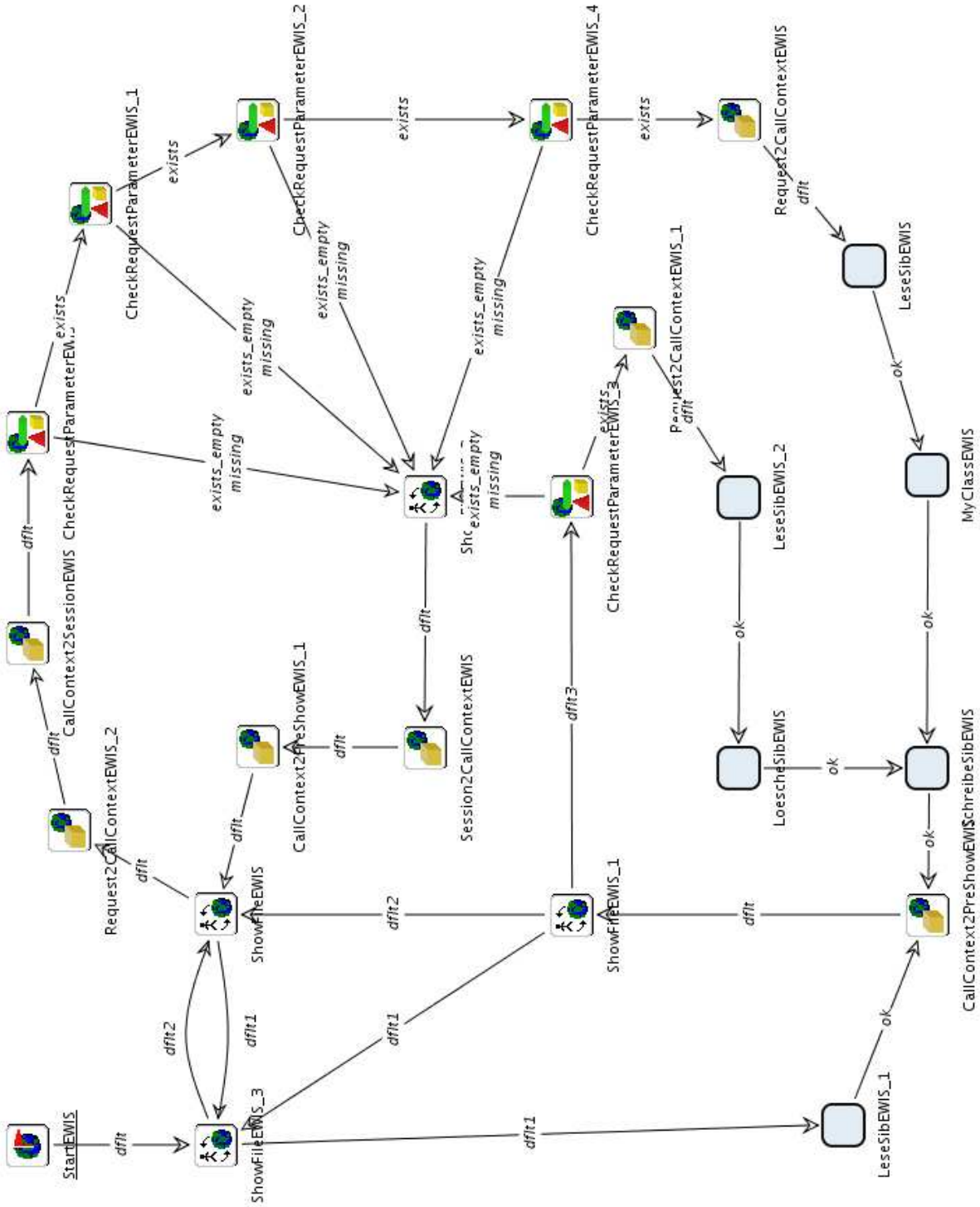

Abbildung 3.4: Service Logic Graph:Kickerstatistics

Abbildung 3.5: Screenshot der Tippverwaltung

Handhabung wurde sich für eine XML basierte Dateistruktur entschieden, da hier über das Baumkonzept beliebige Buchungen angehängt werden können.

3.7 Hangman

Das Ziel des Warm-Up-Projekts war, das bekannte Spiel *Hangman* als Web Anwendung zu realisieren. Um das zu erreichen, wurde die Entwicklungsumgebung jABC mit jEWIS benutzt.

Beim Start des Spiels wird ein Wort aus einer Datei nach einem Zufallsprinzip geladen und jede seiner Buchstaben wird durch einem Strich dem Anwender repräsentiert. Diese Aufgabe wird durch den für diese Anwendung geschriebenen Initialize-SIB erfüllt.

Der Anwender wiederrum kann versuchen das versteckte Wort zu erraten, indem er beliebige Buchstaben aus der ihm vorgelegten Alphabetliste aussucht. Das Gebiet, aus dem das Wort stammt, wird als Tipp angezeigt. Der Übersichtlichkeit halber werden sowie die bis zum gegebenen Zeitpunkt gewählten, als auch die zuletzt geratene Buchstabe angezeigt.

Wenn der Spieler eine Buchstabe errät, dann wird diese an der richtigen Stelle im Wort eingeblendet. Diese Aufgabe übernimmt der WordCheck-SIB. Abbildung 3.10

Das Spiel ist erfolgreich beendet, wenn das ganze Wort richtig erraten wurde, nämlich wenn alle seine Buchstaben mindestens ein Mal aus der Liste gewählt worden sind und dabei nicht mehr als sechs Fehlgriffe gemacht wurden.

Das Spiel ist erfolglos beendet, wenn der Anwender mehr als sechs Fehler beim Buchstabenraten gemacht hat.

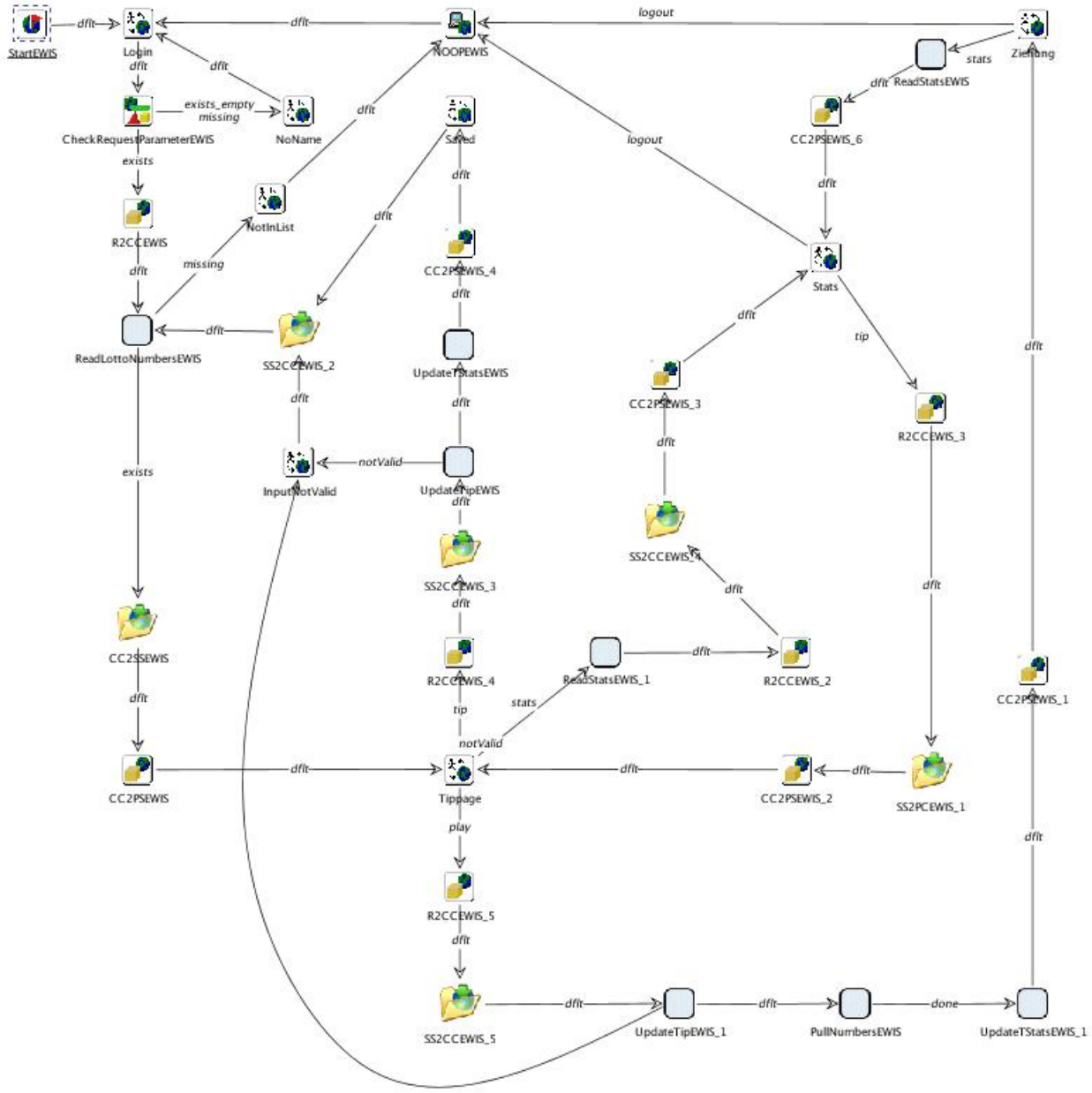

Abbildung 3.6: jABC Modell der Lotto Anwendung

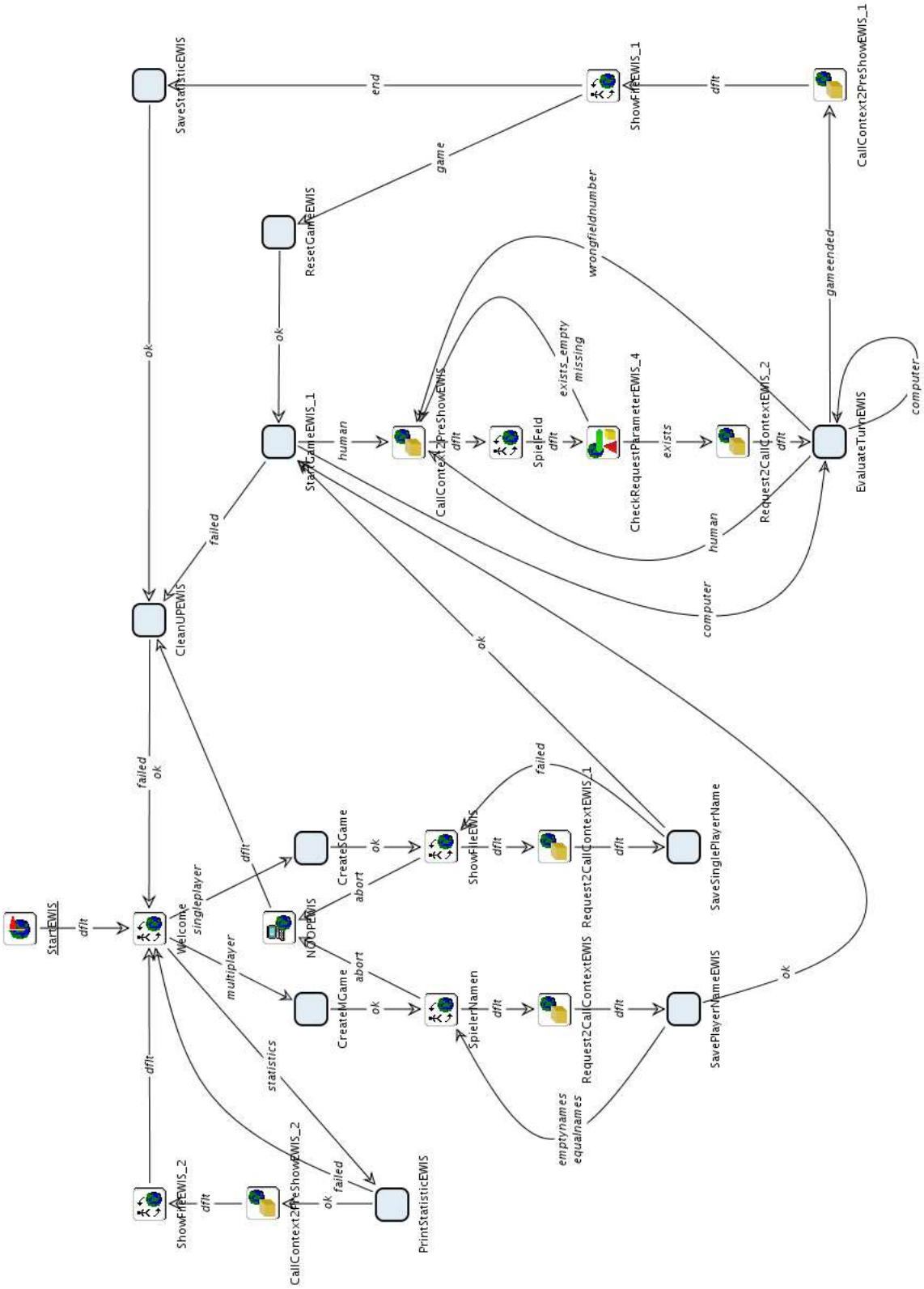

Abbildung 3.7: Modell des TicTacToe-Spiels

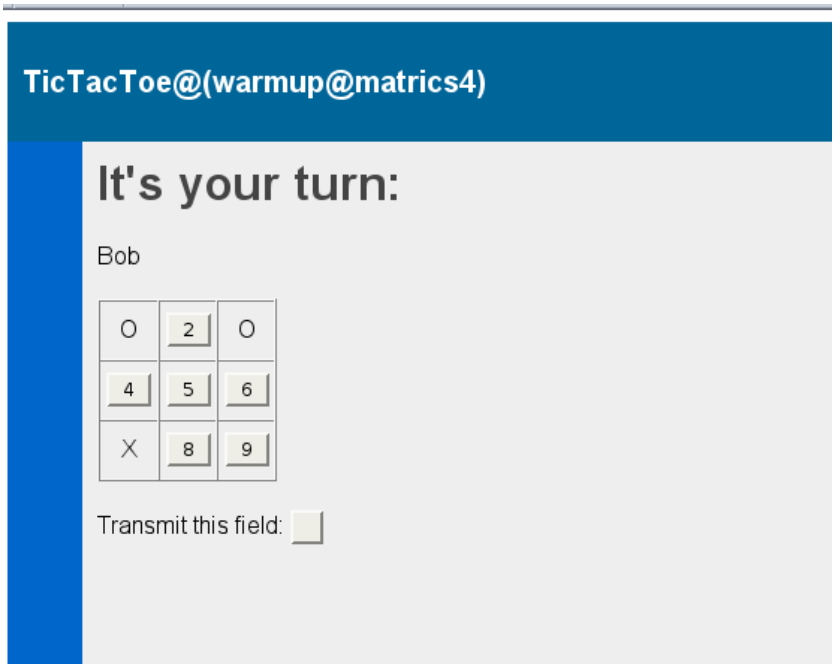

Abbildung 3.8: Screenshot des TicTacToe-Spielfeldes

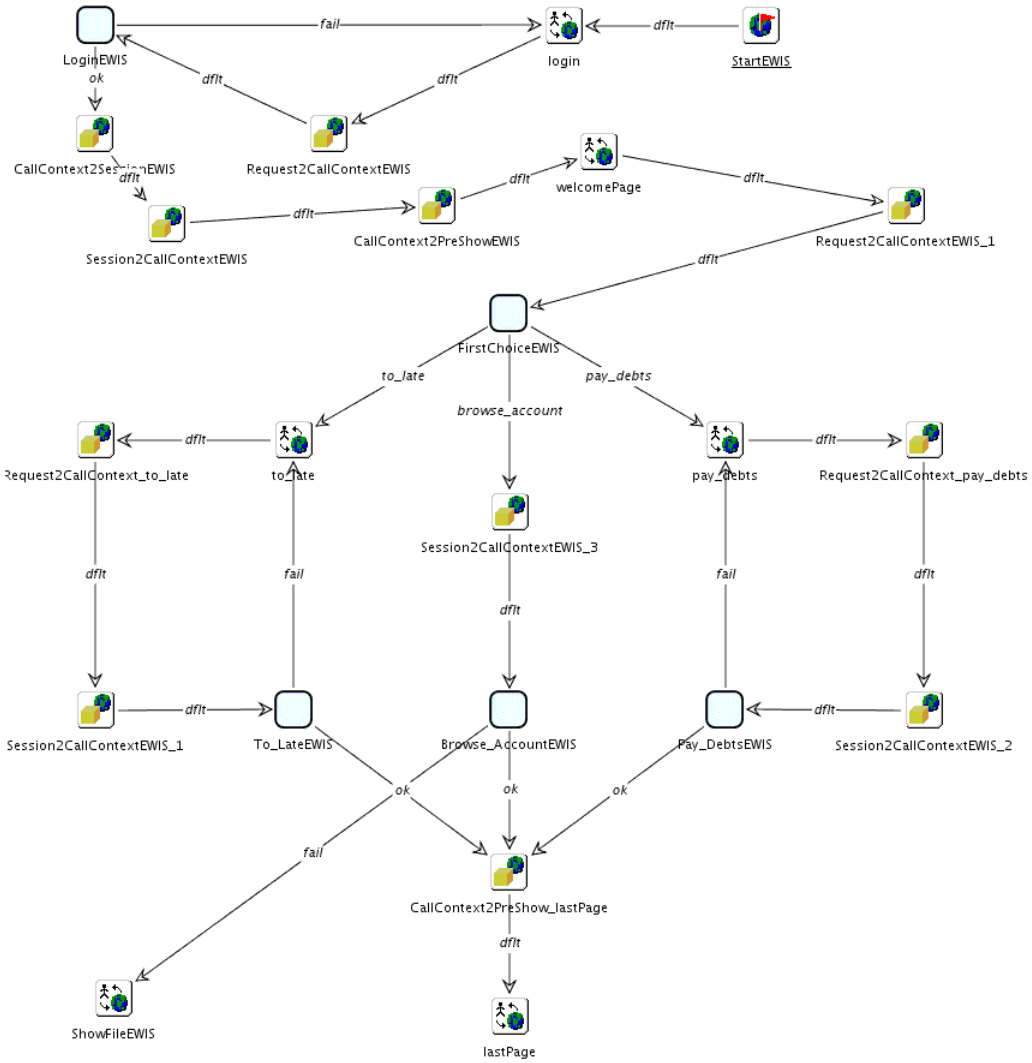

Abbildung 3.9: jABC Modell des Treasurer-Programms

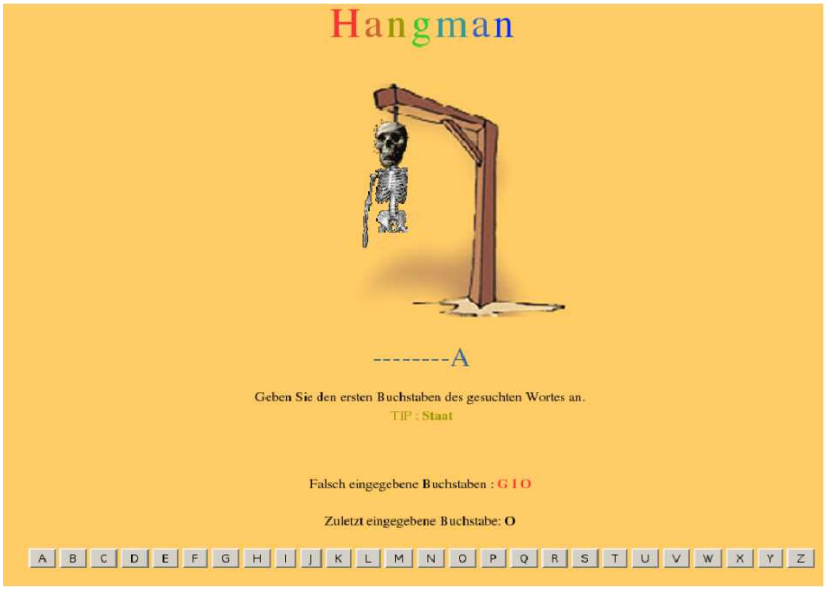

Abbildung 3.10: Die Webanwendung Hangman

Kapitel 4

Kurzthemen

Als Vorbereitung auf die Arbeit mit der MaTRICS, wurden die Teilnehmer in vier Gruppen aufgeteilt, die jeweils zu verschiedenen Themen SIB-Bibliotheken anfertigen sollten, von denen sich ein großer Teil als nützlich erweisen sollte. Auf Grund des Umfangs der einzelnen Bibliotheken kann und soll hier keine detaillierte Beschreibung folgen. Stattdessen wird nach einer kurzen Einführung in die einzelnen Bibliotheken, deren Funktionsweise anhand von Beispielen veranschaulicht.

4.1 SIB Bibliothek auf Basis der JAIN-SIP API zur Kommunikationssteuerung via SIP

Die Aufgabe bestand darin eine SIB-Bibliothek zu designen und zu implementieren, über die verschiedene Kommunikationsfunktionen gesteuert werden können. Diese Funktionen sollten auf dem Session Initiation Protocol (SIP) basieren. Dies ist ein Signalisierungs Protokoll, das den Signalaustausch zur Kommunikation zwischen ein oder mehreren Endpunkten über das Internet festlegt. Zur Implementierung konnte die JAIN-SIP API genutzt werden, um auf einem höheren Abstraktionsniveau zu arbeiten. Diese API ist eine standarisierte Java Schnittstelle über die auf die Funktionen und Elemente des SIP Protokolls zugegriffen werden kann. Zudem wurde das Projekt jain-sip-applet-phone als Schnittstelle genutzt, da die umfangreiche Behandlung der SIP Protokollstapel und deren Aufbau von diesem Programm implementiert wurden.

4.1.1 Design

4.1.1.1 Anwendungsfalldiagram

Um ein möglichst implementierungsnahes Design zu erhalten entsprechen die Anwendungsfälle, wie anhand des Anwendungsfalldiagramms 4.1 erkennbar, den verschiedenen

Nachrichtentypen, die das Session Initiation Protocol zur Verfügung stellt, dabei wird zwischen Request- und Response-Nachrichten unterschieden.
Die Anwendungsfälle im Einzelnen sind

- der Registervorgang, der durch ein REGISTER-Request durchgeführt wird
- das Session Management, das über INVITE-Requests gesteuert wird
- die einem BYE-Request entsprechende Beendigung einer Session
- die Einstellung verschiedener Optionen über OPTION
- die Reaktion auf ein erhaltenes INVITE-Request, nämlich das Annehmen per OK-Response bzw. das Ablehnen über ein BUSY-HERE-Response.

4.1.1.2 Aktivitätsdiagram

Anhand des Aktivitätsdiagramms 4.2, das den Aufbau einer Session darstellt, ist der Kommunikationsfluss zwischen User Agent Client, also dem Nutzer der SIB Bibliothek, dem User Agent Server, d.h. dem gewählten Gateway und dem Verbindungspartner (Callee) nachvollziehbar. Eine initiale INVITE-Nachricht führt dabei zu einem Response des Servers und der Weiterleitung des INVITE zum Callee, bei dem wiederum ein Response bewirkt wird und zudem eine accept-Nachricht erzeugt werden kann. Diese verschiedenen nebenläufigen Aktionen werden im positiven Fall beim Client wieder zusammengeführt und führen zum Erstellen einer Verbindung der Endpunkte.

4.1.2 Implementierung

Um das jain-sip-applet-phone als Schnittstelle zu nutzen, wurde die Klasse MaTRICS-MessengerManager implementiert, die von der Klasse MessengerManager des applet-phone abgeleitet ist. Durch eine Instanz dieser Klasse können die von dem applet-phone implementierten Methoden, mit entsprechenden Parametern aufgerufen werden. Die Klasse SipManager ist eine weitere Manager Klasse, die zur Instanziierung des MaTRICS-MessengerManager dient. Sie stellt verschiedenste Methoden bereit, auf die von den SIBs zugegriffen wird.Zusätzlich wird sie als Observer auf einkommende Requests und Responses eingesetzt. Über folgende SIBs können nun die SIP-Funktionen in Anwendungen eingebunden werden:

- AnswerCall: Es wird ein eingehender Ruf beantwortet.
- Register: Es wird eine Registrierung angestoßen.
- Unregister: Die Registrierung wird aufgehoben.

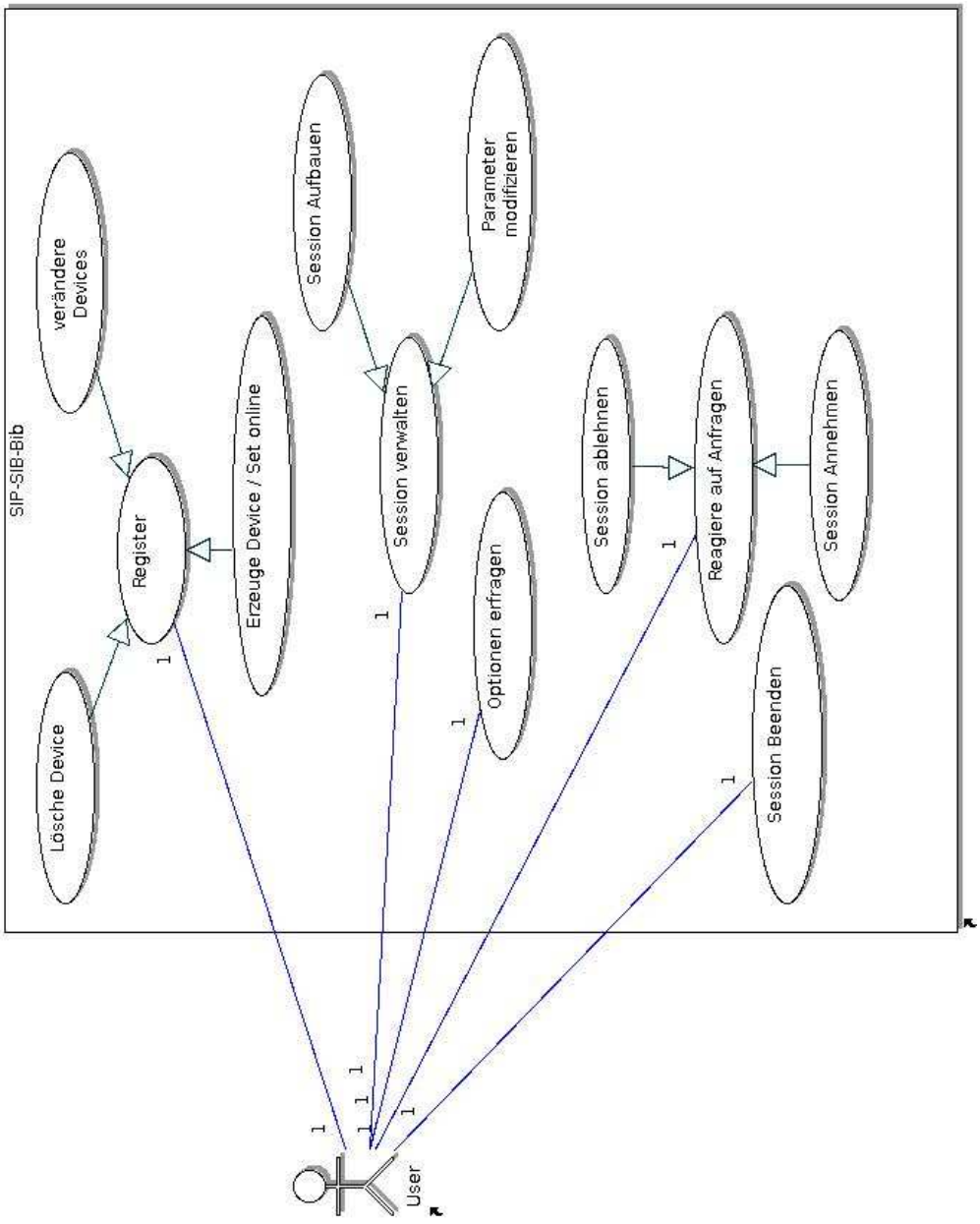

Abbildung 4.1: Anwendungsfalldiagramm der SIB Bibliothek

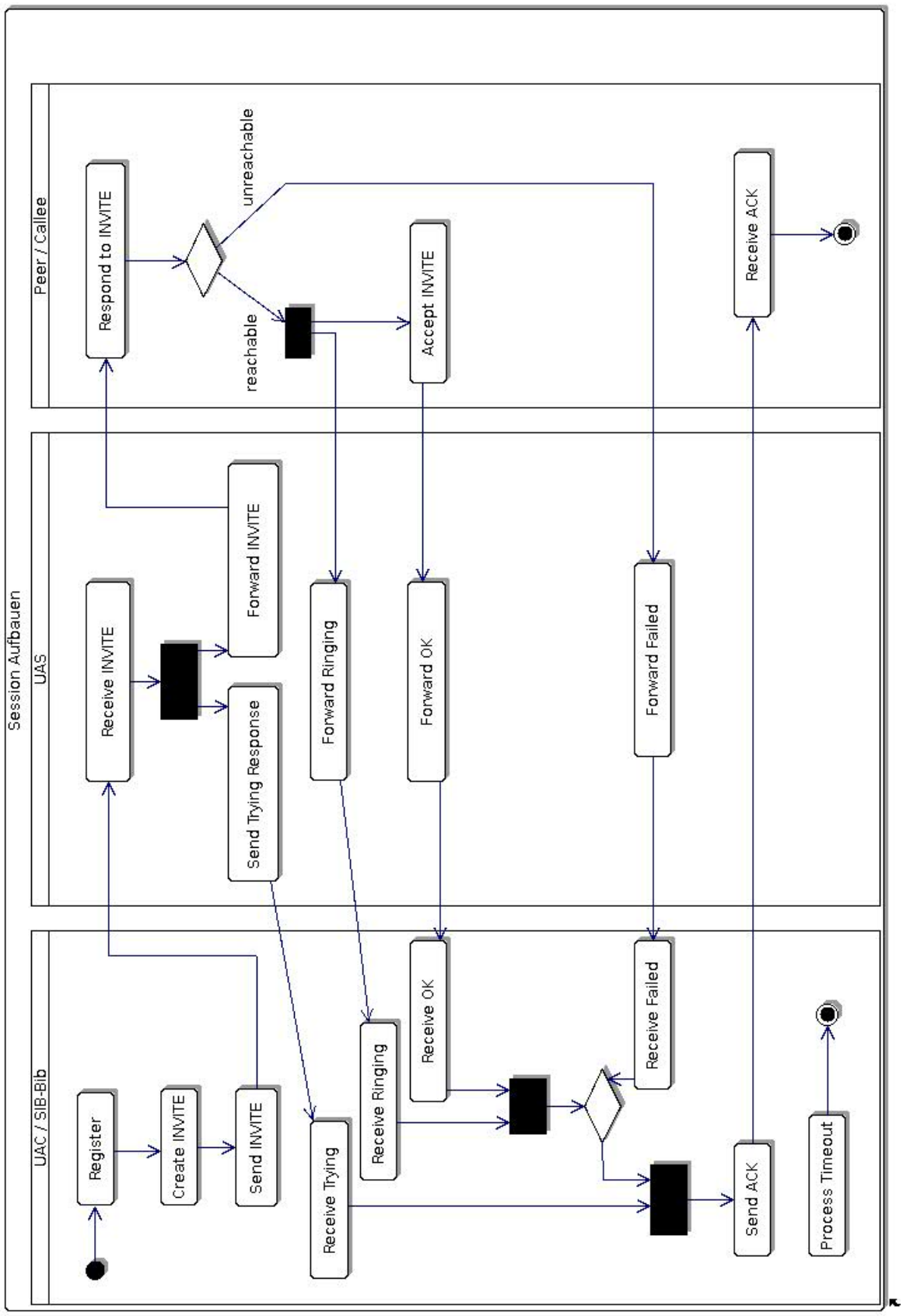

Abbildung 4.2: Aktivitätsdiagramm zum Aufbau einer Session

- EndCall: Eine Session wird abgebaut.
- Call: Der Aufbau einer Session wird initialisiert.
- CancelCall: Ein zuvor versuchter Sessionaufbau wird abgebrochen.
- DenyCall: Ein eingehender Anruf wird abgeblockt.
- StartManager: Eine Instanz des SipManagers wird mit den für eine Registrierung erforderlichen Parametern erzeugt.

4.1.3 Beispiel

Zur Veranschaulichung wurde ein SIP-Service gebaut, mit dem eine einfache Verbindung aufgebaut und beendet werden kann, sowie die weiteren verschiedenen Funktionen der oben beschriebenen SIBs genutzt werden können. Zunächst muss ein User seinen Usernamen, sowie sein Passwort und die Adresse seines Registrars angeben, beim dem ein SIP-Account eingerichtet wurde. Sobald der Benutzer registriert ist, hat er die Möglichkeit einen Anruf zu tätigen oder einen ankommenden Anruf anzunehmen, bzw. abzulehnen. Befindet er sich in einer Verbindung so kann er diese beenden. Den Workflow dieses Services veranschaulicht Abbildung (4.3)

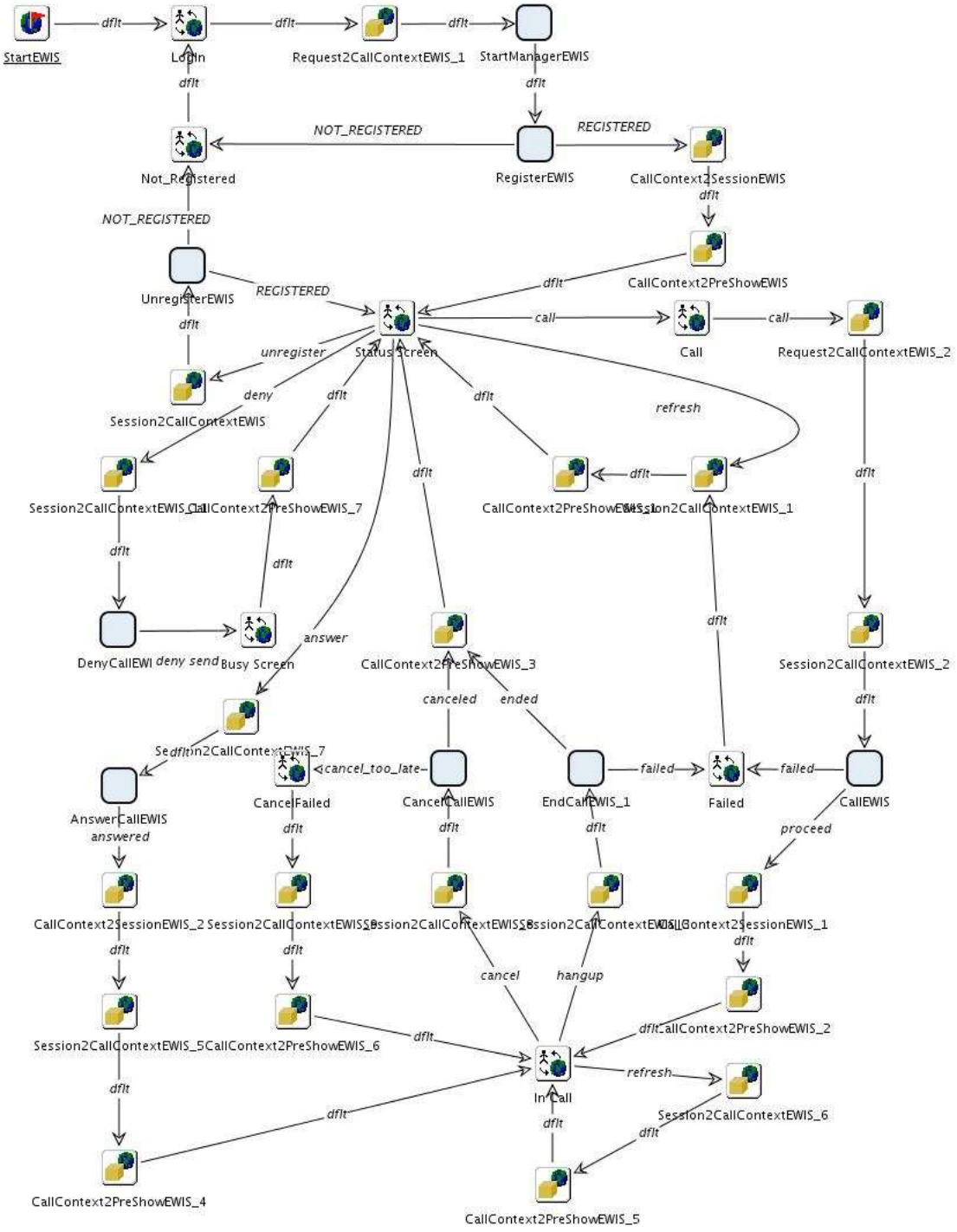

Abbildung 4.3: Workflow des SIP-Service

4.2 SIB-Bibliothek zur Verwaltung von Asterisk über das Asterisk-Manager-Interface

Unsere Aufgabe bestand darin eine SIB-Bibliothek zur Verwaltung von Asterisk zu konzipieren und zu implementieren. Dazu sollte die von Asterisk bereitgestellte *AMI(Asterisk Management Interface)* und die Asterisk-Java-API (siehe http://asterisk-java.org) genutzt werden.

AMI ermöglicht den entfernten Zugriff auf Asterisk und damit die Rekonfiguration seines Verhaltens zur Laufzeit. Mithilfe der Asterisk-Java-API kann die *AMI*-Kommunikation mit Asterisk etwas abstrahiert werden: es sind lediglig die Antworten des Asterisk *manuell* zu parsen und zu interpretieren.

4.2.1 Design

Aus Sicht des Benutzers sollte es keine methodischen Unterschiede in der Anwendung verschiedener Anwendungsfälle geben, deshalb entschieden wir uns die Management-Sessions zustandslos zu halten. Diese Abstraktion führte zu einem sehr generischen Konzept der Anwendungsfälle 4.4. Das Aktivitätsdiagramm 4.5. verdeutlicht nochmals wie die Kommunikation durch impliziete log-in bzw. log-off Befehle zustandslos gehalten wird. Zur Überwachung eines Asterisk wurde ein Asterisk-Event-Listener entworfen. Der Event-Listener lauscht auf eingehende Events und stößt für jedes eingehende Event die Ausführung eines (zuvor definierten) Web-Service an. Das Sequenzdiagramm 4.8 beschreibt die Infratruktur des Event-Listeners.

4.2.2 Implementierung

Die zentralen Komponenten der Implementierung sind die Factory-Klassen *LiveFactory* und *CommandFactory*. Diese dienen als Bindeglied zwischen SIB-Klassen und den *Asterisk-Java-APIs* und stellen für alle wichtigen *AMI*-Aktionen eine Methode bereit, die die komplette Kommunikation übernehmen und entweder eine gültige Antwort zurückgeben oder eine Exception werfen.

Die *LiveFactory* übernimmt dabei alle Aktionen die über die Live-API des Asterisk-Java Pakets realisiert werden konnten wogegen die *CommandFactory* für Aktionen zuständig ist die nicht direkt über das *AMI* sondern nur im *CLI(Command Line Interface)* verfügbar sind. Dabei werden *CLI*-Befehle mithilfe der AMI-Aktion *command* direkt an die Asterisk-*CLI* übertragen. Allerdings besteht die Antwort der *command*-Aktion nur aus den Textzeilen die bei Ausführung des *CLI*-Befehls ausgegeben werden. In den meisten Fällen müssen die Textzeilen erst in eine sinnvolle Datenstruktur überführt werden.

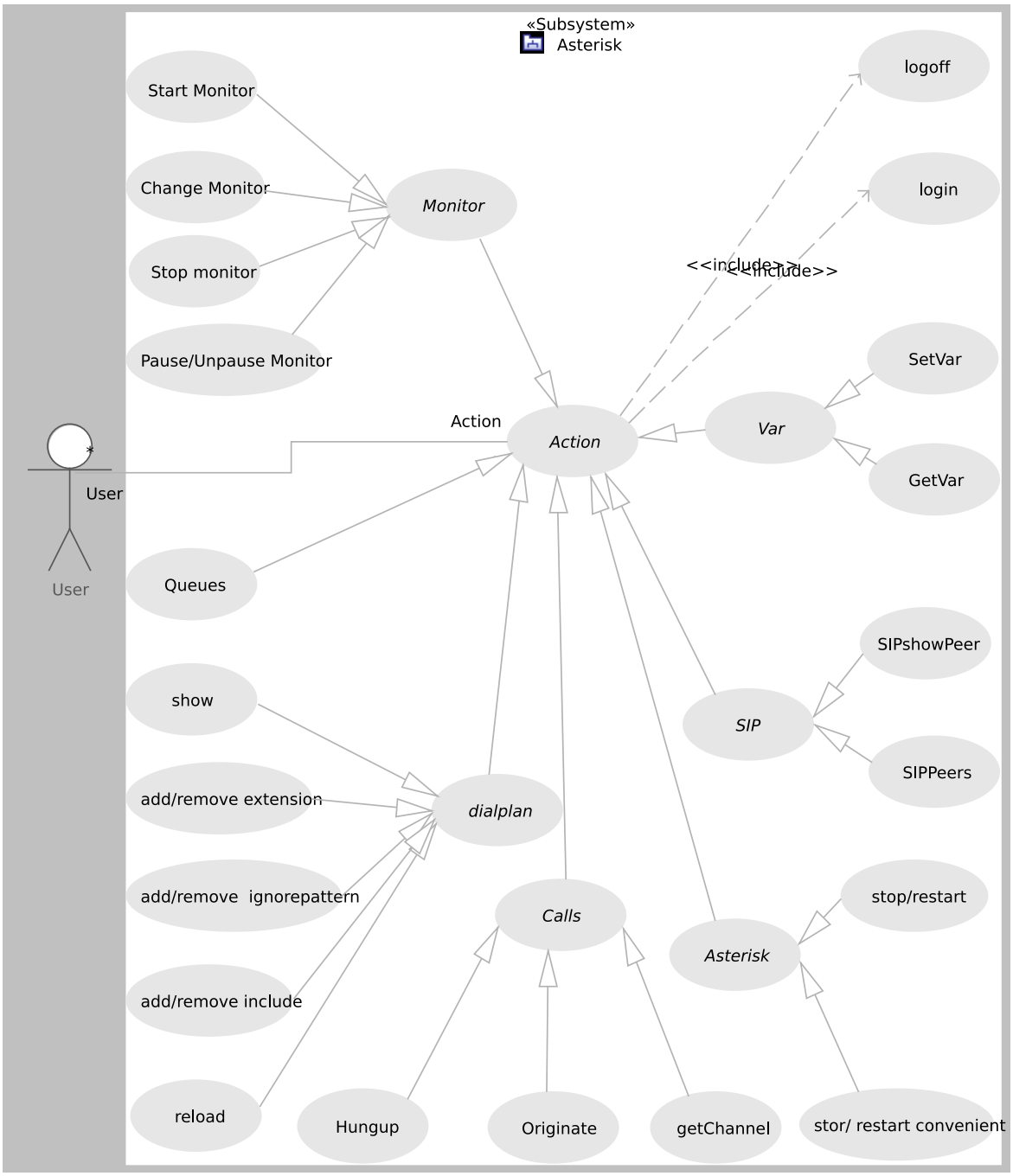

Abbildung 4.4: Anwendungsfalldiagramm

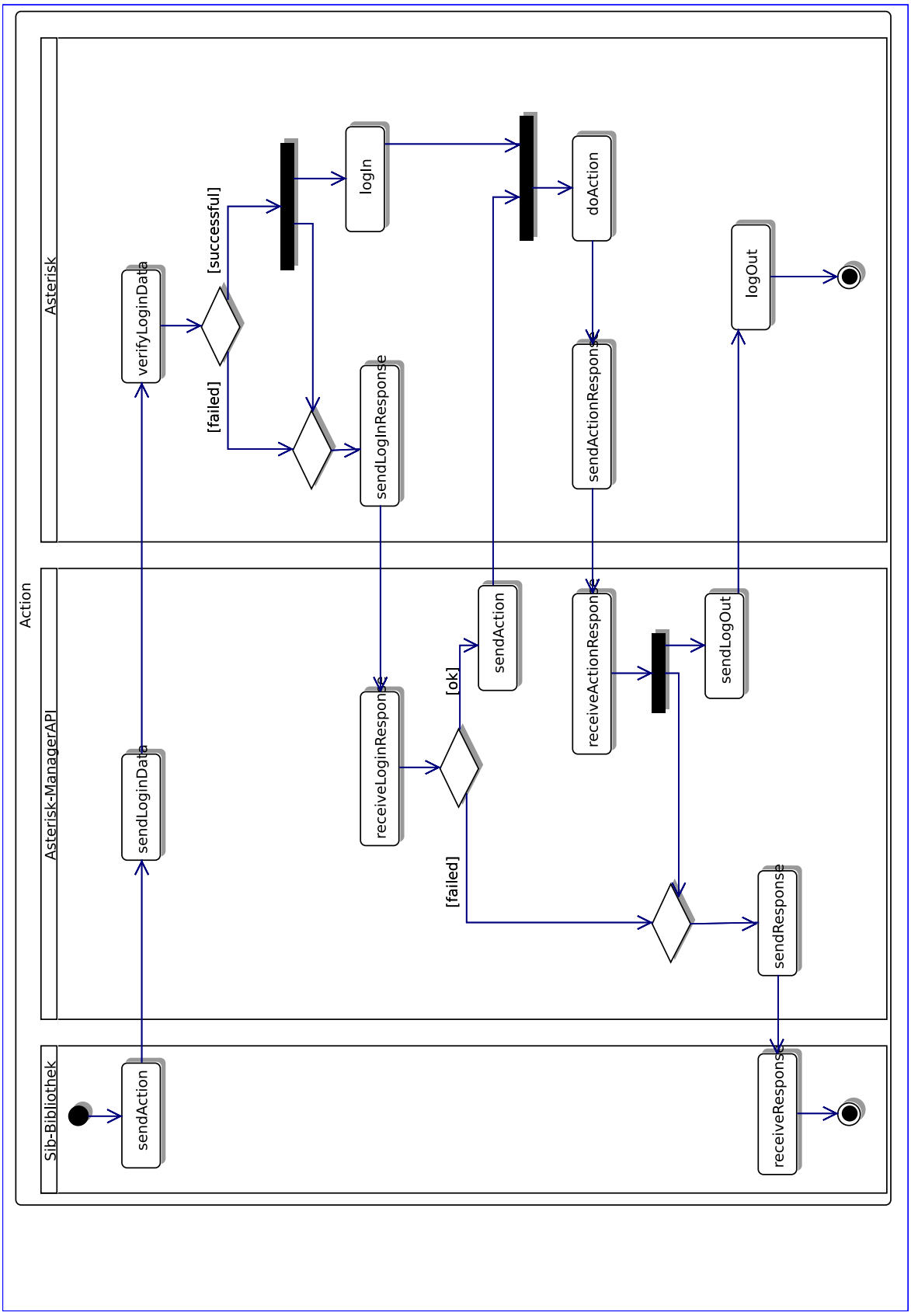

Abbildung 4.5: Aktivitätsdiagramm einer Manager-Aktion per SIB-Bibliothek

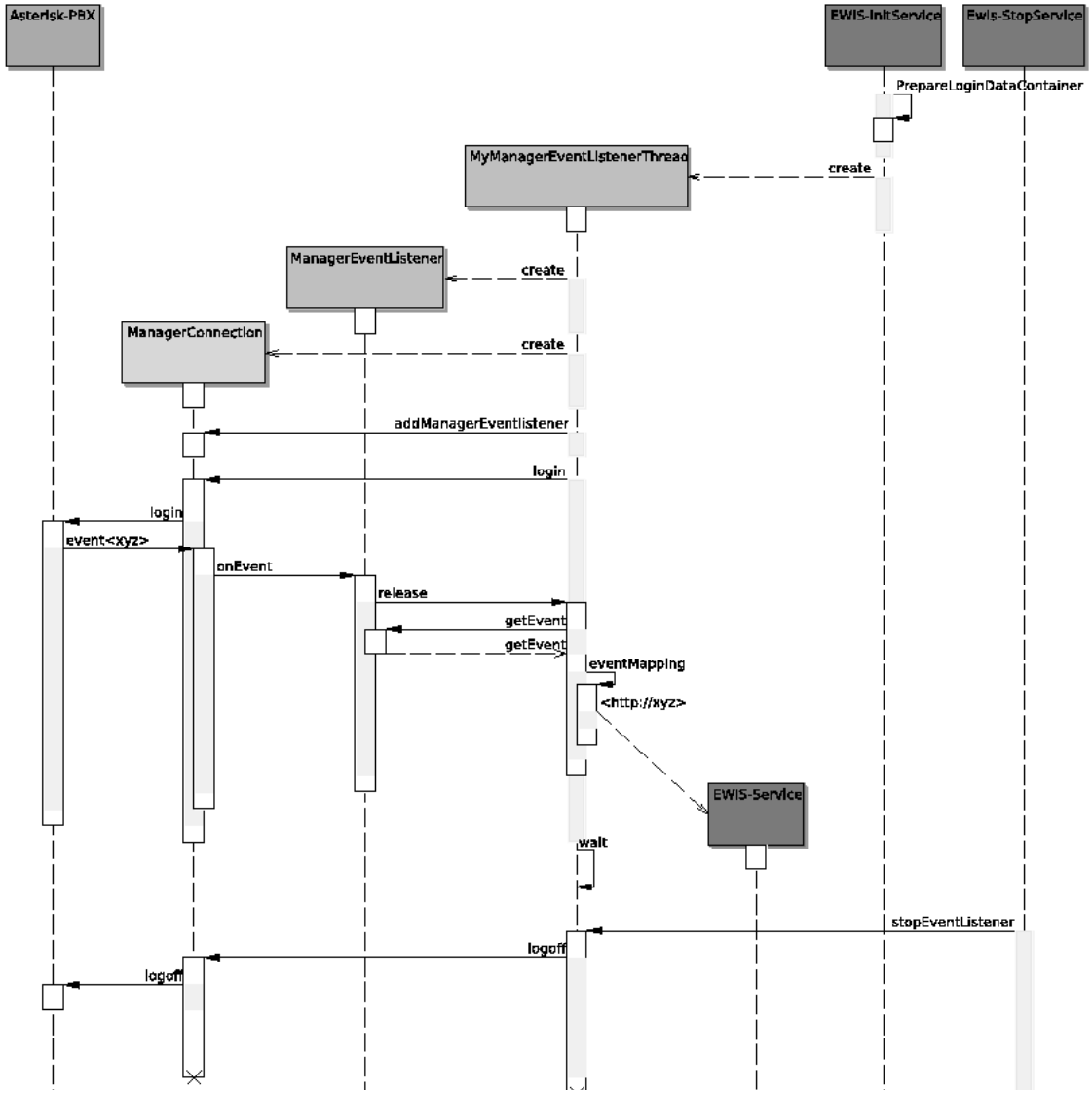

Abbildung 4.6: Sequenzdiagramm: Event-Listener wird initialisiert und empfängt eine Event

Besonders deutlich wird das beim *Dialplan*: die Abfrage des aktuelle Dialplans erfolgt über den Befehl *dialplan show*, dessen Antwort in Abbildung 4.7 zu sehen ist.

Diese Ausgabe wird nun in die Container-Klasse *StructuredDialplan* 4.7 transformiert. .

Nach der Fertigstellung der Factory-Klassen, diverser Datenstrukturen, Parser und Event-Listener-Komponenten wurden folgende SIB implementiert:

- Die SIBs *GetDialplanEWIS*, *DialplanReloadEWIS*, *GetGlobalVariableEWIS*, *DialplanAddIgnorePatternEWIS*, *DialplanAddIncludeEWIS*, *DialplanAddExtensionReplaceEWIS*, *DialplanRemoveExtensionEWIS*, *DialplanRemoveIgnorePatternEWIS*, *DialplanRemoveIncludeEWIS*, *DialplanRemovePriorityEWIS*, *SetGlobalDialplanVariableEWIS*, *GetAllGlobalsEWIS* können für die Darstellung und bearbeitung des Dialplans verwendet werden.
- Zur Darstellung und Verwaltung von Channles stehen folgende SIBs zu Verfügung: *GetChannelsEWIS*, *GetChannelByNameEWIS*, *GetChannelByIDEWIS*, *ChannelHangupEWIS*, *GetChannelVariableEWIS*, *SetChannelVariableEWIS*, *StartMonitorEWIS*, *ChangeMonitorEWIS*, *PauseMonitorEWIS*, *UnPauseMonitorEWIS*, *StopMonitorEWIS*, *GetQueuesEWIS*.
- SIP-Peers können mit den SIBs *GetSIPPeersEWIS*, *GetSIPPeerInfoEWIS*, *OrginateToApplicationEWIS*, *OrginateToExtensionEWIS* und *SIPReloadEWIS* dargestellt und verwaltet werden.
- Die Arbeit des Event-Listeners steuern die SIBs *InitializeManagerEventListenerEWIS*, *InitializeEventMapEWIS*, *StopManagerEventListenerEWIS*, *RemoveEventsEWIS*, *EventCounterEWIS*, *GetEventMappingEWIS*, *AddEventMappingEWIS*.
- Der Astersik selbst kann ebenfalls gesteuert werden. Dazu dienen die SIBs *GetAsteriskVersionEWIS*, *RestartAsteriskEWIS*, *StopEWIS*, *ConsoleDialEWIS*, *ConsoleHangupEWIS*.
- Die SIBs *PrepareLoginDataEWIS* und *PrepareLoginDataInitContextEWIS* erzeigen eine Instanz der Klasse *LoginDataContainer*. Diese enthält alle Daten die zur initialisierung einer Management-Session benötigt werden und muss jedem SIB der eine *AMI*-Aktion darstellt mitgegeben werden.

4.2.2.1 Event-Listener

Der Eventlistener besteht aus den zwei Klassen MyManagerEventListener und MyManagerEventListenerThread. MyManagerEventListenerThread baut bei Initialisierung eine Verbindung zum Asterisk auf, und hält diese offen, bis zur Ausführung der stopEventlistener() Methode. Über diese Klasse ist es möglich zur Laufzeit weitere Events zu registrieren, sowie vorher registrierte Events zu entfernen. MyManagerEventListener enthält

```
[ Context 'default' created by 'pbx
  [ Context 'internal' created by 'pb
    '1' =>
          1. Answer()
          2. SayAlpha(Hal
          3. SayAlpha(Pet
```

Abbildung 4.7: Die textuelle Darstellung des Dialplans

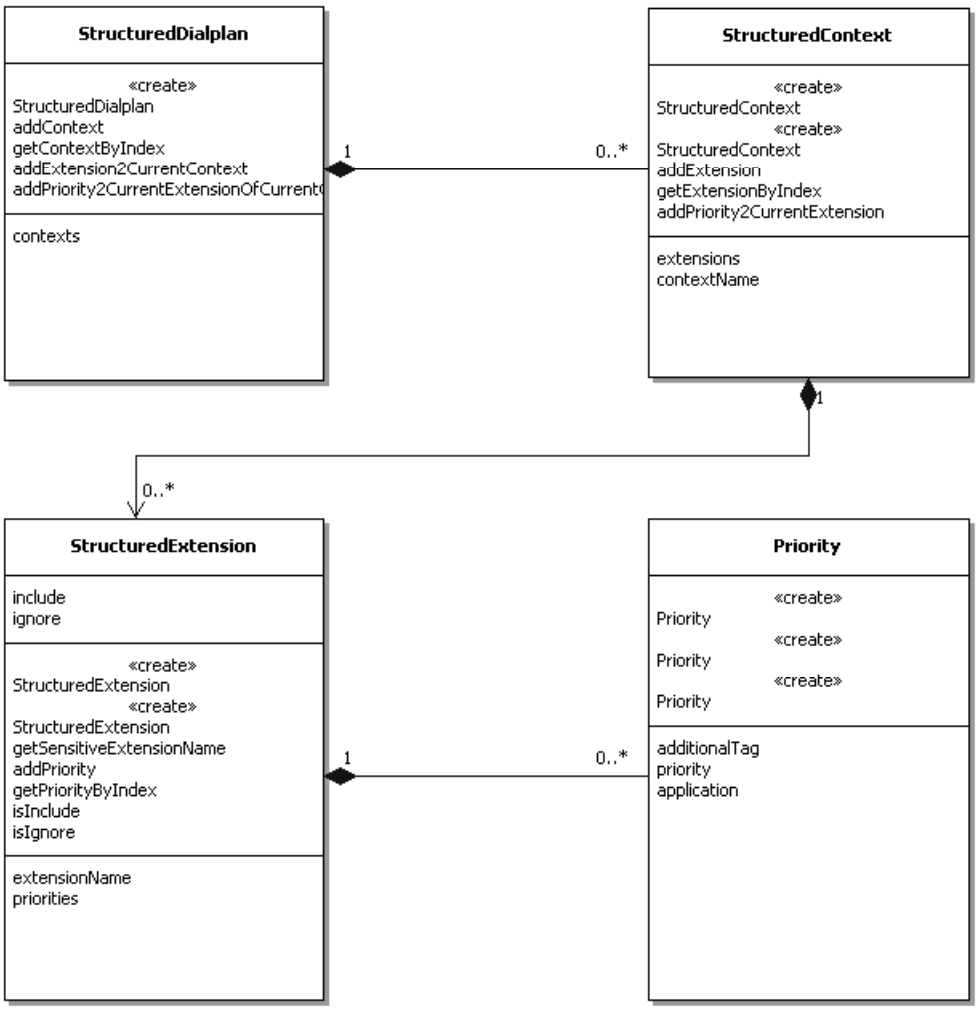

Abbildung 4.8: Klassendiagramm: Die tiefe Struktur eines Dialplans

eine Hashmap der registrierten Events und ihren zugeordneten Webservices. Bei Auftreten eines Events wird die Methode onManagerEvent() ausgeführt, die das Aufrufen des registrierten Webservices veranlasst, falls das Event registriert wurde.

4.2.3 Beispiele

4.2.3.1 Neustart des Asterisk

Dieses Beispiel zeigt wie mithilfe der SIB-Bibliothek ein Web-Service modelliert werden kann der einen Asterisk neustarten lässt.

Zunächst wird mit dem *PrepareLoginDataContainerEWIS*-SIB ein LoginDataContainer erzeugt, der von allen folgenden AMI-SIBs benötigt wird. Die Parameter des LoginDataContainers sind:

- serverAddress_key: die IP-Adresse des Asterisk
- serverPort_key: die Port-Nummer des Asterisk
- username_key: der Benutzername des Manager-Kontos
- secret_key: das Passwort des Manager-Kontos

Mit dem *GetAsteriskVersionEWIS*-SIB wird die Versionsnummer des Asterisk ermittelt, was nur als zusätztliche Information dient und nicht weiter benötigt wird. Schließlich wird mit dem SIB *RestartAsteriskEWIS* der Neustart initiiert.

4.2.3.2 Dialplan-Verwaltung

In diesem Beispiel werden fast alle SIBs zur Verwaltung des Dialplans verwendet, was dieses Beispiel zu einem produktiven und praxisrelevanten Dienst macht.

Wie immer wird ein LoginDataContainer benötigt, anschließend holt der SIB *GetDialplanEWIS* den Dialplan und sorgt dafür, dass dieser zu einem *StructuredDialplan*-Object geparst wird. Das *StructuredDialplan*-Object wird dann mit Velocity abgerollt und auf der HTTP-Seite dargestellt 4.10.

Mit den SIBs *DialplanRemovePriorityEWIS, DialplanRemoveExtensionEWIS, DialplanRemovePriorityEWIS, DialplanAddExtensionReplaceEWIS* werden Möglichkeiten zur Manipulation des Dialplans zur Verfügung gestellt.

Weitere (und detailiertere) Beispiele könne der Dokumentation[1] entnommen werden.

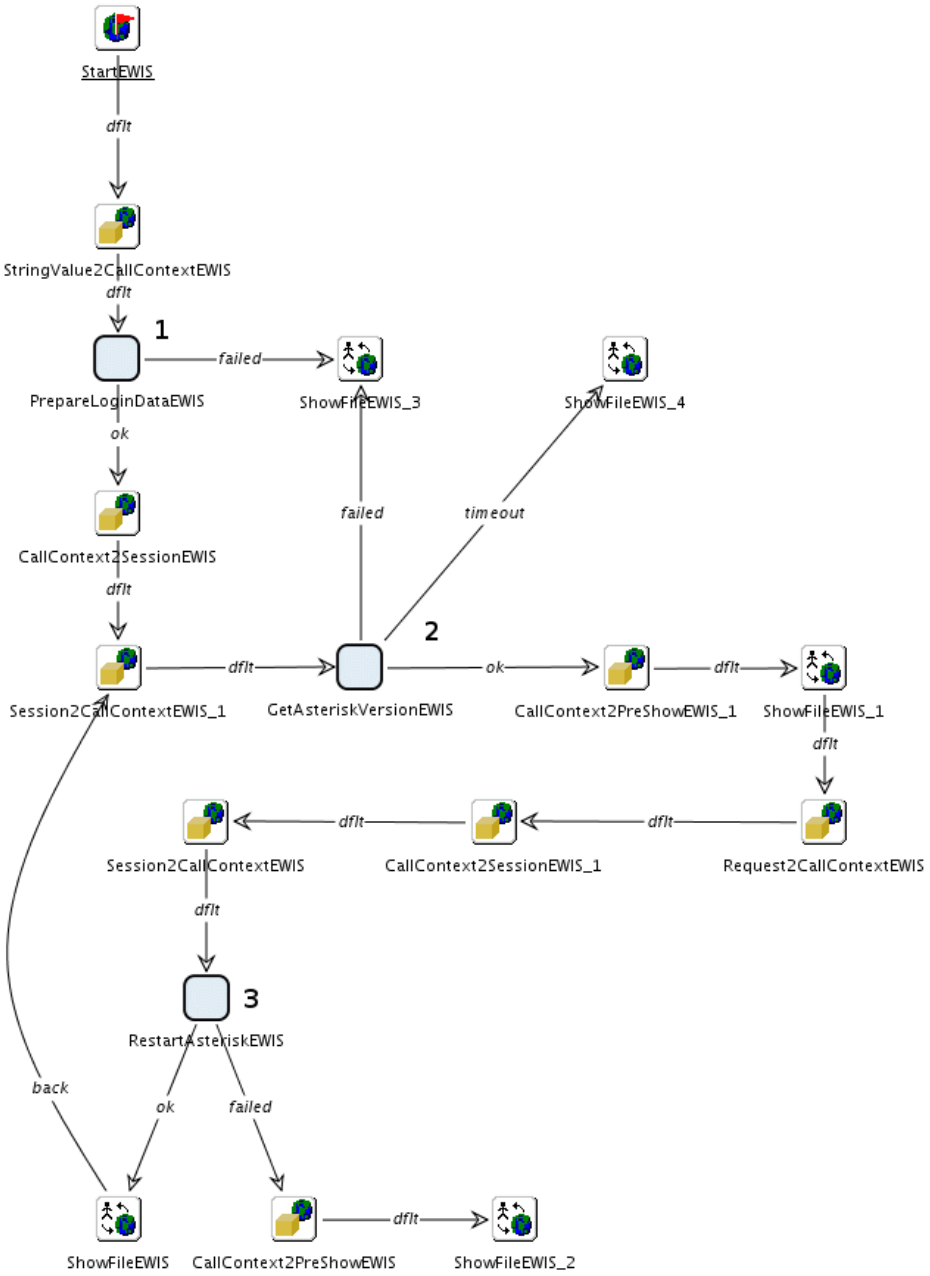

Abbildung 4.9: Workflow: Neustart eines Asterisk

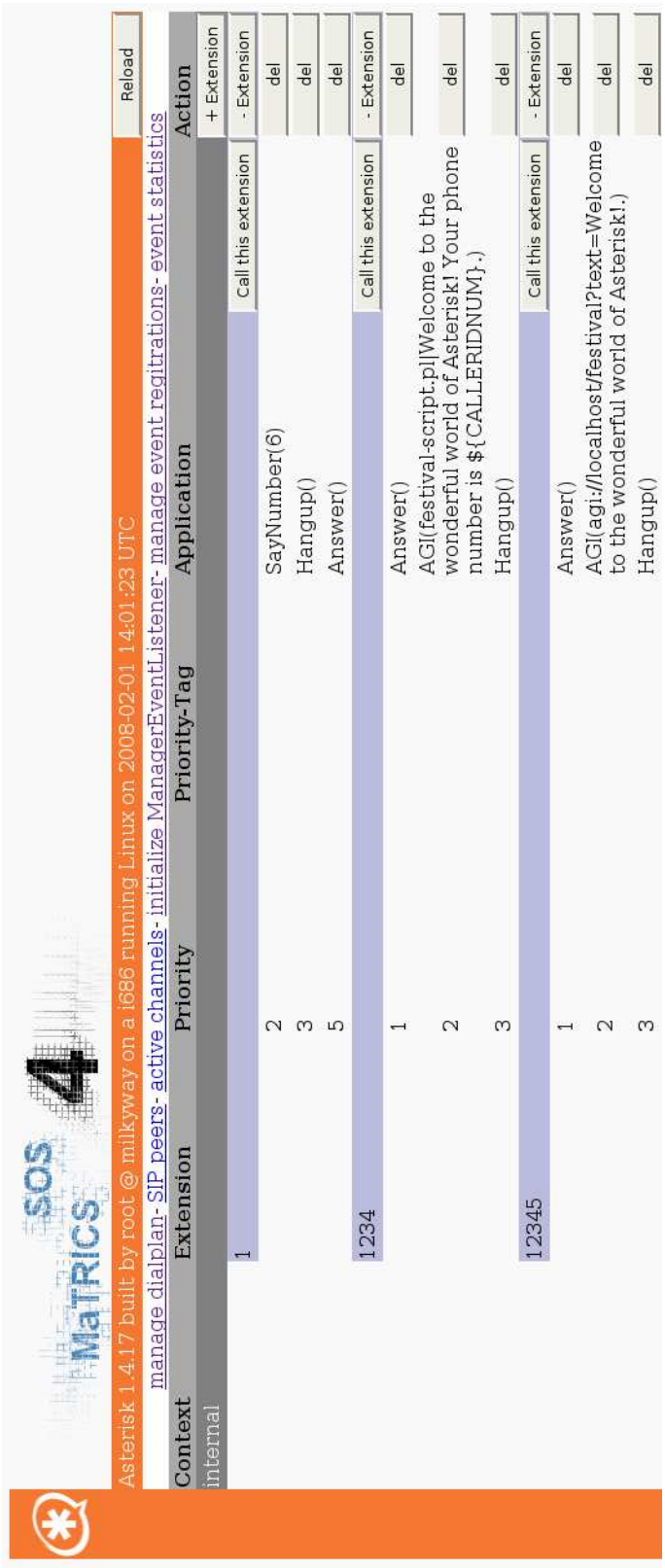

Abbildung 4.10: Screenshot des Dialplan-Verwaltungs-Dientes

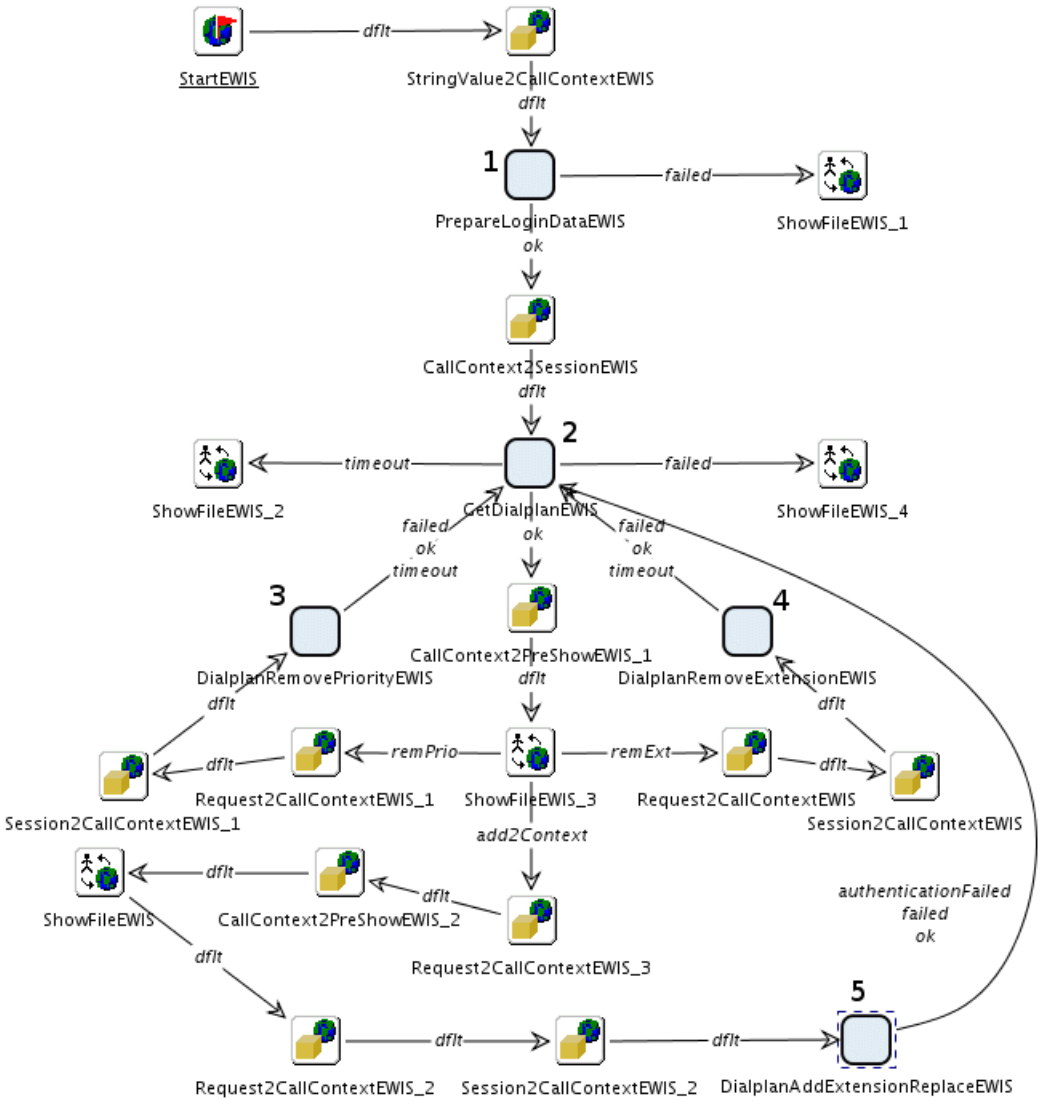

Abbildung 4.11: Workflow des Dialplan-Verwaltungs-Dientes

4.3 SIB Bibliothek zur Modifikation von ASCII-Dateien

4.3.1 Einleitung

Der Asciieditor ist ein nützliches Paket um Konfigurationsdateien verschiedener Systemen und Anwendungen zu bearbeiten.

Die gestellte Aufgabe, unterteilt sich in zwei Teile. Als erstes sollte eine generelle und erweiterbare SIB Bibliothek zur Modifikation von textbasierten Dateien erstellt werden. Die Ziele für diesen ersten Teil waren die Realisierung und Implementierung von grundlegenden Operationen, wie Suchen, Ersetzen, Löschen, Einfügen und Kopieren. Die komplexeren Operationen des Editors sollten mittels Verwendung von regulären Ausdrücken realisiert werden. Eine XML basierte Grammatik musste entworfen werden, um die Textdateien auf rudimentäre Weise validieren zu können. Die Grammatik musste die jeweils validen Schlüsselwörter mit dazugehörigen Argumenten enthalten. Desweiteren musste die Grammatik fähig sein Informationen über den Kontext wiederzugeben, um so den richtigen Gebrauch der Schlüsselwörter zu realisieren.

Das Design sollte mit Hilfe von UML Diagrammen erstellt werden. Ein Anwendungsfalldiagramm sollte einen ersten Eindruck von den Möglichkeiten des Programms vermitteln. Für alle Abläufe musste ein Aktivitätsdiamgramm erstellt werden. Das Klassendiagramm gibt einen Überblick, bevor mit der Implementierung des Codes begonnen wird. Um zeitliche Abhängigkeiten darzustellen wurden Sequenzdiagramme genutzt.

Der zweite Teil bestand aus der Implementierung, dem Testen und der Verifikation der SIBs. Eine gute Möglichkeit um den gesamten Ablauf zu zeigen sind exemplarische Beispiele, welche die syntaktische Modifikation von Ascii Dateien zeigt.

Um diese Ziele zu realisieren, erstellten wir ein Asciieditor-Paket, welches Klassen und Methoden für alle Arten von Operationen beinhaltet (s.o.). Dieses Editor-Paket wurde mit dem jEWIS Plug-In von jABC erstellt.

Die Interaktion mit unseren Paketen läuft über eine Controllerklasse. Nur diese Klasse, der Editor selbst, ist nach außen sichtbar. Diese Klasse nimmt Parameter von außerhalb an und gibt die Ergebnisse der gewünschten Operationen an den Aufrufer zurück. Die Asciieditor-Klasse instanziiert die Virtualeditor Klasse, welche die Kernklasse in diesem Projekt ist. Diese Klasse interagiert mit der Dateimanager-Klasse, welche alle Art von I/O Operationen übernimmt. Abgeschlossen wird das Paket durch die folgenden drei Klassen: einer RegEx-Engine-Klasse, die die entsprechenden Anweisungen auf Basis regulärer Ausdrücke ausführt, einer XML-Engine-Klasse, die u.a. für die Validierung der Grammatik zuständig ist, und einer Cursor-Engine-Klasse, die ein komfortable Steuerung innerhalb eines Dokumentes (Springen, Gehe zu,...) ermöglicht.

4.3.2 Design

4.3.2.1 Anwendungsfalldiagramm

Dieses Diagramm gibt einen kurzen Überblick über alle Anwendungen, die man mit dem Asciieditor durchführen kann. Als erste Aktion muss der Nutzer eine Asciidatei laden. Danach kann der Nutzer aus unterschiedlichen Aktionen wählen.

Erstmal kann der Nutzer Asciidateien modifizieren. Die Modfikation kann wieder in zwei Teile eingeteilt werden. Im ersten Teil kann der Nutzer kopieren, einfügen, überschreiben. Dies geschieht immer mittels eines regulären Ausdruckes. Der andere Teil benötigt ein explizites Finden eines regulären Ausdrucks als Vorbedingung. Danach kann/können die gefundenen Position(en) gelöscht, ersetzt und kopiert werden. Die Validierung einer textbasierten Asciidatei gegen eine Grammatik, die im XML-Format vorliegen muss, ist eine weitere Operation. Mit einer gegebenen XML-Grammatik hat der Nutzer die Möglichkeit, mit den benötigten Parametern eine neue Ascii-Datei zu erstellen und so Templates von Dateien zu erzeugen (s. Abb. 4.12).

4.3.2.2 Aktivitätsdiagramm

Zuerst versucht der Nutzer eine Datei zu laden. Das System überprüft, ob die Datei valide ist und ob jemand anderes zur Zeit auf die Datei zugreift. Sollte der Benutzer Zugriff erhalten, wird die Datei für den Nutzer reserviert und schreibgeschützt. Danach legt der Nutzer den einzufügenden Text und die gewünschte Position für diesen Text fest (Offset). Das System überprüft die Offset-Position. Wenn die Position valide ist, wird die Einfügeoperation durchgeführt. Nun kann der Nutzer entscheiden, ob die geänderte Asciidatei validiert werden soll. Wenn der Nutzer validieren mö, muss er dazu ein XML-Schema laden. Das XML-Schema selbst wird vorher noch auf Konsistenz und Validität gegenüber einer Metagrammatik hin überprüft, bevor die Valdierung der Asciidatei gegen dieses Schema startet. Der Nutzer erhält abschließend das Ergebnis der Validierung.

Unabhängig von der Validierungsentscheidung kann der Nutzer im nächsten Schritt entscheiden, ob die Datei gespeichert werden soll. Zuletzt muss der Nutzer das Asciidatei schließen und den Schreibschutz zu entfernen. (s. Abb. 4.13)

4.3.2.3 Sequenzdiagramm

Dieses Sequenzdiagramm zeigt einen erfolgreichen Copy&Paste-Vorgang im Offset-Modus. (s. Abb. 4.14)

4.3.2.4 Klassendiagramm

Der Dateimanager übernimmt alle Aufgaben, die bzgl. der I/O anfallen.

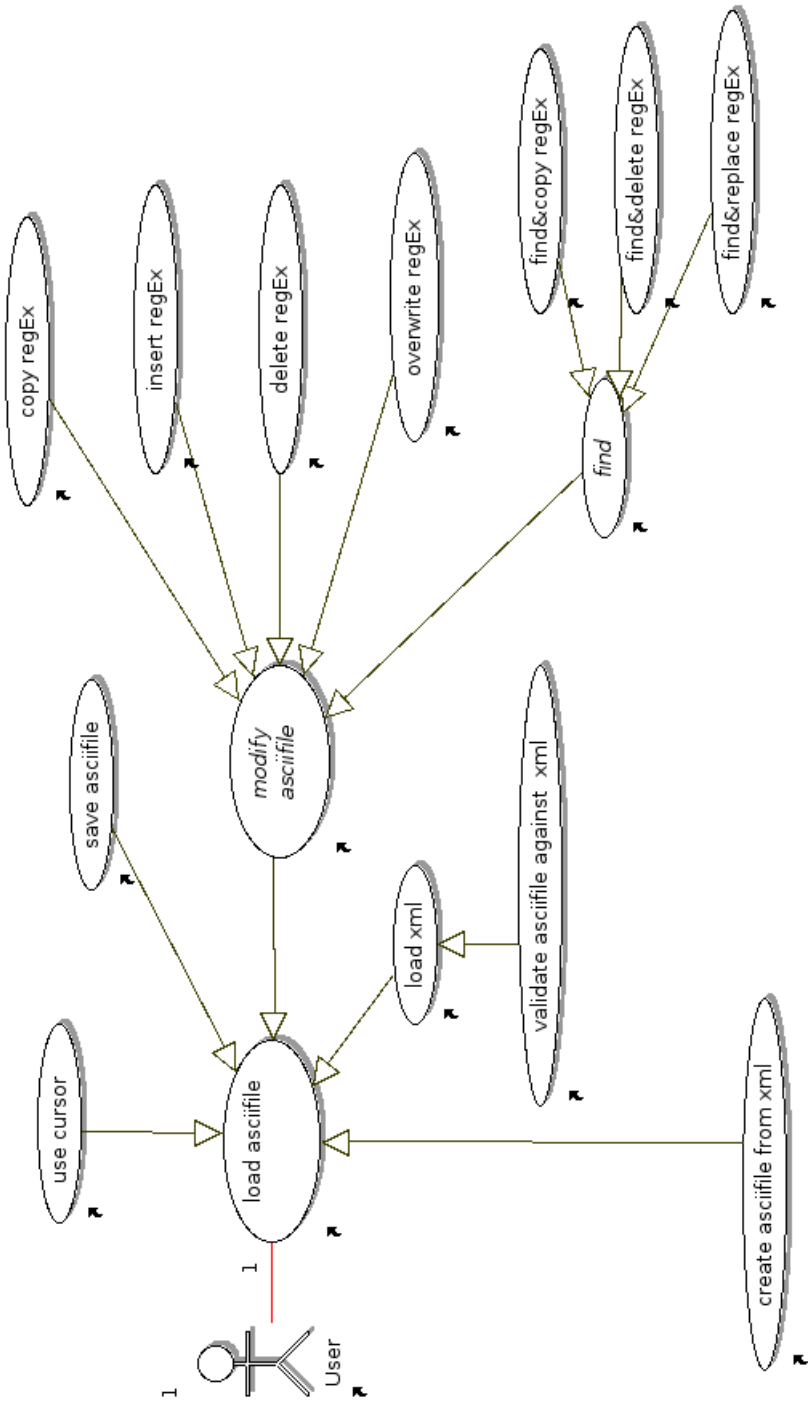

Abbildung 4.12: Anwendungsfalldiagramm der SIB-Bibliothek zur Modifikation von ASCII-Dateien

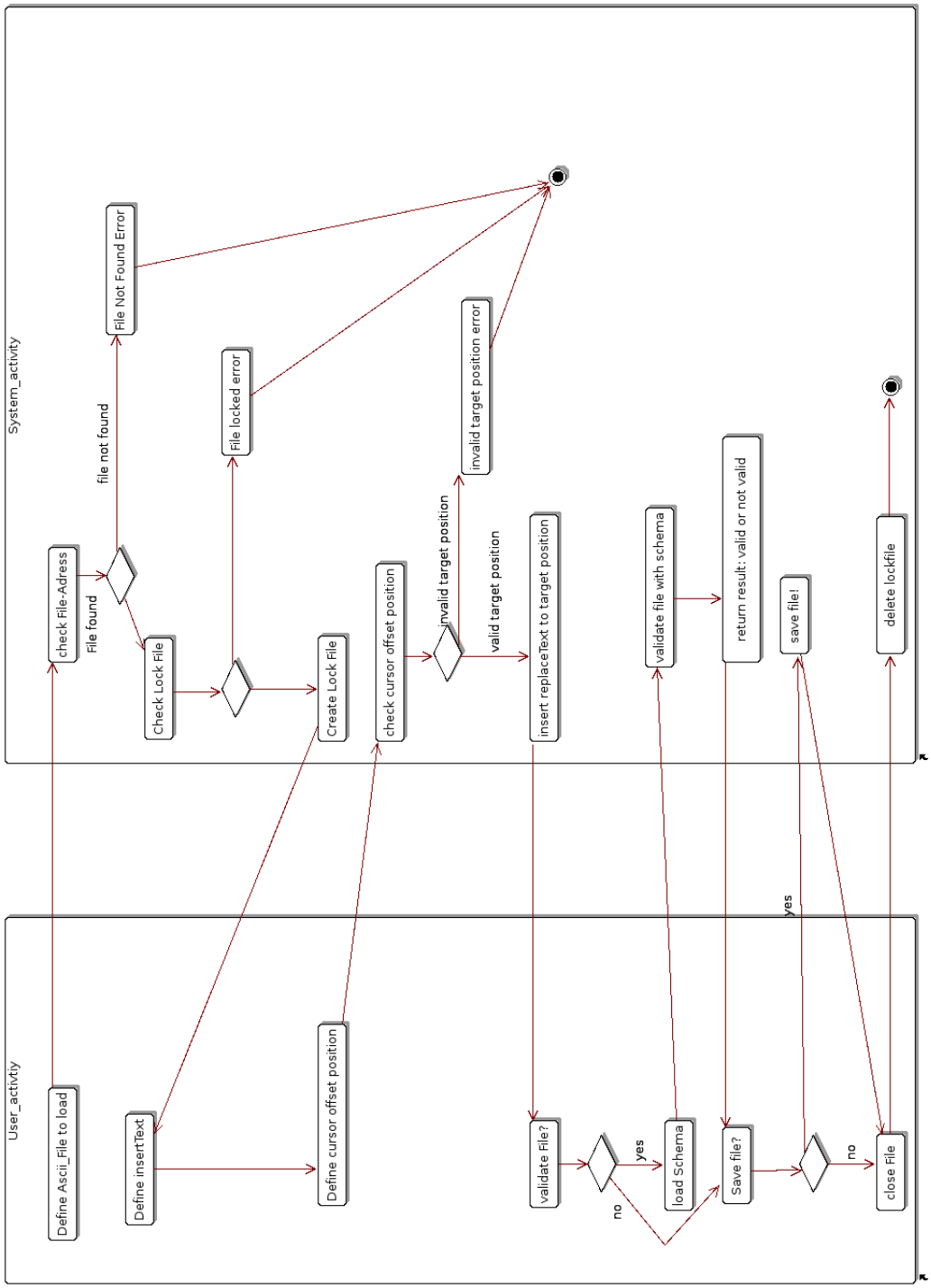

Abbildung 4.13: Aktivitätsdiagramm für SIB Bibliothek zur Modifikation von ASCII-Dateien Insert

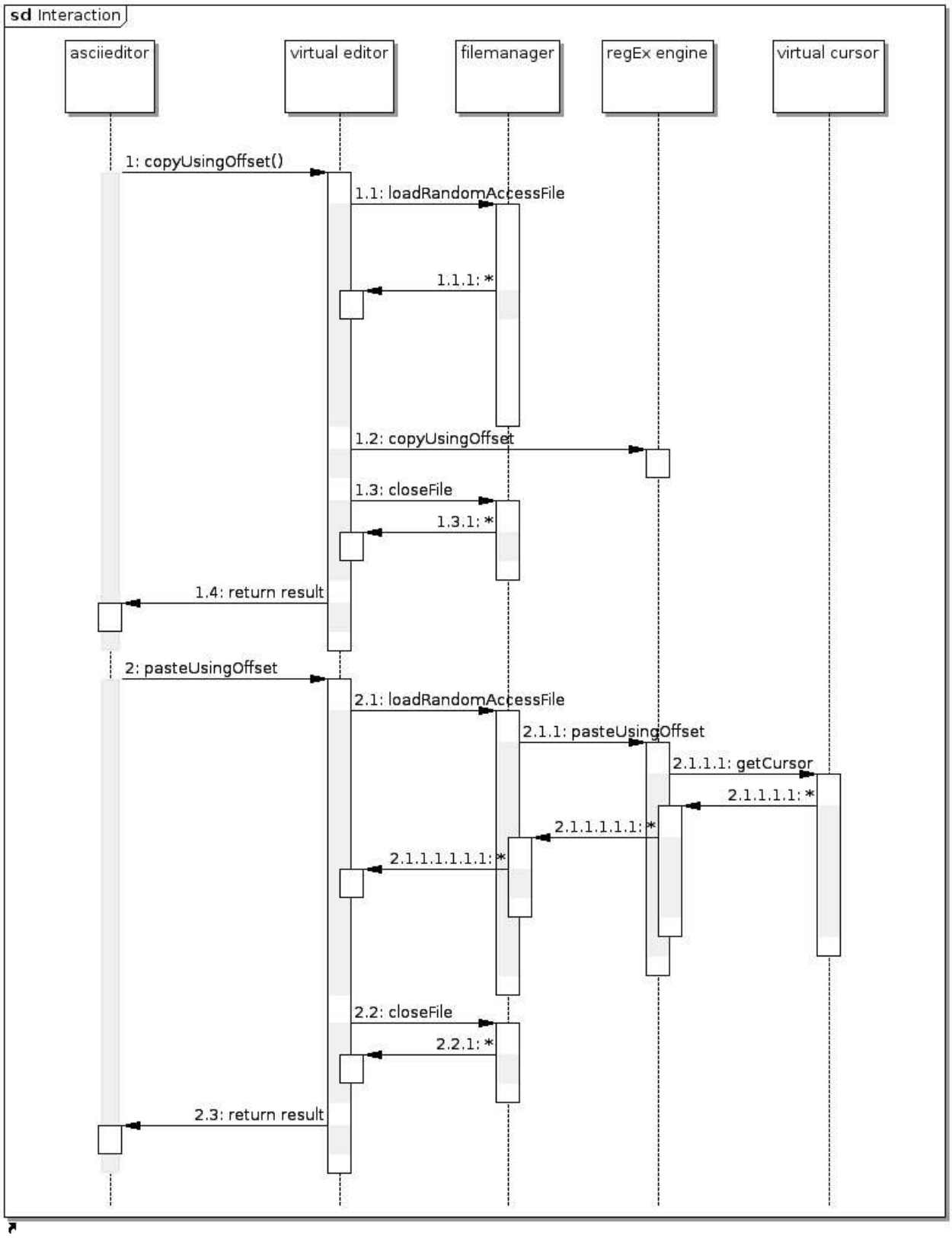

Abbildung 4.14: Sequenzdiagramm SIB Bibliothek zur Modifikation von ASCII-Dateien

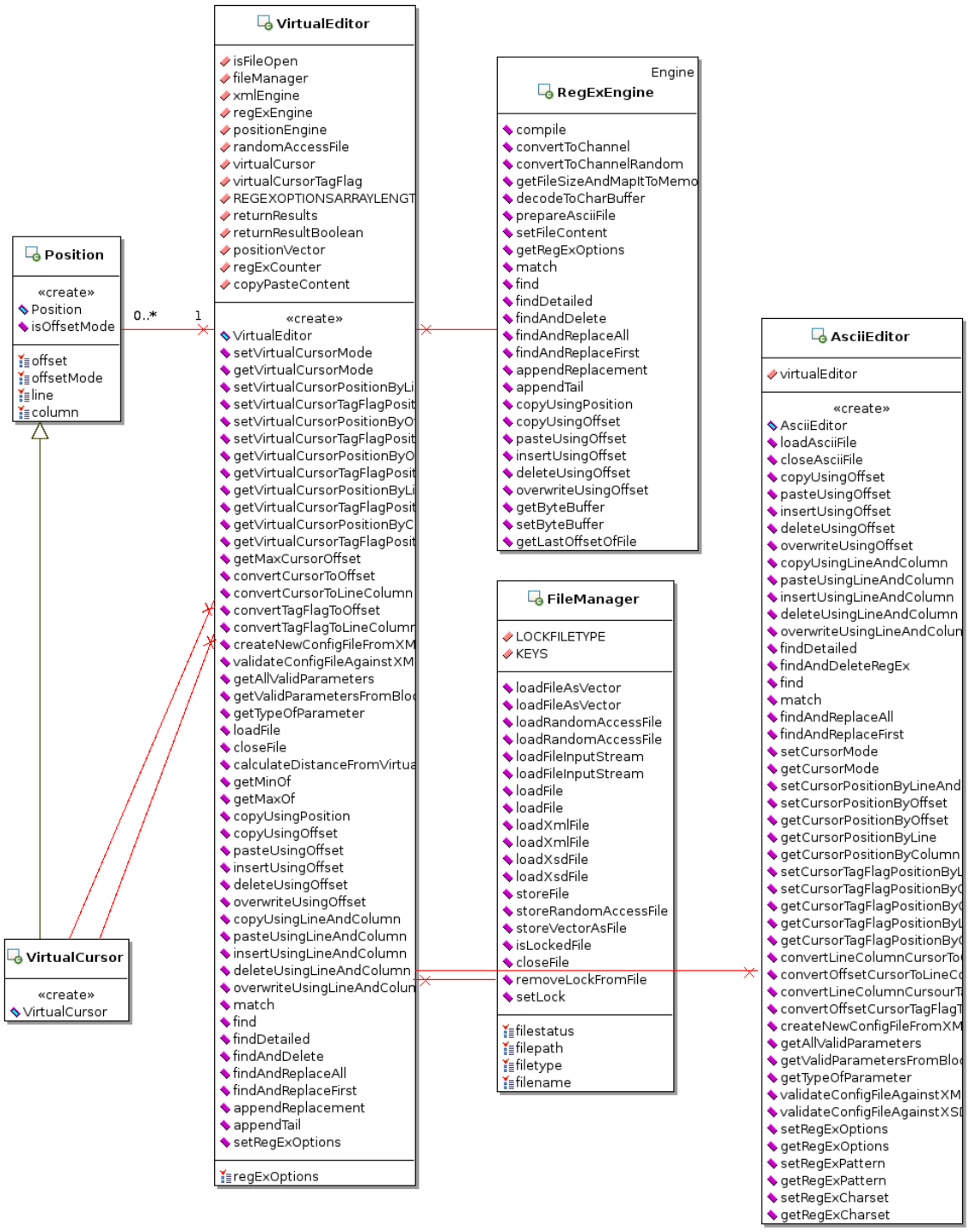

Abbildung 4.15: Klassendiagramm SIB Bibliothek zur Modifikation von ASCII-Dateien

Die RegEx Engine ist eine volle Implementierung vom java.util.regex. Alle bereitgestellten Operation der API sind implementiert.

Die XML-Engine, macht alles was mit der Validierung zu tun hat und kann auch eine neue Konfigurationsdatei (eine Art Template) erstellen.

Die Position-Engine vereinfacht die Arbeit mit den zu bearbeitenden Dateien. Hiermit können Offset-Position in Zeilen/Spalten-Positionen transferiert werden und vice versa. (s. Abb. 4.15)

4.3.3 Implementierung

Es gibt unterschiedliche SIBs die in Klassen unterteilt werden können

- **AsciiFile** - Beinhaltet Dateioperationen
  - AsciiFileClose
  - AsciiFileLoad
- **BuildAsciiEditor** - Starten des Asciieditors
  - BuildAsciiEditor
- **Cursor** - Bewegung des Cursor
  - CursorConvertLineAndColumnToOffset
  - CursorConvertTagFlagLineAndColumnToOffset
  - CursorConvertTagFlagOffsetToLineAndColumn
  - CursorGetMode
  - CursorGetPositionByLineAndColumn
  - CursorGetPositionByOffset
  - CursorGetTagFlagPositionByLineAndColumn
  - CursorGetTagFlagPositionByOffset
  - CursorSetMode
  - CursorSetPositionByLineAndColumn
  - CursorSetPositionByOffset
  - CursorSetTagFlagPositionByLineAndColumn
  - CursorSetTagFlagPositionByOffset
- **Modify** - Textoperationen
  - ModifyCopyUsingLineAndColumn
  - ModifyCopyUsingOffset

ModifyDeleteUsingLineAndColumn

ModifyDeleteUsingOffset

ModifyFind

ModifyFindAndDeleteRegEx

ModifyFindAndReplaceAll

ModifyFindAndReplaceFirst

ModifyFindDetailed

ModifyInsertUsingLineAndColumn

ModifyInsertUsingOffset

ModifyMatch

ModifyOverwriteUsingLineAndColumn

ModifyOverwriteUsingOffset

ModifyPasteUsingLineAndColumn

ModifyPasteUsingOffset

- **Option** - Einstellungen des Asciieditors

  OptionGetRegExCharset

  OptionGetRegExPattern

  OptionGetRegExProperties

  OptionSetRegExCharset

  OptionSetRegExPattern

  OptionSetRegExProperties

- **Util** - sonstiges

  UtilCreateNewConfigFileFromXML

  UtilGetAllValidParameters

  UtilGetTypeOfParameter

  UtilGetValidParametersFromBlock

  UtilValidateConfigFileAgainstXML

4.3.4 Beispiel: Copy&Paste

1. Wie immer braucht der jABC Graph einen Start SIB zum Anfang.
2. Um alle Operationen des Asciieditors nutzen zu können, ist es notwendig den BuildAsciiEditor SIB zu benutzen.
3. Anschließend muss eine Asciidatei geladen werden mit dem AsciiFileLoad SIB. In diesem Beispiel, erhält der AsciiFileLoad SIB die Inputparameter über den String-Value2CallContext SIB.
4. Im nächsten Schritt legen wir die Offset Position und die Tagflag Offset Position für den zu kopierenden Teil fest. In diesem Beispiel, erhält der CursorPositionByOffset SIB und CursorSetTagFlagByOffset SIB, ihre Input Parameter über den StringValue2CallContext SIB.
5. Der ModifyCopyUsingOffset SIB puffert den Teil zwischen der aktuellen Offset Position und Offset TagFlag Position.
6. Jetzt kann die Offset Position geändert werden mit dem CursorPositionByOffset SIB, um die Stelle festzulegen an die kopiert werden soll.
7. Als letztes kopiert der ModifyPasteUsingOffset SIB den Teil im Buffer an den aktuelle Offset Position.

4.4 SIB-Bibliothek zur serviceorientierten Erzeugung von Graphen und Diagrammen

4.4.1 Einleitung

Da es in der MaTRICS an vielen Stellen zu einer sehr beträchtlichen Menge an Daten kommt, die dem Benutzer innerhalb eines Webservices dargestellt werden soll, soll in diesem Teilprojekt eine Schnittstelle zu einer API geschaffen werden, mit der es möglich ist, Diagramme aus einer vorhandenen Datenmenge zeichnen zu lassen. Diese Diagramme sollen innerhalb der MaTRICS genutzt werden können, allerdings soll die Implementierung so modular geschehen, dass auch eine Nutzung in anderen, mit dem jEWIS modellierten Webservices möglich ist. Weiterhin sollen die Diagramme genau dann erzeugt werden, wenn sie benötigt werden, also wenn der Nutzer des Webservices ein Diagramm anfragt. Dies garantiert, dass die dargestellten Daten innerhalb der Diagramme immer aktuell sind.
Zum Zeichnen der Diagramme wurde die API JFreeChart [2] genutzt, die unter der GNU Lesser General Public Licence (LGPL) lizensiert ist. Diese API ermöglicht das Erstellen

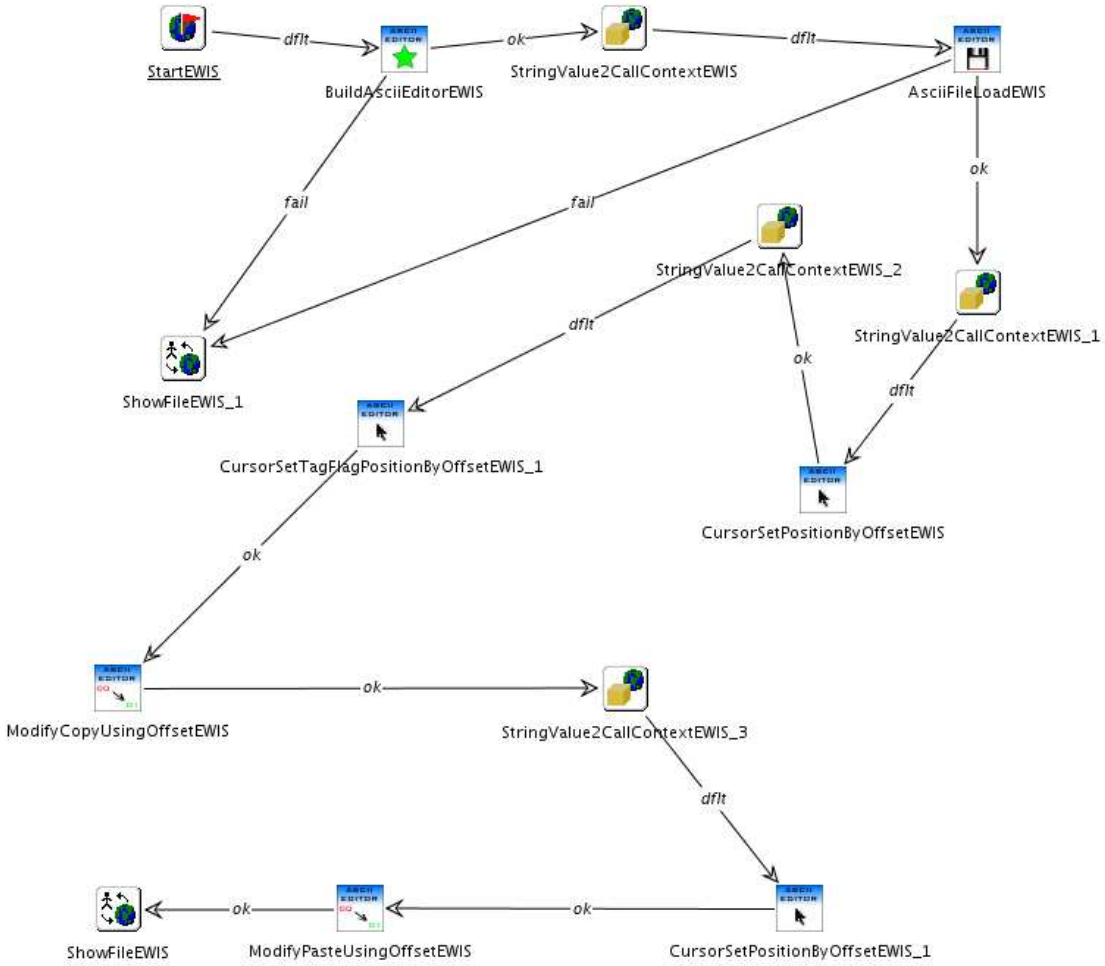

Abbildung 4.16: Beispiel jABC Graph Copy and Paste

von Diagrammen, die zunächst als Objekt existieren. So können erzeugte Diagramme weiter bearbeitet und konfiguriert werden. Es ist möglich, diverse Diagrammtypen erstellen zu lassen (Tortendiagramme, Balkendiagramme, „normale“ Graphen, ausgefallene Darstellungen und vieles mehr). Weiter lassen sich die erzeugten Diagramme bis ins kleinste Detail an die persönlichen Vorstellungen anpassen (Schriftart der im Diagramm enthaltenen Texte, verwendete Farben der Linien oder Flächen, Abstände, etc.). Wenn die Diagrammerstellung und Anpassung abgeschlossen ist, lässt sich das erzeugte Diagramm als jpg, png oder gif Datei auf der Festplatte abspeichern. Alternativ kann auch eine Darstellung für eine Java Oberfläche erzeugt werden, was für dieses Teilprojekt nicht interessant war.

Innerhalb dieses Projekt wurden folgende Interaktionsmöglichkeiten mit der API entwickelt, um ein Diagramm innerhalb eines Webservices zu erzeugen und darzustellen:

- Importieren von vorhanden Daten in das Diagramm
- manuelles Einfügen von Daten in ein Diagramm
- Erzeugen eines Diagramms
- Anpassungsmöglichkeiten für die angebotenen Diagrammtypen (Hier wurde eine Auswahl der durch die API möglichen Anwendungen umgesetzt)
- Speichern des Bildes auf dem Webserver zur späteren Darstellung auf einer Webseite

4.4.2 Design

Hier soll ein genauerer Blick auf die Verwendung der Schnittstelle und die Abläufe beim Zeichnen eines Diagramms gewagt werden. Aufgeteilt ist dieser Abschnitt durch die unterschiedlichen Arbeitschritte, die zum Zeichnen eines Diagrammes durchgeführt werden müssen. Dabei sollen die wesentlichen Konzepte kurz vorgestellt und der grobe Zusammenhang verdeutlich werden

4.4.2.1 GlobalDataContainer

Es wurde ein Container entwickelt, der während der Phase, in der die Daten für das Diagramm zusammengetragen werden, diese gruppiert abspeichert und nachdem das Diagramm erzeugt wurde, dieses speichert um weitere Einstellungsmöglichkeiten vorzunehmen. Der erste Schritt zum Zeichnen eines Diagrammes muss das Initialisieren des GlobalDataContainers durch das SIB **InitGlobalDataContainer** sein.

4.4.2.2 Serien

Die Daten der Diagramme werden bei der verwendeten API JFreeChart [2] in Kategorien eingeteilt, so genannte Serien. Diese Serien werden in den Diagrammen bei bestimmten Diagrammtypen (Balkendiagramme, Liniendiagramme, etc.) genutzt, um Daten die zur gleichen Kategorie gehören farblich gleich darzustellen. In der aktuellen Umsetzung ist die Zuordnung der Diagrammeinträge zu Kategorien wie folgt realisiert. Während der Initialisierung des **GlobalDataContainers** wird eine „default“ Serie angelegt. Solange keine neue Serie definiert wurde, ist diese „default“ Serie als aktive Serie markiert. Diagrammeinträge, die in den **GlobalDataContainer** gespeichert werden, werden immer der als aktiv gekennzeichneten Serie zugeordnet. Wenn nun eine neue Serie durch das SIB **InitSeries** initialisiert wird, hierbei muss ein neuer Name für die Serie angegeben, so wird diese Serie im **GlobalDataContainer** als aktiv gekennzeichnet und bleibt auch so lange aktiv, bis eine neue Serie im **GlobalDataContainer** abgelegt wird.

4.4.2.3 Datenimport und Datenmanipulation

Um Daten für das Diagramm in den **GlobalDataContainer** zu schreiben, sollte eines der beiden SIBs **ManualDataModification** oder **ImportDataEntries** verwendet werden. Mit können **ManualDataModification** Daten direkt eingegeben werden. **ImportDataEntries** bietet die Möglichkeit, eine vorhandene Datenstruktur in den Container zu übernehmen. Hier werden als Datenstrukturen Array, Liste und Hashmap angeboten. Die vorhandene Datenstruktur muss natürlich ein gewisses Muster vorweisen, welches von der Schnittstelle vorgegeben wird.

4.4.2.4 Diagramm erzeugen

Durch die Nutzung des SIBs **CreateDiagram** wird ein Diagramm erzeugt und im **GlobalDataContainer** gespeichert. Hier können schon einige Details zum Diagramm festgelegt werden, unter anderem, welcher Diagrammtyp gezeichnet werden soll. Dieses erzeugte Diagramm ist eine virtuelle Konstruktion, die ein weiteres „Feintuning“ der Diagramme ermöglicht.

4.4.2.5 Feintuning

Um das mit dem **CreateDiagramm**-SIB erzeuge Diagramm anzupassen, könnnen folgende SIBS genutzt werden:

- **ChangeGlobalDesign**
- **ChangePieChartDesign**
- **ChangeXYChartDesign**

- **ChangeCategoryChartDesign**

Das SIB **ChangeGlobalChartDesign** bietet die Möglichkeit, Eigenschaften anzupassen, die alle Diagrammtypen gemeinsam haben, die anderen 3 SIBs hingegen sind auf die Eigenschaften des entsprechenden Diagrammtypen spezialisiert.

4.4.2.6 Diagramme zeichnen

Die SIBs **DrawDiagram** und **DrawMultipleCharts2Image** dienen dazu, das virtuelle Diagramm als Bild auf der Festplatte abzulegen, damit es im Webservice in einer HTML-Seite angezeigt werden kann. Als zusätzliche Funktion wurde durch das SIB **DrawMultipleDiagrams2Image** die Möglichkeit implementiert, mehrere erzeugte Diagramme in ein Bild zu zeichnen. So hat man beispielsweise die Möglichkeit, die selben Daten auf unterschiedliche Weise darstellen zu lassen.

In den Abbildungen 4.17 und 4.18 soll durch ein Activitätsdiagramm der Ablauf zum Zeichnen eines Diagramms genauer dargestellt werden

4.4.3 Einsatz im jEWIS

Nachfolgend soll in Abbildung 4.19 ein kleines Beispiel für einen Service Logic Graph zur Erstellung eines Kuchendiagramms dargestellt werden. Wie zu sehen ist, ist die Anzahl der SIBs, die benötigt werden recht überschaubar, allerdings braucht man zum Zeichnen eines Diagramms mindestens vier SIBs der entwickelten Schnittstelle.

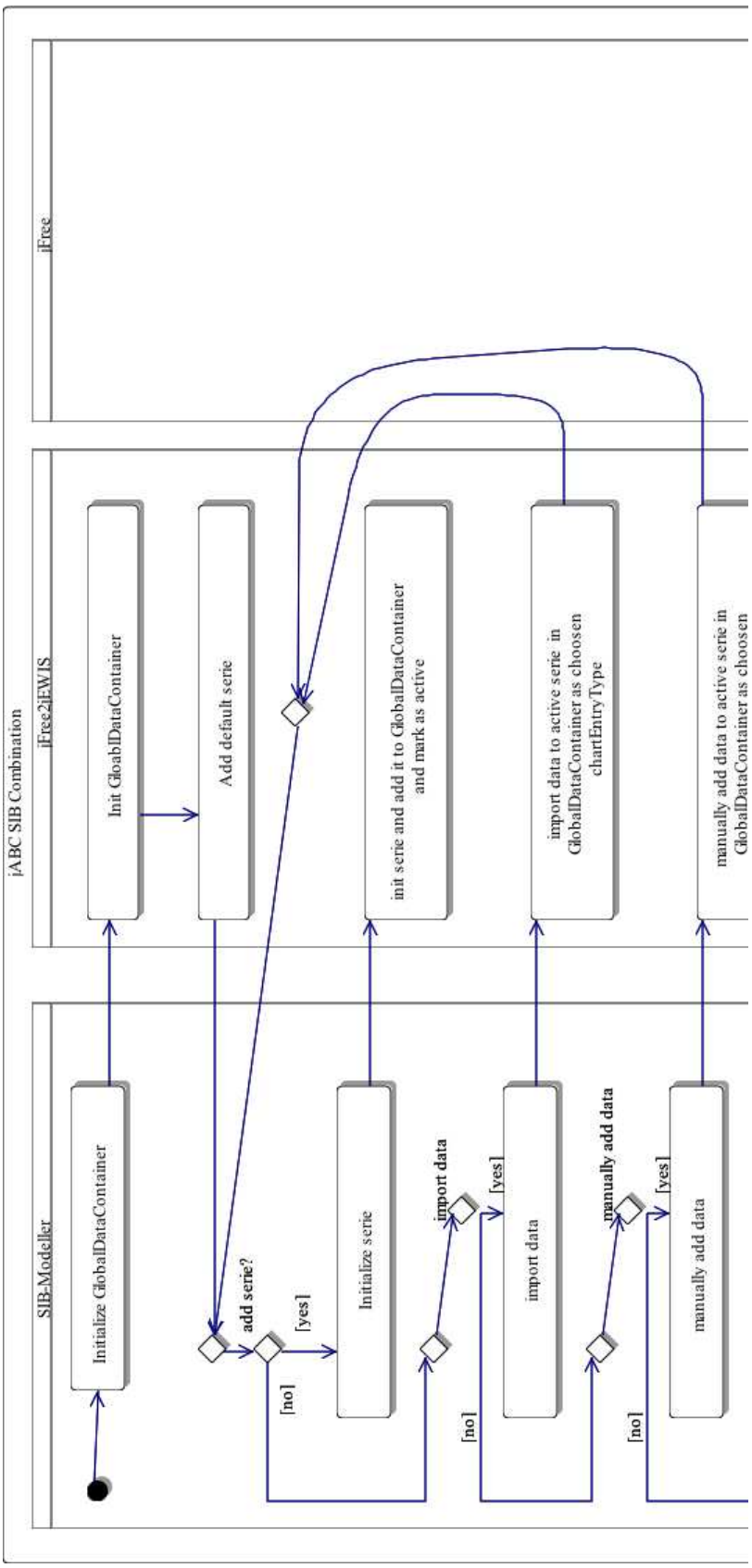

Abbildung 4.17: Aktivitätsdiagramm: Erzeugen eines Kuchendiagramms

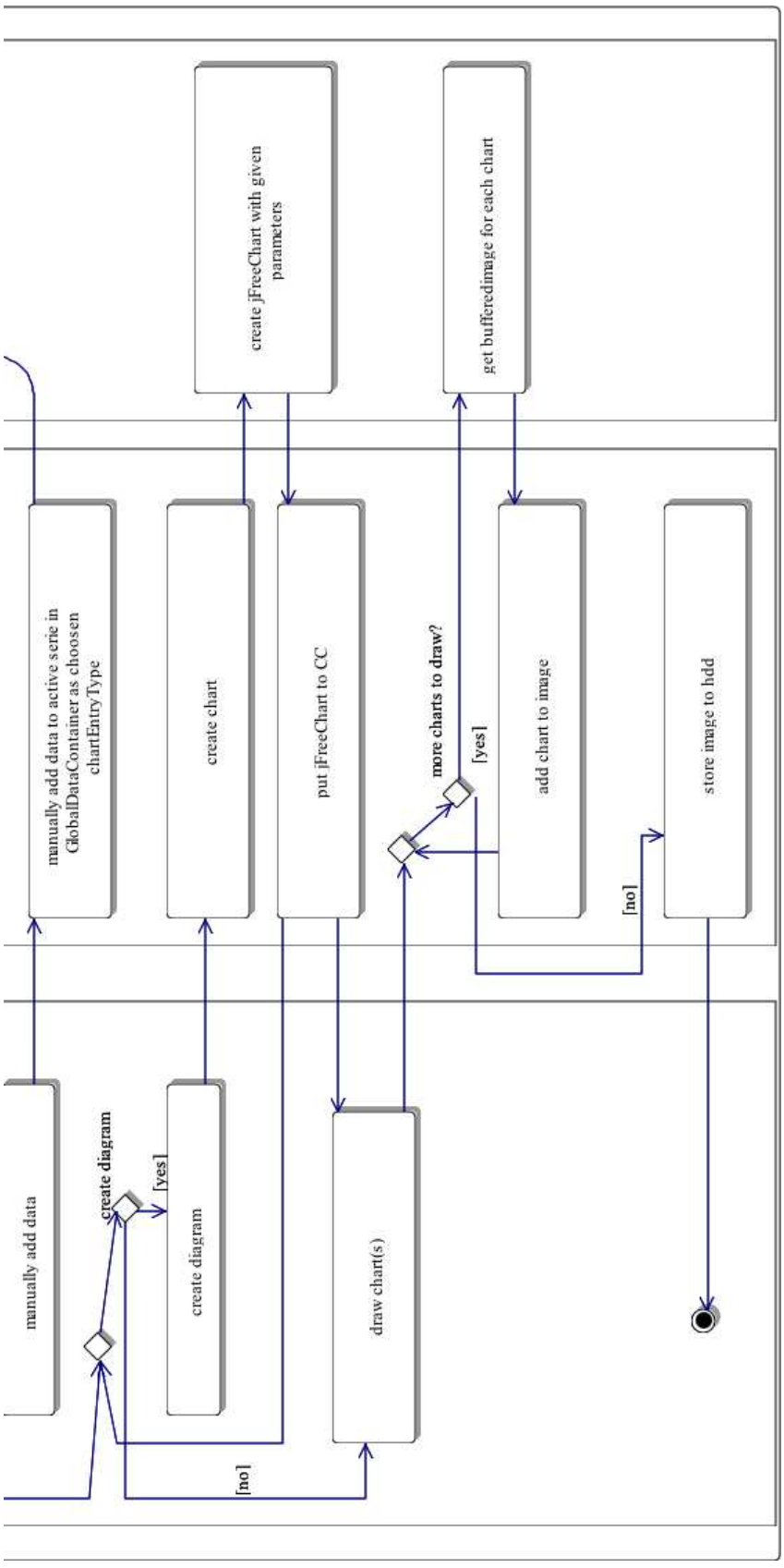

Abbildung 4.18: Aktivitätsdiagramm: Erzeugen eines Kuchendiagramms 2

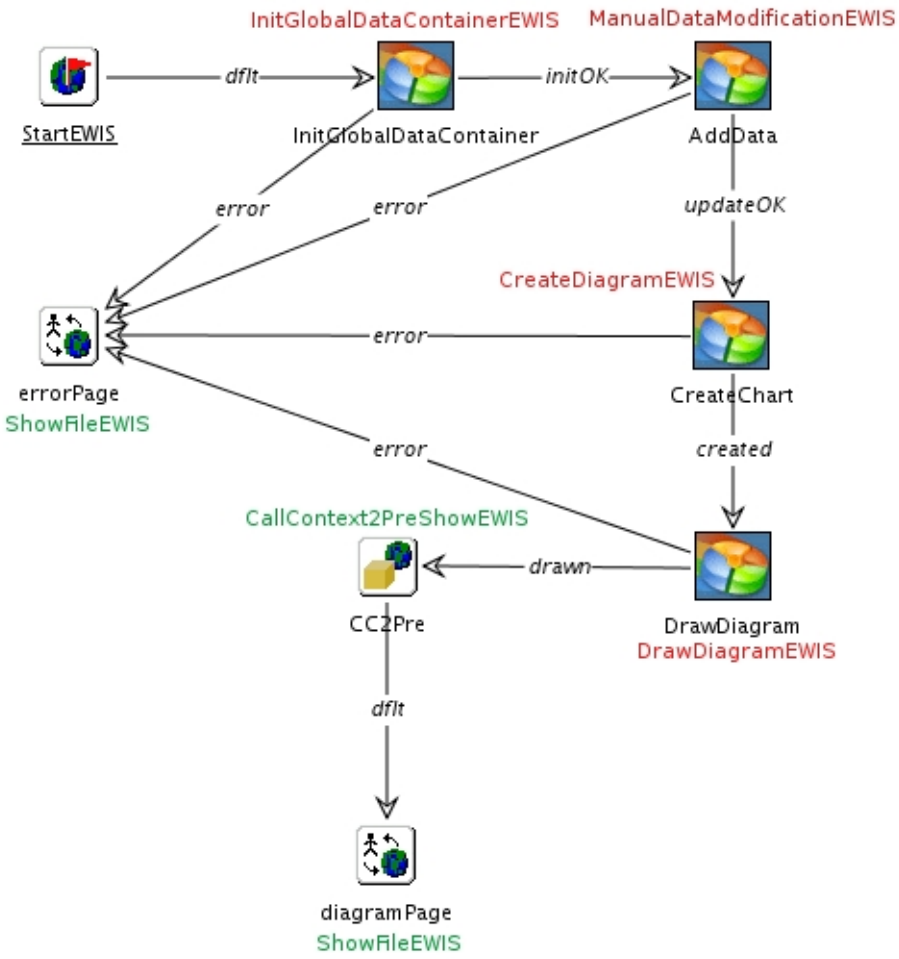

Abbildung 4.19: Service Logic Graph zum Zeichnen eines Kuchendiagramms

Kapitel 5

Gruppenarbeit

5.1 Thema 1: Entwicklung eines WSDL-Adapters zur Orchestrierung von Web-Services

WSDL ist eine Beschreibungssprache für Web-Services. Dabei ist die Beschreibung eines Web Services unabhängig von der Implementierungssprache der Applikation und beschreibt zum einen das Interface des Services und zum anderen die Bindung dieses Interfaces an ein konkretes Protokoll. Eine Ansammlung von statischen HTML-Seiten, die untereinander verlinkt sind, lassen sich bereits mit Hilfe von WSDL spezifizieren. Das Input-Verhalten wird dabei durch Hyperlinks, oder Formulare definiert. Als Output kommen in diesem Fall immer nur HTML-Ausgaben in Frage.

Mit Hilfe solcher Beschreibungen lassen sich bereits verschiedene Web-Services miteinander verbinden. Dabei soll der Zugriff auf einen Web-Service über einen speziellen Adapter erfolgen. Die Umsetzung dieses Adapters stellt die Hauptaufgabe dieses Themas dar. Die Orchestrierung von Web-Services erfolgt auf ServiceLogic-Ebene des MaTRICS-Frameworks, wobei die Kommunikation mit den orchestrierten Web-Services über diesen Adapter erfolgt. Auf Orchestrierungsebene werden dann die Assoziationen der zu übertragenden Daten mit den Input-Variablen eines Web-Services (hier eine HTML-Seite) hergestellt. Somit muss für jede HTML-Seite eine WSDL-Spezifikation existieren, woraus der zu entwickelnde Adapter die Information zum Erzeugen eines Requests gewinnt. Ein Response kann ebenfalls mit Hilfe der WSDL-Spezifikation (Output) interpretiert werden. Beispielsweise können so beliebige Ausgaben einer HTML-Seite in Variablen gekapselt werden, die dann wiederum auf ServiceLogic-Ebene verarbeitet werden können.

Das Design wurde unter Zuhilfenahme von UML erstellt. Dabei ist zu überlegen, wie sich WSDL-Spezifikationen für beliebige HTML-Seiten auf einfache Weise erstellen lassen. Für die Verarbeitung von WSDL können beliebige (open-source) Frameworks verwendet werden.Als Abschluss wurde eine englische Dokumentation erstellt, die das Design des Adapters und die Integration in MaTRICS beschreibt.

5.1.1 Konzept bei der Entwicklung des WSDL-Adapters

Das Endprodukt besteht hauptsächlich aus drei Teile: WSDL-Generator (Kapitel 5.1.2.1), WSDL-Parser (Kapitel 5.1.2.2) und als letztes die Ausführung von Web-Service-Applikationen (Kapitel 5.1.2.3).

5.1.2 Design des WSDL-Adapters

5.1.2.1 WSDL-Generator

Die Hauptfunktion des Generators ist das Estellen von validen WSDL-Dateien über bereits existierende Web-Services zur Entwurfszeit. Dieser basiert auf dem Open-Source-Tool Mozswing[9], welches das Starten von Web-Browser-Instanzen erlaubt. Auf dieser Funktionalität basieren einige Add-Ons, die entwickelt wurden, um die WSDL-Generierungsfunktion zu unterstützen.
Ein *event listener* wurde entwickelt, damit eine WSDL-Datei während der Mozswing-Web-Session erstellt werden kann. Somit kann ein Modellierer/Entwickler viele Elemente einer Webseite wie Form oder Link als Operationen in der WSDL-Datei definieren. Nach der Modellierungsphase wird die WSDL-Datei in dem Dateisystem gespeichert.

Im folgenden Diagramm (Abbildung 5.1) werden alle Klassen aufgelistet, die zu dem Open-Source-Tool Mozswing gehören. Um die Anforderungen der Aufgabe und des Entwicklerteams zu erfüllen wurde die Klasse *MozwingPopupCreator* entsprechend erweitert.

5.1.2.2 Parsen und Erstellen von WSDL-Methoden-Objekten

Der WSDL-Parser analysiert und validiert eine ihm übergebene Datei nach bestimmten Vorgaben. Details bezüglich der Realisierung sind aus der Projektdokumentation [11] zu entnehmen.

Das Klassendiagramm (Abbildung 5.2) stellt die zusätzlich implementierten Klassen dar, die den WSDLAdapter ausmachen.

Die WSDL-Klasse parst bei Instanziierung in ihrem Konstruktor eine übergebene WSDL-Datei. Sie extrahiert die darin enthaltenen Angaben über die Erreichbarkeit sowie die Methoden des Web-Services in mehreren Schritten.
Während die WSDL-Klasse generelle Informationen über den Web-Service direkt über entsprechende get-Methoden zur Verfügung stellt, werden die Methoden des Web-Service mit Name, HTTP-Methode und Parameterliste seperat in Objekte der Klasse *WSDLMethod* gekapselt und als Vektor verwaltet. Hier wird auch mit Hilfe der Aktion *XPath Angeben* festgelegt, welche Teile einer Antwort des Web-Services für den Nutzer relevant sind. Diese Erweiterung ist nur verfügbar im Zusammenspiel mit WSDL-Dateien, die mit dem oben beschriebenen WSDL-Generator erzeugt worden sind.

Das Sequenzdiagramm (Kapitel 5.3) verdeutlicht den genauen Ablauf des Parsingvorgangs mit der Extraktion der einzelnen Knoten und dem anschließenden Ablegen in die entsprechenden Vektoren. Bis auf Knoten vom Typ <*types*> müssen alle vorgesehenen Knoten in der WSDL-Datei enthalten sein, sonst bricht der Parser mit einer Exception ab.

Im weiteren Verlauf werden die Objekte der WSDL-Klasse von der HTMLUnit-Klasse genutzt, um eine Anbindung zu dem repräsentierten Web-Service aufzubauen und seine Methoden anzustoßen.

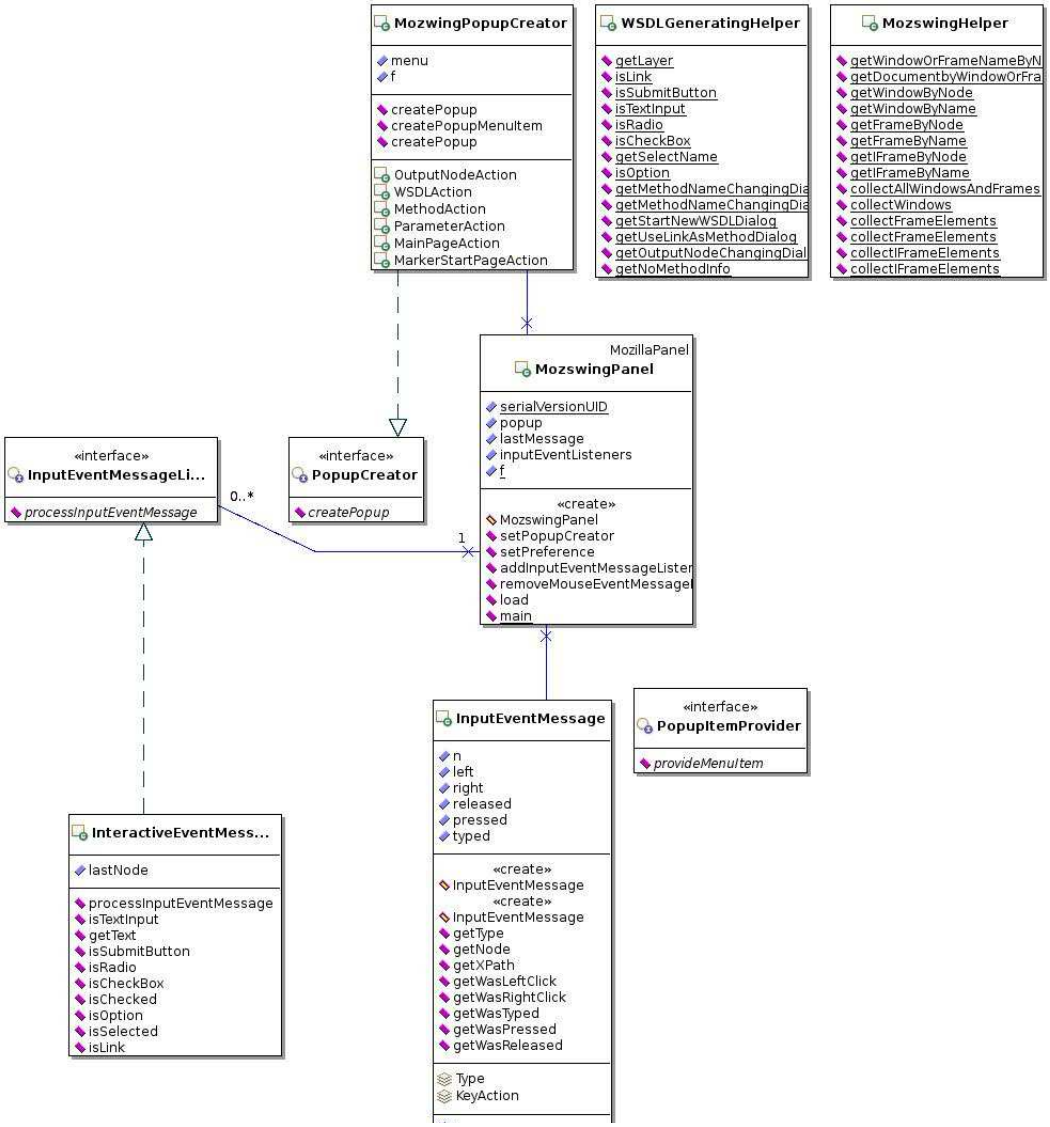

Abbildung 5.1: Klassendiagramm des erweiterten *MozwingPopupCreators*

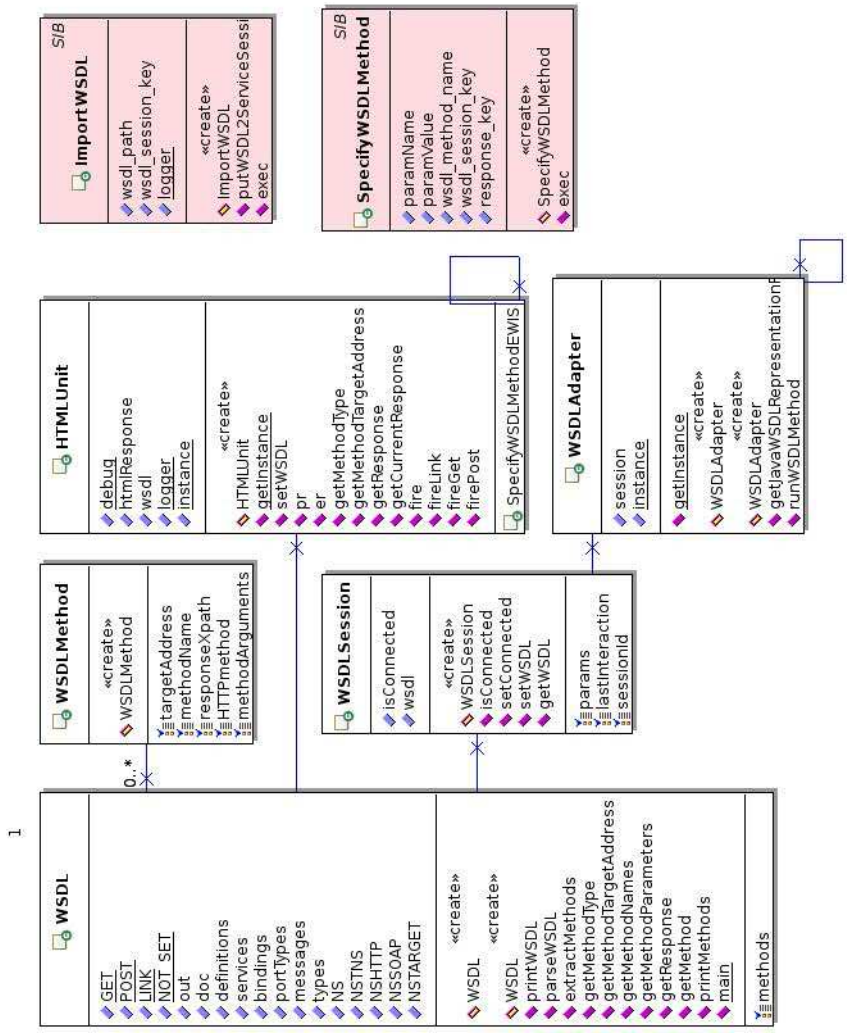

Abbildung 5.2: Klassendiagramm des WSDL-Generators und der HTMLUnit

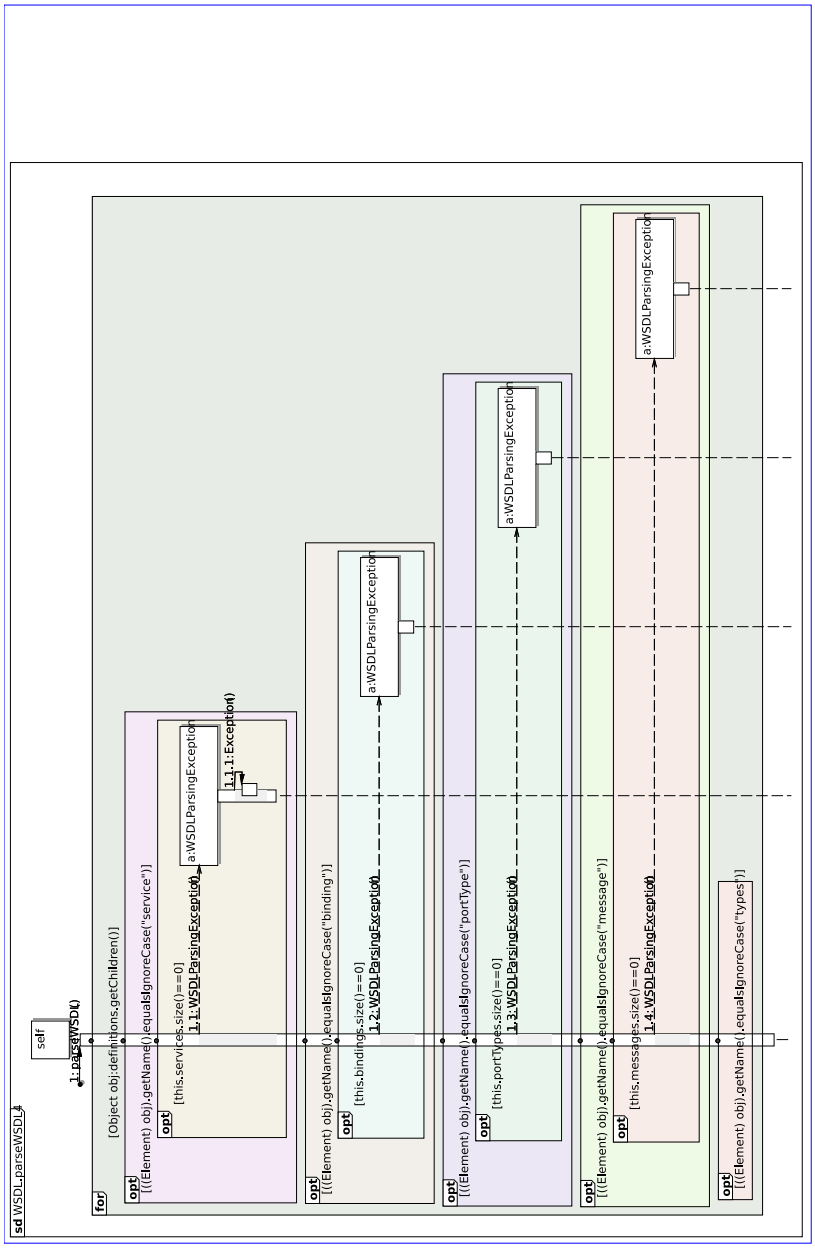

Abbildung 5.3: Sequenzdiagramm der Methode parseWSDL()

Die WSDL-Klasse interpretiert *HTTP GET*- und *HTTP POST*-Methoden. Eine mögliche Interaktion mit SOAP wurde ausgeschlossen, da im Rahmen der gestellten Aufgabe eine Verbindung zwischen dem jABC und anderen nicht-SOAP-basierten Webservices erstellt werden musste.

5.1.2.3 Ausführen und Request/Response-Handling zur Laufzeit via HTMLUnit

In diesem Teilprojekt wurde die Webservice-Anbindung realisiert und die *Requests* und *Responses* ausgewertet. Dazu wurde das Open-Source-Tool HTMLUnit[8] verwendet.

HTMLUnit ist ein frei verfügbarer, in Java implementierter Web-Browser, der ohne graphische Bedienungsoberfläche realisiert ist und über seine API das Ansteuern von Webseiten ermöglicht.
Besonders interessant ist die Anwendung durch seine Fülle an komfortablen Möglichkeiten, das Verhalten eines Benutzers mit einem normalen Web-Browser auf einer Webseite zu simulieren; sprich das Ausfüllen von Formularen, das Anklicken von Hyperlinks oder auch die Interaktion mit JavaScript- und Ajax-Komponenten. Anwendung findet HTMLUnit vor allem beim automatisierten Testen von Web-Seiten.

Die im WSDL-Adapter realisierte HTMLUnit-Klasse arbeitet mit der HTMLUnit-Version, die unter [8] zum Download bereitsteht.
Die Klasse benutzt die HTMLUnit-API um auf Grundlage der in den WSDL-Objekten gespeicherten Informationen *Requests* zum Web-Service abzusetzen und die darauf folgenden Antworten auszuwerten.
Die Unterscheidung zwischen *GET*- und *POST*-Methoden, das einfache Folgen von Hyperlinks und die Verwaltung der Methoden-Parameter wird von der Klasse übernommen. Wird mit einer vom WSDL-Generator erzeugten WSDL gearbeitet, werden am Ende nicht nur die Antworten bereitgestellt, sondern auch entsprechend der XPath-Angaben auf die relevanten Elemente beschränkt.

5.2 Übersicht über die SIB-Bibliothek

Für das Einbinden der WSDL-Adapter Funktionalität in ein jABC Projekt stehen zwei jABC-SIBs zur Verfügung. Zusätzlich wurde ein weiteres SIB zum Parsen von Antworten der Mantis-Bugtracker-Applikation entwickelt. Dieses wird im weiteren Projektverlauf eine Rolle für den Einsatz des Adapters spielen.

ImportWSDLEWIS Das SIB wird zum Einlesen einer WSDL Datei benutzt. Es ist zu beachten, dass nur absolute Pfadangaben oder File-Objekte aus dem CallContext (Eingabe mit einem führeden *cc:*) akzeptiert werden. Im Anschluss an das Einlesen kann das SpecifyWSDLMethod-SIB benuzt werden um die verfügbaren Methoden anzustossen.

SpecifyWSDLMethodEWIS stösst im Anschluss an ImportWSDL eine ausgewählte WSDL-Methode an.

ParseMantisResponseEWIS ist speziel für die Verarbeitung einer Antwort des Matis-Bugtrackers entwickelt.

5.3 Thema 2: Integration einer Notification Komponente auf Basis von JAIN-SIP und Asterisk-Java

Innerhalb der MaTRICS werden alle Benachrichtigungen mit Hilfe des Notification-Management verarbeitet. Dabei lässt sich für jeden Konfigurationsdienst und für jeden ConfigClient der Empfänger einstellen. Zudem können Nachrichten anhand ihrer Güte (Critical, Error, Notify, Debug, etc.) und Dringlichkeit verschieden verarbeitet werden. Beim Design des NotificationManagements wurde besonderen Wert auf Modularität und Erweiterbarkeit gesetzt. Zur Zeit lassen sich somit Nachrichten via SMS und EMail verschicken.

Im Rahmen dieser Aufgabe soll das NotificationManagement erweitert werden, so dass auch Notifications in Form von Sprachnachrichten, die zuvor aufgenommen wurden, via Telefon getätigt werden können. Die Kommunikation erfolgt dabei auf Basis des SIP-Protokolls. Nachrichten können zum Einen mit Hilfe der JAIN-SIP API erstellt und an einen SIP-Provider geschickt werden, der dann die eigentliche Kommunikation mit dem Endgerät regelt. Zum Anderen lassen sich Nachrichten auch direkt von einer PBX, wie Asterisk erstellen und weiterleiten. Dazu soll die Asterisk-Java API verwendet werden.

Innerhalb der MaTRICS wurde ein Dienst entwickelt, mit dem das Management von Sprachnachrichten ermöglicht wird. Beispielsweise können bereits zuvor aufgenommene Audio- Dateien für bestimmte Empfänger aneinander gehangen werden, so dass dadurch ein anderer Ansagetext entsteht.

Das Design wurde unter Zuhilfenahme von UML erstellt. Dabei ist auch zu überlegen, wie sich Interaktionen eines Benutzers (z.B. durch Drücken einer Taste auf dem Telefon) in die Notification-Komponente integrieren lassen. Als Abschluss wurde eine englische Dokumentation erstellt, die das Design bzw. die Erweiterung des NotificationManagements beschreibt.

5.3.1 Konzept der Integration einer Notification-Komponente auf Basis von JAIN-SIP und Asterisk-Java

Zur Realisierung wurde diese Aufgabe in zwei Teilprojekte unterteilt.
Das TemplateManagement (Kapitel 5.3.2) ist eine Erweiterung der MaTRICS-Notification-Komponente und ermöglicht das Versenden von benutzerdefinierten Benachrichtigungen.

Das andere Teilprojekt ist die Integration der VoIP-Technologien (Asterisk (Kapitel 5.3.3.1) und Jain-SIP (Kapitel 5.3.3.2)) in das Notification-Management der MaTRICS.

5.3.2 Template-Management

Die Notification-Komponente der MaTRICS unterstützte bisher nur textbasierte Benachrichtigungen wie Email, SMS, ICQ usw.
Bei einer VoIP-Benachrichtigung reicht es nicht immer aus nur eine Textnachricht zu generieren und diese über einen Kommunikationskanal zu versenden. Eventuell sollen bei bestimmten Benachrichtigungstypen zuvor generierte Audiodateien abgespielt werden. Um dieses und andere denkbare Szenarien (z.B. Benutzerinteraktion) abzudecken, wurde die Notification-Komponente um das Template-Management erweitert.

5.3.2.1 Design und Integration

Das Template-Management soll einfache Aktionen zur Verwaltung von Templates unterstützen.
Ein *Template* besteht aus mindestens einem *Announcement* und aus beliebig vielen Paaren von *Options* und *Targets*. Das *Announcement* beinhaltet die eigentliche Nachricht, die je nach Kommunikationskanal als Text versendet oder anderweitig verarbeitet wird. Das *Option*-Objekt beinhaltet eine mögliche Aktion (z.B. ein DTMF-Symbol), mit der der Empfänger auf die Benachrichtigung reagieren kann. Das *Target* enthält dann die Ziel-URL zu dem Dienst, der diese Aktion ausführt.

Die Abbildung 5.4 gibt einen Überblick über das Template-Management.

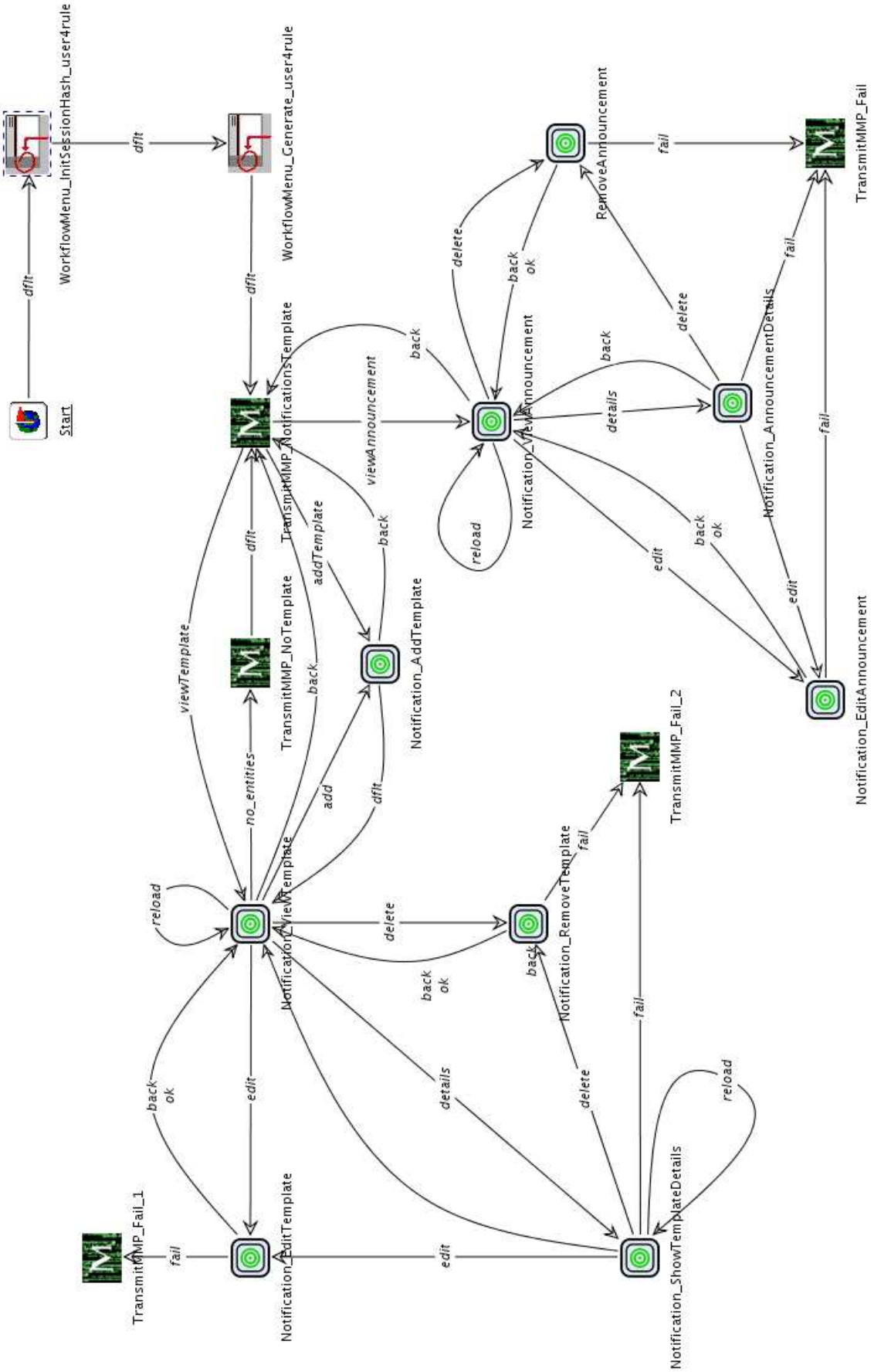

Abbildung 5.4: Template-Management - Das jABC-Modell

5.3.2.2 Persistenz

Zur persistenten Datenhaltung wurden fünf Business-Klassen implementiert, deren Zusammenhang im Diagramm (Abbildung 5.5) dargestellt werden.

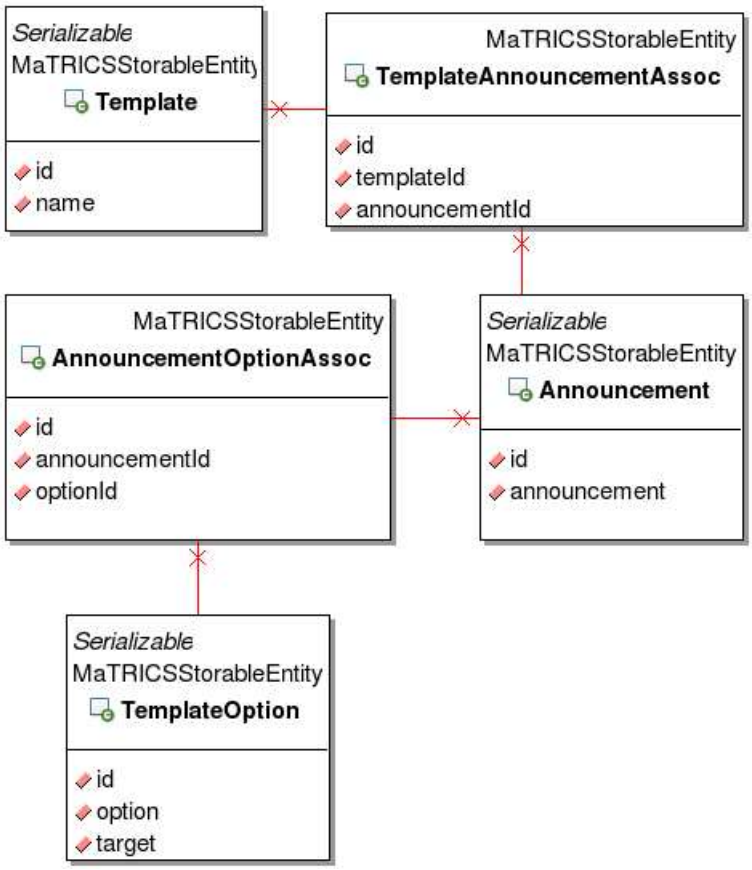

Abbildung 5.5: Template-Management: Datenbankschema

Zu jedem dieser Business-Klassen wurde ein *SmartAdministrator* geschrieben. Dieser ist für das Speichern, Laden und Ändern der jeweiligen Objekte zuständig. Der Zugriff auf die Administratoren erfolgt über die Klasse *TemplateManager*.

5.3.3 Integration von Asterisk und Jain-SIP in das Notification-Management der MaTRICS

Um Notifications zu versenden, müssen diese zunächst in die *NotificationQueue* eingefügt werden, was der versendene Service selbst bewerkstelligt. Mit Hilfe der Notifi*cationEngine* wird unterschieden, welche verschiedenen Kontakttypen die Abonnenten der Notifications benutzen. Anschließend werden die Notifications in die entsprechenden Queues(z.B. *EMailQueue* für Email, *PhoneQueue* für Sprachnachrichten, *SMSQueue* für SMS) eingeordnet.
Um die Übertragung von Sprachnachrichten zu ermöglichen wird eine zentrale Stelle der Integration in das bestehende NotificationManagement benötigt, die *NotificationPhone-Engine*. Diese ist ein Thread, der ständig (nach Ablauf eines vordefinierten Zeitintervalls) ausgeführt wird.
Hier werden die Notification-Objekte aus der *PhoneQueue* geholt, die vorher von der *NotificationEngine* in die Queue eingefügt wurden. Anschließend wird ein Anruf zu der in dem Notification-Objekt enthaltenen Kontakt-URL aufgebaut und die darin enthaltene Nachricht übertragen.
Wurde beim Abonnieren ein Template angegeben, muss die Nachricht vor der Übertragung dem Template entsprechend zusammengesetzt werden. Vor der Abarbeitung der Queue wird ermittelt, ob die VoIP-Kommunikation mittels Asterisk oder Jain-SIP erfolgen soll.

5.3.3.1 Asterisk in der NotificationPhoneEngine

Nachdem eine Nachricht aus der Queue geholt wurde, wird mit Hilfe des Makros *CallUser* eine Telefonverbindung aufgebaut. Anschließend wird der *Asterisk* aufgefordert, aus der Nachricht eine Audiodatei zu erzeugen und diese an den Empfänger zu übertragen.

Wurde die Notification mit einem Template abonniert, so kann der Empfänger über ein DTMF-Signal die Ausführung eines Dientes anstoßen. Sinnvoll wäre eine Beschreibung über die Belegung der DTMFs innerhalb der Nachricht.

Die Abbildung (5.6) zeigt den Schleifenrumpf, der die Benachrichtigungen aus der *PhoneQueue* verarbeitet.

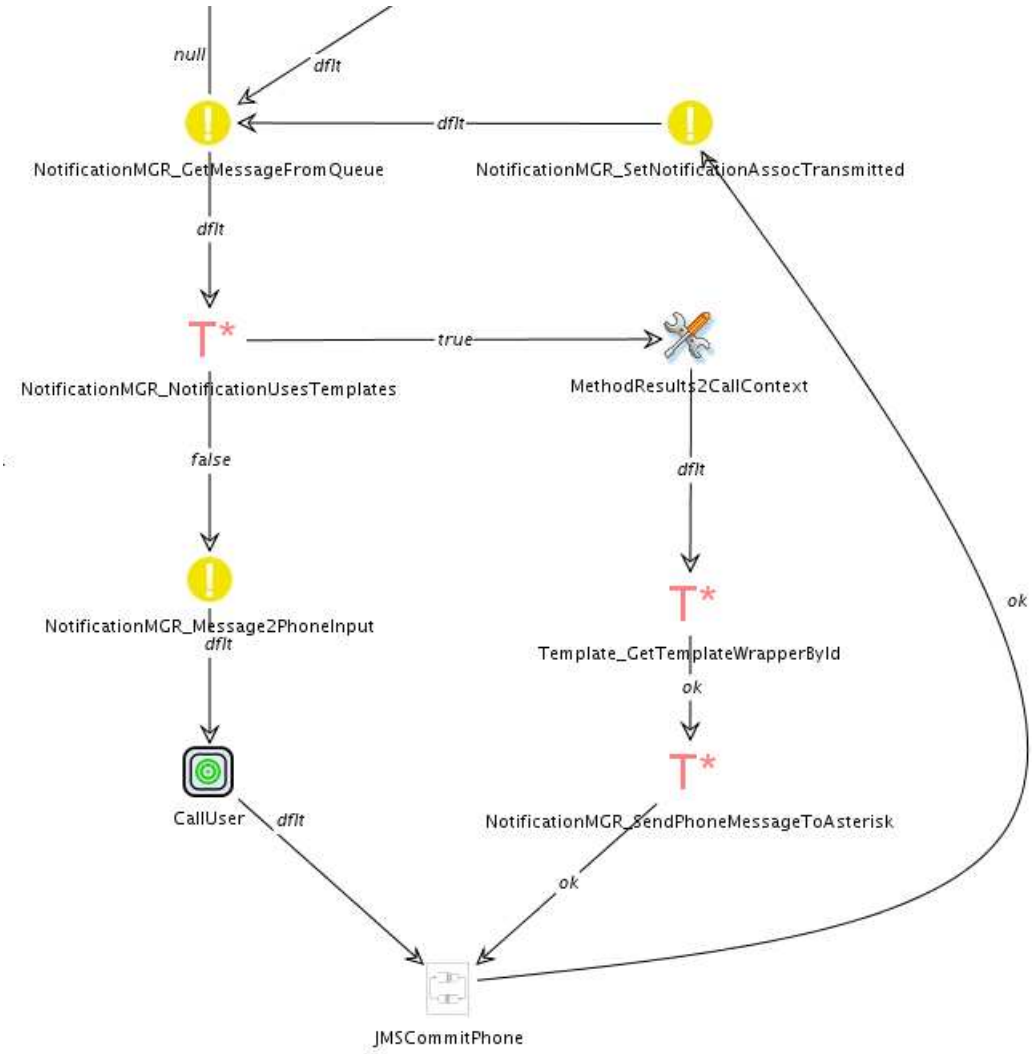

Abbildung 5.6: Asterisk in der NotificationPhoneEngine

5.3.3.2 Jain-SIP in der NotificationPhoneEngine

Soll eine Notification mit Hilfe der Jain-Sip API verschickt werden, so wird sich zunächst bei einem SIP-Provider registriert. Anschliessend wird eine Nachricht aus der Queue geholt. Es wird getestet, ob die Notification mit einem Template abonniert wurde. Fällt dieser Test negativ aus, so wird in dem Makro *Jainsip callUser* mit Hilfe der Konsolenbefehle Festival und Sox aus der Nachricht eine Audiodatei erzeugt, danach eine Telefonverbindung aufgebaut und die Nachricht an den Empfänger übertragen. Wurde ein Template genutzt, so werden zunächst nacheinander die Nachrichten in Audiodateien umgewandelt und in einem Vektor gespeichert. Nachfolgend wird dieser Vektor in dem Makro *SendMessagesAndHangup* verarbeitet, eine Verbindung etabliert und mit der Übermittlung begonnen. Nachdem die Nachrichten vollständig übermittelt wurde, wird die Verbindung unterbro-

chen. Das Abspielen existierender Audiodateien sowie die Interaktion mit Benutzern wurde nicht realisiert.

Die Abbildung (5.7) veranschaulicht die obigen Ausführungen der Nachrichtverarbeitung.

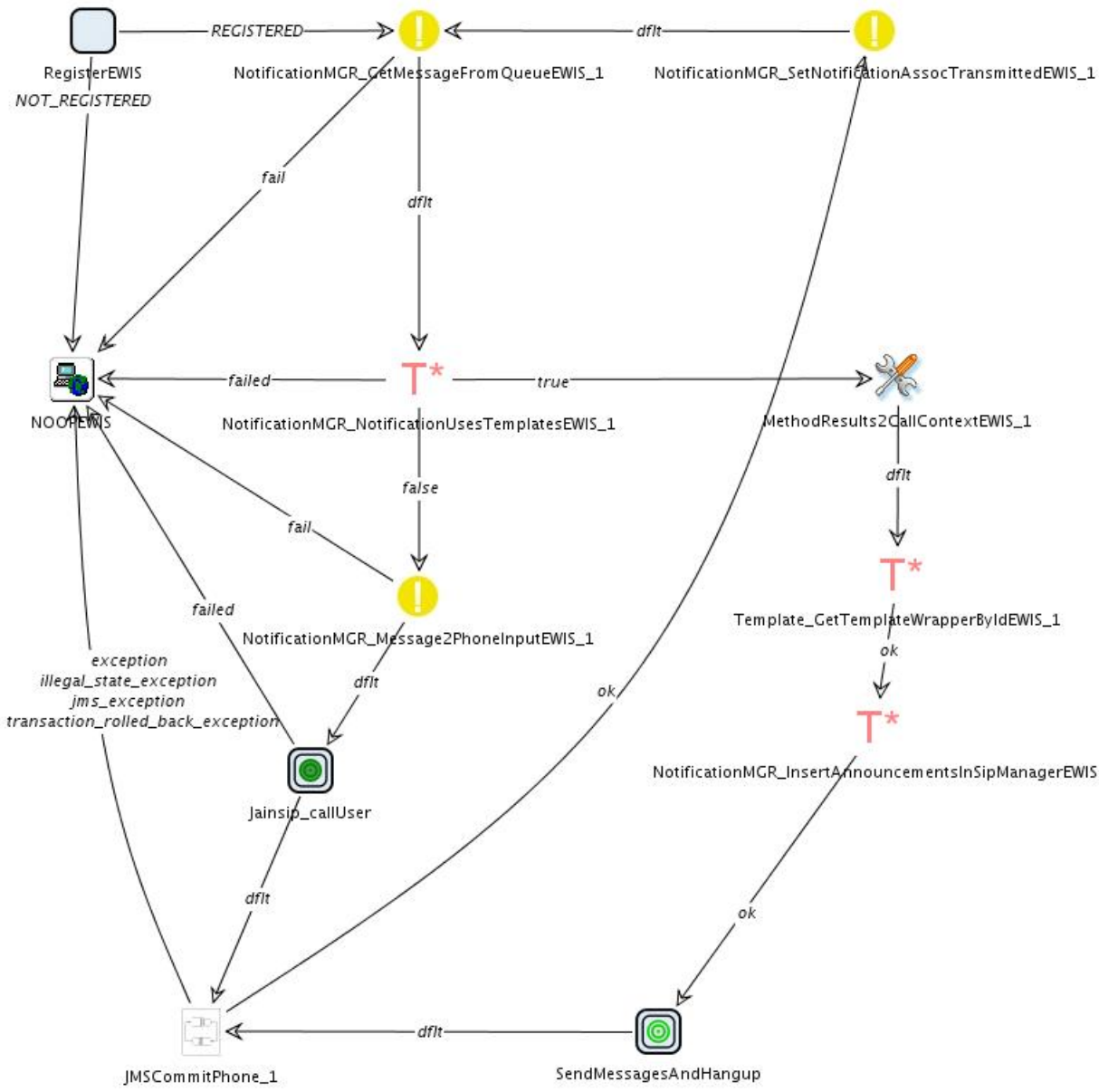

Abbildung 5.7: Jain-Sip in der NotificationPhoneEngine

Kapitel 6

Hauptthema: Entwicklung eines Konfigurationsdienstes für Web-Portale

6.1 Hauptthema: Aufgabenbeschreibung

Das Unternehmen LS Five Programming Systems möchte seinen Kunden in Zukunft ein selbst konfigurierbares Web-Portal zur Verfügung stellen. Im ersten Release, welches am 18.07.2008 online gestellt wird, sollen zunächst nur die Grundfunktionalitäten für das Einrichten und Verwalten der Webspaces der Kunden für die Operateure von LS Five Programming Systems enthalten sein. Ferner soll gelten:

- Jeder Kunde hat eine eigene Portnr für sein Web-Portal in der Form http:milkyway.cs.uni-dortmund.de:PORTNR
- Jeder Kunde kann selbstständig mit seinem Account die Inhalte via FTP auf sein Portal hochladen, dabei hat er nur Zugriff auf seine eigenen Daten.
- Die Operateure von LS Five Programming Systems können Web-Portale anlegen, löschen und sperren. Des weiteren können sie auch Kunden-Accounts verwalten.
- Für jeden Kunden gib es einen Datensatz, der die Email-Adresse und Telefonnr. beinhaltet.
- Kunden sollen bei Problemen oder Störungen via Email und Telefon informiert werden.
- Anbindung an das Bugtracking-Ticket- System Mantis zur Verarbeitung von Fehlern und Änderungswünschen.

Der technische Leiter von LS Five Programming Systems betonte, dass die gesamte Konfiguration über das Configuration-Framework MaTRICS abgewickelt werden soll, da dies bereits alle notwendigen Adapter zu den Fremdsystemen enthält und über eine ausgeklügelte JobFlow-Engine verfügt. Zudem können die Kundenstammdaten in der MaTRICS eigenen Datenbank gespeichert werden.

Konfigurationsaufträge werden in MaTRICS als Tasks bezeichnet, die wiederum aus kleineren Einheiten, den Jobs, bestehen. Ein Task ist erfolgreich abgeschlossen, wenn alle seine Jobs erfolgreich beendet wurden. Falls die Ausführung eines Jobs scheitert wird der gesamte Task als fehlerhaft gekennzeichnet und alle bereits durchgeführten Modifikationen werden rückgängig gemacht (Rollback).

Als Serversoftware soll für die Kundenportale der Apache Webserver 1.3.x eingesetzt werden. Der Dateiupload der Kundendaten erfolgt für die Kunden via FTP. Dazu wird als FTP-Server der ProFTPd eingesetzt.

LS Five Programming Systems setzt für seine zahlreichen Software-Projekte das zentral installierte Bugtracker-System Mantis ein, in dem auch Kunden Probleme und Anregungen zu entsprechenden Projekten einpflegen können. Für den Web-Portal Service wurde im Bugtracker ein neues Projekt angelegt, in das alle Probleme mit dem Dienst, sowie Serverausfälle eingetragen werden.

Mittels Service-Orchestration soll auf das vorhandene Bugracking-System, welches nur eine Web-Schnittstelle zur Verfügung stellt, von MaTRICS zugegriffen werden. Somit kann jede Veränderung der Konfiguration eines Web-Portals eines Kunden als neues *Ticket* aufgenommen werden. Des weiteren soll der Bugtracker das Projekt Web-Portal Service zyklisch auf neue *Bugs* abfragen, die dann im zu entwickelnden Konfigurationsdienst für die Operateure von LS Five Programming Systems angezeigt werden. Zudem sollen die Operateure über ihr Telefon über jeden neuen Bug informiert werden.

Die Aufgabe besteht in der Entwicklung eines Konfigurationsdienstes für den Web-Portal Service der LS Five Programming Systems. Dabei muss zum Einen die Konfiguration für den Apache Web-Server und den ProFTPd erzeugt werden. Dies geschieht mit Hilfe der SIB-Bibliothek des von Gruppe 4 erstellten Editors zur Modifikation von ASCII-Dateien. Dabei erfolgt die Steuerung des Editors SIB-basiert als LocalJob innerhalb der JobFlow-Engine der MaTRICS. Da die Konfigurationsdatei httpd.conf des Apaches recht komplex ist, beschränken wir uns hier nur auf das Hinzufügen bzw. Verändern und Löschen von VirtualHost-Einträgen, wobei jedes Web-Portal eines Kunden durch genau einen VirtualHost-Eintrag, der nur für statisches Web (ohne CGI, etc.) konfiguriert ist, abgebildet wird.

Die Alarmierung der Kunden und Operateure via Telefon erfolgt mit Hilfe der Erweiterungen des Notification-Managements durch das Zwischenthema 2. Dazu sind für die verschiedenen Notification-Typen geeignete Templates bzw. Voice-Dateien zu erstellen. Zum Anderen soll mittels des WSDL-Adapters des Zwischenthemas 1 auf das Mantis Bugtracking-System über die Web-Schnittstelle zugegriffen werden. Dabei müssen neue Bugs für ein Projekt angelegt, und die gesamte Bug-Liste eines Projektes ausgelesen werden können. Die Orchestrierung findet in dem zu entwickelnden Konfigurationsdienst der MaTRICS statt. Für das zyklische Abfragen des Bugtrackers auf neue Bugs kann innerhalb der MaTRICS ein Thread gestartet werden, in dem der Zugriff über den Adapter auf Mantis erfolgt.

Im ersten Schritt ist eine Art Pflichtenheft zu erstellen, welches die Funktionalitäten des zu entwickelnden Konfigurationsdienstes beschreibt und die genaue Zeitplanung und Einteilung der Personen enthält. Dabei soll das Design unter Zuhilfenahme von UML erstellt werden. Nach Abnahme durch die technische Abteilung von LS Five Programming Systems erfolgt die Umsetzung und Implementierung innerhalb der MaTRICS. Als Abschluss ist eine englische Dokumentation zu erstellen, die den Ablauf und die Konzepte des zu entwickelnden Konfigurationsdienst beschreibt.

6.2 Realisierung

6.2.1 Erweiterung und Anpassung des AsciiEditors

Basierend auf der bisherigen Implementierung des AsciiEditors mußte dieser zur Umsetzung des Hauptthemas erweitert und angepasst werden. Im Folgenden werden zunächst die zusätzlichen SIBs aufgezählt, die während des Hauptthemas zusätzlich durch die MaTRICS-Teilnehmer erstellt wurden.

6.2.1.1 Erweiterung der Grammatik

Um den gestiegenen Anforderungen an die Operationen, die mit Hilfe des AsciiEditor realisiert werden sollen, gerecht zu werden, wurde die ursprüngliche Grammatik erweitert und ist nun detailreicher als noch in der Gruppenphase.
Da von Anfang an auf eine XML-basierte Grammatik gebaut wurde, läuft die neuere Version auch mit Programmen, die auf der älteren Grammatik basieren. Allerdings werden die zusätzlichen Informationen der XML-Grammatikdatei nicht mehr ausgelesen und ausgewertet.

6.2.1.2 Implementierung

Es gibt unterschiedliche SIBs, die in Klassen unterteilt werden können

- **AsciiFile** - Beinhaltet Dateioperationen
  AsciiConvertFileToStringBuffer
  AsciiConvertStringBufferToFile
- **BuildAsciiEditor** - Starten des Asciieditors
  BuildBlock
- **Cursor** - Bewegung des Cursor
  CursorGetEndOfFileAsOffset
- **Modify** - Textoperationen
  ModifyAddAsciiFileSezzion
- **Util** - sonstiges
  UtilGetAllBlockNames
  UtilGetWordWrapCharacter

6.2.1.3 AsciiEditor-Makros

Zur Umsetzung der gestellten Aufgabe werden durch die Gruppe AsciiEditor die folgenden vier Makros zur Verfügung gestellt.

- **Insert** - Erzeugen von Einträgen in der jeweiligen Datei httpd.conf, proftpd.conf

  PortalManagement_AsciiEditor_InsertHttpdMacro

  PortalManagement_AsciiEditor_InsertProftpdMacro

  PortalManagement_AsciiEditor_InsertUserProFTPD

- **Delete** - Löschen von Einträgen in der jeweiligen Datei httpd.conf, proftpd.conf

  PortalManagement_AsciiEditor_DeleteMacro

  PortalManagement_AsciiEditor_DeleteUserMacro

Im Folgenden wird exemplarisch jeweils einer der Makros beider Gruppen vorgestellt.

6.2.1.4 InsertHttpdMacro

InsertHttpdMacro (Abbildung 6.1) dient dem Einfügen eines Eintrags in die Konfigurationsdatei.
Dem Makro werden hierbei die zu verarbeitende Konfigurationsdatei als StringBuffer übergeben sowie die Information, welche Daten in diese Datei eingefügt werden sollen.
Da der AsciiEditor ein Dateiobjekt benötigt, auf dem die anstehenden Operationen (einfügen, ändern, löschen) durchgeführt werden können, wird das Objekt von einem StringBuffer-Objekt über ein File-Objekt in ein RandomAccessFile-Objekt gecastet.
Dieser Objekttyp ermöglicht wahlfreies Löschen und Schreiben auf der Datei.
In unserem Fall handelt es sich um die Information, welches Portal angelegt werden soll. Dafür wird dem AsciiEditor die entsprechende Portnummer übergeben. Mithilfe der Grammatik, die wiederum die Konfigurationsdatei beschreibt, werden alle notwendigen Parameter ausgelesen, aufbereitet und mit der Laufzeitinformation (der Portnummer) in die Konfigurationsdatei geschrieben.
Am Ende der Einfügeoperation wird der Anfangs durchgeführte Castvorgang entgegengesetzt gestartet (RandomAccessFile -> File -> StringBuffer), weil die umgebende Architektur diesen Buffer wieder aufnimmt.

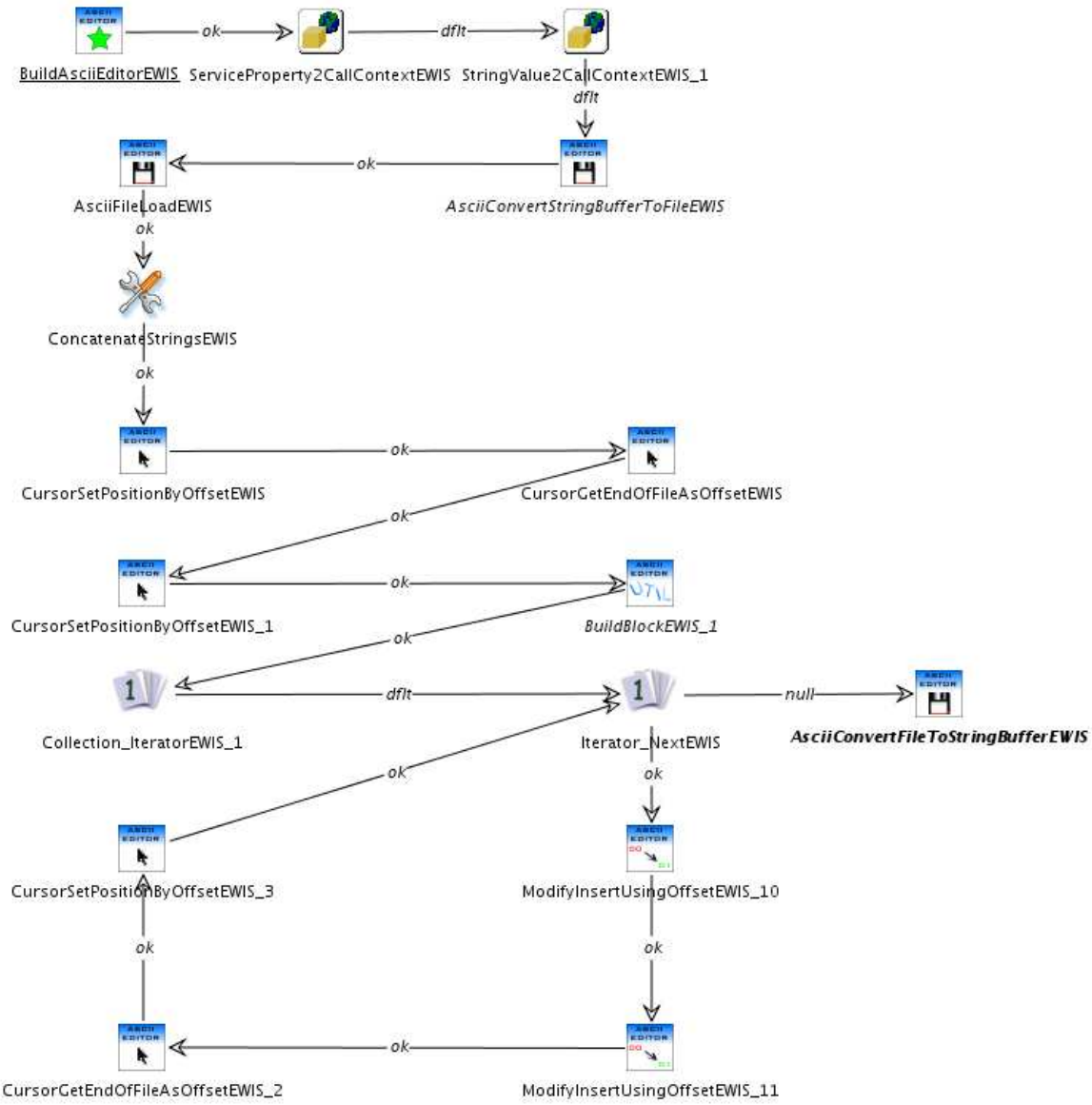

Abbildung 6.1: AsciiEditor: InsertHttpdMacro

6.2.1.5 InsertUserProFTPD

Das Makro *InsertUserProFTPD* (Abbildung 6.2) ist für das Anlegen eines Benutzers in der Benutzerverwaltungsdatei des ProFTPD Servers zuständig. Im Allgemeinen heißt die Datei ftpd.passwd, sofern man nicht explizit einen anderen Namen hierfür vergibt. Da das Passwort aus dem PortalManagement unverschlüsselt übertragen wird, aber verschlüsselt in der ftpd.passwd abgelegt werden muß, wird es mittels einer Implementierung des Unix-Befehls *crypt* umgewandelt.
Alle anderen Informationen zu einem Benutzer werden im Klartext in der Datei abgelegt,

u.a. die UserID, GroupID und sein Homeverzeichnis.

Änderungen an dieser Verwaltungsdatei können zur Laufzeit geschehen. Der FTP Server muß hierfür nicht neu gestartet werden.

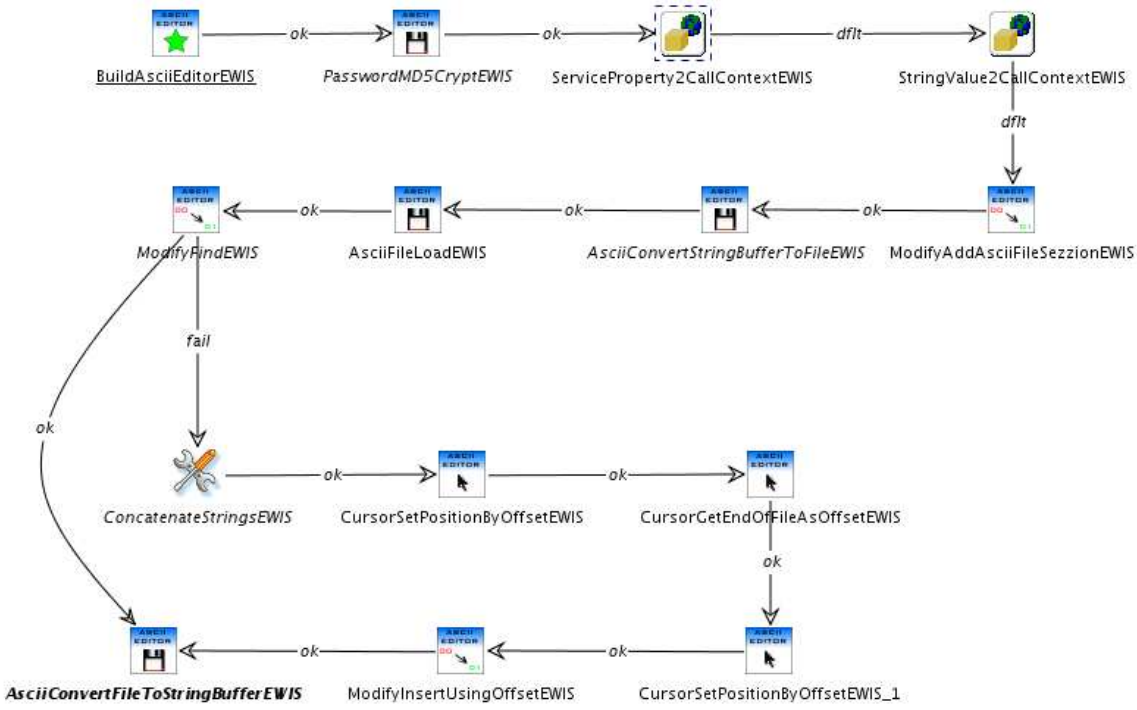

Abbildung 6.2: AsciiEditor: InsertUserProFTPDMacro

6.2.1.6 DeleteMacro

Dieses Makro (Abbildung 6.3) löscht einen Eintrag in der gegebenen Konfigurationsdatei. Ähnlich dem Einfügen werden hierbei von außen die gleichen Informationen übergeben. Nach der Löschoperation wird ein StringBuffer-Objekt erwartet.
Im Gegensatz zum Einfügen basiert das Löschen losgelöst von der Grammatik auf der Verwendung von regulären Ausdrücken, da diese die Möglichkeit besitzen, auch manuell angereicherte Portalinformationen komplett zu löschen.

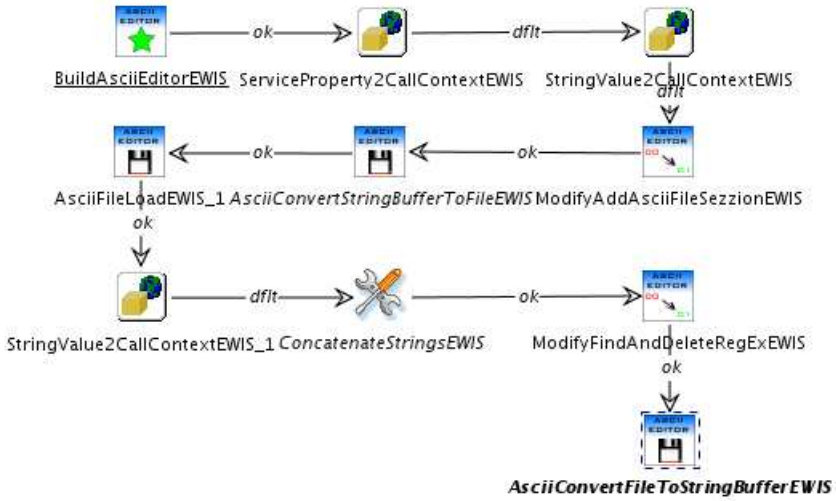

Abbildung 6.3: AsciiEditor: DeleteMacro

6.3 Portal-Management

6.3.1 Einführung

Das Portal-Management ist ein eigenständiger (jedoch auf dem MaTRICS-Framework basierender) Web-Service. Es soll alle grundlegenden Funktionen zur Verwaltung von Web-Portalen bereitstellen.
Ein Web-Portal besteht aus einem VirtualHost auf einem Web-Server und einem FTP-Zugang, um den Inhalt des VirtualHost verwalten zu können.

Zusätzlich sollen Nutzer der Web-Portale die Möglichkeit haben, Fehler, Probleme und Wünsche in einen Bugtracker einzutragen.
Als Bugtracker soll das Open-Source-Tool Mantis[12] dienen. Die Einträge des Bugtrackers sollen automatisiert ausgelesen und über die Notification-Komponente der MaTRICS an zuständige Administratoren verschickt.

6.3.2 Design

Das Anwendungsfalldiagramm (Abbildung 6.4) verdeutlicht welche Verwaltungsoptionen einem Administrator zur Verfügung stehen.
Außer des üblichen Erzeugens und Entfernens von Web-Portalen hat der Administrator die Möglichkeit, den Status eines Portals zu ändern (*enabled*, *disabled*) und ein neues Passwort für das Portal eines Nutzers zu vergeben.

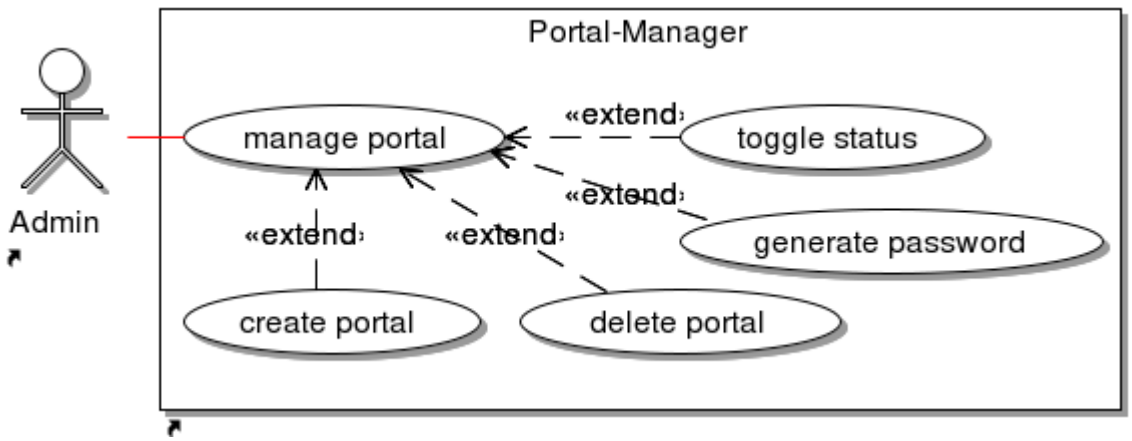

Abbildung 6.4: Anwendungsfalldiagram des Portal-Managers

6.3.2.1 Persistenz

Die Datenbankebene der MaTRICS musste lediglich um zwei Buisiness-Klassen (*Portal* und *UserPortalAssoc*) und die entsprechenden *SmartAdministrator*- bzw. Manager-

Klassen erweitert werden.
Die Klasse *Portal* enthält alle für ein Web-Portal relevanten Informationen. Die Klasse *UserPortalAssoc* assoziiert einen Benutzer mit einem Portal, dies wird in dem folgenden Klassendiagramm (Abbildung 6.5) dargestellt.

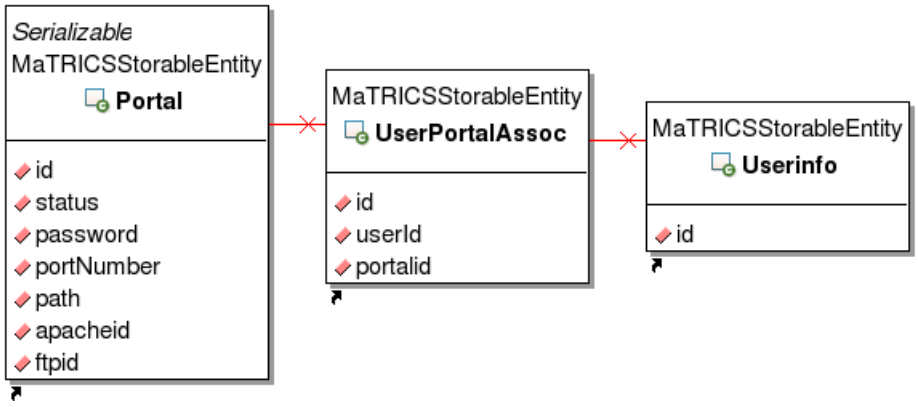

Abbildung 6.5: Portal-Manager Datenbankschema

6.3.3 Entwicklung

Im jABC-Modell (Abbildung 6.6) wurde jeder Anwendungsfall in ein Makro gekapselt. Nachfolgend werden beispielhaft die Implementierungen einzelner Anwendungsfälle erläutert.

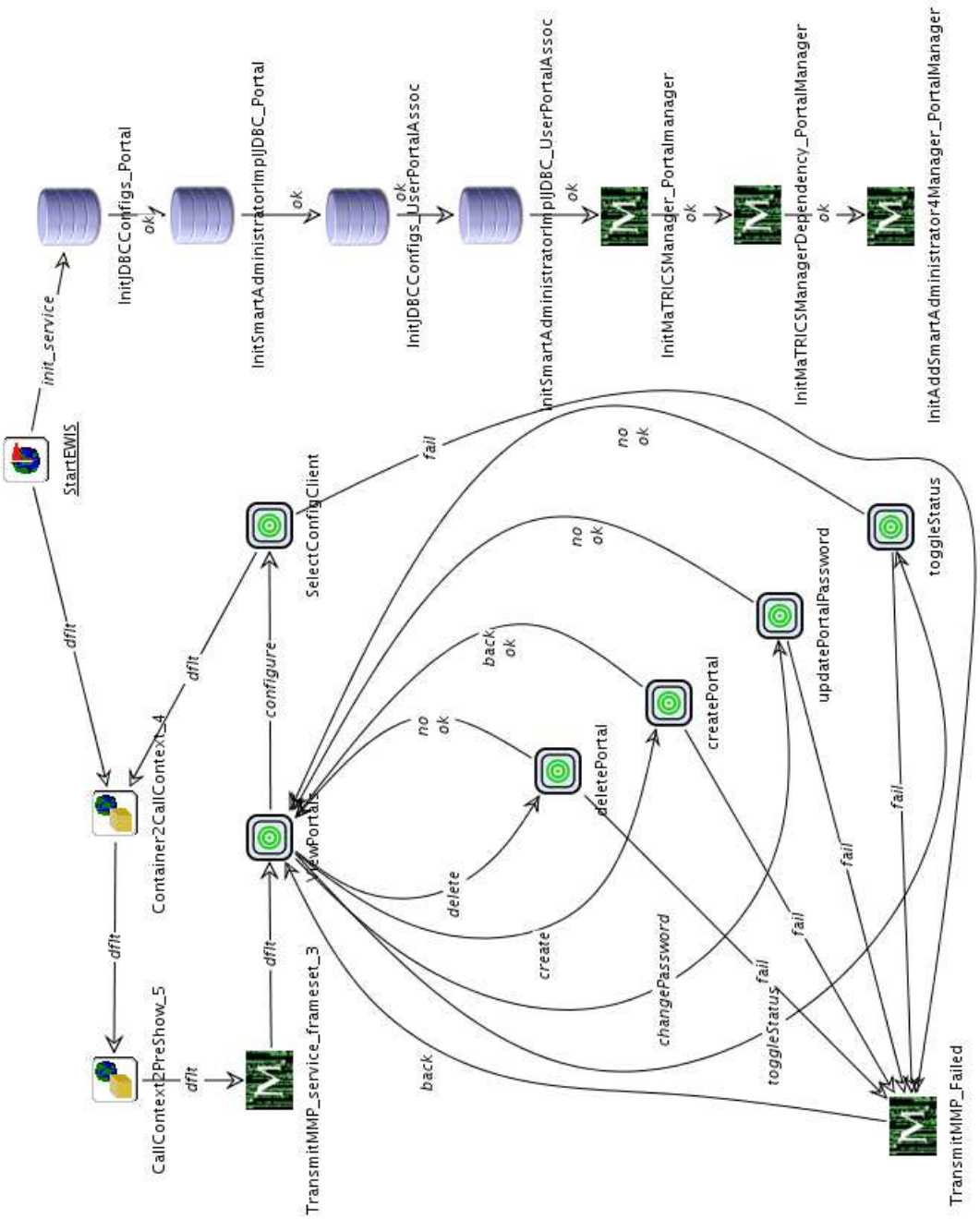

Abbildung 6.6: Portal Management - Das Modell

6.3.3.1 Anwendungsfall: *create portal*

Das Einrichten eines Portals besteht aus einer Reihe von Aufgaben:
Aktualisieren der Datenbank, Konfiguration des FTP-Servers bzw. HTTP-Servers,

Konfiguration des Bugtrackers.

Zunächst wird ein Portal-Objekt (6.3.2.1) erzeugt, mit einem (existierenden) MaTRICS-Nutzer assoziiert und gespeichert. Die Server werden mit Hilfe des MaTRICS-JobManagements konfiguriert (6.3.3.3) und abschließend der Mantis-Zugang eingerichtet.

6.3.3.2 Anwendungsfall: *generate password*

Sollte ein Nutzer sein Passwort vergessen haben oder aus anderen Gründen ein neues Passwort benötigen, kann der Administrator den Portal-Manager veranlassen ein neues Passwort zu generieren. Dieses wird dem Nutzer auf allen verfügbaren Kommunikatiionskanälen (Email, Telefon, SMS) mitgeteilt. Danach wird die Passwortdatei des FTP-Servers aktualisiert über das JobManagement.

6.3.3.3 Konfiguration der Server

Die Konfiguration des FTP- bzw. des HTTP-Servers erfolgt über das JobManagement der MaTRICS. Dazu wird für jeden Anwendungsfall ein *Task* modelliert.
Ein *Task* besteht aus mehreren *Jobs*, für die bei Bedarf eine Eindeutige Ausführungsreihenfolge definiert werden kann. Zur Laufzeit müssen beide Server im MaTRICS HostManagement als *ConfigClients* eingerichtet werden.
Die Abbildung 6.7 zeigt den *Task CreatePortalTask*, welcher den Anwendungsfall *create portal* abdeckt.

CreatePortalTask besteht aus folgenden - in ihrer Ausführungsreihenfolge aufgelisteten - zwölf *Jobs*:

- *GlobalUndoSaveFileRevisions*: In diesem *Job* werden die aktuellen Versionsnummern der Konfigurationsdateien *proftpd.conf*, *httpd.conf* und *proftp.passwd*, damit sie bei Fehlschlagen des *Tasks* wiederhergestellt werden können.
- *ConfigProFTP* (Abbildung 6.8) konfiguriert (mithilfe des AsciiEditors) die Datei *proftpd.conf*.
- *TransmitProFTPConfig* überträgt die neue *proftpd.conf* zum entspechenden *ConfigClient*.
- *ConfigProFTPUser* trägt den Zugangsdaten des Portals und den Pfad des Portals in die *proftp.passwd* ein.
- *TransmitProFTPUserConfig* überträgt die neue *proftp.passwd* zum entspechenden *ConfigClient*.
- *TransmitProFTPUserConfig* führt eienen Neustart des FTP-Servers durch.

- *CreateDocumentRoot* richtet ein Verzeichnis für das neue Portal ein.
- *ConfigApache* konfiguriert (mithilfe des AsciiEditors) die Datei *httpd.conf*.
- *TransmitApacheConfig* überträgt die neue *httpd.conf* zum entspechenden *ConfigClient*.
- *RestartApache* führt eienen Neustart des HTTP-Servers durch.
- *CreateMantisBug* (Abbildung 6.9) erzeugt einen neuen Mantiseintrag (Kapitel 6.3.3.4) über die Erstellung des neuen Portals.
- *GlobalUndo*: Dieser Job wird ausgeführt, falls ein anderer Job nicht beendet werden konnte. Die Konfigurationsdateien werden auf die zuvor gesicherten Versionen zurückgesetzt und das Portal wird aus der Datenbank entfernt. Zudem wird ein neuer Mantiseintrag erstellt.

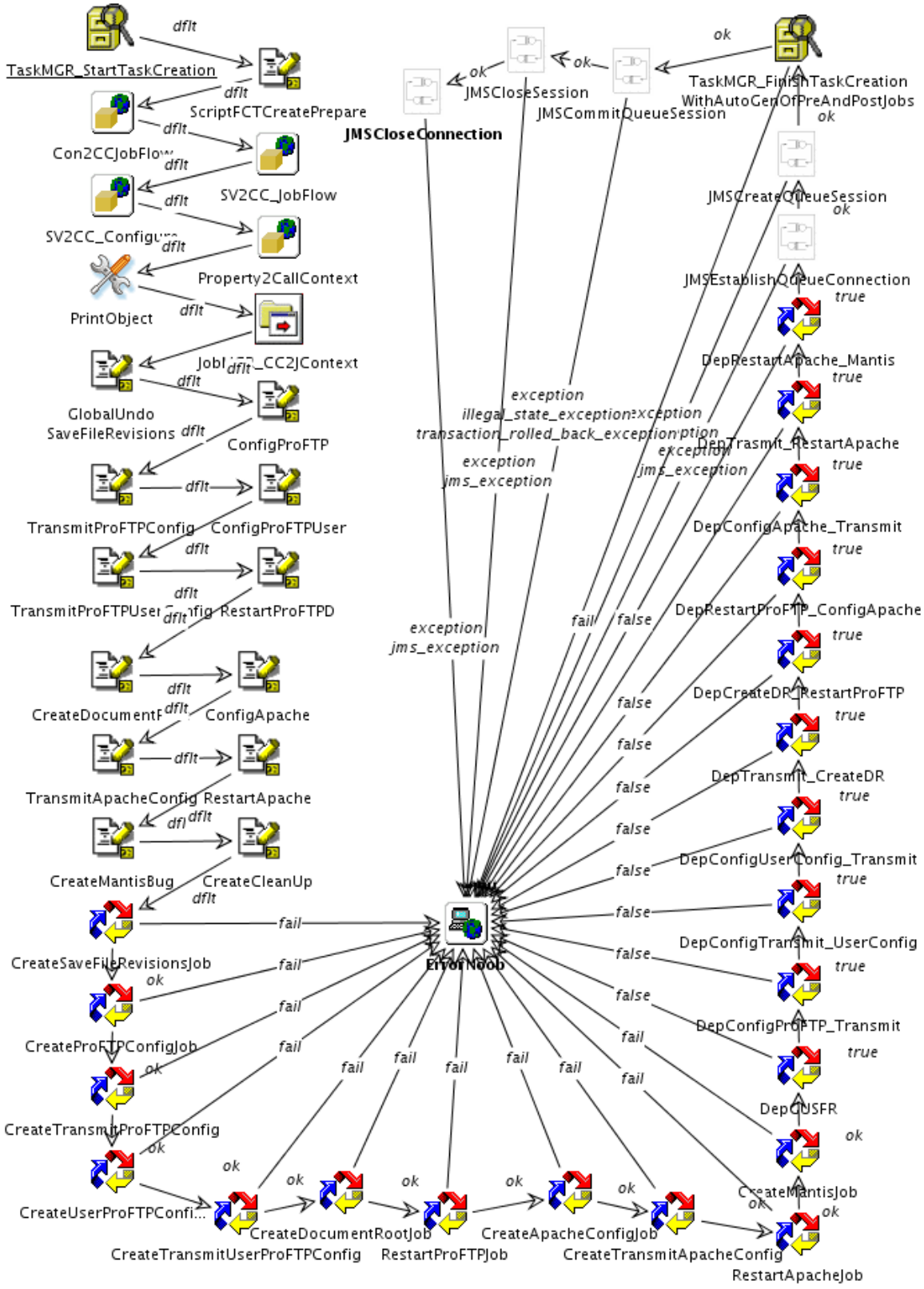

Abbildung 6.7: jEWIS-Graph des CreatePortalTask

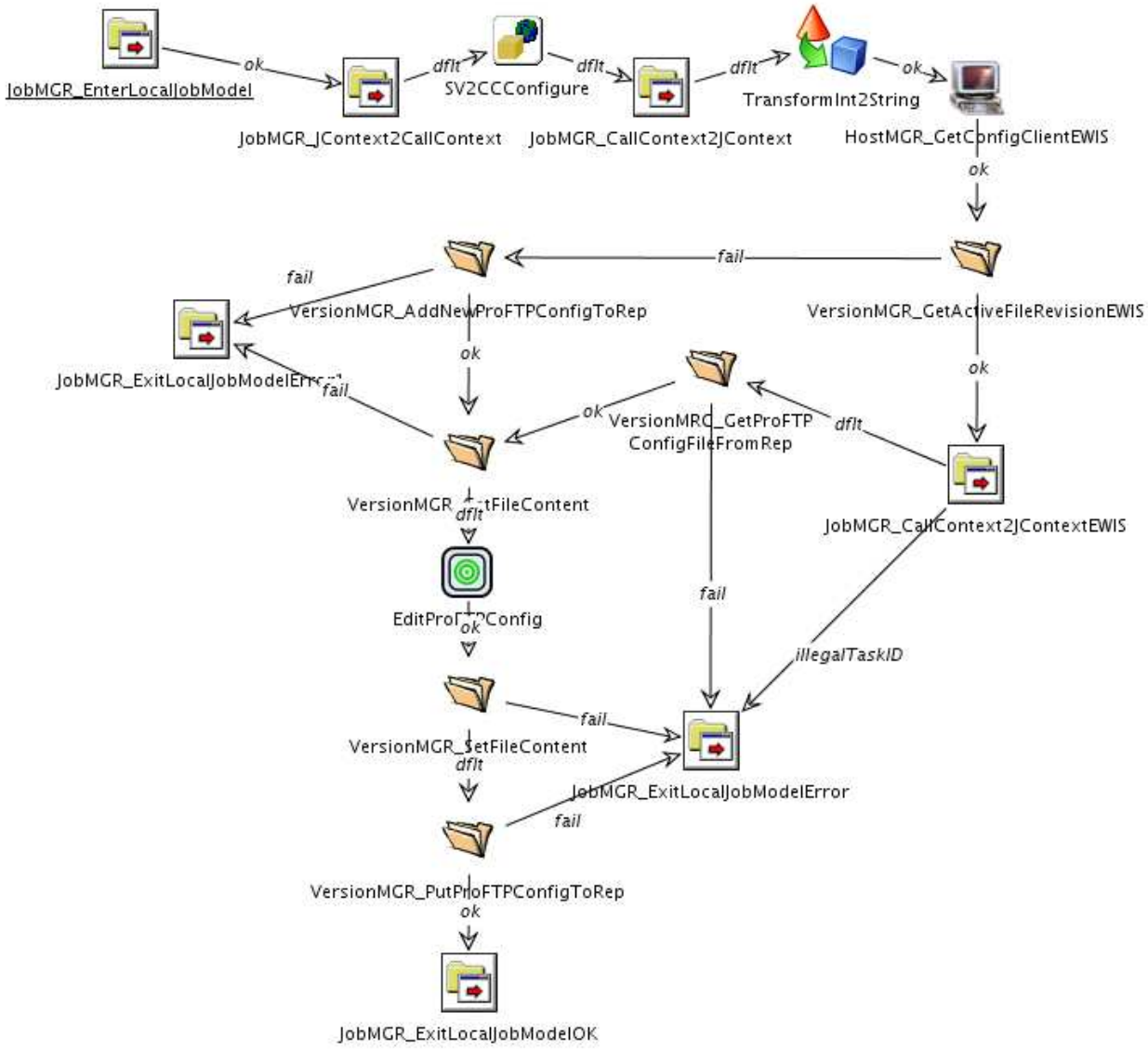

Abbildung 6.8: Konfigurtionsjob für den FTP-Server

6.3.3.4 Anbindung des Mantis-Bugtrackers

Immer, wenn ein neues Portal für einen Benutzer angelegt wird, wird hierfür ein Eintrag im Mantis BugTracker angelegt. Dieser Eintrag enthält Informationen über das neue Portal (z.B. die Portalnummer). Danach werden periodisch die neue Einträge ausgelesen (mittels XPATH der Antwort, definiert über die WSDL dieser Webseite) und die eindeutige durch Mantis generierte ID für jeden Eintrag übermittelt. Dieser Schlüssel (ID) dient dann dem Zugang zum Ändern des jeweiligen Status des Eintrags. Der Status des Eintrags wird auf geschlossen (close) gesetzt, da es sich hierbei lediglich um eine Information und keinen

Fehler (Bug) im engeren Sinne handelt.

- *LoginGraphSIB* Dieses Makro kümmert sich um das Einloggen in den MantisBug-Tracker (Startseite).
- *ImportReportBug* Importiert die zugrundeliegende WSDL der Webseite.
- *ReportBug* Mittels Request-Response werden hier die Einträge im Mantis abgefragt.
- *ParseMantisResponse* Das SIB dient dem Parsen der Antwort des vorherigen SIBs zur weiteren Verwendung.
- *ImportChangeStatusWSDL* Liest die WSDL ein, die den Status eines Mantis Bugs ändern kann.
- *ChangeStatusOfBug* Das Ergebnis des vorherigen Parsens dient dieser WSDL Methode als Inputparameter für den nächsten Request-Response.

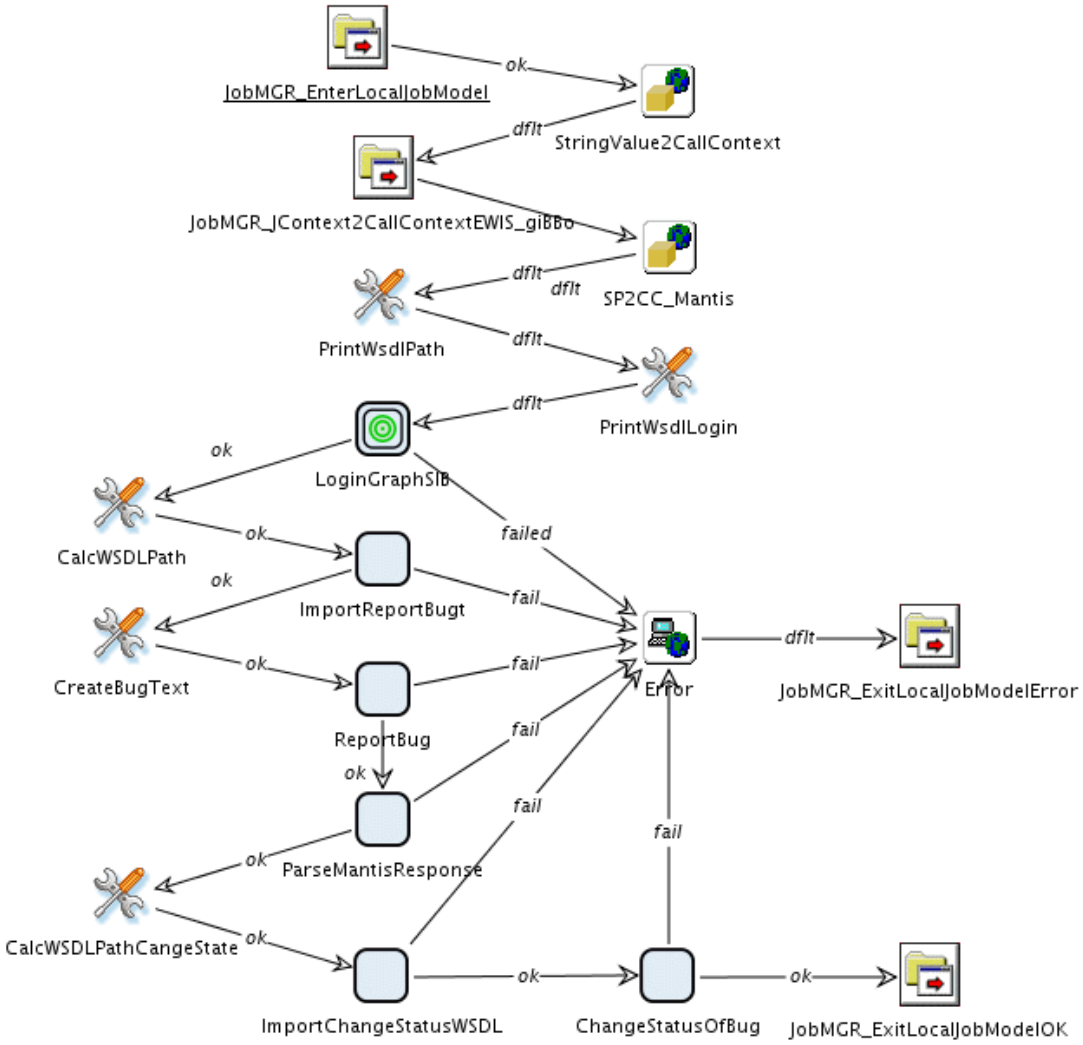

Abbildung 6.9: Konfigurationsjob für das Einstellen von Mantiseinträgen

Der folgende jABC Graph dient dazu automatisiert neue Bugs auszulesen. Er ist als Makro in dem Thread *MantisBugcheckerThread* (6.11) eingebettet. Dieser Thread wird in regelmäßigen Abständen angestoßen und informiert mithilfe des *Notification*-Managements die zuständigen Admins über das Vorhandensein neuer Bugs.

- *LoginGraphSIB* Dieses Makro kümmert sich um das Einloggen in den MantisBug-Tracker (Startseite).
- *ImportGetBugList* Importiert die WSDL zum Einstellen der Filterfunktionen von Mantis sowie die Fehler (Bugs) auslesen zu können.
- *ApplyFilter* und ApplyFilter_Again Führt die WSDL Methode ApplyFilter aus, damit nur noch neue Fehler (Bugs) ausgelesen werden.

- *GetBugList* Führt die WSDL Methode ViewIssues aus und liefert die Menge (Tabelle) aller Mantisbugs wieder.
- *ParseMantisResponse* Aufbereitung der Daten des vorherigen SIBs.

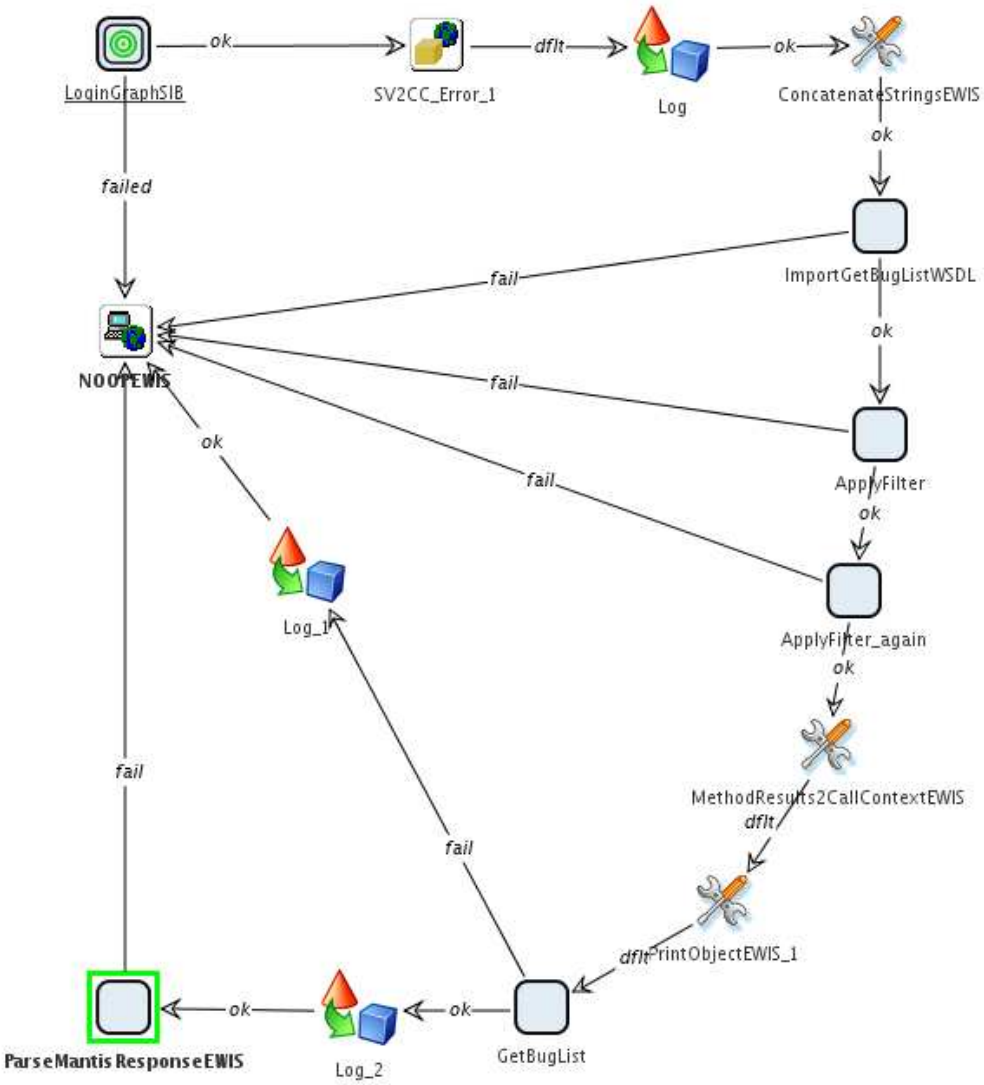

Abbildung 6.10: Abfragen von neuen Mantiseinträgen

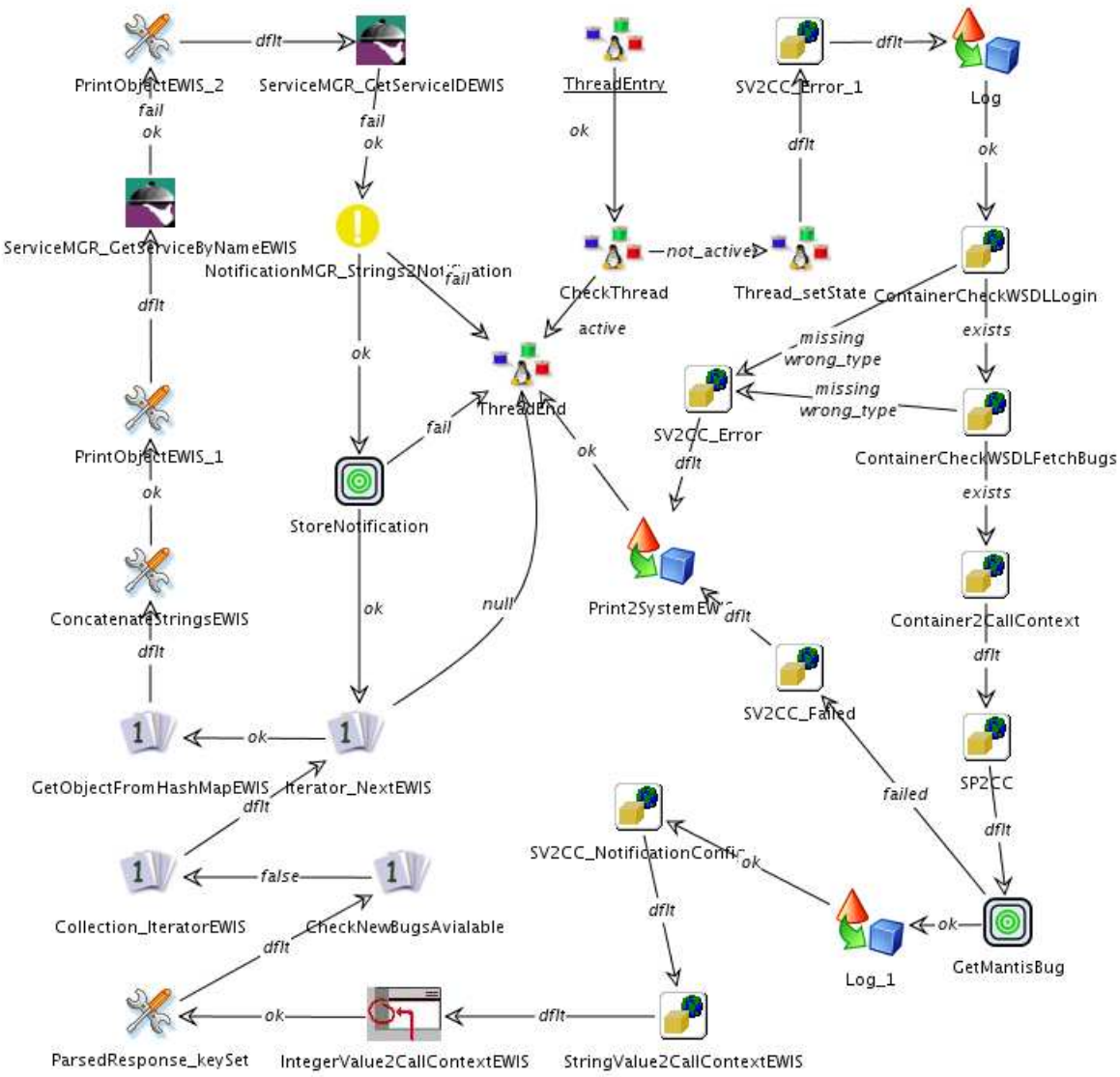

Abbildung 6.11: Thread für das Abfragen von neuen Mantiseinträgen

Kapitel 7

Fazit

Die Aufgaben der PG waren in Teilprojekte gegliedert die im Verlauf der PG an Aufwand und Komplexität zunahmen. Dabei wurden die Teilgruppen stets größer, wodurch die Teilnehmer gezwungen waren sich immer besser zu organisieren - was nicht immer gelang.

Am Anfang stand die Einarbeitung in das jEWIS-Plugin und das jABC im Vordergrund. Danach sollten sich die Teilnehmer selbständig in Drittanbieter-Software einarbeiten und eigene SIB-Bibliotheken entwickeln.
Anschließend lag der Fokus auf Orchestrierungs-Technologien und dem MaTRICS-Framework. Zuletzt sollten alle Teilprojkte mithilfe des MaTRICS-Frameworks in einem Beispieldienst vereint, und somit auch getestet werden.

Bei der Integration der Teilprojekte in die MaTRICS war paralleles arbeiten nicht möglich. Das lag vor Allem daran, dass vor Beginn des Hauptthemas nur wenige PG-Teilnehmer direkt in Berührung mit der MaTRICS kamen. Von den übrigen Teilnehmern fanden die meisten keine Zeit (oder Motivation) sich in das relativ komplexe Projekt einzuarbeiten. Natürlich führte dies zu weiteren Verzögerungen.

Ein Teil der Entwicklungswerkzeuge stellte sich als große Frustrationsquelle heraus. Ständig neue und unerklärliche Fehler wirkten sich sehr negativ auf die Motivation der Teilnehmer aus.

Die technische Ausrüstung war zum Anfang der PG ausreichend, sodass alle vierzehn Teilnehmer einen Arbeitzplatz fanden. Zum Ende hin änderte sich dies dramatisch: teilweise standen der PG nur vier Arbeitsplätze zur Verfügung und die Performance der Server lies zu wünschen übrig (cvs-update dauerte bis zu zwei Minuten).

Für viele Teilnehmer war die PG die erste Erfahrung mit praxisnaher Softwareentwicklung. Auch Teilnehmer die zuvor Praxiserfahrung gesammelt hatten, konnten ihr Wissen auf viele neue Technologien und Themenbereiche ausdehnen. Zudem wurden viele Softskills (Selbsorganisation, Teamfähigkeit, Sprachliche Kompetenz(englisch)) trainiert.

Die Betreuer waren immer sehr motiviert und glänzten stets mit großen Fachwissen, sodass die PG für alle Teilnehmer eine der positiveren Erfahrung im Studium war.

Literaturverzeichnis

[1] *Design and implementation of a SIB-library for management of Asterisk via Asterisk-Manager-Interface*, Dokumentation

[2] *http://jfree.org/*

[3] Andreas Karbach, *SIP Die Technik*, Vieweg 2005

[4] Anatol Badach, *Voice over IP*,Hanser 2005

[5] *http://www.sipforum.org/*, Internetquelle

[6] *http://www.voip-info.de/*, Internetquelle

[7] Weerawarana, S. et al., *Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WS-Reliable Messaging, and More, Prentice Hall 2005*

[8] *http://htmlunit.sourceforge.net/*

[9] *http://sourceforge.net/projects/mozswing*

[10] *http://www.w3.org/TR/wsdl*

[11] *Design and implementation of a SIB-library for service-orientated generation of graphs and diagrams*, Dokumentation

[12] *http://www.mantisbt.org/*