INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Benutzer-Verwaltungsanwendung Erstellung für den CIP-Pool am Institut für Statistik/LMU

Viktoriya Serbina und Alexander Velkov

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Fortgeschrittenenpraktikum

Benutzer-Verwaltungsanwendung Erstellung für den CIP-Pool am Institut für Statistik/LMU

Viktoriya Serbina und Alexander Velkov

Aufgabensteller:	Prof. Dr. Dieter Kranzlmüller Prof. Dr. Friedrich Leisch (Statistik Institut, LMU)
Betreuer:	Dr. Nils gentschen Felde Manuel J. A. Eugster (Statistik Institut, LMU)
Abgabetermin:	8. September 2010

LMU-Arbeit

Wir versicheren, dass wir diese Ausarbeitung selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet haben.

München, den 8. September 2010

. . . . . .
(Unterschrift der Kandidaten)

Abstract

Der CIP-Pool ("Computer-Investitions-Program" für die Lehre an deutschen Hochschulen) am Institut für Statistik stellt Rechner- und Anwendungskapazitäten für Lehrveranstaltungen zur Verfügung und ermöglicht darüberhinaus Studierenden, PCs, Internet und spezielle Anwendungen für Ausbildungszwecke nutzen zu können.

Das Vorgehen der Vergabe von Kennungen, mit denen die Studierenden sich anmelden können, war bislang ziemlich komplex, hatte viel Zeit in Anspruch genommen und wurde oft manuel gepflegt. Um den administrativen Aufwand für die Erstellung und Verwaltung von CIP-Benutzerkonten für die CIP-Betreuer zu erleichtern, wurde im Rahmen unseres Projektes eine Benutzerverwaltungsanwendung - Account Management Application for Statistical Institute (AMASI) - entwickelt.

Diese Anwendung ermöglicht den Administratoren am Department, Benutzerkonten und deren Rollen und Berechtigungen für die Nutzung der angebotenen Dienstleistungen zu verwalten. Unser Benutzerverwaltungsystem besteht aus zwei Komponenten - Frontend und Backend. Über das Frontend können Daten eingetragen und in einer Datenbank gespeichert werden. Das Frontend ruft anschließend das Backend auf, um entsprechende Berechtigungen für die Benutzerkonten zu setzen.

Inhaltsverzeichnis

1 Einführung

Heutzutage ist es für Studierende immer, Zugang zu neuen Technologien an Hochschulen zu bekommen. CIP-Pools ("Computer-Investitions-Program"/ für die Lehre an deutschen Hochschulen) stellen Rechner- und Anwendungskapazitäten für Lehrveranstaltungen zur Verfügung und ermöglichen darüberhinaus Studierenden, PCs, Internet und spezielle Anwendungen für Ausbildungszwecke benutzen zu können.

Der CIP-Pool am Statistikdepartement der LMU ist auf mehrere Räume verteilt, in denen Studierende mit Statistik als Haupt- oder Nebenfach spezielle Anwendungen für Statistische Auswertungen unter Windows und Linux benutzen können. Das resultiert in einer Vielzahl an Nutzern, die einen reibungslosen Zugriff auf Computer und Anwendungsdienste erwarten. Für das Department mit mehr als 500 Zugangsberechtigte - verteilt in unterschiedliche Benutzerrollen - stellt die Verwaltung der CIP-Benutzerkonten eine große Herausforderung dar. Das gilt vor allem dann, wenn die Benutzerkonten in mehreren Systemen, Verzeichnisdiensten oder Datenbanken gepflegt werden sollen.

Die Benutzerinformationen werden zur Zeit großenteils in verschiedenen Dateien gespeichert, die manuell synchron gehalten werden müssen. Auch die Zugangsverwaltung für die verschiedensten Dienste wird per Hand erledigt und erfordert oft, gleiche Daten mehrmals eingeben zu müssen. Diese Aufgaben nehmen viel Zeit in Anspruch und werden von verschiedenen Administratoren erledigt. Aueßerdem kann es zu Übertragungsfehlern kommen. Das Departement setzt teilweise spezifisch angepasste Anwendungslösungen ein und möchte keine grundlegenden Veränderungen an diesen Systemen vornehmen. Die Einführung neuer- und die Verbesserung der Qualität bestehender Dienste ist ein ständig fortlaufender Prozess und muss ebenfalls unterstützt werden. Um den administrativen Aufwand dieser Aufgaben zu reduzieren, wird im Rahmen unserer Arbeit eine an die Anforderungen angepasste Lösung entwickelt und umgesetzt. Das Ziel der Lösung ist eine einfache, leicht erweiterbare, praktische und systemübergreifende Verwaltung von Benutzern und Benutzerinformationen anzubieten und zudem die Rollenprofile und Kontrolle über Dienstzugriffe zu ermöglichen.

Im Rahmen dieses Praktikums haben wir ein rollen-basiertes Benutzerverwaltungssystem entwickelt, dem wir den Namen "AMASI" gegeben haben. AMASI bietet eine Web-Schnittstelle, mit deren Hilfe die Adminstratoren die Benutzer und deren Zugriff auf Dienste verwalten können, die am Institut für Statistik angeboten werden.

In dieser Arbeit wird AMASI vorgestellt, beginnend mit der Anforderungsanalyse in Kapitel 2, wo zuerst die Ist- und Soll-Zustände erfasst werden, die zur Entwicklung eines allgemeinen Entwurfes führen. Da AMASI hauptsächlich aus zwei logisch getrennten Komponenten

besteht, werden diese in Kapitel 3 und 4 mit Entwurf und Implementierung vorgestellt. Die Frontend Komponente von AMASI wird von Viktoriya Serbina entwickelt und vorgestellt, das Backend von Alexander Velkov. In Kapitel 5 wird die Installation von AMASI beschrieben sowie die Migration von dem alten System auf das neue. Eine kurze Zusammenfassung des gesammten Projektes wird in Kapitel 6 gemacht.

AMASI ist seit sechs Monaten am Institut für Statistik in Betrieb und wird erfolgreich in der Administration der CIP-Pool Umgebung eingesetzt. Das System hat die administrativen Prozesse bezüglich der Benutzerverwaltung und Dienstzugriffe vereinfacht und mit seiner praktischen Bedienung und leichten Erweiterbarkeit überzeugt.

2 Anforderungsanalylse

In diesem Kapitel wird zunächst die Aufnahme der Ist-Zustand am Department erfasst. Dies ist für die Problemidentifikationen nötig und führt zur gewünschten Zielsetzung und den Anforderungen, die in Kapitel Soll-Zustand beschrieben werden. Anschließend wird ein generelles Konzept zur Realisierung vorgestellt, wodurch die Grundlagen des neuen Systems bestimmt werden.

2.1 Ist-Zustand

Das CIP-Netz am Institut für Statistik umfasst 43 Arbeitsplätze, die unter *Windows XP* und *Linux Debian* laufen, einen Drucker-Server, der unter *Windows Server 2003* und zwei Hochleistungsrechner für statistische Auswertungen ("Rechenknechte"), die unter Debian laufen. Derzeit sind ca. 500 Benutzer im System mit quasi unbefristetem Zugang eingetragen, und pro Jahr werden ungefähr 100 neue Benutzerkonten beantragt. Zusätzlich gibt es spezielle Kurs- und Gast-Nutzerkonten, die für eine befristete Zeit aktiviert werden. Der CIP-Pool stellt den nutzenden Beschäftigten und Studierenden eine Reihe von Diensten zur Verfügung: Arbeiten unter *Windows*- und *Linux* Betriebssystemen, Freidrucke und Zugang zum internen Wiki Infoportal, um studienbezogene Informationen oder eine Reihe von departmentsinternen Anleitungen zu erhalten.

Die Benutzerverwaltung ist zur Zeit komplex gestaltet und umfasst mehrere Schritte, die von verschiedenen Administratoren durchgeführt werden. Benutzerdaten wie Name, E-Mail Adresse, Studienfach werden von einem Administrator in einer Excel-Datei verwaltet. Diesem Administrator wird als Rolle Admin1 (oder Administrator1) in den folgenden Beispielen zugeordnet. Die anderen Administratoren sind - je nach Fachwissen und sonstigem Arbeitsbereich - für die technische Realisierung und Verwaltung der Benutzerkonten zuständig. Die Rolle Admin2 (oder Administrator2) stellt schematisch diese Administratoren in den folgenden Beispielen dar. Es ist leicht vorzustellen, dass durch diese Bereichstrennung ein hoher Kommunikationsbedarf zwischen den einzelnen Administratoren die Folge ist.

Um sich ein besseres Bild vom Benutzerverwaltungsablauf zu schaffen, betrachten wir folgende Anwendungsfälle:

1. Vergabe eines CIP-Pool Benutzerkonto

2. Änderung eines Passwortes eines Benutzerkontos

3. Freischaltung eines zusätzlichen Dienstes für ein existierendes Konto

4. Deaktivierung eines Dienstes für ein Benutzerkonto

5. Löschung eines CIP-Pool Benutzerkontos

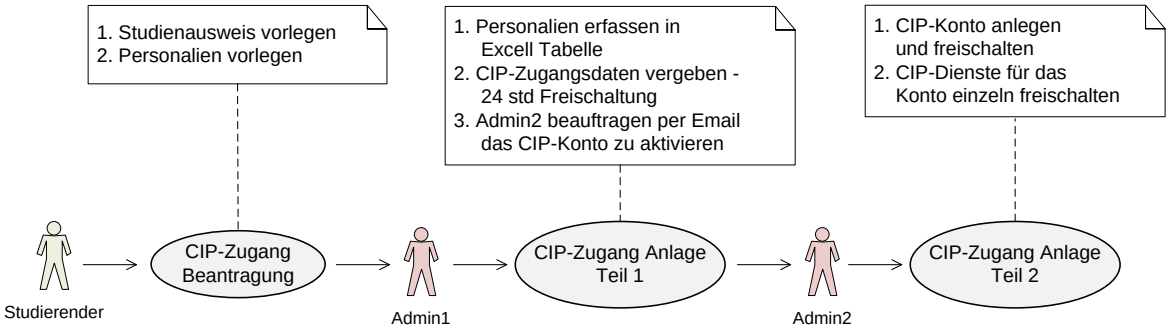

Abbildung 2.1: Jetziger Ablauf von Vergabe und Verwaltung eines CIP-Pool Benutzerkontos

In Abbildung 2.1 wird der Ablauf von Vergabe eines CIP-Pool Benutzerkontos (Anwendungsfall 1) und das Aktivieren des Windows-Dienstes graphisch dargestellt. Benötigt ein Studierender ein Benutzerkonto am Department, geht er erst zum Admin1 und legt seine Personalien sowie seine Immatrikulationsbescheinigung vor. Dieser Administrator fasst seine Daten in einer Excel-Tabelle zusammen und teilt ihm seine Anmeldedaten (Kennung und Passwort) mit. Diese sind allerdings noch nicht aktiv. Sie werden erst nach der technischen Erzeugung des Kontos mit entsprechenden Diensten durch Admin2 freigeschaltet (in der Regel innerhalb von 24 Stunden), der per E-Mail von Admin1 dazu aufgefordert wird.

Die technische Freischaltung bedeutet generell das Durchführen folgender Aktionen:

1. Erzeugung eines Arbeitsverzeichnisses. Im Institut für Statistik werden die Daten der Nutzer in Verzeichnisse in einem Pool zur Verfügung gestellt, der auf einem ZFS Dateisystem organisiert ist.

2. Erzeugung eines Benutzers auf dem Domain Server. Dabei wird das in Schritt 1 erstellte Arbeitsverzeichnis dem neuen Benutzerkonto zugewiesen.

3. Aktivierung des Windows-Zuganges für den Benutzer. Mittels *samba* wird eine NT Domäne zur zentralen Authentifizierung und Autorisierung von Benutzern im CIP-Rechnernetz des Instituts für Statistik realisiert. Das neue Benutzerkonto muss in *samba* [23] registriert werden. Da Linux auch als Betriebsystem im CIP-Netz benutzt werden kann, stellt *NIS* [28] ein Verzeichnis für Benutzernamen und Passwörter zur Verfügung, die für die Autorisierung und Authentifizierung unter Linux benutzt wird. Um einem Benutzer Zugang zu Linux-Diensten zu geben, muss das Benutzerkonto in die NIS-Tabellen eingetragen werden.

Schritte 2 und teilweise 3 werden mittels der freien Hilfssoftware *Webmin* [25] erledigt, Schritt 1 sowie weitere spezifische Dienstanpassungen für das Benutzerkonto werden manuell durchgeführt (z.B Einträge in die NIS Tabellen).

Um die Vergabe von Benutzerkonten zu beschleunigen, werden ca. 100 Zugänge auf dem Domain Server im Voraus erzeugt, z.B. mit Benutzernamen cip07400 bis cip07500. Diese werden dann bei Bedarf (Admin1 verlangt die Freischaltung von einem neuen Benutzerkonto) von Admin2 mit Personeninformationen und Passwort ergänzt und freigeschaltet. Nach der erfolgreichen Durchführung der oben beschriebenen Schritte kann sich der Studierende mit dem neuen Benutzerkonto im CIP-Pool anmelden.

Dieser Anwendungsfall ist in Tabelle 2.1 zusammengefasst.

Anwendungsfallname	**Neuanlage von einem Benutzerkonto**
Kurze Beschreibung	Erstellung eines Benutzerkontos und Freischaltung von entsprechenden Diensten
Auslöser	Der Studierende fordert ein CIP-Benutzerkonto an.
Vorbedingung	Der Studierende darf ein CIP-Konto beantragen - Statistik als Haupt-/Nebenfach, Teilnahme an einem Kurs am Institut für Statistik, etc.
Nachbedingung	Es wird ein Benutzerkonto angelegt. Der Studierende bekommt Anmeldedaten.
Beteiligte Akteure	Studierende, Administrator1, Administrator2
Primärszenario	1. Der Studierende kommt zu Administrator1. 2. Der Studierende legt seine Personalien vor. 3. Die relevanten Daten werden von Administrator1 in einer Excel-Tabelle erfasst. 4. Administrator1 fordert per E-Mail Administrator2 auf, ein neues Benutzerkonto zu erzeugen. 5. Administrator2 legt das Benutzerkonto an und schaltet die entsprechenden Dienste frei.

Tabelle 2.1: Ist-Zustand: Neuanlage von einem Benutzerkonto

Tabellen 2.2, 2.3, 2.4 und 2.5 beschreiben weitere Anwendungsfälle: “Änderung eines Passwortes”, “Freischaltung eines zusätzlichen Dienstes für ein existierendes Benutzerkonto”, “Deaktivierung eines Dienstes für ein Benutzerkonto”, ”Löschung eines CIP-Benutzerkontos”

Aus den oben beschriebenen Anwendungsfällen kann man sehen, dass an allen Abläufen mehrere Administratoren beteiligt sind. Administrator1 ist immer die selbe Person und Ansprechperson für alle administrativen Vorgänge im CIP-Pool. Administrator2 ist je nach Aufgabenbereich ein anderer. Das führt dazu, dass alle Abläufe verzögert werden und deren Durchführungszeit nicht vorhergesagt werden kann.

Anwendungsfallname	**Änderung eines Passwortes**
Kurze Beschreibung	Studierender vergisst sein CIP-Password. Es muss ein neues gesetzt werden.
Auslöser	Der Studierende hat das Passwort vergessen und fordert ein neues an.
Vorbedingung	Der Studierende hat ein CIP-Benutzerkonto.
Nachbedinung	Es wird ein neues Passwort gesetzt.
Beteiligte Akteure	Studierende, Administrator1, Administrator2
Primärszenario	1. Der Studierende kommt zu Administrator1. 2. Der Studierende legt seine Personalien vor. 3. Administrator1 teilt dem Studierenden sein neues Passwort mit. 4. Administrator1 fordert per E-Mail Administrator2 auf, das neue Passwort zuzuteilen. 5. Administrator2 ändert das Passwort für das Benutzerkonto innerhalb 24 Stunden.

Tabelle 2.2: Ist-Zustand: Änderung eines Passwortes

Anwendungsfallname	**Freischaltung eines zusätzlichen Dienstes für ein existierendes Benutzerkonto**
Kurze Beschreibung	Freischaltung eines zusätzlichen Dienstes - z.B. Linux-Zugang - für einen Studierenden mit aktivem Benutzerkonto.
Auslöser	Der Studierende fordert einen zusätzlichen Dienst.
Vorbedingung	Studierender hat schon ein Benutzerkonto.
Nachbedingung	Es werden entsprechende Dienste für den Studierenden freigeschaltet.
Beteiligte Akteure	Studierende, Administrator1, Administrator2
Primärszenario	1. Der Studierende kommt zu Administrator1. 2. Administrator1 schreibt eine Email an Administrator2 und fordert ihn den Dienst für den Studierenden freizuschalten. 3. Administrator2 schaltet für den Studierenden den entsprechende Dienst frei.

Tabelle 2.3: Ist-Zustand: Freischaltung des zusätzlichen Dienstes für schon existierendes Konto

Anwendungsfallname	**Deaktivierung eines Dienstes für ein Benutzerkonto**
Kurze Beschreibung	Ein Dienst - z.B. Windows-Zugang - wird für ein Benutzerkonto deaktiviert.
Auslöser	Der Administrator möchte aus organisatorischen Gründen den Windows-Zugang für ein Benutzerkonto deaktivieren.
Vorbedingung	Für den Studierenden wurde bereits ein Benutzerkonto mit betreffendem Dienst aktiviert.
Nachbedingung	Der Dienst wird für das Benutzerkonto deaktiviert.
Beteiligte Akteure	Administrator1, Administrator2
Primärszenario	1. Administrator1 fordert Administrator2 per E-Mail auf, den Dienst für das Benutzerkonto zu deaktivieren. 2. Administrator2 deaktiviert den Dienst für das Benutzerkonto und teilt Administrator1 das erfolgreiche Ergebnis mit.

Tabelle 2.4: Ist-Zustand: Deaktivierung eines Dienstes für ein Benutzerkonto

Anwendungsfallname	**Löschung eines CIP-Benutzerkontos**
Kurze Beschreibung	Löschung eines CIP-Benutzerkontos und Deaktivierung von allen Diensten.
Auslöser	Administrator1 fordert die Löschung eines CIP-Benutzerkos.
Vorbedingung	Das zu löschende CIP-Benutzerkonto existiert.
Nachbedingung	Ein CIP-Benutzerkonto wird gelöscht. Die Dienste, die für dieses Konto freigeschaltet waren, werden deaktiviert.
Beteiligte Akteure	Administrator1, Administrator2
Primärszenario	1. Administrator1 fordert Administrator2 per E-Mail auf, ein CIP-Benutzerkonto zu löschen. 2. Administrator2 löscht das CIP-Benutzerkonto.

Tabelle 2.5: Ist-Zustand: Löschen eines CIP-Benutzerkontos

Die gespeicherten Daten eines Benutzerkontos sind in der Exel-Datei nicht vollständig erfasst, weil dort zwar die Personalien eingetragen sind, nicht aber die dienstbezogenen Informationen. Deshalb kann nicht garantiert werden, dass keine groben Fehler auftreten, beispielsweise das Erstellen von zwei Benutzerkonten für denselben Studierenden.

2.2 Soll-Zustand

Um den administrativen Aufwand im CIP-Pool zu verbessern, wurden vom Institut für Statistik verschiedene Anforderungen gestellt. Die Anwendungsfälle, beschrieben in Kapitel Ist-Zustand 2.1, erfassen nur die menschlichen Akteure. Ziel ist, deren Anzahl zu reduzieren und damit sowohl die Fehleranfälligkeit zu minimieren als auch die Aufgaben-Ausführungszeit zu verkürzen.

Die CIP-Pool-Verantwortlichen möchten ihre Organisation professionalisieren und den CIP-Pool als zentrale Dienstleistung mit verschiedensten Diensten darstellen, wie z.B. Zugang zu Arbeitsplatzrechnern oder Zugang zum Wiki. Unsere Aufgabe war, ein Benutzerverwaltungssystem zu programmieren, mit dem die Aufgaben des täglichen Ablaufs ohne besonderes technisches Wissen erledigt werden kann. Vergabe und Verwaltung von CIP-Pool-Benutzerkonten, Rollen und relevanten Diensten sollen an einer einzigen zentralen Stelle erfolgen.

Das Institut für Statistik bietet folgende Dienste für seine CIP-Nutzer an: Windows, Linux und Wiki Zugänge. Zusätzlich wurde ein neuer Dienst - die Webanwendung GForge [9] - im Rahmen unseres Praktikums angefordert. Dieses muss installiert und konfiguriert und in das neue Benutzerverwaltungssystem eingebunden werden. Man möchte, dass die neue Anwendung mit vorhandenen Systemen und Diensten kooperiert, so dass keine grundlegenden Veränderungen an der existierenden Infrastruktur vorgenommen werden müssen. Der Zugang zu diesen Diensten soll auf Rollen basieren, die in diesem Benutzerverwaltungssystem angenommen werden können.

Wir betrachten nun die selben Anwendungsfälle wie im Kapitel Ist-Zustand beschrieben und vergleichen die alten und neuen Abläufe. Auslöser, Vor- und Nachbedingung sowie kurze Beschreibung sind in jedem Anwendungsfall gleich geblieben. Es werden nur die Unterschiede betrachtet.

Abbildung 2.2 zeigt, wie der Ablauf bei der Vergabe eines CIP-Benutzerkontos aussehen soll.

Dieser Anwendungsfall ist auch in Tabelle 2.6 beschrieben. Hier, sowie bei den anderen Anwendungsfällen (Tabellen 2.7-2.10), wird die Anlage von einem Benutzerkonto nur noch von einem Administrator erledigt.

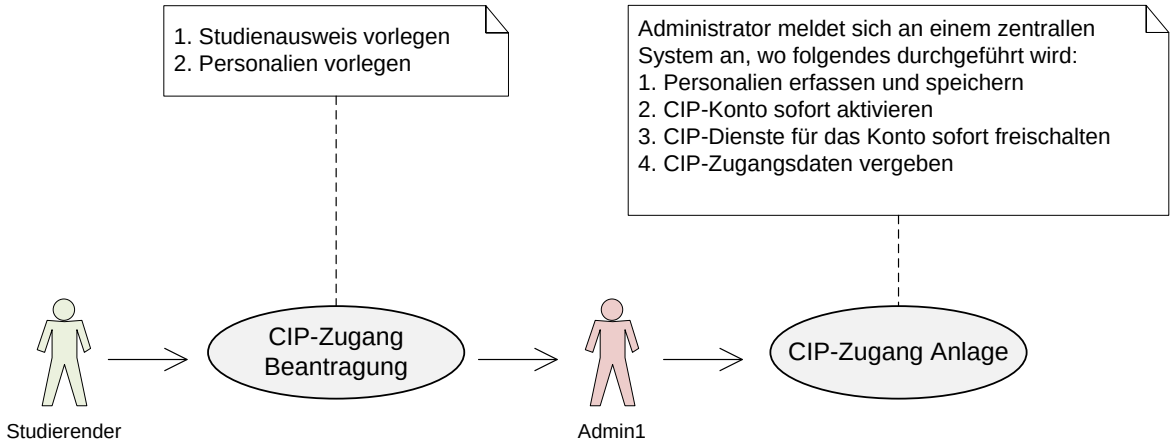

Abbildung 2.2: Soll-Zustand bei der Vergabe von CIP-Pool Benutzerkonto

Anwendungsfallname	**Neuanlage eines Benutzerkontos**
Beteiligte Akteure	Studierende, Administrator1
Primärszenario	1. Der Studierende kommt zu Administrator1. 2. Der Studierende legt seine Personalien vor. 3. Administrator1 trägt seine Daten in einer Eingabemaske ein und spezifiziert, welche Dienste für das Konto aktiviert werden sollen. 4. Administrator1 generiert das Benutzerkonto. 5. Administrator1 informiert den Studierenden über seine Anmeldedaten, mit denen er sich gleich anmelden kann.

Tabelle 2.6: Soll-Zustand: Neuanlage von einem Benutzerkonto

Anwendungsfallname	**Änderung eines Passwortes**
Beteiligte Akteure	Studierende, Administrator1
Primärszenario	1. Der Studierende kommt zum Administrator1. 2. Der Studierende legt seine Personalien vor. 3. Administrator1 ändert sein Passwort für alle Dienste, die für sein Konto aktiviert sind. 4. Administrator1 teilt dem Studierenden sein neues Passwort mit.

Tabelle 2.7: Soll-Zustand: Änderung eines Passwortes

Anwendungsfallname	**Freischaltung des zusätzlichen Dienstes für schon existierendes Benutzerkonto**
Beteiligte Akteure	Studierende, Administrator1
Primärszenario	1. Der Studierende kommt zum Administrator1. 2. Der Studierende legt seine Personalien vor. 3. Administrator1 schaltet für den Studierenden den neuen Dienst frei.

Tabelle 2.8: Soll-Zustand: Freischaltung des zusätzlichen Dienstes für schon existierendes Konto

Anwendungsfallname	**Deaktivierung eines Dienstes für ein Benutzerkonto**
Beteiligte Akteure	Administrator1
Primärszenario	1. Administrator1 deaktiviert der Dienst für das Benutzerkonto.

Tabelle 2.9: Soll-Zustand: Deaktivierung eines Dienstes für ein Benutzerkonto

Anwendungsfallname	**Löschung eines CIP-Benutzerkontos**
Beteiligte Akteure	Administrator1
Primärszenario	1. Administrator1 löscht ein CIP-Benutzerkonto.

Tabelle 2.10: Soll-Zustand: Löschen von einem CIP-Benutzerkonto

2.3 Konzept

Die wichtigste Voraussetzung für die Realisierung des Benutzerverwaltungssystems ist die Entwicklung des Konzepts. Das Konzept muss den CIP-Pool als Dienstleistungssystem so nah wie möglich abbilden und die Soll-Zustand Use-Cases definiert in Tabellen 2.6, 2.7, 2.8, 2.9, 2.10 realisieren. Der CIP-Pool umfasst eine Reihe von Diensten, die für die Nutzer angeboten werden. Die Nutzer werden in verschiedene Kategorien ("Rollen") unterteilt und bekommen den Zugriff zu entsprechenden Diensten. Um CIP-Nutzern den Zugriff zu den Anwendungen zu ermöglichen, die für ihre Arbeit/Studium benötigt werden, muss zunächst dieser Nutzer im Benutzerverwaltungssystem erfasst werden. Darüberhinaus müssen entsprechende Benutzerkonten angelegt und die benötigten Dienste freigeschaltet werden.

2.3.1 Begriffe und Definitionen

In Folgenden definieren wir einige abstrakte Begriffe, Momente und deren Zusammenhänge, die als Basis-Elemente für die Entwicklung unseres Systems dienen.

Service Ein Service ist die Zugriffskontrolle für einen bestimmten Dienst, wie z.B. Windows. Installation und Verwaltung der Dienste werden an anderer Stelle im Departement erledigt und sind nicht in unserem Benutzerverwaltungssystem vorgesehen.

Ganz allgemein hat jeder möglicher Benutzer entweder Zugriff auf einen bestimmten Dienst oder nicht. Das heißt, dass ein Service für einen Benutzer entweder aktiviert ist oder deaktiviert ist. Sobald ein Service für einen Benutzer aktiviert ist, können weitere Diensteigenschaften verändert werden, beispielsweise durch eine Änderung des Passwortes.

Services können von anderen abhängig sein, d.h. ein bestimmter Service kann nur dann für ein Benutzerkonto aktiviert werden, wenn die Services, von denen dieser abhängig ist, zuvor dafür freigeschaltet wurden.

Damit die Services festgelegt werden können, betrachten wir nun genauer, welche Schritte vom Administrator2 (siehe Ist-Zustand) durchgeführt wurden, um einem CIP-Nutzer den Zugang zu Windows und/oder Linux im CIP-Pool zu ermöglichen:

- Administrator2 wird aufgefordert, ein CIP-Nutzerkonto mit Windows-Zugang für einen Studierenden zu erzeugen
- Erzeugen eines Arbeitsverzeichnisses für das neue Benutzerkonto
- Festlegen der Größe des neuen Arbeitsverzeichnisses
- Erzeugen eines neuen Nutzers auf dem Domain Server

- Eintragen des Nutzers in die samba-Konfigurationsdateien (Windows-Zugang)
- Eintragen des Nutzers in die NIS-Konfigurationsdateien (Linux-Zugang)

Am oben geschilderten Vorgang sieht man, dass der Prozess in vier Phasen sinnvoll aufgeteilt werden kann: “Arbeitsverzeichnis”, “Domainserver-Nutzer”, “samba-Eintrag” und “NIS-Eintrag”. Diese Unterteilung ist notwendig, damit die Windows- und Linux-Zugänge jederzeit für einzelne CIP-Benutzer an- oder ausgeschaltet werden können. Die Benutzerdaten in den Arbeitsverzeichnissen müssen für beide Zugänge zur Verfügung stehen.

Folgende Dienste werden im System als Services abgebildet:

- Arbeitsverzeichnis im ZFS (**directory**): Zuständig für Erzeugen/Löschen von einem ZFS Arbeitsverzeichnis.
- Domainserver-Nutzer (**user**): Zuständig für Erzeugen/Löschen/Passwort ändern von einem Domainserver-Nutzer. Dieser Service ist vom Service *directory* abhängig.
- samba-Eintrag (**windows**): Zuständig für Erzeugen/Löschen/Passwort ändern von einem Eintrag in der samba Konfigurationsdatei auf dem Domainserver. Dieser Service ist vom Service *user* abhängig.
- NIS-Eintrag (**linux**): Zuständig für Erzeugen/Löschen/Passwort-ändern von einem Eintrag in den NIS Tabellen auf dem Domainserver. Dieser Service ist vom Service *user* abhängig.
- GForge-Nutzer (**gforge**): Zuständig für Erzeugen/Löschen/Passwort ändern von einem GForge-Nutzer. GForge beinhaltet eine Wiki-Anwendung, wodurch der alte Wiki-Dienst ersetzt wird.

Rolle Rollen sind eine Abbildung der verschiedenen CIP-Pool-Benutzertypen. Folgende Rollen werden im Benutzerverwaltungssystem für das Institut für Statistik festgelegt:

- Administratoren: Haben volle Zugriffskontrolle auf das Benutzerverwaltungssystem
- Statistik-Hauptfach: Studierende mit Hauptfach Statistik
- Statistik-Nebenfach: Studierende mit Nebenfach Statistik
- Beschäftigte: Mitarbeiter und Mitarbeiterinnen des Instituts für Statistik
- Externe: Zu dieser Rolle gehören Benutzerkonten mit gesonderte Freischaltung

Rollen bündeln und steuern den Zugriff zu einer Menge von Services, die für die Benutzerkonten dieser Rolle freigegeben werden.

Die Rolle “Administrator” unterscheidet sich von den anderen Rollen insofern, dass sie Zugriff für die Verwaltung des hier beschriebenen Systems haben, d.h. Benutzerkonten dieser Rolle können Services, Benutzerkonten und weitere Rollen erzeugen, verwalten und löschen.

Benutzerkonto Jeder CIP-Nutzer benötigt ein Benutzerkonto, das zu einer Rolle gehört. Damit bekommt er die entsprechenden Services, die für diese Rolle festgelegt sind.

Benutzerkonten gehören mindestens zu einer Rolle. Gehört ein Benutzerkonto zu mehreren Rollen, sind alle Services, die durch die verschiedenen Rollen angegeben sind, für das Benutzerkonto freizuschalten. Services können für das Benutzerkonto unabhängig von den Rollen deaktiviert oder aktiviert werden. Ein Nutzer, der zum Beispiel zur Rolle “Statistik-Hauptfach” gehört und damit keinen Zugiff auf einen Mitarbeiter-Service hat, kann für diesen ausnahmsweise freigeschaltet werden. Abbildung 2.3 zeigt die Beziehungen zwischen den definierten abstrakten Elementen.

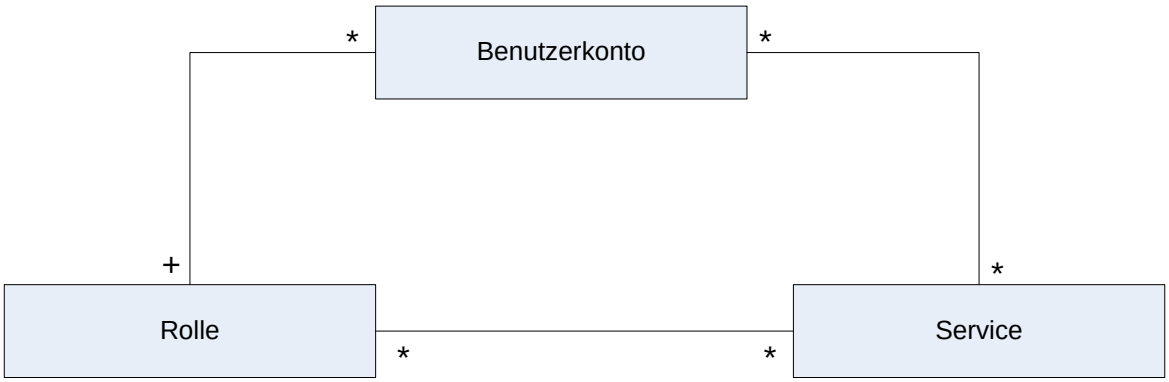

Abbildung 2.3: Beziehungen zwischen Baukomponenten des Benutzerverwaltungssystems

Ein Benutzerkonto, das zu einer oder mehreren Rollen gehören kann, bekommt die Freischaltung für mehrere Services. Ein Service kann für beliebige Benutzerkonten de-/aktiviert werden und kann mehreren Rollen zugewiesen sein.

Services, Rollen und Benutzerkonten sowie deren Zusammenhänge werden wir weiter als Basis-Elemente für die Entwicklung des Benutzerverwaltungssystems benutzen. Zunächst sollen diese Elemente mit ihren Attributen beschrieben werden. Diese Attribute sind grundlegend für das Datenbankschema des Systems (siehe Abschnitt 2.3.3).

Im Bereich der Benutzerkonten werden die Daten der Personen erfasst, die den CIP-Pool nutzen. Jedes Benutzerkonto wird durch Name, Vorname, Kontaktdaten beschrieben. Benutzername, Password und Rolle werden vom Department vergeben. Studierende haben - im Unterschied zu den Mitarbeitern - eine Reihe von ihr Studium betreffende Informationen (Matrikelnummer, Semester, Fach), wodurch ein oder mehrere Studiengänge mit Studienbeginn festgehalten sind. Die Rollen und die Services werden durch Namen, allgemeine Beschreibung und andere systemspezifische Informationen beschrieben. Die Rollen stehen

mit einer Reihe von Services in Verbindung. Die Services sind voneinander abhängig, die Abhängigkeiten müssen im System abgebildet werden.

2.3.2 Allgemeiner Entwurf

Das Benutzerverwaltungssystem besteht aus drei Komponenten - Frontend, Backend und Datenbank. Für die Eingabe der Daten braucht man eine Benutzerschnittstelle: Frontend. Hier bearbeiten die Administratoren die Benutzerinformationen, die in der Datenbank gespeichert werden. Das Frontend übergibt die eingegebenen servicebezogenen Informationen an das Backend zur Verarbeitung. Das Backend hat die Aufgabe, den Zugriff von den Benutzerkonten zu den reellen Diensten zu ermöglichen, die durch die Services beschrieben sind.

Abbildung 2.4 stellt den Entwurf des neuen Systems allgemein graphisch dar.

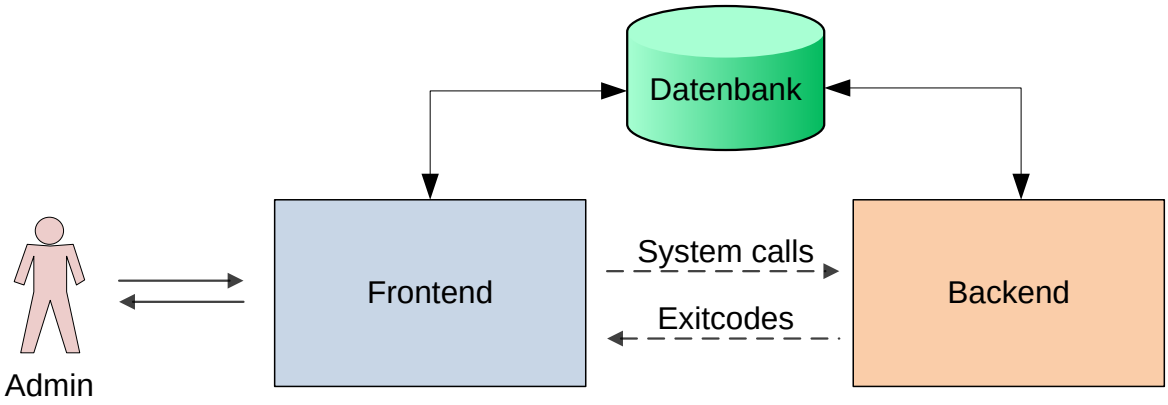

Abbildung 2.4: Konzept

Entwurf und die Implementierung von Frontend und Backend wird in getrennten Kapiteln vorgestellt. Das Frontend wurde von Viktoriya Serbina entwickelt, das Backend von Alexander Velkov. Wie die Datenbank von beide Komponenten benutzt wird, beschreibt das nächste Kapitel zum Datenbankschema. Darauf folgt die Beschreibung der Kommunikationsschnittstelle zwischen Frontend und Backend.

Das Backend stellt die Schnittstelle zur Verfügung, mit welcher die Kommunikation mit dem Frontend stattfindet. Die Kommunikation zwischen Frontend und Backend erfolgt mittels System-Aufrufen (System Calls). Das Backend *antwortet* auf ein System Call mit einem Exitcode. Dieser Exitcode gibt an, ob das Durchführen der angestoßenen Aktion erfolgreich war (Exitcode=0) oder nicht (Exitcode=1). Falls ein Fehler in der Ausführung auf der Backend Seite aufgetreten ist, soll das Backend diese Information in Form von Log-Dateien bereitstellen. Die Backend Schnittstelle definiert Informationen, die für die korrekte Ausführung der gewünschten Aktion bereitgestellt werden müssen. Die Backend API definiert folgende Parameter, deren Übergabe zu vollziehen ist:

- *Service Name* - um welchen Service handelt es sich,
- *Aktion* - welche Aktion muss durchgeführt werden,
- *Benutzername* - für wen ist diese Aktion durchzuführen,
- *Passwort* - gegebenenfalls das Passwort dieses Benutzers.

Der *Service Name* ist einer der fünf bereits definierten Service Namen (siehe Seite 12):

- directory
- user
- windows
- linux
- gforge

Der *Benutzername* muss zu einem Benutzerkonto in der AMASI- Datenbank gehören. Möglicherweise reicht der Benutzername allein für die korrekte Ausführung der Aktion aber nicht aus. Falls das Backend mehr Informationen über einen Benutzer benötigt, können diese in der AMASI-Datenbank ausgesucht werden. Eine Voraussetzung dafür ist, dass die nötigen Informationen in der Datenbank bereits abgespeichert sind. Für diese Aufgabe sorgt das Frontend des Benutzerverwaltungssystems. Der Benutzername ist der Primärschlüssel der Account Tabelle in der Datenbank. Damit können alle Benutzerinformationen eindeutig gefunden werden. (siehe Abschnitt 2.3.3)

Die Werte der Parameter *Aktion* sind fest definiert:

- ACTIVATE - um einen Service für ein Benutzerkonto frei zu schalten,
- DEACTIVATE - um einen Service für ein Benutzerkonto auszuschalten,
- PASSWORD - um das Passwort von einem Benutzer für einen Service zu ändern.

ACTIVATE- und DEACTIVATE-Aktionen sind in der Regel für jeden Service definiert. Dies ist die direkte Abbildung der Realität, denn der Zugriff für einen Dienst ist für einen Benutzer entweder eingeschaltet oder ausgeschaltet.

Für die meisten Dienste wird ein Passwort in Kombination mit dem Benutzernamen benötigt. Ziel für die Nutzung aller angebotenen Dienste in der CIP-Infrastruktur ist die Anmeldung mit immer den gleichen Login-Daten. Die Benutzerkonto-Passwörter werden gehasht mit dem Algorithmus SHA1 und in der Datenbank abgespeichert. Die verschiedenen Dienste benutzen

verschiedene Hash-Funktion-Algorithmen, um die Passwörter eventuell in eigenen Datenbanken abzuspeichern. Der Wert in der AMASI-Datenbank kann nicht in dieser Form den verschiedenen Diensten übergeben werden. Deshalb muss das Passwort in Clear-Text als weiterer Parameter an das Backend übergeben werden, falls es für die Ausführung der gewünschten Aktion erforderlich ist. Dies ist der Fall bei Freischaltung oder Änderung des Passwortes für einen Service (Aktionen ACTIVATE und PASSWORD).

2.3.3 Datenbank

Wir definieren hier die Struktur der Datenbank ausgehend von Informationen über und Beziehungen zwischen den Basis-Elementen *Benutzerkonto*, *Rolle* und *Service*.

Die Informationen und Beziehungen zwischen den Elementen werden in einer relationalen Datenbank gespeichert. Das ergibt ein Datenbankschema, wie es in Abbildung 2.5 grafisch dargestellt ist. Die genaue Bedeutung der einzelnen systemspezifischen Felder wird im späteren Verlauf dieser Arbeit erklärt.

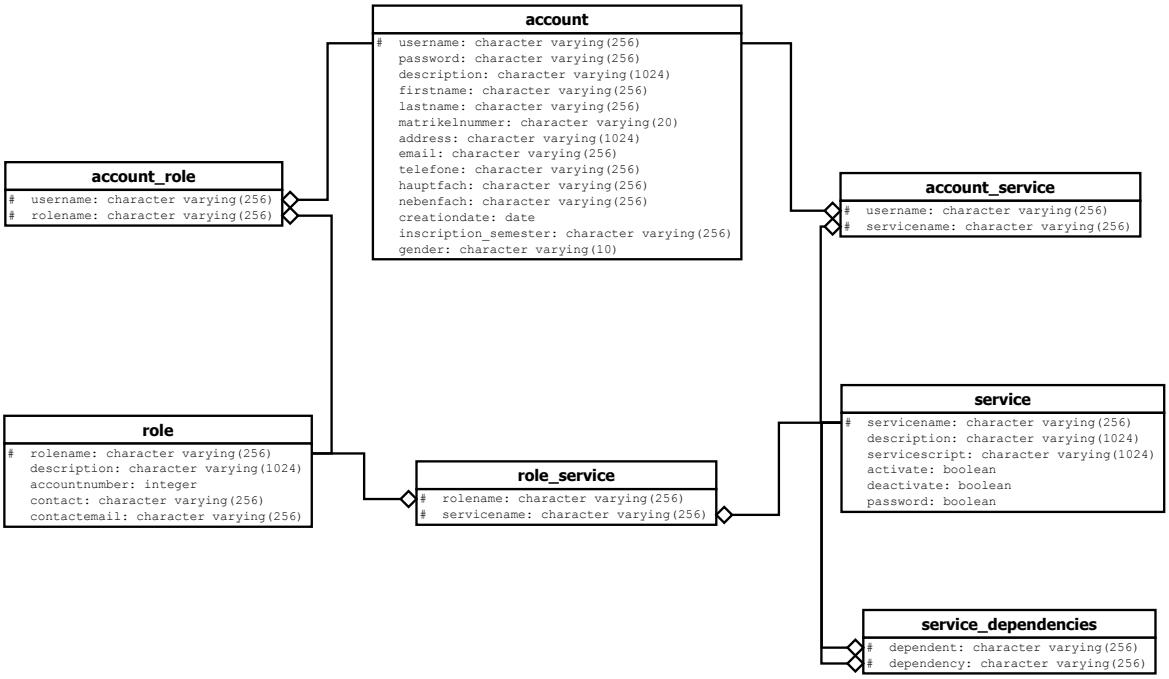

Abbildung 2.5: AMASI Datenbankschema

Jedem Basis-Element entspricht eine Tabelle in der Datenbank. Es wird durch seinen Namen eindeutig identifiziert, diese Namen sind als Primärschlüssel festgelegt. Die Beziehungen zwischen den Basis-Elementen sind durch zusätzliche Tabellen realisiert. Diese Tabellen sind account_role (zwischen Benutzerkonto und Rolle), account_service (zwischen Benutzerkonto und einem Service), role_service (zwischen Rolle und Service) und service_dependencies (Abhängigkeiten zwischen Services). Diese Tabellen haben als Primärschlüssel die Zusammensetzung der Fremdschlüssel der entsprechenden Basis-Elemente Tabellen.

3 Frontend

Mit dem Frontend wird der Teil unserer Anwendung bezeichnet, mit der der Benutzer direkt interagiert und in welchem die Logik der Informationsteuerung und Datenspeicherung implementiert ist.

3.1 Entwurf

Die Anwendung besteht aus zwei Teilen. Eine wird von den Administratoren für die Verwaltung von Nutzerkonten im CIP-Pool-Umgebung am Institut für Statistik verwendet. Die zweite wird von den Nutzern für die Passwortänderung benutzt.

Das Frontend muss zwei benutzerfreundliche Schnittstellen zur Verfügung bereitstellen (siehe Abbildung 3.1):

- eine nur für die Administratoren, wo sie die Elemente anlegen, ändern, löschen und suchen können (admin);
- eine für die CIP-Nutzer, wo sie ihr Passwort ändern können (public);

Die Aufgabe des Frontends ist, den Aufbau der beiden Schnittstellen sowie die Kommunikation mit Datenbank und Backend technisch und logisch zu beschreiben.

Wie man in Abbildung 3.1 sehen kann, haben wir uns mit der Realisierung des Frontends für eine Anwendung entschieden, die plattformunabhängig ist und die ganze Applikationslogik übernimmt. Um die CIP-Benutzerkonten zu erstellen und zu verwalten, wird dem Administrator Interface (admin) zur Verfügung gestellt, mit dessen Hilfe er das System ansprechen kann. Mit Hilfe dieser grafischen Oberfläche kann er die Daten eintragen, anschauen und verwalten. Auch für die CIP-Nutzer steht ein Interface (public) zur Verfügung, durch welches sie das Passwort ändern können. Bei beiden Interfaces sollen die Eingaben auf Korrektheit überprüft werden können.

Nach Überprüfung auf Korrektheit der eingegebenen Werten werden die Informationen in der Datenbank gespeichert (siehe Kapitel 2.3.3). DB-Handler leitet diese Daten an die Datenbank weiter (DB-Handler bietet eine Schnittstelle, die die Kommunikation zwischen Frontend und Datenbank ermöglicht).

Servicebezogene Informationen müssen an das Backend zur Verarbeitung übergegeben werden, was als Service Handler realisiert wird. Dieser bietet eine Schnittstelle, die die Kommunikation zwischen Frontend und Backend ermöglicht (siehe Kapitel 2.3.2).

Abbildung 3.1 stellt den ganzen Frontend-Entwurf nochmal grafisch dar.

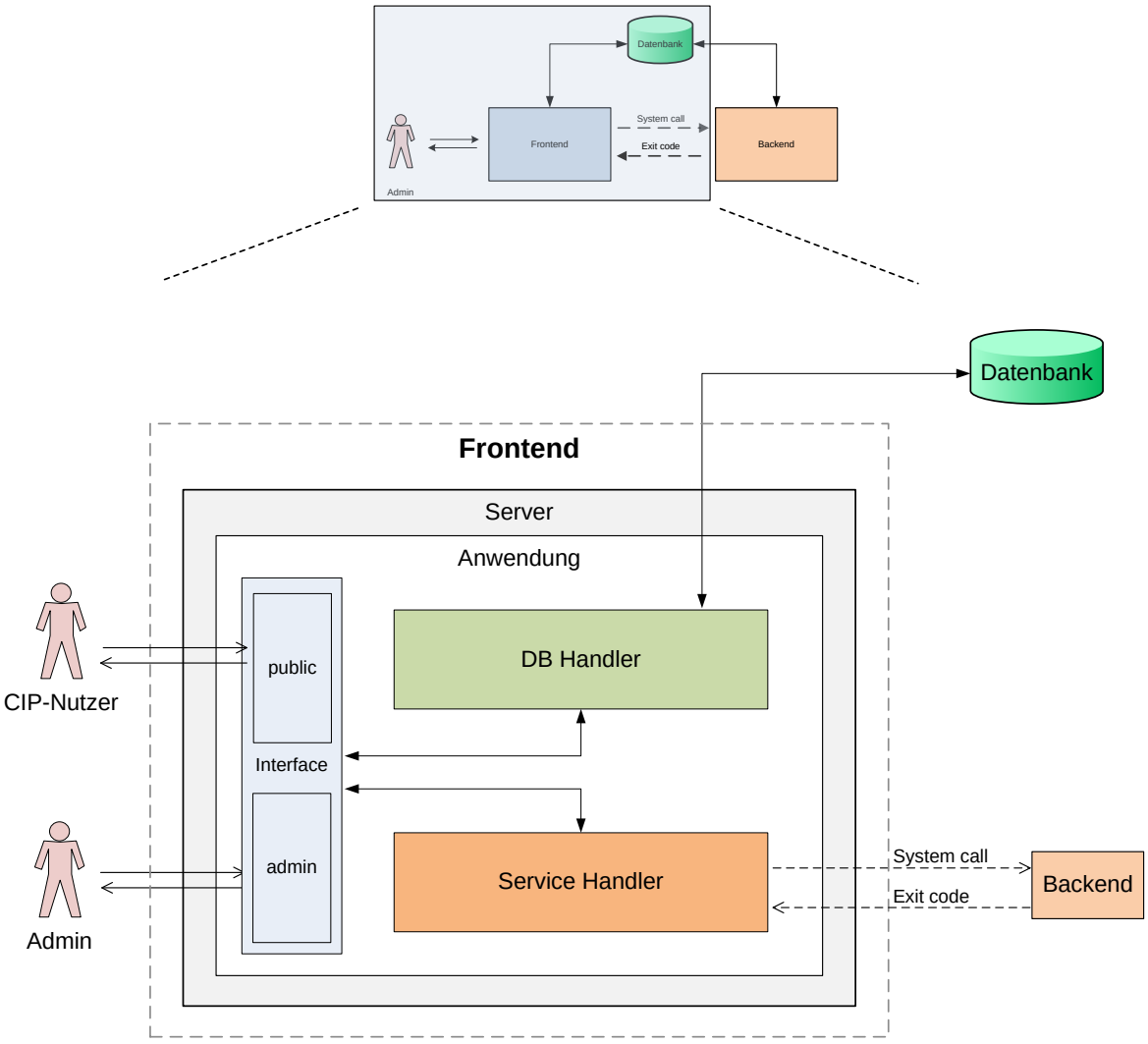

Abbildung 3.1: Entwurf des Frontends

Wie in Kapitel 2.3.1 erklärt wurde, soll es möglich sein, die folgenden Basiselemente durch das Interface zu verwalten: Benutzerkonten, Rollen, Gruppenkonten und Services.

Auf jedem dieser Elemente können folgende Aktionen angewendet werden:

- Neuanlage: Bei Neuanlage werden, dem Basis-Element entsprechend, bestimmte Felder ausgefüllt und nach der Überprüfung in der Datenbank gespeichert. Für bestimmte Elemente können auch die entsprechenden Backend-Aktionen ausgeführt werden.

- Änderung: Bei einer Änderung werden die schon angelegten Elemente geändert. Hier können jedoch die Namen von Elementen nicht geändert werden, da sie als Schlüssel in der Datenbank definiert sind.

- Suche: Durch die Filtersuche können die Basiselemente der Anwendung gefunden und einzeln modifiziert oder gelöscht werden. Für jedes Basis-Element werden eigene Suchkriterien angeboten, aber ansonsten läuft die Suche bei jedem Basis-Element gleich ab.

- Löschung: Bei einer Löschung wird ein Basis-Element aus der Datenbank entfernt.

Nach jeder Aktion wird am Ende das Ergebnis zusammengefasst. Dabei werden immer die relevanten Informationen dargestellt, die darüber berichten, was erfolgreich/nicht erfolgreich durchgeführt ist (z. B. wird für die Neuanlage von einem Benutzerkonto immer angezeigt, ob die Aktivierung von Services erfolgreich abgeschlossen ist).

Bei Neuanlage und Änderung werden die eingegebenen Werte auf Korrektheit überprüft. Die Eingabemasken für Neuanlage und Änderung sind gleich.

Im Weiterem wird auf jedes Basis-Element mit seinen spezifischen Aktionen näher eingegangen.

Services

Unter Services werden die Dienste, die im Kapitel 2.3.1 erklärt sind, angelegt und verwaltet.

- Bei Neuanlage müssen Servicename und Pfad zum Service Skript (Backend Schnittstelle), das für diesen Dienst entsprechend vom Backend ausgeführt wird, Aktionen und Serviceabhängigkeiten angegeben werden. Die Aktionen sind: Activate (Dienst-Aktivierung), Deactivate (Dienst-Deaktivierung) und Password (Passwortänderung zwingend für den “Dienst”). Falls eine Aktion für den Dienst nicht relevant ist, z.B. Passwortänderung, wird sie nicht ausgewählt. Diese Aktionen werden mit Hilfe von Service-Handler für die Benutzerkonten an Backend weitergeleitet, der diese auszuführen hat.

  Aus den schon angelegten Services kann man auswählen, von welchen der Dienst abhängig sein soll, d.h. welche Dienste für ein Benutzerkonto aktiviert/deaktiviert werden müssen, bevor es selbst aktiviert/deaktiviert wird. Diese Abhängigkeit gilt z.B. für die Services user und directory. D.h. dass das Arbeitsverzeichnis im ZFS (directory) zuerst angelegt werden muss, danach der Domainserver-Nutzer (user) selbst. Dementsprechend wird erst nach der Deaktivierung von Service user directory deaktiviert.

- Nach Aufruf der Löschfunktion, muss der Vorgang noch bestätigt werden. Danach wird der Service zuerst für die Benutzerkonten, für die dieser Service aktiviert war, deakti-

viert. Es werden dabei die Serviceabhängigkeiten berücksichtigt. Erst wenn die Deaktivierung für die Benutzerkonten erfolgreich abgeschlossen ist, werden die Abhängigkeiten zwischen Service und Rollen, Service und Benutzerkonten sowie der Service selbst aus der Datenbank gelöscht.

Rollen

Rollen sind eigentlich die abstrakten Begriffe verschiedener CIP-Pool- Benutzertypen. Sie sind im Kapitel 2.3.1 aufgelistet. Die Rolle Administrator ist es Spezielles, da das Benutzerkonto mit dieser Rolle den Zugriff zur admin-Anwendung besitzt. Von daher muss mindestens ein Benutzerkonto mit dieser Rolle existieren. Zur public-Anwendung haben die Nutzer unabhängig von ihren Rollen Zugriff.

- Bei Neuanlage einer Rolle sind Rollename, Name und E-Mail-Adresse der betreffenden Person einzugeben, sowie die Services, die deren Benutzerkonten bekommen müssen.

- Falls bei einer Änderung die Services hinzugefügt oder entfernt werden, werden sie auch für die Benutzerkonten, die schon zu dieser Rolle gehören, entsprechend aktiviert oder deaktiviert. Die Serviceabhängigkeiten werden dabei berücksichtigt. Falls das Benutzerkonto nicht nur zu einer Rolle gehört, wird überprüft, welche Services die anderen Rollen besitzen. Ein Service wird dann deaktiviert, wenn zu ihm keine andere Rolle des Benutzerkontos gehört.

- Nach Aufruf der Löschfunktion, muss der Vorgang noch bestätigt werden. Danach werden zuerst die Services von Benutzerkonten, die zu dieser Rolle gehören, deaktiviert (allerdings nur die Services, die für diese Rolle gespeichert waren und falls keine andere Rolle von dem Benutzerkonto den Service besitzt). Falls die Deaktivierung erfolgreich durchgeführt wurde, werden die Benutzerkonten, die nur zu dieser Rolle gehört haben, die Abhängigkeiten zwischen Benutzerkonto und Service, Benutzerkonto und Rolle, Rolle und Service sowie auch die Rolle selbst aus der Datenbank gelöscht.

Gruppenkonten

Da im Institut für Statistik mehrere Konten als eine Einheit (z.B. Benutzerkonten für einen Kurs) angelegt und verwaltet werden müssen, wurde ein weiteres Element, das “Gruppenkonto” in unsere Anwendung eingeführt. Dieses ist eine spezifische Rolle mit dem Unterschied, dass bei der Neuanlage von Gruppenkonto außer Informationen, die bei den Rollen anzugeben sind, auch die Anzahl der Benutzerkonten angegeben wird. Diese Benutzerkonten werden zusammen mit dem Gruppenkonto angelegt. Die Benutzernamen sind aus Gruppenkontonamen und durchlaufender Nummer (1 bis angegebene Anzahl) gebildet. Sie bekommen die Services freigeschaltet, die für das betreffende Gruppenkonto ausgewählt wurden.

Bei Gruppenkontoänderung gibt es die Möglichkeit, die Anzahl der zugehörigen Benutzerkonten zu ändern. Falls die Anzahl reduziert wird, werden die letzten übrigen Benutzerkonten gelöscht. Falls die Anzahl erhöht wird, werden die nötigen Benutzerkonten zusätzlich erstellt. Ansonsten werden Suche, Änderung und Löschung genauso wie bei Rollen durchführt.

Benutzerkonten

Die Benutzerkonten entsprechen den Nutzern des CIP-Pools von dem Institut für Statistik und umfassen die Informationen über die dort gemeldeten Personen, ihre Anmeldedaten und die jeweils zu nutzenden Services.

- Bei Neuanlage eines Benutzerkontos sind folgende Informationen einzugeben:
  - Personalien: alle relevanten Daten zu einer Person: Vorname, Nachname und E-Mail sind Pflichtfelder; Adresse, Telefon, Matrikelnummer, Hauptfach, Nebenfach, Studienbeginn, Geschlecht und Bemerkungen optional.
  - Anmeldedaten: Benutzername und Kennwort. Der Benutzername muss eindeutig sein und wird für Studenten automatisch aus der Matrikelnummer und einem "s" vorne generiert, für die Mitarbeiter besteht er einfach aus dem Nachnamen. Das Password bekommt auch einen automatisch generierten Random-Wert. Beides kann aber bei Neuanlage geändert werden.
  - Rollen und Services: Wie oben erwähnt, sind die zur Auswahl gestellten Rollen und Services diejenigen, die schon vorher angelegt sind. Man kann ein Benutzerkonto ohne Services anlegen, es muss aber mindestens eine Rolle ausgewählt werden.
- Nach dem Aufruf von Löschfunktion, muss der Vorgang noch bestätigt werden. Danach werden die Services für dieses Konto deaktiviert. Falls die Deaktivierung erfolgreich durchführt wird, werden die Beziehungen zwischen Benutzerkonto und Service, Benutzerkonto und Rolle sowie das Benutzerkonto selbst aus der Datenbank gelöscht.

Anwendung für die Passwortänderung

Damit die CIP-Pool-Nutzer das Passwort für die Dienste, die vom Administrator für sie freigeschaltet sind, ändern können, wurde diese zusätzliche Anwendung implementiert. Dort hat der Nutzer nach der Anmeldung die Möglichkeit, sein Passwort zu ändern. Wenn der Nutzer sein Passwort ändert, wird dies für alle für ihn freigeschalteten Services erfolgen.

Diese Anwendung bietet momentan nur eine Funktion (Passwortänderung). Sie kann aber in Zukunft weiterentwickelt werden, sodass z.B. die Nutzer auch E-Mail-Adresse oder Adresse

ändern können.

3.2 Implementierung

In diesem Kapitel wird die Realisierung des Frontend-Entwurfs detaillierter beschrieben und auf die notwendigen Ressourcen für die Zusammenstellung der Projektumgebung näher eingegangen.

Das Frontend ist als eine Web-Anwendung mit dem Einsatz von folgenden Technologien realisiert: JSP, JSTL und benutzerdefinierte Tags, Hibernate, JavaScript. Der Vorteil der Web-Anwendung ist, dass sie plattformunabhängig ist, von überall steuerbar und auf sie mithilfe des Browsers zuzugriefen ist. Im Weiteren wird auf diese Technologien näher eingegangen.

JSP

Da unsere Applikation durch den Browser gesteuert wird, werden für die Darstellung JSP (Java Server Pages) eingesetzt. JSP [12] ist eine Technologie, die von Sun Microsystems entwickelt wurde und die Erzeugung von dynamischem Web-Inhalt ermöglicht. Unsere Anwendung ist auf mehreren .jsp-Dateien aufgebaut, die in Allgemeinem aus einem HTML-Grundgerüst mit JSTL und benutzerdefinierten Tags bestehen, z.B. welcome.jsp, index.jsp, sowie die .jsp-Dateien einzelner Menüpunkte (siehe Abbildung 3.2).

Abbildung 3.2:

Tomcat

Die JSP-Seiten müssen auf einem Applikationsserver ausgeführt werden. Hier wird Tomcat [24] als Web-Server eingesetzt. Tomcat ist ein Web-Container bzw. Applikationsserver, von ASF (Abk. für: Apache Software Foundation) entwickelt. Es implementiert die Spezifikation für Servlet und JSP der Firma Sun Microsystems und stellt eine Umgebung zur Ausführung von Java-Code auf Webservern bereit. Es ist auch eine freie Software, also aus dem Internet kostenlos herunterzuladen.

Mit Hilfe von JDBC Realms [1] bietet Tomcat eine Authentifizierung und Zugriffssteuerung. Ein Realm gibt Tomcat Informationen zu einer Datenbank, wo die Benutzernamen und Passwörter zu finden sind, die auf eine Applikation (oder mehrere Applikationen) zugreifen dürfen, und auch die Rollen, denen die gültigen Benutzer zugeordnet sind.

Die Konfiguration von JDBC Realm erfolgt in der Datei server.xml. In unserer Applikation werden die Benutzer ihren zugehörigen Parametern mit der Datenbank abgeglichen und so authentifiziert. Die Tabelle in der Datenbank heißt hier “account” und die Spalte “benutzername”. Der Benutzername soll zu einer Rolle “admin” gehören. Die Beziehungen zwischen Benutzernamen und Rolle sind in der Tabelle “account_role” gespeichert. Die Passwörter der Benutzer werden mit SHA (Secure Hash Algorithm) verschlüsselt.

Der Realm ist aber ein Teil bei der Authentifizierung mit Tomcat. Die weiteren Teile (login-config, security-role und security-constraints) sind in der Datei web.xml (siehe Listing 3.1) von unserer Arbeit enthalten.

```
<security-constraint>
            <web-resource-collection>
                    <web-resource-name>admin</web-resource-name>
                    <description>Accessible by authenticated users of
                    the admin role</description>
                    <url-pattern>*.jsp</url-pattern>
                    <http-method>GET</http-method>
                    <http-method>POST</http-method>
                    <http-method>PUT</http-method>
                    <http-method>DELETE</http-method>
            </web-resource-collection>
            <auth-constraint>
                    <description>These roles are allowed access
                    </description>
                    <role-name>Administratoren</role-name>
            </auth-constraint>
    </security-constraint>

    <login-config>
            <auth-method>FORM</auth-method>
            <realm-name>MyFirst Protected Area</realm-name>
            <form-login-config>
                    <form-login-page>/index.jsp</form-login-page>
                    <form-error-page>/errorLogin.jsp</form-error-page>
```

```
        </form-login-config>
    </login-config>

    <security-role>
        <description>AMASI administators role</description>
        <role-name>Administratoren</role-name>
    </security-role>
```

Listing 3.1: Abschnitt aus der web.xml-Datei

In `<security-constraint>` werden die Zugriffsrechte auf einen Bereich der Applikation definiert, der in `<web-resource-collection>` vorgegeben ist. In unserem Fall heißt die Anwendung admin (in web.xml von public-Anwendung - public), die Authentifizierung des Nutzers wird dann überprüft, wenn er eine .jsp-Seite innerhalb der Anwendung aufrufen will (`<url-pattern>`), wofür die HTTP-Methoden (`<http-method>`) wie POST, GET, PUT und DELETE angewendet werden.

¡login-config¿ ist das Element, das konfiguriert wurde, damit der Nutzer authentifiziert werden kann. In unserer Anwendung passiert die Authentifizierung durch HTML-Form in der Datei index.jsp. Diese Datei ist eine Einloggen-Form, die die Felder “Benutzername” und “Passwort” beinhaltet, die als “j_username” und “j_password” benannt werden müssen. Falls Benutzer erfolgreich authentifiziert wurde, wird er weiter zu welcome.jsp weitergeleitet. Bei Fehlschlagen der Authentifizierung (falsches Passwort eingegeben), kommt man zu error.jsp.

Einen Zugriff auf die Applikation haben nur Administratoren (`<auth-constraint>`). Falls der gültige Nutzer, der kein Administrator ist, sich anzumelden versucht, wird die Fehlerseite mit dem Status-Code 401 zurückgegeben.

Im Unterschied zur admin-Anwendung steht in `<auth-constraint>` in der public-Anwendung anstatt “Administratoren” nur ein *. Das bedeutet, dass jedes Benutzerkonto unabhängig seiner Rolle auf diese Anwendung zugreifen darf.

Hibernate

Als Persistenzmedium wurde eine Relationale Datenbank verwendet. In unserer Anwendung wollen wir keine SQL-Statements verwalten, sondern gewöhnliche Objekte mit Attributen und Methoden in der Datenbank speichern und aus entsprechenden Datensätzen wiederum Objekte erzeugen. Es sollen auch Beziehungen zwischen Objekten auf entsprechende Datenbank-Relationen abgebildet werden. Programmdaten in der Relationalen Datenbank zu speichern sowie die Daten aus diesem System zu laden, erfordert die Einführung einer weiteren Abstraktionsebene, weil objektorientierte Programmdaten und Relationale Datenbanksysteme unterschiedliche Konzepte zur Organisation von Daten umsetzen. Um dieses Problem zu lösen, verwendet man OR-Mapper (Objektrelationale Mapper). Statt für jede Anwendung eigene OR-Mapper zu schreiben, verwenden wir ein Persistenzframework.

Als Persistenzframework, das die Objekte zur Datenbank zwecks Speicherung weiterleitet, wird Hibernate [10] benutzt. Es steht unter eine Open-Source-Lizenz im Internet und kann daher lizenzkostenfrei eingesetzt werden. Hibernate funktioniert mit fast jeder relationalen Datenbank und kann wegen seiner Flexibilität und Leistungsfähigkeit in verschiedenen Projekten als Persistenzschicht eingesetzt werden. Daher wird Hibernate auch in diesem Projekt zur Persistenz von Java-Objekten verwendet.

In unserer Anwendung werden die Daten nicht mittels SQL-Statements gespeichert, sondern objektorientiert in der Datenbank. Hibernate bietet eine zentrale Konfigurationsdatei (hibernate.cfg.xml), um die Datenbank sowie die OR-Mappings zwischen Datenbank und Objekten darzustellen. In dieser Konfigurationsdatei werden verschiedene Einstellungen für Hibernate festgelegt, wie zum Beispiel SQL-Dialect, JDBC-Treiber und Zugangsdaten für die Datenbank. In folgender Listing 3.2 wird die Konfigurations-Datei für Hibernate aus der Implementierung dieser Arbeit gezeigt:

```
<?xml version='1.0' encoding='UTF-8'?>
<!DOCTYPE hibernate-configuration PUBLIC
          "-//Hibernate/Hibernate Configuration DTD 3.0//EN"

"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd">
<hibernate-configuration>
<session-factory>

    <property name="connection.url">jdbc:postgresql://localhost/amasi
    </property>
    <property name="connection.username">admin</property>
    <property name="connection.password"></property>

    <property name="connection.driver_class">org.postgresql.Driver
    </property>
    <property name="dialect">org.hibernate.dialect.PostgreSQLDialect
    </property>
    <property name="transaction.factory_class">
    org.hibernate.transaction.JDBCTransactionFactory</property>
```

Listing 3.2: Abschnitt aus der zentralen Hibernate-Konfigurationsdatei (hibernate.cfg.xml)

In Listing 3.3 ist der Ausschnitt aus der Konfigurationsdatei für unsere Applikation zu sehen. Sie zeigt das Mapping für das Benutzerkonto. Es gibt eine Java-Klasse “Account” und die Tabelle mit gleichem Namen in der Datenbank. Mit “id” wird der Primärschlüssel identifiziert. Die normalen Attribute sind in property-Tags den Tabellenspalten zugeordnet, z.B. entspricht das Attribut “password” mit Typ “string” der Spalte “password” in Datenbanktabelle “account”.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC "-//Hibernate/Hibernate Mapping
DTD 3.0//EN" "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
```

```
<hibernate-mapping package="de.lmu.stat.server.hibernate.mappingObjects" >

   <class name="Account" table="account">

     <id name="benutzername" column="benutzername" type="string" />

     <property name="password" type="string" column="password"/>
     <property name="description" type="string" column="description"/>
     <property name="vorname" type="string" column="vorname"/>
     <property name="nachname" type="string" column="nachname"/>
```

Listing 3.3: Abschnitt aus der Hibernate-Konfigurationsdatei für die Anwendung

Um diese Konfiguration zu ermöglichen, müssen die entsprechenden Tabellen in der Datenbank und denjenigen Java-Classen (Entities) angelegt werden. Hier sind es Benutzerkonten (Tabelle "account" in DB und Java-Klasse *Account.java*), Rollen (Tabelle "role" in DB und Java-Klasse *Role.java*), Services (Tabelle "service" in DB und Java-Klasse *Service.java*) und Serviceabhängigkeiten (Tabelle "service_dependencies" in DB und Java-Klasse *ServiceRelationship.java*).

Die zentrale Rolle spielt hier die Java-Klasse *HibernetUtils.java*. Hier werden Hibernet gestartet und alle relevanten Methoden für Speicherung, Löschung, Modifizierung und Suche in der Datenbank definiert. In der Klasse *HibernetUtils.java* sind folgende Methoden implementiert:

- `save()`-Methode bekommt ein Objekt als Parameter und speichert es in der Datenbank;
- `delete()`-Methode bekommt ein Objekt als Parameter und löscht es, falls es in der Datenbank existiert;
- `update()`-Methode bekommt ein Objekt als Parameter und aktualisiert die geänderten Werte eines Objekts in der Datenbank;
- `getFromDB()`-Methode bekommt als Parameter einen String id und eine Klasse und gibt ein Objekt zurück;
- `searchManyInDB()`-Methode bekommt als Parameter die Angaben für die HQL-Statements und gibt mehrere Objekte zurück, die diesen Angaben entsprechen.

JSTL und Benutzerdefinierte Tag Libraries

Für die Realisierung des Frontends werden in Ergänzung zu JSPs auch JavaServer Pages Standard Tag Library (JSTL [13]) und benutzerdefinierte Tag Bibliotheken benutzt. JSTL ermöglicht es, die JSPs vom Java-Code fernzuhalten und unterstützt Aufgaben, wie Darstellungslogik oder auch Iterationen über Arrays und Listen.

Für Backend und Datenbank wurden benutzerdefinierte Tag Bibliotheken angelegt. Sie bestehen aus drei Teilen:

1. Tag Handler (Kompilierte Java Datei): Es ist ein Objekt, das während der Ausführung von einer JSP-Seite aufgerufen wird, um einen benutzerdefinierten Tag zu evaluieren. Diese Java-Klassen müssen für ihre Attribute Get- und Set-Methoden beinhalten. Falls diese Klasse von der Klasse TagSupport erbt, müssen auch Methoden `doStartTag()` und `doEndTag()` überschrieben werden. Wenn der Start-Tag eines benutzerdefinierten Tags erreicht ist, wird der Tag Handler initialisiert und die Methode `doStartTag()` aufgerufen. Wenn der Ende-Tag erreicht ist, wird die Methode `doEndTag()` aufgerufen. Außerdem hat der Tag Handler eine API, die mit JSP-Seite zu kommunizieren erlaubt. Der Eingangspunkt zu dieser API ist ein PageContext-Objekt, durch welchen der Tag Handler auf alle impliziten Objekte von JSP-Seite zugreifen kann.

2. TLD-Beschreibungsdatei: Dies ist eine XML-Datei, wo der benutzerdefinierte Tag Bibliothek und Tags selbst beschrieben werden. Hier sollen die Tag-Namen, die Referenz auf die Java-Datei, die möglichen Parameter und zusätzliche Eigenschaften des Tags zusätzlich zu der Parameter definiert werden. Die TLDs werden von einem Web-Container verwendet, um die Tags zu validieren. Die Wurzel von TLD ist das taglib-Element. Hier in den Tag-Elementen werden Tags definiert. Jeder Tag wird durch eindeutigen Namen, Tag Handler sowie Body-Typ beschrieben. Auch die Tag-Attribute werden hier in attribute-Elementen definiert. Für jedes Attribut muss ein Name (name-Element) spezifiziert werden, ob es ein Pflichtattribut sein soll (required-Element) und ob der Wert durch einen Ausdruck ermittelt werden kann (rtexprvalue -Element). Z.B.:

   ```
   <attribute>
         <name>attr1</name>
         <required>true|false|yes|no</required>
         <rtexprvalue>true|false|yes|no</rtexprvalue>
   </attribute>
   ```

3. Codefragmente zur Einbindung in die JSP-Datei: Sie müssen zuerst eine Referenz auf die TLD-Datei und einen Taglib-Präfix definieren (`<%@ taglib uri="..." prefix="..." %>`). Die Verwendung erfolgt dann über HTML-Tags mit dem Taglib-Präfix und dem Tag-Namen (`<meinetaglib:MeinCustomTag ... >`).

In unserer Anwendung sind folgende benutzerdefinierte Tags definiert:

- Für die Datenbank (DB-Handler):

  – TagSaveAccount: Speichern eines Benutzerkontos in der Datenbank. In der TLD-Datei sind folgende Attribute für diesen Tag definiert: benutzername (erforderlich), description, password, vorname, nachname, matrikelnummer, adresse, tel, email, hauptfach, nebenfach, inscriptionsemester, roles, services, modify, gender. Als Tag Handler wurde die Klasse *TagSaveAccount.java* angegeben. Falls es sich um die Änderung des Benutzerkontos handelt, wird in der `doStartTag()`-Methode

erst die Methode `getFromDB()` aufgerufen, um das Benutzerkonto-Objekt in der Datenbank zu finden. Danach werden die entsprechenden Methoden (`save()` oder `update()`) aufgerufen, um dieses Objekt in der Datenbank zu speichern.

- TagDeleteAccount: Löschen eines Benutzerkontos aus der Datenbank. In TLD-Datei sind folgende Attribute für diesen Tag definiert: benutzername (erforderlich). Als Tag Handler wurde die Klasse *TagDeleteAccount.java* angegeben. Dort in der `doStartTag()`-Methode wird erst die Methode `getFromDB()` aufgerufen, um das Benutzerkonto-Objekt in der Datenbank zu finden und danach die Methode `delete()` aus, um dieses Objekt aus der Datenbank zu löschen.

- TagSaveRole: Speichern einer Rolle in der Datenbank. In TLD-Datei sind folgende Attribute für diesen Tag definiert: rolename (erforderlich), description, accountnumber, contact, contactemail, services, modify. Als Tag Handler wurde die Klasse *TagSaveRole.java* angegeben. Dort in der `doStartTag()` wird erst die Methode `getFromDB()` aufgerufen, um das Rolle-Objekt in der Datenbank zu finden, falls es sich um die Änderung von Rolle handelt. Danach werden die entsprechenden Methoden (`save()` oder `update()`) aufgerufen, um dieses Objekt in Datenbank zu speichern.

- TagDeleteRole: Löschen einer Rolle aus der Datenbank. In TLD-Datei ist folgendes Attribut für diesen Tag definiert: rolename (erforderlich). Als Tag Handler wurde die Klasse *TagDeleteRole.java* angegeben. Dort in der `doStartTag()`-Methode wird erst die Methode `getFromDB()` aufgerufen, um das Rolle-Objekt in der Datenbank zu finden und danach die Methode `delete()` aus, um dieses Objekt aus der Datenbank zu löschen.

- TagSaveService: Speichern eines Services in der Datenbank. In der TLD-Datei sind folgende Attribute für diesen Tag definiert: servicename (erforderlich), description, activate, deactivate, edit, password, servicescript, modify, dependencies. Als Tag Handler wurde die Klasse *TagSaveService.java* angegeben. Dort in der `doStartTag()`-Methode wird erst die Methode `getFromDB()` aufgerufen, um das Service-Objekt in der Datenbank zu finden, falls es sich um die Änderung vom Service handelt. Danach werden die entsprechenden Methoden (`save()` oder `update()`) aufgerufen, um dieses Objekt in der Datenbank zu speichern.

- TagDeleteService: Löschen eines Services aus der Datenbank. In der TLD-Datei ist folgendes Attribut für diesen Tag definiert: servicename (erforderlich). Als Tag Handler wurde die Klasse *TagDeleteService.java* angegeben. Dort in der `doStartTag()`-Methode wird erst die Methode `getFromDB()` aufgerufen, um das Service-Objekt in der Datenbank zu finden und danach die Methode `delete()` aus, um dieses Objekt aus der Datenbank zu löschen.

- TagGetRoleServices: gibt die entsprechenden Services für die ausgewählten Rollen zurück. In TLD-Datei sind folgende Attribute für diesen Tag definiert: roles (erforderlich), key (erforderlich). Als Tag Handler wurde die Klasse *TagGetRoleServices.java* angegeben. Dort in der `doStartTag()`-Methode werden die Services

für die angegebenen Rollen in der Datenbank gefunden und in einem Vektor gespeichert.

  - TagFind: Suche in der Datenbank. In TLD-Datei sind folgende Attribute für diesen Tag definiert: what (erforderlich), filterProperty, filterMatch, filterValue, vectorName, listName, orderby, object. Als Tag Handler wurde die Klasse *TagFind.java* angegeben. Dort in der `doStartTag()`-Methode werden die Werte für die HQL-Anfrage erfasst und als Parameter in der Methode `searchManyInDB()` weitergegeben. Die resultierenden Daten werden in einem Vektor in PageContext gespeichert.

- Für Backend (Service-Handler):

  - TagExecuteScript: Ausführen eines Backend-Skripts. In der TLD-Datei sind folgende Attribute für diesen Tag definiert: benutzername (erforderlich), password (erforderlich), servicename (erforderlich) und action (erforderlich). Als Tag Handler wurde die Klasse *TagExecuteScript,java* angegeben. Dort in der `doStartTag()`-Methode werden die Backend Skripte entsprechend den übergebenen Parametern ausgeführt.

  - TagTopologicalOrder: Sortierung einer Reihenfolge nach bestimmten Abhängigkeiten. In TLD-Datei sind folgende Attribute für diesen Tag definiert: services (erforderlich), resultKey (erforderlich), reverse. Als Tag Handler wurde die Klasse *TagTopologicalOrder.java* angegeben. Dort in der `doStartTag()`-Methode werden die Services topologisch sortiert. Für die topologische Sortierung wird eine Bibliothek TopologicalSort vom Hibernate (org.hibernate.envers.tools.graph.TopologicalSort) benutzt.

Außer benutzerdefinierten Tags wurden auch die EL (Expression Language) Funktionen für JSPs definiert. Als Tag Handler für beide Funktionen wurde die Klasse *AmasiElFunctions.java* angegeben. In TLD-Datei sind folgende Funktionen beschrieben:

- `specialContains()` bekommt zwei String-Parameter und liefert einen Boolean-Wert zurück. Das Ziel der specialContains-Funktion ist zu bestimmen, ob sich ein Wert innerhalb einer komma-separierten Liste befindet. In dieser Funktion ist der erste Parameter der gesuchte Wert, der zweite ist ein String, der innerhalb dieser Funktion in eine Liste umgewandelt wird.

- `regexpMatches()` bekommt zwei String-Parameter und liefert einen Boolean-Wert zurück. Die Funktion regexpMatches prüft, ob ein Wert dem regulären Ausdruck entspricht. Hier sind die Parameter sowohl der Wert selbst als auch der reguläre Ausdruck.

- `encryptText()` bekommt folgende Parameter: text (erforderlich), key (erforderlich) und method. Mit dieser Funktion wird der angegebene Text (text) verschlüsselt.

JavaScript und Tag Bibliotheken für die Überprüfung der Felder

Für die Überprüfung von eingetragenen Daten auf Korrektheit werden JavaScript und JSTL benutzt. JavaScript wird jedes Mal für die Überprüfung von Pflichtfeldern und die richtige Schreibweise (z.B. E-Mail) eingesetzt.

Mit Hilfe von JSTL wird überprüft, ob der Datensatz mit gleichem Schlüssel schon in der Datenbank existiert. Dafür wird TagFind benutzt, der in der Datenbank schaut, ob ein Eintrag mit dem angegebenen Schlüssel existiert. Die Fehler werden entsprechend kenngezeichnet.

Logging

Um den Durchlauf der Applikation zu beobachten und mögliche Fehler zu erkennen, haben wir Logging benutzt. Dafür wurde das Java Logging-Bibliothek Log4j [15] ausgewählt. Die Konfiguration erfolgt über die Konfigurationsdatei log4j.cfg, wo die Eingaben zur Ausgabe-Stufe von Meldungen (hier INFO-Level allgemeine Ausgaben) und Appender (Datei, wohin die Standartausgaben geschrieben werden) definiert werden (siehe Listing 3.4). Die Logging Statements wurden im Quellcode eingefügt und die Ausgabe in einer .log Datei gespeichert, damit sie zu einem späteren Zeitpunkt analysiert werden könnten.

```
log4j.rootLogger=INFO, R

log4j.appender.R=org.apache.log4j.RollingFileAppender
log4j.appender.R.File=/root/amasi/log/amasi_webapp.log
log4j.appender.R.MaxFileSize=1MB
log4j.appender.R.MaxBackupIndex=10
log4j.appender.R.layout=org.apache.log4j.PatternLayout
log4j.appender.R.layout.ConversionPattern=
%d{dd MMM HH:mm:ss} %-5p (%F %M:%L) - %m%n

log4j.logger.de.lmu.stat=DEBUG

log4j.logger.org.hibernate.SQL=INFO
```

Listing 3.4: Ausschnitt aus der Konfigurationsdatei log4j.cfg

JUnit Tests

In der Test-Phase wurde das Framework JUnit [14] zur Durchführung von Black-Box-Tests benutzt. Das Ziel ist, durch automatisierte Tests die DB-Speicherung zu prüfen, da sich schnell erkennen lässt, ob ein Fehler passiert ist. JUnit bietet die Möglichkeit, gezielt zu testen. Darüber hinaus werden Fehler weitaus schneller erkannt und können kostengünstiger

behoben werden. Bei jeder Datenbank-Änderung,oder auch bei Änderungen in den Java-Entities (*Account.java*, *Rolle.java* und andere) oder einer Hibernate-Konfiguration sollen die JUnit-Tests erweitert und angestoßen werden, ohne das ganze Projekt auf dem Tomcat Container einzusetzen.

4 Backend

Das Backend umfasst Logik und Bearbeitung der Anbindung zwischen den Diensten und den AMASI Accounts. Das Backend steuert den Zugriff von Benutzern auf alle zur Verfügung stehenden Dienste.

4.1 Entwurf

Das Backend wird vom Frontend aufgerufen. Das Frontend hat zuvor alle nötigen Informationen über Benutzer und Service in der AMASI-Datenbank abgespeichert. Das Backend muss in der Lage sein, sich die für ihn relevanten Informationen aus der Datenbank zu holen und die gewünschte Aktion durchzuführen. Dabei wird hauptsächlich mit externen Systemen kommuniziert, d.h. die Durchführung jeglicher Aktionen zu protokollieren ist unerlässlich, damit alle Backend Vorgänge jederzeit nachvollziehbar werden. Besonders im Fehlerfall ist es nötig, möglichst viel Information bezüglich des Vorfalls herausfinden zu können. Außerdem soll es einfach sein, das Backend zu erweitern, d.h. neue Service-Implementierungen zu integrieren.

Hier noch einmal die wichtigsten Entwicklungsmerkmale des Backends:

- eine einheitliche Schnittstelle anbieten
- leicht erweiterbar sein
- benötigte Informationen aus der Datenbank holen können
- die gewünschte Aktion durchführen und das Resultat zurückmelden
- Protokollierung jeglicher Vorgänge

Das Backend ist getrennt und unabhängig vom Frontend. Diese Modularität hat viele Vorteile. Für die Implementierung von weniger oft durchgeführten Aktionen ist es wünschenswert, das Frontend nicht extra dafür anpassen zu müssen. Stattdessen können diese mit einem Skript leicht realisiert werden, indem die Backend Schnittstelle direkt angesprochen wird. Der spätere Prozess der Migration zum neuen System ist damit leichter zu realisieren und wird im Kapitel “Migration” genauer erklärt werden.

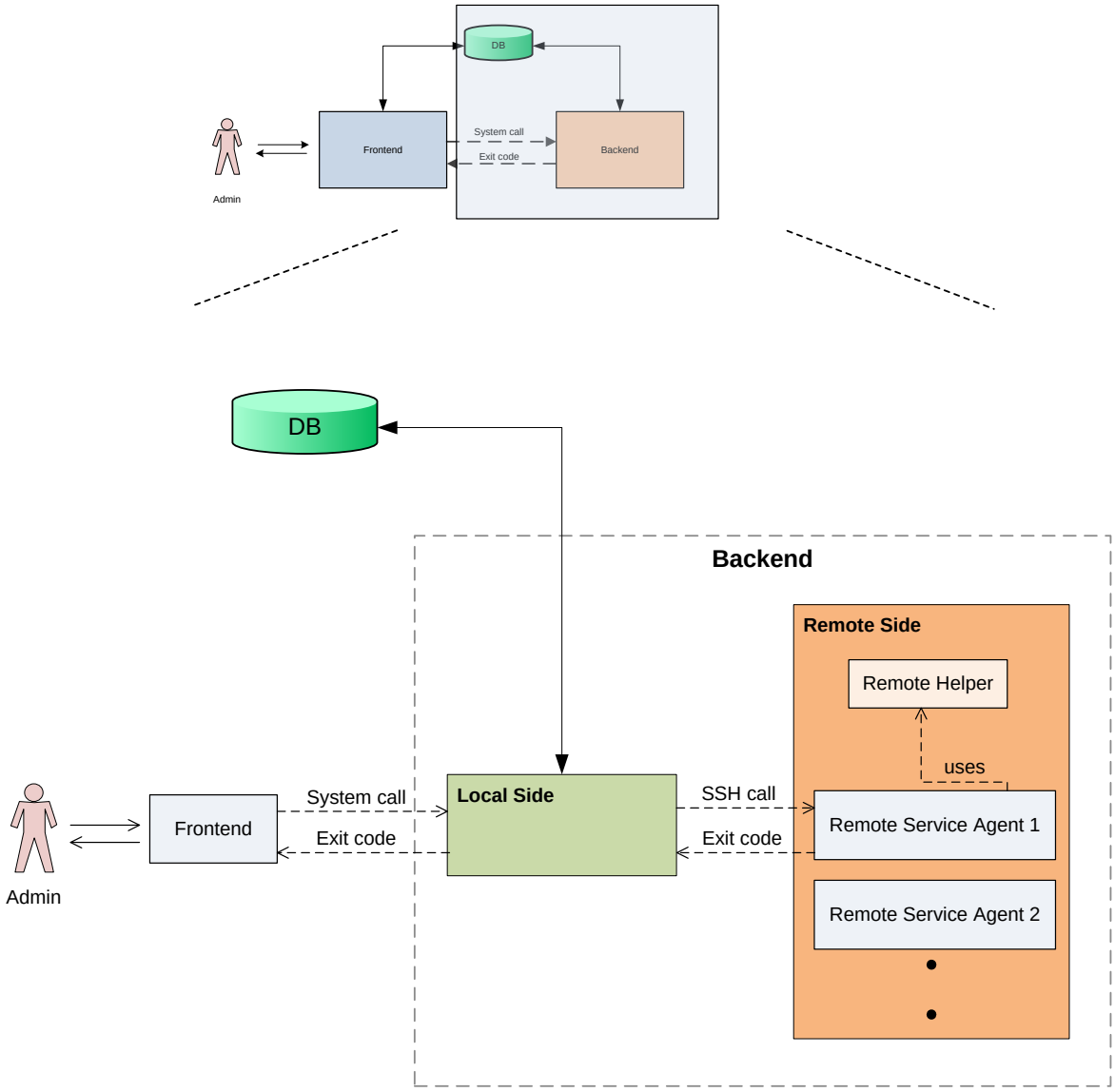

Abbildung 4.1: Backend

Das Backend ist in zwei logisch getrennte Teile unterteilt. Der allgemeine Entwurf des Backends wurde in Diagramm 4.1 dargestellt. Hier wird vom *lokalen* bzw. *entfernten* Teil gesprochen in Bezug auf die Maschine auf der das Frontend und ggf. die Datenbank von AMASI installiert ist. Diese Aufteilung wurde gemacht, um einen logischen System-Schnitt zu realisieren. Der lokale Teil (Local Side) bietet die Schnittstelle zum Backend, wie es im Kapitel Anforderungsanalyse beschrieben wurde, und kommuniziert mit dem entfernten Teil (Remote Side), der die eigentlichen Dienstanpassungen vornimmt. Der entfernte Teil besteht aus einem Agent pro Service. Diese Agents befinden sich auf dem entfernten Rechner, wo die spezifischen Dienste, verbunden mit den AMASI Services, lokalisiert sind. Zum Beispiel liegt der entfernte Windows Agent (remoteWindowsAgent.pl) auf dem Domain-Server und ist zuständig für das Aktivieren/Deaktivieren von einem *samba* Nutzer. Wenn das Backend aufgefordert wird, einen bestimmten Service für ein Benutzerkonto freizuschalten, wird der lokale Teil angesprochen, der die Aufgabe übernimmt, die gewünschte Aktion durchzuführen. Der lokale Teil wird alle benötigten Informationen aus der AMASI Datenbank sammeln und wird anschließend den entfernten Agent aufrufen, der die Dienstanpassungen vornimmt. Anschließend wird das Resultat der Aktion an den lokalen Teil weitergegeben, was dem Frontend zurückgemeldet wird. Falls der betreffende Dienst auf dem lokalen Teil läuft, ist es nicht unbedingt nötig, einen zusätzlichen Agent aufzurufen. Diese Aufgabe kann in diesem Fall von dem lokalen Teil übernommen werden.

Die Abhängigkeiten zwischen den Services (siehe Anforderungsanalyse) wird nicht in der "Backend"-Implementierung berücksichtigt. Allein das Frontend kennt diese Abhängigkeiten und sorgt dafür, dass das Backend korrekt und in der richtigen Reihenfolge aufgerufen wird.

4.2 Implementierung

Folgende Service-Agents wurden im Rahmen unseres Praktikums entworfen und implementiert: Directory Agent, User Agent, Windows Agent, Linux Agent und Gforge Agent. Diese Agents realisieren die Anforderungen, die in der Anforderungsanalyse für die entsprechenden Services Directory, User, Windows, Linux und Gforge beschrieben sind. Die Agents werden von einen lokalen Teil des Backends kontrolliert aufgerufen. In diesem Kapitel wird der Aufbau des lokalen Teils (Local Side) des Backends sowie die Schnittstellen und Abhängigkeiten der entfernten Agents (Remote Side) beschrieben. Die Umsetzung von zweien wird detaillierter geschildert: User Agent und Windows Agent. Deren Realisierungen zeigen zwei verschiedene Ansätze der Implementierung, die für weitere Agent-Implementierungen benutzt werden können.

Die Implementierung des Backends ist in der Skriptsprache Perl realisiert. Perl ist für den praktischen Einsatz entwickelt worden und bietet schnelle und einfache Programmierbarkeit, Vollständigkeit und Anpassungsfähigkeit. Es gibt eine große Menge frei verfügbarer Module für Perl, die für verschiedene Anforderungen entwickelt wurden. Alle Perl Module, die in der Implementierung des Backends eingesetzt werden, sind auf der CPAN Online-Repository für Perl Module verfügbar. Perl erlaubt auch die objektorientierte Programmierung, was

in der Implementierung der Local Side benutzt wird. Perl ist eine mächtige Programmiersprache, die eine sehr flexible Schreibweise erlaubt, wodurch sie allerdings unübersichtlich und unstrukturiert werden kann, falls nicht ordentlich kommentiert und/oder unorthodoxe Schreibweise benutzt wird. In der Backend-Implementierung wurde dafür gesorgt, dass der Code an relevanten Stellen mit Kommentaren erläutert wird, sodass eventuelle Anpassungen leichter und schneller für den Implementierer zu ermöglichen sind. Außerdem bietet jedes Script Help-Ausgaben, die seine Benutzung erläutern, wie am Beispiel der Remote Directory Agent in Listing 4.1 zu sehen ist. Diese Ausgaben helfen, die konkreten Schnittstellen anschaulich zu machen.

```
<action> : ACTIVATE, DEACTIVATE, RIGHTS
<benutzername> : a valid username
Example : perl remoteDirectoryAgent.pl ACTIVATE johnny
```

Listing 4.1: Help Ausgabe von remoteDirectoryAgent.pl

4.2.1 Local Side

Der Lokale Teil des Backends ist objektorientiert programmiert und folgt großteils dem Verhaltensmuster des Vermittlers (Mediator Pattern [27]). Der Entwurf ist im Klassendiagramm 4.2 dargestellt.

Die *Base* Klasse ist eine Basisklasse und enthält einen generellen Konstruktor, der von allen Unterklassen benutzt wird. Darüberhinaus bietet sie nützliche Funktionen, die von Unterklassen benutzt werden können. Somit werden Funktionen, die von mehr als einer der konkreten Base-Unterklassen benötigt werden, zusammengefasst, sodass die Implementierung nur einmal erfolgt. Damit wird gewährleistet, dass die Funktionalität einheitlich und weniger fehleranfällig realisiert wird. Die Service Aktionen, beschrieben in der Anforderungsanalyse (siehe Seite 15), werden von drei spezifischen Funktionen in der Basisklasse abgebildet - activate(), deaktivate() und password(). Diese Funktionen werden von den Unterklassen überschrieben, falls die konkrete Funktionalität unterstützt ist.

Der Backend Mediator realisiert die Schnittstelle zum Backend. Dieser überprüft die Korrektheit des Aufrufes zum Backend, nämlich ob alle benötigten Informationen übergeben sind. Falls das der Fall ist, erzeugt er eine Instanz der entsprechend abgeleiteten Klasse und führt die angeforderte Aktion aus. Dies geschieht, indem die spezifische Funktion von dieser Instanz aufgerufen wird. Das Resultat dieser Aktion wird in Form eines Exitcodes als Resultat vom Backend an den Aufrufer (in der Regel das Frontend) zurückgegeben.

Base

Die Base Klasse ist eine Basisklasse, die nicht direkt instanziert wird, sondern das Gerüst für die konkreten abgeleiteten Klassen stellt. Diese Klasse ist in Base.pm implementiert.

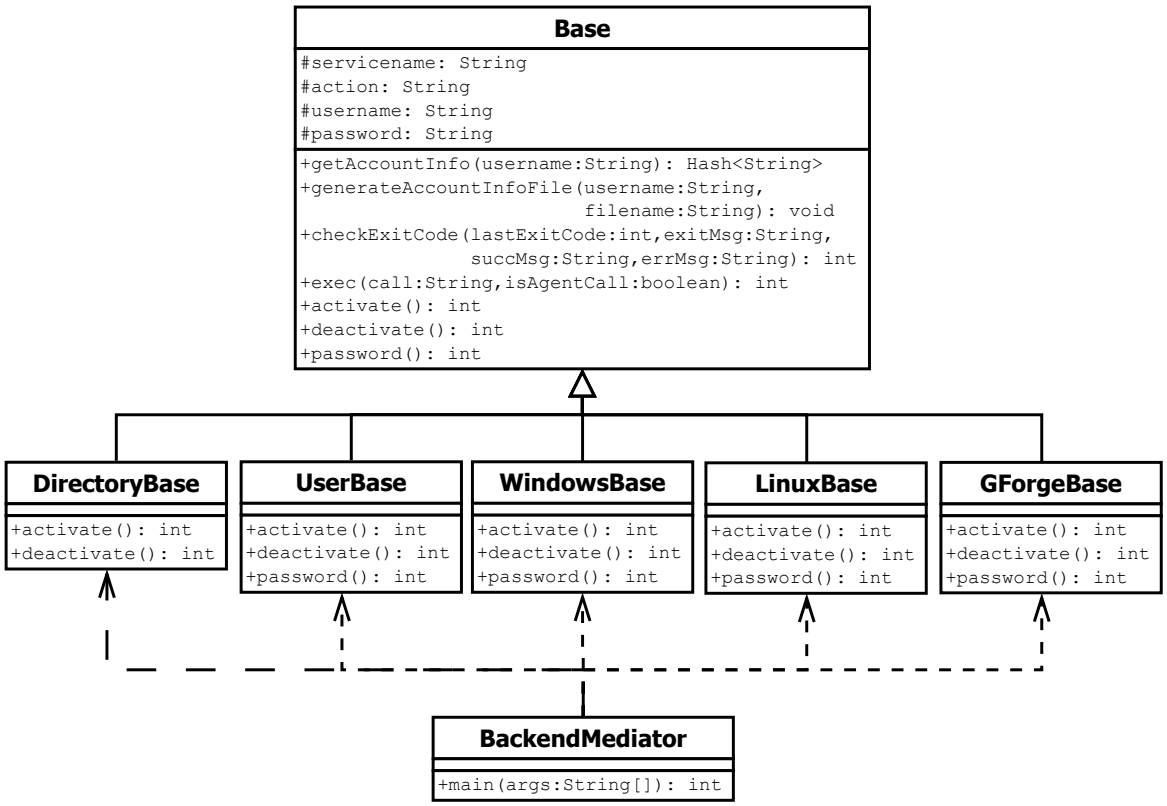

Abbildung 4.2: Klassen Diagramm von dem lokalen Teil des Backends

Konstruktor

Der Konstruktor der Basisklasse wird mit den angegegebenen Paramter *service*, *action*, *username* und *password* initialisiert. Außerdem wird ein Logger Objekt bereitgestellt, der für alle Funktionen zur Protokollierung benutzt wird. Die Initialisierung des Loggers beinhaltet die Konfiguration von spezifischen Konfigurationsparametern definiert, in der log.conf File. Diese Konfigurationen legen bestimmtes Verhalten des Loggers fest - Ziel Log-Datei (`/root/amasi/log/amasi_backend.log`), *Logger level* (z.B. DEBUG, INFO, ERROR), Struktur der Logmeldungen und Umgang mit der Log-Datei (Rotieren am Anfang des Monats). Alle Objekte der *Local Side* schreiben Log-Ausgaben auf den lokalen Rechner. Die Log Datei befindet sich unter /root/amasi/log/amasi.log. Für weitere Details über die Verzeichnisstruktur von AMASI siehe Kapitel “Installation und Migration”.

Abhängigkeiten Die Base Klasse stellt die generellen Abhängigkeiten von Perl Modulen, die für die korrekte Arbeit der Local Side notwändig sind:

- `DBI` - Perl-DBI bietet ein API für den Datenbankzugriff unter Perl.
- `DBD::Pg` - ein DBI Treiber für die PostgreSQL Datenbank. Dieser Modul zusammen mit `DBI` ist nötig um der Zugriff zur AMASI Datenbank zu ermöglichen. Diese Funktionalität wird in den Funktionen `getAccountInfo()` und `generateAccountInfoFile()` benutzt.
- `Log::Log4perl` - Ein Perl-Modul für kontrollierten Umgang mit Log-Anweisungen. Dieses Modul wird verwendet, um den Protokollierungsprozess des Backends zu realisieren.

Funktionen Drei der Funktionen in der Base Klasse werden von den abgeleiteten Klassen überschrieben, falls das bezüglich der angebotenen Funktionlität notwändig ist. Diese Funktionen sind `activate()`, `deactivate()` und `password()`. Die Basisklasse hat diese Funktionen bereits als *nicht implementiert* realisiert. Die Funktionen liefern Exitcode=1 zurück, falls sie nicht überschrieben sind. Außer diesen speziellen Funktionen werden mehrere Hilfsfunktionen definiert, die für alle abgeleiteten Klassen zur Verfügung stehen:

- `Hash<String> getAccountInfo ( String benutzername )`

  holt alle Account Informationen für einen Benutzer aus der Datenbank und gibt sie als ein `Hash<String>` Objekt zurück.

- `int generateAccountInfoFile ( String benutzername, String fileName )`

  benutzt die Funktion `getAccountInfo`, um alle Benutzerinformationen in einem File zu speichern. Die Informationen werden zeilenweise im folgenden Format gespeichert:

`key<-->value`. key ist der Name der Spalte in der Datenbank und value ist dessen Wert (z.B. `matrikelnummer<-->2232842`). Zusätzlich wird das Ergebnis der Funktion als Resultat zurückgegeben. Falls das Benutzerkonto für den angegebenen Benutzernamen nicht gefunden wird oder der File nicht erstellt werden kann, ist das Resultat der Funktion 1.

- `int checkExitCode (int lastExitCode, String exitMessage, String successMessage, String errorMessage)`

  Die Funktion überprüft `lastExitCode` und schreibt entsprechend die Strings `exitMessage`, `successMsg` und `errorMsg` als Log-Meldungen aus. Der Exitcode `0` wird als Erfolg interpretiert und jeden anderen Exitcode wird als nicht erfolgreich angesehen.

- `int exec( String call, String isAgentCall )`

  Diese Funktion ist für das Aufrufen bestimmter Kommandos. Der Parameter `isAgentCall` gibt an, ob es sich um einen SSH-Aufruf zum entfernten Agent handelt oder nicht. Ist das der Fall, dann werden andere Log-Meldungen geschrieben als im Normalfall. Das Kommando, spezifiziert mit dem Parameter `call`, wird aufgerufen und anschließend wird die Funktion `checkExitCode` benutzt, um den Exitcode zu analysieren und entsprechende Log-Meldungen auszugeben.

Base-Unterklassen

Es gibt fünf Base-Unterklassen für jede der definierten Services - DirectoryBase, UserBase, WindowsBase, LinuxBase, GForgeBase. Diese Unterklassen von Base sind in den Dateien DirectoryBase.pm, UserBase.pm, WindowsBase.pm, LinuxBase.pm, GForgeBase.pm implementiert. Diese haben als Aufgabe, alle nötigen Informationen bezüglich einer konkreten Aktion für den jeweiligen entfernten Agent zu sammeln und zu übergeben. Jeder dieser Klassen überschreibt je nach Bedarf die Funktionen `activate()`, `deactivate()` und `password()` aus der Oberklasse Base. Diese Funktionsimplementierungen könnten die Dienstanpassungen auch direkt durchführen (z.B. wenn der dazugehörige Dienst auf dem lokalen Rechner läuft). Bei unserer Implementierung aller Services aber wurde ein entfernter Agent angelegt.

Auf die Implementierung von zwei Funktionen wird hier näher eingegangen - `activate()` von UserBase und `deactivate()` von WindowsBase.

UserBase:activate() Die Funktion `activate()` ist wie folgt strukturiert:

- Hole alle Informationen für ein Benutzerkonto und speichere sie in einer Datei auf der lokalen Maschine.

- Ersetze das Passwort aus der Datenbank mit dem `cleartext` Passwort in dieser Datei

- Stelle die Datei für den entfernten Agent auf dem entfernten Rechner bereit
- Rufe den entfernten Agent auf, um die Aktion durchzuführen
- Lösche alle temporären Dateien, falls die Aktion erfolgreich war

Der Quellcode für diese Funktion ist in Listing 4.2 veranschaulicht.

```
sub activate{
    my $object = shift;
    $logger = $object->{logger};

    # Create a temporary file with all the information for this account
    $username = $object->{benutzername};
    my $temp_file = $tempDirPath . "account." . $username;

    $logger->debug( "Generate account info file " .
                    "for benutzername '$username' at $temp_file." );

    my $result = $object->generateAccountInfoFile( $username, $temp_file );

    # If an error occured then we don't
    # need to proceed and exit with an error.
    if ( $result != 0 ) {
        return 1;
    }

    #replace hashed password from DB with the cleartext one
    `perl -i -n -e 'print if not /password/i' $temp_file`;
    `echo "password<-->$object->{password}" >> $temp_file`;

    #Copy the file containing the account information on the remote server
    $logger->debug( "Attempting to copy account " .
                    "info file to the remote host" );

    my $scpCommand = "scp $temp_file $remoteUser\@$remoteIP:$tempDirPath";
    $result = $object->exec ( $scpCommand );

    # If an error occured then we don't need to proceed.
    if ( $result != 0 ){
        return 1;
    }

    # ACTIVATE command
    my $sshCommand = "ssh $remoteUser\@$remoteIP " .
                     "'perl $remoteAgent ACTIVATE $temp_file'";
    $result = $object->exec ( $sshCommand, 'true' );

    # If an error occured then we don't need to
    # proceed and leave back all files as debug information.
    if ( $result != 0 ){
```

```
        return 1;
    }

    #Clean the files on the local machine and on the server machine
    $logger->debug( "Delete temp account info file" .
                    " from local and from remote machine..." );

    my $clean1 = "rm -f $temp_file";
    $object->exec ( $clean1 );

    my $clean2 = "ssh $remoteUser\@$remoteIP 'rm $temp_file'";
    $object->exec ( $clean2 );


    $logger->info( "Successfully created user '$username'" .
                   "on the domain server." );

    exit 0;
}
```

Listing 4.2: Funktion activate() — Ausschnitt aus UserBase.pm

WindowsBase:deactivate() Die Funktion `deactivate()` ist weniger aufwendig, da außer den Benutzernamen keine zusätzliche Information für die Aktion benötigt werden. Eigentlich wird nur direkt der entfernte Agent aufgerufen, um die Aktion auszuführen. Der Quellcode ist in Listing 4.3 gezeigt.

```
sub deactivate{
        my $object = shift;
        my $logger = $object->{logger};

        # DEACTIVATE ACTION
        $logger->info ( "Deactivating 'Windows' service. " .
                        "Account: '$object->{benutzername}'." );

        my $sshCommand = "ssh $remoteUser\@$remoteIP " .
                         "'perl $remoteAgent DEACTIVATE $object->{benutzername}'";
        my $result = $object->remoteCall ( $sshCommand );

        return $result;
}
```

Listing 4.3: Funktion deactivate() — Ausschnitt aus WindowsBase.pm

BackendMediator

Der Backend Mediator wird als ein Skript realisiert (BackendMediator.pl). Dieses Skript trägt die Verantwortung, den Aufruf auf Richtigkeit zu überprüfen, d.h. ob die Backend

Schnittstelle korrekt angesprochen wird. Die andere Aufgabe des Backend Mediators ist die Analyse des Aufrufes. Aufgrund der übergebenen Parameter *service* und *action*, wird eine konkrete Instanz einer Base-Unterklasse erzeugt und die passende Funktion aufgerufen. Die Implementierung der Überprüfung und die Analyse des Aufrufes wird in Listing 4.4 gezeigt, was ein Ausschnitt aus dem BackendMediator.pl Skript veranschaulichen soll.

```
#test if the call is correct
if (    defined $ARGV[0] && defined $ARGV[1] && defined $ARGV[2] &&

        ( $ARGV[0] eq "directory" || $ARGV[0] eq "user" ||
          $ARGV[0] eq "windows" || $ARGV[0] eq "linux" ||
          $ARGV[0] eq "gforge" ) &&

        ( ( $ARGV[1] eq "ACTIVATE" && defined $ARGV[3] ) ||
          ( $ARGV[1] eq "PASSWORD" && defined $ARGV[3] ) ||
          ( $ARGV[1] eq "DEACTIVATE" ) ) ) {

                $service = $ARGV[0];
                $action = $ARGV[1];
                $username = $ARGV[2];
                $password = $ARGV[3];

} else {
        $logger->error( "Exiting. Factory not properly called." );
        printf ( "%s\n", "Usage: perl $me <service> <action> " .
                 "<benutzername> ?<cleartextPassword>?" );
        printf ( "\t %s\t%s\n", "<service>", ": directory, user, " .
                 "windows, linux, gforge" );
        printf ( "\t %s\t%s\n", "<action>", ": ACTIVATE, DEACTIVATE, " .
                 " PASSWORD" );
        printf ( "\t %s\t%s\n", "<benutzername>", ": a valid username" );
        printf ( "\t %s\t%s\n", "<cleartextPassword>",
                 ": the password in cleartext to set for the user account" );
        printf ( "%s\n",
                 "Example : perl $me directory ACTIVATE johnny secret" );
        printf ( "%s\n",
                 "Example : perl $me directory DEACTIVATE johnny" );
        printf ( "%s\n",
                 "Example : perl $me directory PASSWORD johnny newSecret" );
        exit 1;
}

if ( $service eq 'directory' ){
    $agent = LocalDirectory->new( $username, $password );
} elsif ( $service eq 'user' ){
    $agent = LocalUser->new( $username, $password );
} elsif ( $service eq 'windows' ){
    $agent = LocalWindows->new( $username, $password );
} elsif ( $service eq 'linux' ){
    $agent = LocalLinux->new( $username, $password );
} elsif ( $service eq 'gforge' ){
```

```
    $agent = LocalGForge->new( $username, $password );
}

if ( $action eq 'ACTIVATE' ){
    $result = $agent->activate();
} elsif ( $action eq 'DEACTIVATE' ){
    $result = $agent->deactivate();
} else {
    $result = $agent->password();
}

exit $result;
```

Listing 4.4: Backend-Schnittstelle Realisierung — Ausschnitt aus BackendMediator.pl

4.2.2 Remote Side

In diesem Kapitel werden Schnittstellen, Abhängigkeiten und einige Beispiele für die Nutzung von entfernten Agents sowie das Hilfsskript *Remote Helper* vorgestellt. Auf die Realisierung von Remote User- und Windows Agent wird näher eingegangen. Damit wird auch eine Implementierungsmöglichkeit gezeigt, die sich für die schnelle Realisierung von Agents lohnen kann. (siehe Kapitel `HTTP-based Agents`)

Remote Helper

Die Aufgabe vom Remote Helper ist es, Funktionen zusammenzufassen, die von mehr als einem entfernten Agent benötigt werden, sodass die Implementierung nur einmal erfolgt. Die Implementierung der Remote Helper befindet sich in der Datei general_remote.pl.

Abhängigkeiten Die Basis-Module, installiert mit der Perl Umgebung, sind für die Benutzung ausreichend.

Schnittstelle Remote Helper stellt folgende Funktionen zur Verfügung:

- `Hash<String> extractAccountFromFile ( String fileName )`

  ist die Umkehrfunktion der generateAccountInfoFile Funktion (siehe Local Side). Sie extrahiert Benutzerkontoinformationen von einem File, die durch den Parameter fileName angegeben sind. Die Informationen müssen zeilenweise im folgenden Format abgespeichert sein: `key<-->value`. `key` ist der Name und `value` ist dessen Wert (z.B. `matrikelnummer<-->2232842`). Ein `Hash<String>` Objekt mit allen Namen als keys wird als Ergebnis zurückgegeben.

Benutzung und Beispiele Damit die Funktionen im Remote Helper benutzt werden können, muss die Datei erstmal importiert werden. Dies ist in der Regel für alle entfernten Agents notwendig.

```
require "/root/amasi/general_remote.pl";
```

- Parse alle Informationen in der Datei /tmp/account.txt und gib den Benutzernamen aus.

  ```
  my %account = &extractAccountFromFile( "/tmp/account.txt" );
  print "Benutzername: $account('benutzername')"
  ```

Remote Directory Agent

Der Datenserver im CIP-Pool ist auf Basis von ZFS unter Solaris realisiert. Der Remote Directory Agent ist zuständig für das Erzeugen/Löschen eines Arbeitsverzeichnisses für einen Benutzer auf dem ZFS Datenserver. Der Name des Verzeichnisses auf dem Datenserver ist der Benutzername. Der Remote Directory Agent ist in der Datei remoteDirectoryAgent.pl implementiert.

Abhängigkeiten Folgende Perl Module müssen installiert sein:

- `Log::Log4perl`

  Für die Aktion `RIGHTS` muss der Benutzer als Unix Nutzer auf dem Daten-Server existieren.

Schnittstelle Der Remote Directory Agent stellt folgende Schnittstelle zur Verfügung:

- `perl remoteDirectoryAgent.pl <ACTION> <BENUTZERNAME>`

  Zugelassene Werte für `<ACTION>` sind `ACTIVATE`, `DEACTIVATE` und `RIGHTS`. Falls der Agent richtig aufgerufen wird und kein Fehler während der Ausführung der Aktion auftritt, wird '0' als Ergebnis geliefert. In allen anderen Fällen ist das Ergebnis '1'.

Benutzung und Beispiele

- Erzeuge ein Arbeitsverzeichnis für einen Benutzer

```
~# perl remoteDirectoryAgent.pl ACTIVATE johnny
~# result=$?; echo "Das Ergebnis ist: $result.";
```

- Lösche ein Arbeitsverzeichnis

```
~# perl remoteDirectoryAgent.pl DEACTIVATE johnny
```

- Ändere die Rechte des Verzeichnisses mit dem Namen 'johnny' auf dem Unix-User johnny

```
~# perl remoteDirectoryAgent.pl RIGHTS johnny
```

HTTP-basierte Agents

Die Idee von allen HTTP-basierten Agents ist, die HTTP-Schnittstelle von einer WEB-Anwendung anzusprechen, um so die Funktionalität der Anwendungen zu nutzen. Dieser Ansatz wird für die Implementierung von zwei Agents verwendet: Remote User- und GForge-Agent.

Eine "HTTP-Schnittstelle" wird als Aufruf einer Webseite mit bestimmten HTTP-Request Parametern definiert. Das HTTP-Method für das Senden der Parameter kann POST oder GET sein.

```
z.B. GET : https://druckerServer.com/getPrinterQuota.jsp?username=s123466
```

Die Trennung zwischen Funktionalität und Darstellung ist für Web-Anwendungen nicht immer gegeben. Eine saubere Schnittstelle für die Nutzfunktionalität ist oft nicht verfügbar. Da eventuell viel Logik im Webseitencode integriert ist, kann die Benutzung der HTTP-Schnittstelle zwingend nötig sein. Die HTTP-Schnittstelle unterscheidet sich nicht viel von anderen Schnittstellenarten und ist relativ leicht zu implementieren. Außerdem bietet sich eine anschauliche Test-Umgebung für die Überprüfung der Funktionalität an, z.B. durch die Benutzung von einem WEB-Browser. Da diese Schnittstelle normalerweise nicht dokumentiert ist und damit nicht als API bezeichnet werden kann, entsteht die Gefahr von Inkompatibilität mit neueren Versionen der Web-Anwendung. Das Design einer Webseite (z.B. CSS Code, Tabellen usw.) wird oft in verschiedenen Versionen verändert. Die HTTP-Schnittstelle, also der eigentliche Parameter für den Aufruf, verändert sich eher seltener. Unserer Meinung nach wird es relativ leicht sein, eine Anpassung im Falle einer solchen Veränderung in der HTTP-basierten Agenten zu machen.

Uns ist es bewusst, dass aus Managementsicht die Nutzung der HTTP-Schnittstellen von Web-Anwendungen keine saubere Lösung ist. Es kann jedoch als ein guter Kompromiss zwischen Einfachheit der Implementierung und bewusster Gefahr von Veränderung der HTTP Schnittstelle gesehen werden.

Während der Entwicklung von AMASI hat sich die Version von *Webmin* (siehe Kapitel Remote User Agent) von der ursprünglich installierten Version 1.311 auf 1.4.7 verändert. Es gab keine Veränderung in der HTTP-Schnittstelle, die vom Remote User Agent verwendet wird.

Die Implementierung der HTTP-Schnittstellen wird im Backend mit Hilfe der Perl Library `WWW::Mechanize` realisiert. Dieses Modul bietet die Möglichkeit, automatisiert mit einer Web-Seite zu interagieren: WEB-Formulare auszusetzen, WEB-Links aufzurufen, usw. Zusammen mit dem Modul `HTTP::Cookies` für Perl ist es möglich, auch aufwendigere Aufgaben wie z.B. HTTPS-Sessions zu realisieren. Die genaue Anwendung wird im nächsten Kapitel verdeutlicht.

Remote User Agent

Für jedes Benutzerkonto muss man einen Unix-Benutzer auf dem Domain-Server erzeugen. Bis jetzt wurde diese Aufgabe mithilfe der freien WEB-Anwendung Webmin erledigt. Webmin bietet eine WEB-Schnittstelle für das Erledigen von systemadministrativen Aufgaben unter Unix. Webmin ist modular aufgebaut und bietet unter anderem Module für das Erzeugen von Unix, samba und NIS Nutzern. Der Vorteil der Nutzung von Webmin ist, dass es die Konfiguration von Dateien wie /etc/passwd für den Administrator übernimmt, wodurch sich die Fehleranfälligkeit für solche Aktionen vermindert.

Die Verantwortlichen für den CIP Betrieb am Institut für Statistik möchten Webmin weiterhin benutzen können. Im Fall eines Ausfalls der AMASI Anwendung wird es immer noch möglich sein, die Unix-Benutzer leichter zu verwalten. Der User Agent benutzt die HTTP-Schnittstelle von Webmin, um Nutzer zu erzeugen und zu löschen. Zum Zeitpunkt der AMASI Entwicklung stand keine andere API von Webmin zur Verfügung. Inzwischen bietet Webmin eine dokumentierte Perl-Library API für die Funktionalität von den meisten Webmin-Modulen. Trotzdem werden viele weitere Eingabeüberprüfungen im Webseiten-Code realisiert.

Die wichtigsten Vorteile für die Nutzung von Webmin mit seiner HTTP-Schnittstelle sind:

- Eine von einer Unix-Plattform unabhängige Realisierung.

  Die Pfade für die verschiedenen Konfigurationsdateien oder die Namen der Kommandos sind auf den verschiedenen Unix Systemen teilweise unterschiedlich, z.B. das Kommando 'useradd' unter Debian entspricht 'adduser' unter Solaris. Webmin bietet die Möglichkeit, eine einheitliche Schnittstelle für viele Aktionen unter Unix Betriebssystemen zu nutzen. Im Institut für Statistik ist eine Migration der Domain-Server auf Debian geplant. Dies würde relativ wenig Einfluss auf die Implementierung des User Agents haben, vorausgesetzt Webmin ist wieder auf dem neuen System installiert.

- Überprüfungen auf die Korrektheit der Daten

Die Webmin Web-Schnittstelle überprüft die eingegebenen Daten auf syntaktische und semantische Fehler, wie zum Beispiel die Länge eines Passwortes, die Existenz eines Unix-Benutzers oder die Synchronisation der System-Tabellen mit samba-Tabellen. Diese brauchen dann nicht erneut implementiert werden.

- Gleichzeitige Benutzererzeugung in verschiedenen Anwendungen

  Webmin ist modular aufgebaut und bietet die Möglichkeit, durch weitere Übergabe mehrere Module anzusprechen. Dies ist besonders interessant, da man gleichzeitig einen samba- und einen NIS-Nutzer zusammen mit dem Unix-Nutzer erzeugen kann.

Der Remote User Agent ist ein HTTP-basierter Agent, der Aktionen für einen Unix-Benutzer auf dem Domain-Server überprüft und steuert. Der Agent wird vom lokalen Teil aufgerufen, der bereits alle benötigten Informationen bereitgestellt hat. Der Remote Agent benutzt die HTTP-Schnittstelle von Webmin, um die Aktionen `ACTIVATE` und `DEACTIVATE` zu realisieren. Die auf Webmin relevanten Seiten für Einloggen und Nutzergenerierung sind in den Abbildungen 4.4 und 4.3 aufgezeigt. Mit `WWW::Mechanize` Modul können Felder in Formularen "ausgefüllt" oder ein Buttonklick ausgeführt werden, je nach dem was notwendig ist. Hier wird stattdessen die Zielseite der Formulare direkt mit den HTTP-Parameter aufgerufen.

Für die Aktion PASSWORD wird der Einfachheit halber ein weiteres Script aufgerufen. Der Remote User Agent ist in der Datei remoteUserAgent.pl implementiert.

Abhängigkeiten

- `Log::Log4perl`
- `WWW::Mechanize`
- `HTTP::Cookies`
- `expectUserPasswd.exp` script für die `PASSWORD` Aktionsausführung

Interface

- `perl remoteUserAgent.pl <ACTION> <BENUTZERNAME> ?<password>?`

  Zugelassene Werte für `<ACTION>` sind `ACTIVATE`, `DEACTIVATE` und `PASSWORD`. Der Agent analysiert die Ergebnis-HTML-Seiten von Webmin und liefert das entsprechende Resultat an den lokalen Agenten zurück.

-

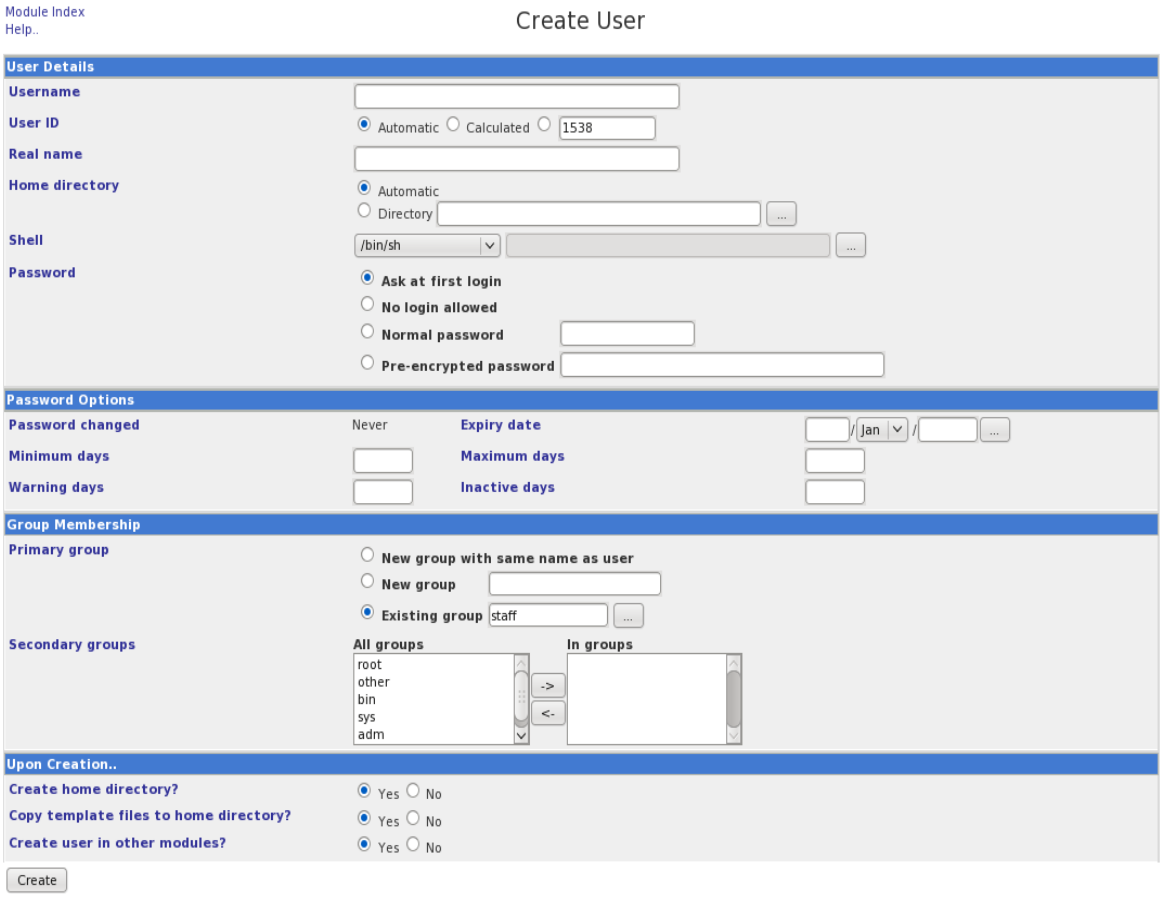

Abbildung 4.3: Screenshot der Seite zum Erzeugung von Nutzer - Webmin

Realisierung Anhand der ACTIVATE Aktion wird die Implementierung beschrieben. Für `DEACTIVATE` ist die Realisierung analog.

Um die Administrations-WEB-Seiten auf Webmin zu erreichen, muss man sich zunächst als Administrator anmelden. Dafür wird ein `WWW::Mechanize` Objekt erzeugt, der für alle folgenden HTTP-Aufrufe benutzt wird. Listing 4.5

```
$logger->debug( "Login on '" . `hostname` .  "'s Webmin via HTTPS." );

#This is my browser
$browser = WWW::Mechanize->new();
$browser->cookie_jar(HTTP::Cookies->new());

#These are the url and credentials for webmin
my $urlLogin = "https://127.0.0.1:10000";

$logger->debug( "Calling page '" . $urlLogin .  "'." );

#call Webmin's login page
$results = $browser->get($urlLogin);
```

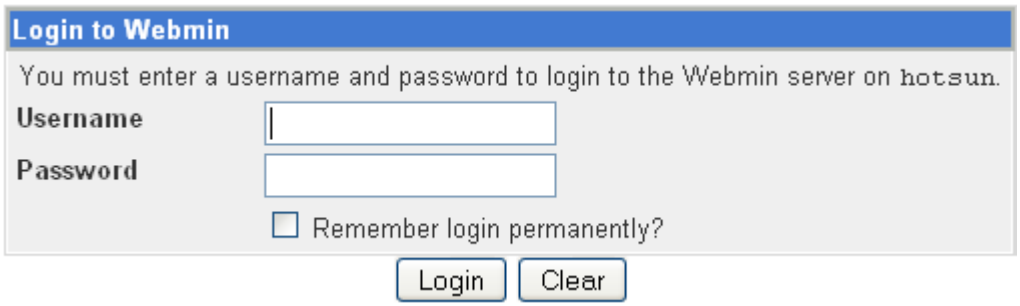

Abbildung 4.4: Loginseite - Webmin

```
$logger->debug( "Authenticating at Webmin as '" .
                $usernameLogin . "' ." );
#fill out the form and submit
my $test = $browser->submit_form(
  with_fields => { user => $usernameLogin, pass => $passwordLogin },
);

#Check if successfully logged in
my $output_page = $browser->content();
my $resultOK = ( !($output_page =~ /failed/) &&
                 !( $output_page =~ /error/ ) );

if ( ! $resultOK ){
    $logger->error( "Authentication at Webmin was not successful." );
    exit 1;
}
```

Listing 4.5: Anmeldung auf Webmin — Ausschnitt aus remoteUserAgent.pl

Sobald der Einlog-Vorgang abgeschlossen ist, kann mit den Operationen Erzeugen bzw. Löschen von Benutzern begonnen werden. Jetzt ist es möglich, die Zielseite für die Nutzererzeugung mit den gewünschten Parametern aufzurufen. Listing 4.6. Dem entspricht das Ausfüllen des Formulars in Abbildung 4.3. Danach wird der `Submit` Button gedrückt. Die nötigen Informationen gehen in die Bildug der URL mit ein, und werden anschließend mit der HTTP-POST Method verschickt. Die Ergebnis Seite wird gespeichert und der Inhalt analysiert.

```
&extractedAccount = &extractAccountFromFile ( $file_name );

$benutzername = $extractedAccount{ 'benutzername' };

if ( ! $extractedAccount{ 'benutzername' } ){
  $logger->error( "There is no benutzername in the specified file '" .
                  $file_name ."'." );
  exit 1;
}

# define the home directory
my $homeDir = $userHomeBase . $benutzername;
```

```
#call Webmin's save user cgi with the appropriate values
my $urlSaveUser = "https://127.0.0.1:10000/useradmin/save_user.cgi?" .
                        "user=" . $benutzername . "&" .
                        "uid_def=1" . "&" .
                        "uid=" . "&" .
                        "real=" . $extractedAccount{ 'vorname' } .
                        " " . $extractedAccount{ 'nachname' } . "&" .
                        "home_base=0" . "&" .
                        "home=" . $homeDir . "&" .
                        "shell=/bin/false" . "&" .
                        "passmode=3" . "&" .
                        "pass=" . $extractedAccount{ 'password' }
                        . "&" .
                        "encpass=" . "&" .
                        "expired=" . "&" .
                        "expirem=1" . "&" .
                        "expirey=" . "&" .
                        "min=" . "&" .
                        "max=" . "&" .
                        "warn=" . "&" .
                        "inactive=" . "&" .
                        "gidmode=0" . "&" .
                        "newgid=" . "&" .
                        "gid=" . $userGroupName . "&" .
                        "sgid=" . "&" .
                        "makehome=0" . "&" .
                        "copy_files=1" . "&" .
                        "others=1";

$logger->debug( "Trying to generate account for '" . $benutzername .
"'." );
$logger->debug( "Calling page '" . $urlSaveUser .  "'." );

$results = $browser->post($urlSaveUser);
```

Listing 4.6: Nutzererzeugung mittels Webmin — Ausschnitt aus remoteUserAgent.pl

Nach der Aktion braucht man nicht eingeloggt bleiben, man kann sich einfach ausloggen. Listing 4.7.

```
#logout from Webmin
$logger->debug( "Logging out from Webmin..." );
$browser->post( "https://127.0.0.1:10000/session_login.cgi?logout=1" );
```

Listing 4.7: Ausloggen vom Webmin — Ausschnitt aus remoteUserAgent.pl

Falls der Nutzer erfolgreich erzeugt wurde, müssen die Rechte für sein Arbeitsverzeichnis auf das ZFS System angepasst werden. Dafür ruft man direkt Remote Directory Agent mit der Aktion RIGHTS auf. Listing 4.8

```
#SET RIGHTS FOR THE USER'S HOME DIRECTORY
```

```
my $setRights = "perl " . $remoteDirectoryAgent .
                " RIGHTS " . $benutzername;

$logger->debug( "Command calling: " . $setRights .  "." );

my $exitMsg = `$setRights 2>&1`;
```

Listing 4.8: RIGHTS Aktion — Ausschnitt aus remoteUserAgent.pl

Remote Windows Agent

Alle Arbeitsplatzrechner im CIP-Pool erlauben die Nutzung von zwei Betriebssystemen - Windows XP und Linux Debian. Um den Dateizugriff zwischen Windows und dem Unix Datenserver zu organisieren, wird die samba Anwendung auf dem Datenserver eingesetzt. Samba regelt Domains, unter denen sich Nutzer authentifizieren können und somit Zugriff auf Dateien auf dem Datenserver haben. Der User Agent hat bereits einen samba Nutzer zusammen mit dem Unix-User erzeugt. Der Windows Agent ist zuständig für das Aktivieren/Deaktivieren/Passwordändern eines SMB Nutzers auf dem Domain-Server. Der Name dieses samba Nutzers ist gleich dem AMASI Benutzernamen.

Der Remote Windows Agent ist in der Datei remoteWindowsAgent.pl implementiert.

Abhängigkeiten

- `Log::Log4perl` Das Script expectWindowsPasswdChange.exp muss an der entsprechenden Stelle auf dem Domain Server für die PASSWORD-Aktion vorhanden sein.

Interface Der Remote Windows Agent stellt die folgende Schnittstelle zur Verfügung:

- `perl remoteWindowsAgent.pl <ACTION> <BENUTZERNAME> ?<password>?`

  Zugelassene Werte für `<ACTION>` sind `ACTIVATE`, `DEACTIVATE` und `PASSWORD` Falls der Agent richtig aufgerufen wird und kein Fehler während der Ausführung der Aktion auftritt, liefert dieser '0' als Ergebnis zurück. In allen anderen Fällen ist das Ergebnis '1'.

Realisierung Der Remote Agent benutzt die Funktionalität, angeboten durch das samba Kommando smbpasswd. Das Kommando hat die Funktion, einen SMB Nutzer zu aktivieren bzw. zu deaktivieren.

Für die ACTIVATE- und DEACTIVATE-Aktionen ergibt sich damit eine relativ leichte Implementierungsmöglichkeit. Listing 4.9

```
if ( $action eq "ACTIVATE" ||
     $action eq "DEACTIVATE" ){

############ (DE)ACTIVATE the account ###################
my $command = $smbCommandPath;

if ( $action eq "ACTIVATE" ){
    $command = $command . " -e ";
} else {
    $command = $command . " -d ";
}

$command = $command . $benutzername;

$logger->debug("samba command performing: " . $command );

$tempE = "Failed to $action Windows service for"
         . " account '$benutzername'.";
$tempS = "Windows service successfully $action" .
         "D for account '$benutzername'.";

my $exitMsg = `$command 2>&1`;
checkExitCode( "true", $exitMsg, $tempS, $tempE );

############ END (DE)ACTIVATE the account ###################
}
```

Listing 4.9: ACTIVATE und DEACTIVATE Implementierungen — Ausschnitt aus remoteWindowsAgent.pl

Für die PASSWORD-Aktion wird wieder das smbpasswd Kommando benutzt. In diesem Fall aber wird eine Interaktion mit der Konsole benötigt. Das Passwort muss eingegeben und nochmals bestätigt werden. Für diese Aufgabe ist das Skript expectWindowsPasswdChange.exp verantwortlich. Das Skript ist in der Expect Programmiersprache realisiert. Der wichtigste Teil aus diesem Skript ist in Listing 4.10 gezeigt. Die Implementierung der `PASSWORD` Aktion ist unter Listing 4.11 zu finden.

```
set arg1 [lindex $argv 0]
set arg2 [lindex $argv 1]

set timeout -1
spawn /usr/sfw/bin/smbpasswd $arg1
match_max 100000
expect -exact "New SMB password:"
send -- "$arg2\r"
expect -exact "\r
Retype new SMB password:"
send -- "$arg2\r"
expect eof
```

Listing 4.10: Ändere samba Passwort — Ausschnitt aus expectWindowsPasswdChange.exp

```
############## BEGIN CHANGE PASSWORD #################

my $expectCall = $expectCommand . " " . $expectWindowsPasswdChange .
                 " " . $benutzername . " " . $cleartextPassword;

$logger->debug( "Expect command executing: " . $expectCall );

$tempE = "Failed to change password for account '$benutzername'";
$tempS = "Password for '$benutzername' successfully changed.";

my $exitMsg = `$expectCall 2>&1`;
checkExitCode( "true", $exitMsg, $tempS, $tempE );

############## END CHANGE PASSWORD #################
```

Listing 4.11: PASSWORD Aktion — Ausschnitt aus expectWindowsPasswdChange.exp

Benutzung und Beispiele

- Aktiviere einen SMB Benutzer

  ```
  ~# perl remoteWindowsAgent.pl ACTIVATE johnny
  ```

- Deaktiviere ein SMB Nutzer

  ```
  ~# perl remoteWindowsAgent.pl DEACTIVATE johnny
  ```

- Ändere das Password für einen SMB Nutzer

  ```
  ~# perl remoteWindowsAgent.pl PASSWORD johnny newSecret
  ```

Remote Linux Agent

Alle Arbeitsplatzrechner im CIP-Pool Statistik erlauben die Nutzung von zwei Betriebssystemen - Windows XP und Linux Debian. Um die Authentifizierung gegen den Domain-Server unter Linux zu regeln, kommt NIS (Network Information Service) zum Einsatz. Der Remote Linux Agent überprüft und steuert das Erzeugen/Löschen/Passwort ändern von Benutzern in den NIS Tabellen auf dem Domain-Server. Der Remote Linux Agent ist in der Datei remoteLinuxAgent.pl implementiert.

Abhängigkeiten Folgende Perl Module müssen installiert sein:

- `Log::Log4perl` Der expectLinuxPasswdChange.exp Script muss für die Aktion `PASSWORD` auf dem Domain-Server an der richtigen Stelle zur Verfügung stehen.

Interface Der Remote Linux Agent stellt folgende Schnittstelle zur Verfügung:

- `perl remoteLinuxAgent.pl <ACTION> <BENUTZERNAME> ?<password>?`

  Zugelassene Werte für `<ACTION>` sind `ACTIVATE`, `DEACTIVATE` und `PASSWORD`. Falls der Agent richtig aufgerufen wird und kein Fehler während der Ausführung der Aktion auftritt, liefert dieser '0' als Ergebnis zurück. In allen anderen Fällen ist das Ergebnis '1'.

Benutzung und Beispiele

- Addiere einen Eintrag in den NIS Tabellen für einen existierenden Domain-Server Benutzer

  ```
  ~# perl remoteLinuxAgent.pl ACTIVATE johnny
  ```

- Lösche den Eintrag für einen Benutzer von den NIS Tabellen.

  ```
  ~# perl remoteLinuxAgent.pl DEACTIVATE johnny
  ```

- Ändere das NIS Passwort

  ```
  ~# perl remoteLinuxAgent.pl PASSWORD johnny newSecret
  ```

Remote GForge Agent

GForge wurde am Institut für Statistik eingeführt, um verschiedene Anwendungen integriert anzubieten. Es stellt ein Portal für die Verwaltung von Diensten wie SVN [3] und Wiki zur Verfügung. Am Department können Studenten mithilfe von GForge für die gemeinsame Arbeit an verschiedensten Projekten in Gruppen zusammengefasst werden. Der administrative Aufwand für diese Aufgabe wird damit deutlich erleichtert. Der Remote GForge Agent ist zuständig für das Erzeugen/Löschen/Passwort ändern von GForge Nutzern. Der Remote GForge Agent ist ein Web-based Agent, der die einzige Schnittstelle zum GForge anspricht. GForge benutzt zur Benutzerverwaltung eine eigene Datenbank. Die Implementierung ist ähnlich dem Remote User Agenten aufgebaut. Der Remote GForge Agent ist in der Datei remoteGForgeAgent.pl implementiert.

Abhängigkeiten Folgende Perl Module müssen installiert sein:

- `Log::Log4perl`
- `DBI`

- `DBD::Pg`
- `WWW::Mechanize`
- `HTTP::Cookies`

Interface Der Remote GForge Agent stellt folgende Schnittstelle zur Verfügung:

- `perl remoteGForgeAgent.pl <ACTION> <BENUTZERNAME> ?<password>?`

  Zugelassene Werte für `<ACTION>` sind `ACTIVATE`, `DEACTIVATE` und `PASSWORD`. Falls der Agent richtig aufgerufen wird und kein Fehler während der Ausführung der Aktion auftritt, liefert dieser '0' als Ergebnis zurück. In allen anderen Fällen ist das Ergebnis '1'.

Benutzung und Beispiele

- Aktiviere einen bereits existierenden GForge Benutzer. Der Benutzer ist bereits im GForge System erfasst, aber deaktiviert. Das Passwort wird nicht verändert und muss nicht als Parameter angegeben werden.

  ```
  ~# perl remoteGForgeAgent.pl ACTIVATE johnny
  ```

- Erzeuge einen neuen GForge Benutzer. Das Passwort muss zwingend als weiterer Parameter gesetzt werden.

  ```
  ~# perl remoteGForgeAgent.pl ACTIVATE johnny secret
  ```

- Deaktiviere einen GForge Nutzer. Der Nutzer wird nicht von der GForge Datenbank gelöscht.

  ```
  ~# perl remoteGForgeAgent.pl DEACTIVATE johnny
  ```

- Ändere das Password für einen GForge Nutzer

  ```
  ~# perl remoteGForgeAgent.pl PASSWORD johnny newSecret
  ```

5 Installation und Migration

In diesem Kapitel wird zuerst ein kurzer Überblick über die benötigte Software Postgresql [21], Java [11] mit Libraries, Tomcat [24] und Hibernate [10], Perl mit Libraries und die Grund Installations- und Konfigurationsschritte gegeben. Außerdem wird die Migration des alten Systems auf das neue in Abschnitt 5.2 vorgestellt.

5.1 Installation und Konfiguration

Die Beschreibungen in diesem Kapitel gehen davon aus, dass man die Software aus den Sourcen installiert. Natürlich ist die Installation von distributionsabhängigen Software-Repositories mit Kommandos wie yum oder apt-get auch möglich. Soweit vorhanden, sollten diese vorher durchsucht werden, um den Aufwand für Installation und Konfiguration zu reduzieren.

Die AMASI-Anwendung Sourcen sowie die relevanten Konfigurationsdateien für die nötige zusätzliche Software sind im SVN des Departements für Statistik i Verzeichnis `fopra` gespeichert. Für neuere Versionen der Software sollten die Konfigurationen unbedingt überprüft und angepasst werden, um ein reibungslosen Betrieb zu gewährleisten.

5.1.1 Postgresql

Version: Postgresql 8.3 oder höher

AMASI benötigt eine Datenbank, um alle seine Informationen zu speichern. Postgresql wird als Datenbank eingesetzt. Postgresql ist ein open-source relationales Datenbanksystem, das unter anderem Programmier-Schnittstellen für Java und Perl anbietet. Somit können das Frontend und das Backend die Datenbank manipulieren. Installation und Konfiguration von Postgresql wie auf `http://www.postgresql.org`.

Nach Installation und grundlegenden Konfigurationen, wie z.B. Benutzer und Zugriffsrechte, muss das AMASI Datenbankschema angelegt werden. Das Schema ist in SVN unter `fopra/datenbank/default-db-1.5.sql` zu finden. Damit die Installation schneller erfolgen kann, empfehlen wir das Einspielen einer anderen Datei, vorhanden unter `fopra/datenbank/initial-db-1.5.sql`. Diese beinhaltet bereits Einträge für AMASI-Administrator-

Rolle und das Administrator-Benutzerkonto. Diese sind für die Anwendung von AMASI zwingend notwendig.

5.1.2 Java & Libraries

Version: Java 6 oder höher

Der Kern der AMASI Anwendung basiert auf Java, und das Frontend ist als eine JSP Wegapplikation realisiert. Die Installation von Java JRE ist deshalb zwingend notwendig. Installation und Konfiguration wie auf `http://java.sun.com/javase/`.

Die folgenden Java-Libraries mit deren Abhängigkeiten müssen auf einem zugänglichen Klassenpfad für die Frontendkompilierung und den -Betrieb installiert werden:

- O/R Mapper - `Hibernate` mit der Session Manager `C3P0`
- Logging - `log4j`
- JSP Libraries - `jstl` und `displaytag`

Die Konfigurationsdateien für diese Libraries liegen in SVN in das Verzeichnis `fopra/amasi/src/de/lmu/stat/server/conf`.

Um Hibernate zu konfigurieren, werden zwei Dateien benötigt: `hibernate.cfg.xml` und `amasi.hbm.xml`. Die erste Datei beschreibt - wie die Anbindung zum AMASI Datenbanktreiber - die Konfiguration der `JDBC3` Funktionalität mittels `C3P0`, Datenbank-URL und -Benutzerdaten, etc. Die zweite Datei beinhaltet das Mapping zwischen den Datenbanktabellen und den Objekten, die in der AMASI Implementation verwendet werden. Der Logger des Frontends ist in der Datei `log4j.cfg` konfiguriert und beschreibt das Format für die Log-Anweisungen und das Handling der Logdatei.

Die Konfigurationsdateien werden mit der Installation von AMASI an die richtige Stelle gesetzt.

5.1.3 Tomcat

Version: Apache Tomcat 5.5 oder höher

Das Frontend von AMASI läuft als Webapplikation unter Tomcat, dass eine Umgebung für das Ausführen von Java Code bereitstellt. Tomcat installieren und konfigurieren wie auf `http://tomcat.apache.org/` zu finden. Nach erfolgreicher Installation ist die Anbindung zur AMASI Datenbank nötig. Als erstes muss der JDBC Treiber für Postgresql mit Tomcat

installiert werden. Dieser kann von der Seite `http://jdbc.postgresql.org/` heruntergeladen werden.

AMASI benutzt das *container managed security* Feature von Tomcat, um die Anmeldung zu den beiden Web-Applikationen admin und public zu regeln. Um authentifizieren zu können, braucht Tomcat Benutzerinformationen, insbesondere Benutzername und zugehöriges Passwort. Zusätzlich werden die Zugriffsrechte auf Rollenebene realisiert. Dafür muss Tomcat auch die Anbindung von Benutzer zu Rollen bekannt sein. Zu diesem Zweck muss ein *JDBC Realm* in der Tomcat-Konfigurationsdatei `server.xml` definiert werden. Der Realm in Listing 5.1 beschreibt die Tabellen in der AMASI Datenbank die von Tomcat benötigt werden. Die account Tabelle beinhaltet Benutzernamen und Passwörter und die `account_role` Tabelle die Rollen, zu der die Benutzer gehören. Zusätzlich wird der Hash-Algorithmus für die Passwörter in der Datenbank angegeben, sodass die Anmelde-Passwörter mit der in der Datenbank gespeicherten angeglichen werden können.

```
<Realm  className="org.apache.catalina.realm.JDBCRealm" debug="0"
        driverName="org.postgresql.Driver" digest="SHA"
        connectionURL="jdbc:postgresql://localhost/amasi"
        connectionName="admin" connectionPassword=""
        userTable="account" userNameCol="benutzername"
        userCredCol="password"
        userRoleTable="account_role" roleNameCol="rolename"/>
```

Listing 5.1: Realm Konfiguration — Ausschnitt aus server.xml

Alle Tomcat Libraries sind für das Kompilieren von AMASI notwendig und müssen auf einem zugänglichen Klassenpfad verfügbar sein.

5.1.4 Apache

Version: Apache 2.0 oder höher

Tomcat liefert ein HTTP-Server mit der Standardinstallation. Dieser ist für die Entwicklung und Testen von AMASI Anwendungen ausreichend. Im CIP-Pool des Statistik Instituts wurde Apache installiert als produktiver HTTP-Server installiert. Installation und Konfiguration wie auf `http://httpd.apache.org/`.

Das Einbinden von Apache und Tomcat erfordert die Installation von den Modulen *mod_jk* unter Apache und der AJP Connector unter Tomcat. Die Konfiguration von mod_jk spezifiziert welche Anfragen an Tomcat weitergeleitet werden müssen. Das AJP Protokoll wird für die Kommunikation zwischen Apache und Tomcat eingesetzt. Mehr Informationen zur Konfiguration der Apache-Tomcat Anbindung sind unter `http://tomcat.apache.org/connectors-doc/` zu finden.

Ziel ist, dass der HTTP-Server von Tomcat keine andere Ports offen hat als der AJP port. Apache hat nur den Port 443 für sichere SSL Verbindungen offen und leitet alle Anfragen auf die AMASI Webapps an Tomcat zum Verarbeitung weiter.

5.1.5 Perl

Version: Perl 5.10 oder höher

Für das Backend ist eine Installation von Perl mit zusätzliche Perl-Modulen nötig. Installation und Konfiguration wie auf `http://www.perl.org`. Folgende zusätzlichen Perl Module können vom CPAN (`http://www.cpan.org/`) gefunden und installiert werden:

- `HTTP::Cookies`
- `WWW::Mechanize`
- `Log4perl`
- `DBI`
- `DBD:Pg`

5.1.6 AMASI

Version: AMASI 1.0

Bevor man AMASI installiert, muss dafür gesorgt werden, dass der AMASI-Server auf alle entfernten Rechner als root ohne Passworteingabe zugreifen kann. Dazu wird ein RSA key auf dem Hostrechner erzeugt und in der Datei `/root/.ssh/authorized_keys` auf jeden der entfernten Server kopiert. Eine kurze Anleitung dafür ist unter `http://wiki.unixboard.de/index.php/SSH_ohne_Passwort` zu finden.

Die Installation von AMASI erfolgt mittels eines Makefiles, das auf dem AMASI Server auszuführen ist. Dieses muss vorher konfiguriert werden:

- Pfad zum SVN Verzeichnis von AMASI
- Installationsverzeichnis
- Hostnamen bzw. IP Adressen aller entfernten Server auf der das Backend zugreift.

Das Frontend und die Local Side von AMASI werden auf dem AMASI-Server installiert. Die entfernten Agents werden jeweils auf dem entfernten Server installiert, beispielsweise wird der remoteDirectoryAgent auf dem Datenserver bereitgestellt. Die AMASI Verzeichnisstruktur muss auf jedem Rechner gleich organisiert sein. Direkt unter dem `amasi` Verzeichnis müssen die entsprechenden Backend Skripte zu finden sein. Im Verzeichnis `amasi/log` befinden sich alle amasi log-Dateien. `amasi/error` beinhaltet Debug-Informationen über einen aufgetretenen Fehler. Das `amasi/temp` Verzeichnis beinhaltet Informationen, die - wie der Name

schon verrät - zur Laufzeit kurzzeitig benötigt werden. Der AMASI Verzeichnisbaum ist am Beispiel von des AMASI Servers in Listing 5.2 dargestellt.

```
amasi:~# tree /root/amasi
/root/amasi
|-- error
|-- DirectoryBase.pm
|-- GForgeBase.pm
|-- LinuxBase.pm
|-- UserBase.pm
|-- Windows.pm
|-- Base.pm
|-- BackendMediator.pl
|-- log
|   |-- amasi.log
|   |-- amasi_webapp.log
|   `-- amasi_webapp.log.1
`-- temp
```

Listing 5.2: AMASI Verzeichnisbaum

Der Makefile erledigt folgende Aufgaben:

Basis

- Sichere alle AMASI log-Dateien sowhl lokal als auch auf allen entfernten Servern
- Lade die AMASI Source Dateien runter vom SVN
- Kompiliere alle nötigen `*.java` Dateien und generiere die AMASI libraries

Frontend

- Kopiere die public und admin Webapplikationen in das Tomcat-Webapps Verzeichnis
- Stelle Tomcat die nötigen AMASI libraries zur Verfügung
- Starte Tomcat neu

Backend

- Erzeuge die AMASI Verzeichnisstruktur lokal und auf allen entfernten Servern
- Schiebe die Local Base Skripten auf den AMASI Server unter `/root/amasi/`

- Kopiere alle entfernten Agents auf den entfernten Server unter `/root/amasi/`

5.2 Migration

Die Migration des alten Systems auf die CIP-Benutzerverwaltung mit AMASI bedeutet das Erzeugen von neuen AMASI Benutzerkonten für alle CIP-Nutzer. Als Ausgangsinformationen haben wir eine Liste mit allen Studenteninformationen und deren zu dieser Zeit gültigen Benutzernamen bekommen. Die Verantwortlichen am Institut hatten eine neue Namensgebung für die neuen CIP-Benutzernamen beschlossen: `s<Matrikellnummer>`.

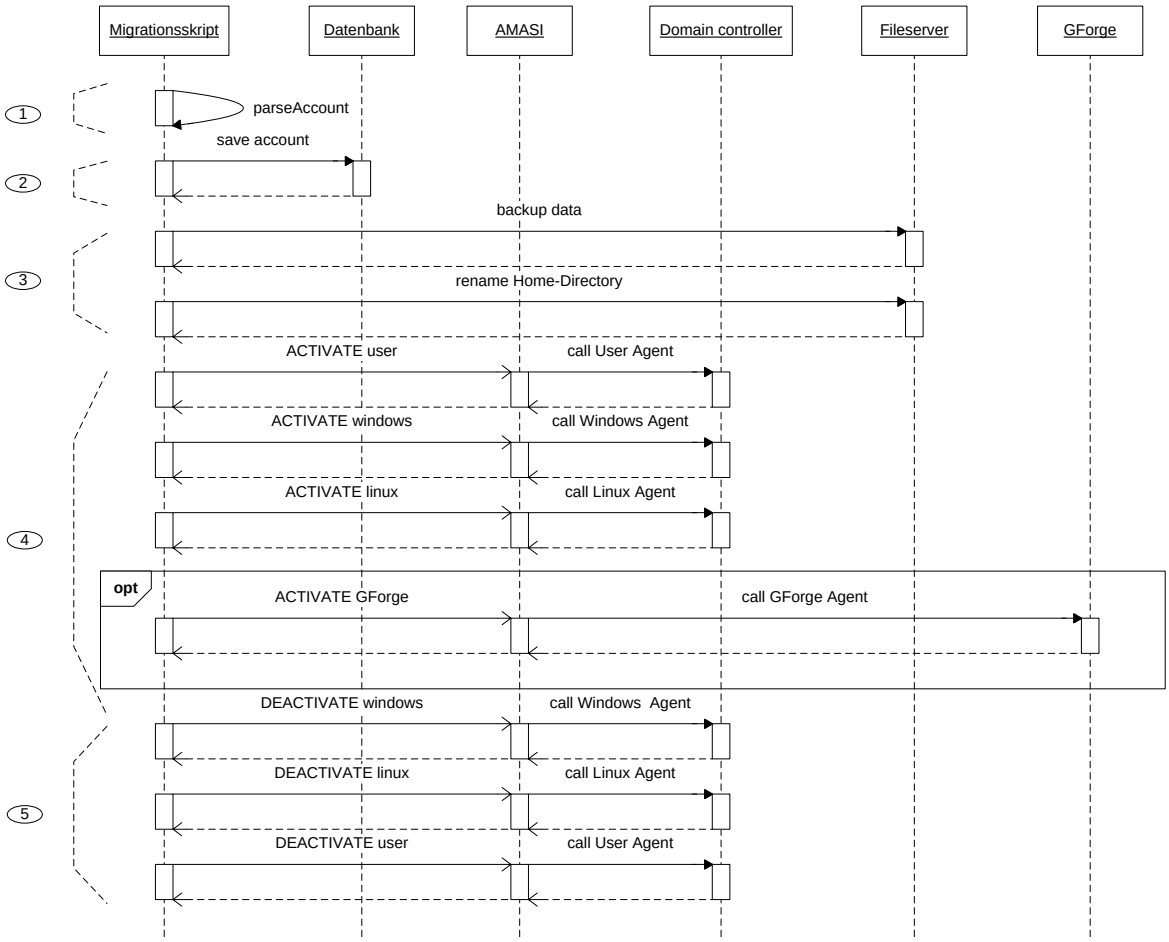

Abbildung 5.1: Sequenzdiagramm Migration

Die Migration wurde mittels eines Skripts realisiert, das sequentiell durch die Liste der alten CIP-Benutzer durchgegangen ist und neue erzeugt hat. Der Migrationsprozess für einen Benutzer ist in Abbildung 5.1 dargestellt, die einzelnen Schritte dabei sind wie folgt:

1. Alle Daten für den Benutzer werden von der Benutzerliste geparst

2. Die Benutzerinformationen werden in der AMASI-Datenbank abgespeichert

3. Alle Dateien des Nutzers sowie seine Profile in Windows und Linux wurden unter dem alten Namen behalten und sichergestellt

4. Mithilfe des Backends wird ein neuer Nutzer auf dem Domain Server mit neuem Benutzernamen erzeugt. Außerdem bekommt dieses Benutzerkonto Windows und Linux Zugänge freigeschaltet. Für die Mitarbeiter wird zusätzlich auch der GForge Service aktiviert.

5. Anschließend wird das alte Benutzerkonto gelöscht. Das bedeutet, dass alle freigeschalteten Dienste dafür deaktiviert werden.

Der große Vorteil der Modularität des Backends für die Migration war, dass die Schnittstelle des Backends direkt angesprochen wurde, ohne das Frontend zu benutzen. Nach der Migration der Benutzerkonten mussten die CIP-Nutzer ihre Passwörter ändern, indem die `public` Web-Anwendung von AMASI benutzt wurde. Die Migration hat über 500 CIP-Benutzerkonten in das neue System migriert.

6 Zusammenfassung

Die Ziele, die am Anfang dieses Projektes gestellt wurden, sind erreicht. Wir haben eine Web-Anwendung realisiert, die den administrativen Aufwand für die Benutzerverwaltung im CIP-Pool am Institut für Statistik erheblich reduziert hat. Vergabe und Verwaltung von Nutzerkonten sowie deren Rollen und Zugriffsberechtigungen für Dienstnutzungen erfolgen jetzt an einer zentralen Stelle. Ein Administrator, der nicht unbedingt Kenntnisse für die technische Realisierung dieser Aufgaben besitzen muss, kann jetzt AMASI benutzen, um diese Aufgaben zu erledigen. Das Benutzerkonto sowie seine Dienstzugriffsberechtigungen werden sofort freigeschaltet, seine Anmeldedaten ihm unmittelbar übergeben. Ein neuer CIP-Nutzer kann somit die angebotenen Dienstleistungen sofort in Anspruch nehmen.

Im Gegensatz zu den Administratoren haben alle sonstigen CIP-Nutzer beschränkten Zugang zu unserem System, die ein Benutzerkonto am Department besitzen. Sie haben somit die Möglichkeit, ihr Passwort für alle für sie freigeschalteten Dienstzugriffe zu setzen. Das System kann jederzeit erweitert werden, so dass die CIP-Nutzer zusätzliche Benutzerkontoinformationen selbst eingeben können und dadurch aktualisieren.

AMASI kooperiert mit vorhandenen Systemen und Diensten, da keine Veränderungen an der existierenden Infrastruktur vorgenommen werden sollten. Jede durchgeführte Aktion wird aufgezeichnet und für eine Analyse zu einem späteren Zeitpunkt zur Verfügung gestellt. Außerdem werden alle Benutzerkontendaten in einer Datenbank gespeichert, wodurch reduntante Informationsspeicherungen nicht mehr vorkommen können.

Wir freuen uns, dieses Projekt erfolgreich realisiert zu haben und bedanken uns für die Betreuung sowohl am Institut für Statistik, v.a. bei Manuel Eugster, als auch bei Dr. Nils gentschen Felde am Institut für Informatik.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[1] Apache Tomcat 6.0: Realm Configuration HOW-TO. http://tomcat.apache.org/tomcat-6.0-doc/realm-howto.html.

[2] Comprehensive Perl Archive Network (CPAN). http://www.cpan.de/.

[3] Subversion. http://subversion.tigris.org/, 2009.

[4] Apache modjk. http://tomcat.apache.org/connectors-doc/, 2010.

[5] Apache Tomcat AJP Connector. http://tomcat.apache.org/tomcat-6.0-doc/config/ajp.html, 2010.

[6] c3p0: JDBC3 Connection and Statement Pooling. http://www.mchange.com/projects/c3p0/index.html, 2010.

[7] Computer-Investitions-Programm. http://web.archive.org/web/20060519013854/http://www.dfg.de/forschungsfoerderung/wissenschaftliche_infrastruktur/wgi/geraete_im_hbfg_verfahren/was_ist_hbfg/cip.html, 2010.

[8] Display tag library. http://displaytag.sourceforge.net/1.2/, 2010.

[9] GForge. http://gforge.org/gf/, 2010.

[10] Hibernate: Relation Persistance for Java. http://www.hibernate.org/, 2010.

[11] Java. http://java.sun.com/javase/, 2010.

[12] JavaServer Pages. http://java.sun.com/products/jsp/index.jsp, 2010.

[13] JavaServer Pages Standard Tag Library. http://java.sun.com/products/jsp/jstl/, 2010.

[14] JUnit Tests. http://www.junit.org/, 2010.

[15] log4j: Logging Services for Java. http://logging.apache.org/log4j/1.2/index.html, 2010.

[16] Perl Modules: DBD:Pg. `http://search.cpan.org/~turnstep/DBD-Pg-2.17.1/Pg.pm`, 2010.

[17] Perl Modules: DBI. `http://search.cpan.org/~timb/DBI-1.609/DBI.pm`, 2010.

[18] Perl Modules: HTTP::Cookies. `http://search.cpan.org/~gaas/libwww-perl-5.834/lib/HTTP/Cookies.pm`, 2010.

[19] Perl Modules: Log::Log4perl. `http://search.cpan.org/~mschilli/Log-Log4perl-1.28/lib/Log/Log4perl.pm`, 2010.

[20] Perl Modules: WWW::Mechanize. `http://search.cpan.org/~petdance/WWW-Mechanize-1.62/lib/WWW/Mechanize.pm`, 2010.

[21] Postgresql. `http://www.postgresql.org/`, 2010.

[22] Postgresql JDBC Driver. `http://jdbc.postgresql.org/`, 2010.

[23] samba.org. `http://www.samba.org`, 2010.

[24] Tomcat. `http://tomcat.apache.org/`, 2010.

[25] Webmin. `http://www.webmin.com/`, 2010.

[26] Wiki. `http://d-nb.info/gnd/4806885-8`, 2010.

[27] D. Alur, D. Malks, and J. Crupi. *Core J2EE Patterns: best practices and design strategies*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2001.

[28] Thorsten Kukuk. Linux nis/nis+ projects. `http://www.linux-nis.org/`, 2007.