Technische Referenz

Mandate based IP-Telephony Administration (Mabata)

Modul:	Bachelorarbeit FS09	
Projektteam:	Fabio Looser	f1looser@hsr.ch
	Ueli Kuratli	ukuratli@hsr.ch
	Pascal Fuchs	pfuchs@hsr.ch
Betreuer:	René Stanger	rstanger@hsr.ch
	Maurin Egler	megler@hsr.ch
Verantwortlicher:	Prof. Beat Stettler	bstettle@hsr.ch
Datum:	12. Juni 2009	

HSR
HOCHSCHULE FÜR TECHNIK
RAPPERSWIL

ins institute for networked solutions

Dokumente des Projekts

Die Struktur der Bachelorarbeitsdokumentation wurde in zwei Dokumente gegliedert:

Bachelorarbeit

Das ist das Hauptdokument der Bachelorarbeit. Darin enthalten sind:

- Auftrag
- Management Summary
- Technischer Bericht
- Projektplan
- Projektmonitoring

Technische Referenz – *aktuelles Dokument*

Dokument für die Weiterentwicklung und interessierte Spezialisten. Darin enthalten sind:

- Entwicklungsumgebung
- Installationsanleitung
- Analyse
- Anforderungsspezifikation
- Design

Inhaltsverzeichnis

Entwicklungsumgebung

Im nachfolgenden Teil der Dokumentation wird die Entwicklungsumgebung behandelt, welche folgendermassen unterteilt ist:

- Arbeitsumgebung
- Testumgebung

1. Entwicklungsumgebung

1.1. Eingesetzte Arbeitsumgebung

Eclipse

- Entwicklungsumgebung: Eclipse IDE for Java EE Developers 3.4.1 mit JDK 6
- Versionskontrollsystem: SVN-Plugin Subclipse (Version 1.4.5)
- Ajax Framework für Java: ICEfaces Projektintegration Version 1.7.2 für Eclipse 3.4

DBMS

- PostgreSQL 8.3
- PgAdmin III 1.8.4 (als Datenbank-Browser)

Applikationsserver

- Apache Tomcat 6.0

Radius Server

- WinRadius 3

SVN

- svn://isvn.hsr.ch/MabataCode

Text-, Grafik- und Designsoftware

- Anzeigeprogramm: Adobe Acrobat Reader 9.0
- Texteditor: Microsoft Word 2007
- Tabellenkalkulation: Microsoft Excel 2007
- Visualisierung: Microsoft Visio 2007
- UML & ERD Modeling: Enterprise Architect 7.0 & JUDE Community 2.5.1

Debugging Software

- Packet Sniffer: Wireshark 1.0

Webbrowser (Testing)

- Mozilla Firefox 3.0
- Microsoft Internet Explorer 7.0

1.2. Testumgebung

1.2.1. Aufbau

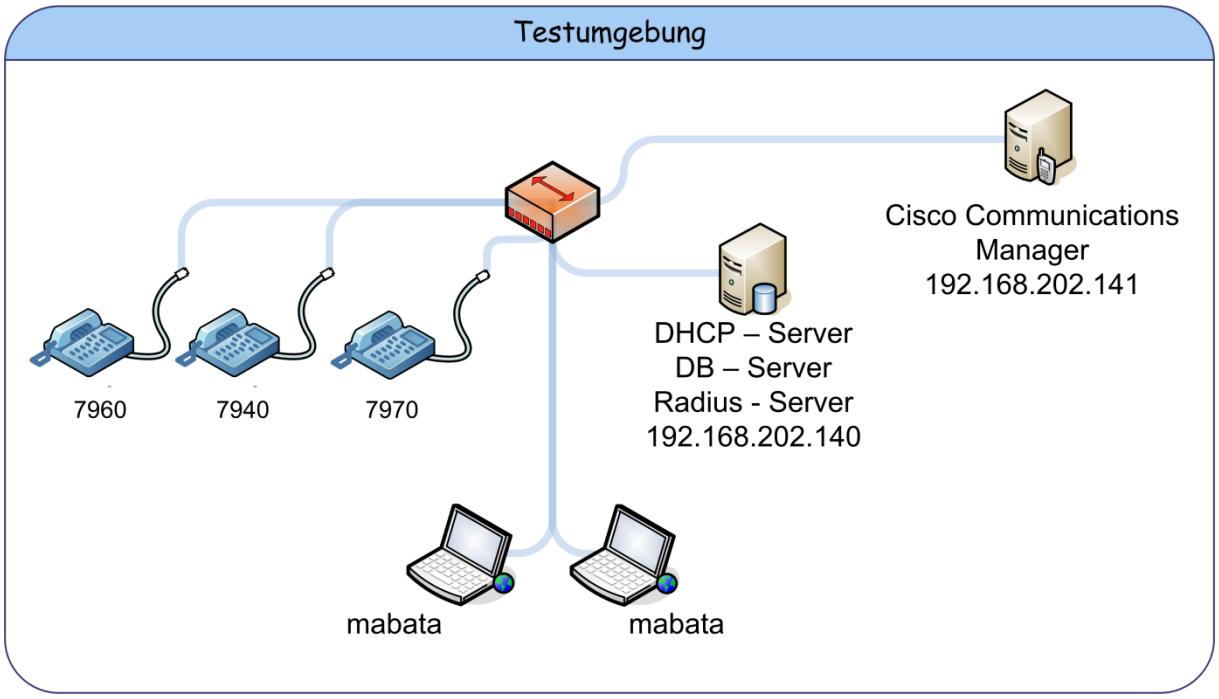


Abbildung 1-1 Übersicht Testumgebung

1.2.2. Communications Manager Konfiguration

1.2.2.1. Services aktivieren

Folgende Services müssen aktiviert werden (Menü: «Cisco Unified Serviceability > Tools > Service Activation»):

- Cisco Communications Manager
- Cisco TFTP
- Cisco Messaging Interface
- Cisco Unified Mobile Voice Access Service
- Cisco IP Voice Media Streaming App
- Cisco CTIManager
- Cisco Extension Mobility
- Cisco Extended Functions
- Cisco WebDialer Web Service
- Cisco AXL Web Service
- Cisco Serviceability Reporter

1.2.2.2. Extension Mobility Service einrichten

Der gerade aktivierte Service «Cisco Extension Mobility» muss nun konfiguriert werden um ihn nutzen zu können. (Menü: «Device > Device Settings > Phone Services»)

- Name: ExtMob
- URL: http://192.168.202.141:8080/emapp/EMAppServlet?device=#DEVICENAME#

ACHTUNG: Weil auf dem Communications Manager kein DNS eingerichtet ist, müssen zusätzliche Parameter angepasst werden, damit die Telefone den Phone-Service Server finden können. (Menü: «System > Enterprise Parameter > URL Service Parameter»)

- URL Authentication: http://192.168.202.141:8080/ccmcip/authenticate.jsp
- URL Directories: http://192.168.202.141:8080/ccmcip/xmldirectory.jsp
- URL Information: http://192.168.202.141:8080/ccmcip/GetTelecasterHelpText.jsp
- URL Services: http:192.168.202.141:8080/ccmcip/getservicesMenü.jsp

Alle weiteren Einstellungen bezüglich des Services (Extension Mobility) werden in den nächsten Schritten (Phones und Phoneprofiles einrichten) aufgelistet.

1.2.2.3. Route Partitions einrichten

Menü: «Cisco Unified CM Administration > Call Routing > Class of Control > Partition»

Name	Beschreibung (Description)
Abteilung1	Swiss Abteilung1
Abteilung2	Swiss Abteilung2
Abteilung3	Swiss Abteilung3

Tabelle 1-1 Route Partitions einrichten

1.2.2.4. Calling Search Spaces einrichten

Menü: «Cisco Unified CM Administration > Call Routing > Class of Control > CSS»

Name	Available Partitions
CSSAbteilung1	Abteilung1 hinzufügen
CSSAbteilung2	Abteilung2 hinzufügen
CSSAbteilung3	Abteilung3 hinzufügen
CSSALL	Alle Partitionen hinzufügen

Tabelle 1-2 Calling Search Spaces einrichten

1.2.2.5. DirNumbers einrichten

Menü: «Cisco Unified CM Administration > Call Routing > Directory Number»

Directory Number	Route Partition	CSS
1070 – 1079	Abteilung1	CSSALL
1060 – 1069	Abteilung2	CSSALL
1040 – 1049	Abteilung3	CSSALL

Tabelle 1-3 DirNumbers einrichten

1.2.2.6. PhoneButton Template erstellen

Menü: «Cisco Unified CM Administration > Device > Device Settings > Phone Button Template»

Für jeden benötigten Telefontyp muss ein eigenes PhoneButton Template erstellt werden.

- Das bestehende PhoneButton Template z.B. «Standard 79XX SCCP» kopieren
- Einen neuen Namen vergeben: «Standard 79XX SCCP BLF»
- Das Layout wie gewünscht anpassen (Feature von «Speed Dial» auf «Speed Dial BLF» ändern)

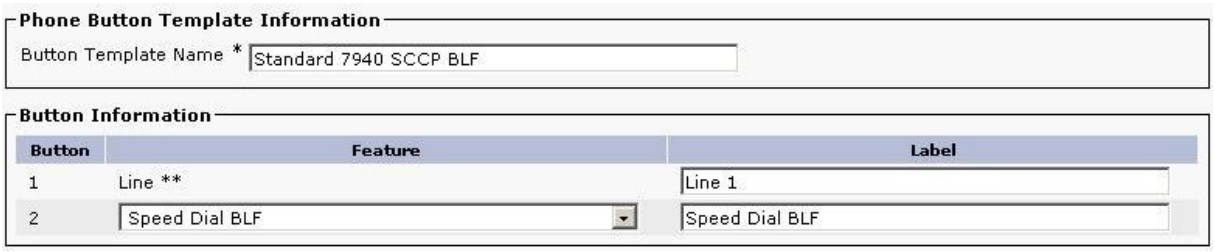


Abbildung 1-2 PhoneButton Template erstellen

1.2.2.7. SoftKey Template erstellen

Menü: «Cisco Unified CM Administration > Device > Device Settings > Softkey Template»

- Das «Standard Assistant» Template kopieren
- Einen neuen Namen vergeben, z.B. «Standard Assistant Pickup»
- Speichern
- Rechts oben über «Relatet Links:» «Configure Softkey» Layout auswählen -> «Go»
- Das Layout wie gewünscht anpassen (die beiden Funktionen «Pick Up» und «Group Pick Up» zur Auswahl hinzufügen)

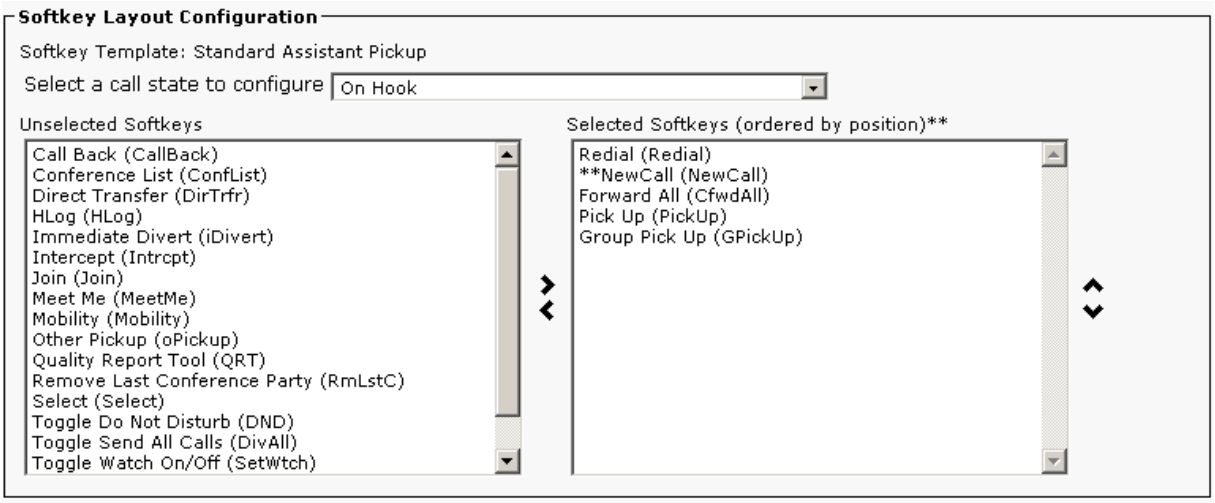


Abbildung 1-3 SoftKey Template erstellen

1.2.2.8. Phones einrichten

Menü: «Cisco Unified CM Administration > Device > Phone»

ACHTUNG: Bei jedem erstellten Phone muss über das Untermenü «Related Links > Subscribe/Unsubscribe Service» der «ExtMob Service» für das definierte Phone abonniert werden.

Phone 7970

- Phone Type: Cisco 7970
- Select the device protocol: SCCP
- MAC Address: 000ED7ABFE04
- Device Pool: Default
- PhoneButton Template: Standard 7970 SCCP
- Device Security Profile: Cisco 7970 - Standard SCCP Non-Secure Profile
- Enable Extension Mobility (aktivieren)

Phone 7960

- Phone Type: Cisco 7960
- Select the device protocol: SCCP
- MAC Address: 00075079D063
- Device Pool: Default
- PhoneButton Template: Standard 7960 SCCP
- Device Security Profile: Cisco 7960 - Standard SCCP Non-Secure Profile
- Enable Extension Mobility (aktivieren)

Phone 7940

- Phone Type: Cisco 7940
- Select the device protocol: SCCP
- MAC Address: 000D65707B8A
- Device Pool: Default
- PhoneButton Template: Standard 7940 SCCP
- Device Security Profile: Cisco 7940 - Standard SCCP Non-Secure Profile
- Enable Extension Mobility (aktivieren)

1.2.2.9. PhoneProfile einrichten

Menü: «Cisco Unified CM Administration > Device > Device Settings > Device Profile»

ACHTUNG: Bei jedem erstellten Phone Profile muss über das Untermenü «Related Links > Subscribe/Unsubscribe Service» der «ExtMob Service» für das definierte Phone Profile abonniert werden.

Phone Profile 7970

- Device Profile Type: Cisco 7970
- Device Profile Name: deviceprofile7970
- PhoneButton Template: Standard 7970 SCCP
- Line[1] wählen: Directory Number: 1070 Route Partition: Abteilung1 CSS: CSSALL

Phone Profile 7960

- Device Profile Type: Cisco 7960
- Device Profile Name: deviceprofile7960
- PhoneButton Template: Standard 7960 SCCP
- Line[1] wählen: Directory Number: 1060 Route Partition: Abteilung2 CSS: CSSALL

Phone Profile 7940

- Device Profile Type: Cisco 7940
- Device Profile Name: deviceprofile7940
- PhoneButton Template: Standard 7940 SCCP
- Line[1] wählen: Directory Number: 1040 Route Partition: Abteilung3 CSS: CSSALL

1.2.2.10. User einrichten

Menü: «Cisco Unified CM Administration > User Management > End User»

User 7970

- User ID: user7970
- Password: user7970
- PIN: 7970
- Last name: user7970
- Available Profiles: deviceprofile7970 hinzufügen
- Device Association (SEP000ED7ABFE04) wählen (nach Usererstellung)

User 7960

- User ID: user7960
- Password: user7960
- PIN: 7960
- Last name: user7960
- Available Profiles: deviceprofile7960 hinzufügen
- Device Association (SEP00075079D063) wählen (nach Usererstellung)

User 7940

- User ID: user7940
- Password: user7940
- PIN: 7940
- Last name: user7940
- Available Profiles: deviceprofile7940 hinzufügen
- Device Association (SEP000D65707B8A) wählen (nach Usererstellung)

Wenn alle Einstellungen vorgenommen wurden, können alle definierten Phones über den Menüpunkt «Cisco Unified CM Administration > Device > Phone > Reset» neu gestartet werden.

1.2.3. Telefon-Konfiguration

Um die Konfiguration der Telefone ändern zu können, muss das Telefon mittels «**#» entsperrt werden. Um nicht bei jedem Telefon eine fixe IP-Adresse vergeben zu müssen, werden sie auf DHCP umgestellt.

Installationsanleitung

Dieser Teil der Dokumentation beinhaltet eine Installationsanleitung, damit die Applikation ohne grossen Aufwand installiert werden kann. Sie beinhaltet folgende Punkte:

- Installationsanleitung für den Endbenutzer
- Installationsanleitung für den Entwickler

2. Installationsanleitung

Im folgenden sind für den Endbenutzer sowie für den Entwickler Installationsanleitungen enthalten.

2.1. Installationsanleitung für Endbenutzer

Dieses Kapitel beinhaltet eine Installationsanleitung um Mabata zu installieren.

2.1.1. Voraussetzungen

WebServer:	Apache Tomcat 6.0.x installiert
Datenbankserver:	PostgreSQL 3.8 installiert
Java Runtime:	1.6 installiert
Communications Manager:	Version 6.x oder 7.x installiert
	AXL Web Service aktiviert (Menü: «Cisco Unified Serviceability > Tools > Service Activation»)
Systemanforderungen:	1 GB Arbeitsspeicher
	Pentium 4 Prozessor
	1000Mbit/s Netzwerkkarte
	2 GB Festplattenspeicher
	Windows XP/Windows 2003 Server (Virtualisierung möglich)

2.1.2. Installation

2.1.2.1. Schritt 1: Datenbank erstellen

Es wird eine leere Datenbank benötigt. Erstellen Sie dazu eine neue Datenbank auf einem bestehenden PostgreSQL Server. Als Encoding muss UTF8 gewählt werden. Auf dem Bild sehen Sie ein Beispiel. Merken Sie sich den Namen der Datenbank sowie den Benutzernamen und das Passwort des Datenbank-Owners.

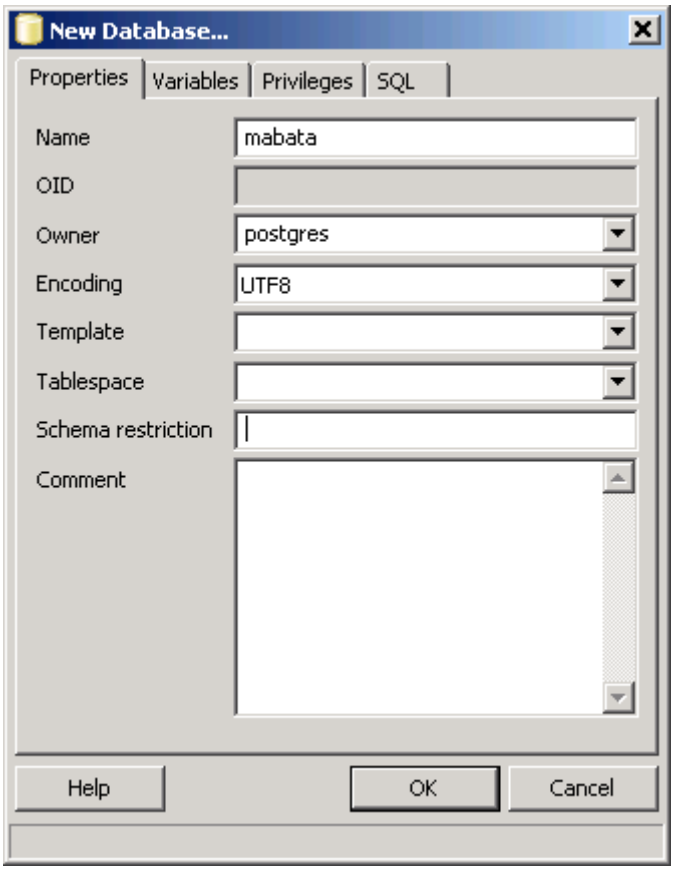


Abbildung 2-1 Erstellen der neuen Datenbank

2.1.2.2. Schritt 2: Mabata deployen

Um Mabata deployen zu können, gibt es zwei verschiedene Möglichkeiten. Schritt 2a beschreibt das manuelle Deployen, Schritt 2b das Automatische.

Schritt 2a: Mabata deployen (manuell)

Im Ordner Mabata auf der CD finden Sie eine Datei namens «Mabata.war». Kopieren Sie diese Datei in das Tomcat Webapps Verzeichnis. Der Pfad bei der Standard-Installation lautet:

`«C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps»`

Entpacken Sie nun die «Mabata.war» Datei mit einem Entpackungsprogramm, z.B. 7-Zip. Das Tomcat Webapps Verzeichnis sollte jetzt aussehen wie im Bild unten dargestellt. Sie können für die Datei auch einen eigenen Namen vergeben. Dieser Name bildet einen Teil der URL. Heisst die Datei «Mabata.war», lautet die URL «http://localhost/**mabata**».

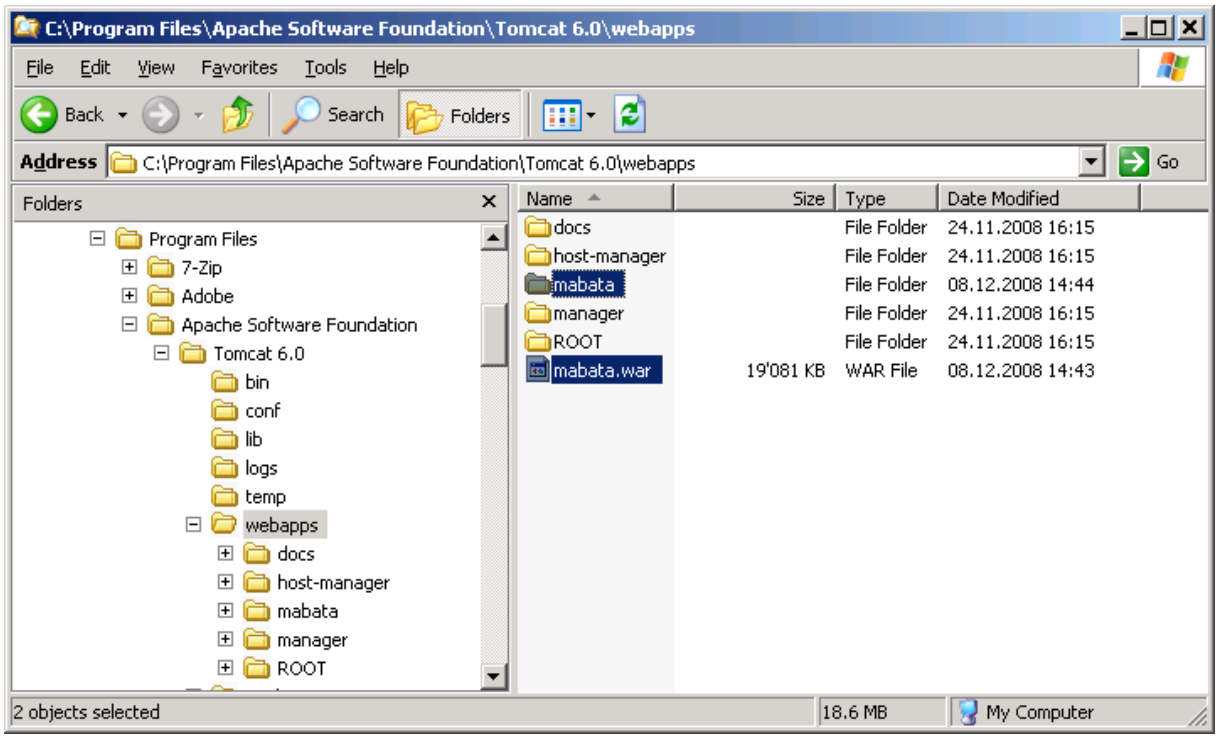


Abbildung 2-2 Tomcat Verzeichnis

Schritt 2b: Mabata deployen (automatisch)

Sie können auch den Tomcat Manager verwenden um die Datei zu deployen. Starten Sie hierfür den Tomcatmanager (Standardinstallation über «http://localhost:8080/manager/html»).

Im Bereich Deploy unter «War file to deploy» können Sie die WAR-Datei suchen und per «Deploy» auf dem Server veröffentlichen.

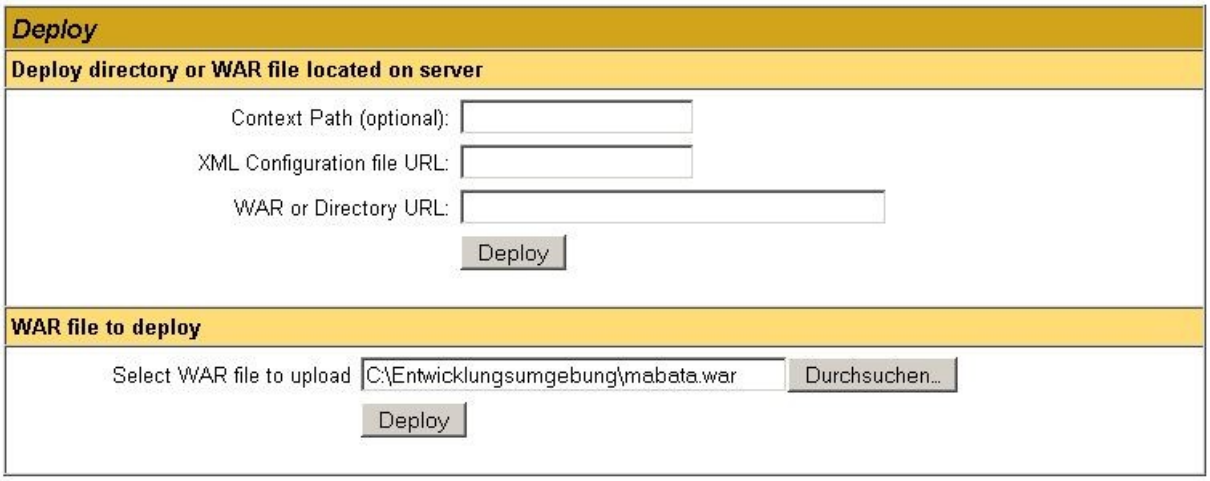


Abbildung 2-3 Tomcat Manager Deploy

Danach erhalten Sie die folgende Meldung:

Abbildung 2-4 Deploy erfolgreich

Um die Konfiguration abzuschliessen muss der Tomcat Server beendet werden.

2.1.2.3. Schritt 3: Mabata Konfiguration laden

Starten Sie nun den Mabata-Konfigurationsassistenten mit einem Doppelklick auf die Datei «Konfigurationsassistent.jar» im Verzeichnis «Mabata/Konfigurationsassistent».

Wählen Sie den in Schritt 2 entpackten Ordner aus:

«`C:\Program Files\Apache Software Foundation\Tomcat 6.0\webapps\mabata`»

Abbildung 2-5 Pfad zur Mabata-Installation

Mittels «Konfiguration laden» wird die bestehende Konfiguration geladen.

2.1.2.4. Schritt 4: Mabata Konfiguration anpassen

Passen Sie nun alle Parameter an Ihre Umgebung an. In der Tabelle werden alle Parameter mit einem Beispiel erklärt.

Parameter	Wert	Beispiel
Debug	Debug Nachrichten des Loginmodules ausgeben	true/false
Mabata Standard Passwort	Standardpasswort, welches die Benutzer erhalten die importiert werden können	m@b@t@2009
Middleware WebService URL	Die URL, unter welcher die Middleware erreichbar ist	http://localhost:8081/webservice?wsdl
Middleware WebService Passwort	Passwort, welches den WebService vor unberechtigten Zugriffen schützt	m@b@t@2009
Extension Mobility URL	URL des Extension Mobility Services wie auf dem Communications Manager eingerichtet (Menü: «Device > Device Settings > Phone Servies» den «ExtMob» Service wählen und die URL herauslesen)	http://localhost:8080/emapp/EMAppServlet?device=#DEVICENAME#
Extension Mobility Name	Name des Extension Mobility Service, wie er auf dem Communications Manager konfiguriert wurde	ExtMob
ID des ExtMob Services	ID des ExtMob Services auf dem Communications Manager (Menü: «Device > Device Settings > Phone Servies» den «ExtMob» Service wählen und dann kann die ID aus der URL im Browser gelesen werden)	00830540-cfaa-e481-06b7-3a1df8e1ed82
Extension Mobility aktivieren	Bestimmt, ob Extension Mobility aktiviert oder deaktiviert werden soll	true/false
Datenbankbenutzer	Benutzername der Datenbank	postgres
Datenbankpasswort	Passwort des Datenbankbenutzers	m@b@t@2009
Datenbankverbindung	Verbindung zur PostgreSQL Datenbank	jdbc:postgresql://localhost:5432/mabata

Tabelle 2-1 Konfigurationsparameter

Werden alle Daten auf derselben Datenbank gespeichert, können die Werte der anderen Datenbanken mittels des Buttons auf der Seite übernommen werden.

Der Parameter «mabata» muss mit jenem der Middleware übereinstimmen. Die «WebServiceURL» enthält den gleichen Wert wie der Parameter «wsdlLocation» der Middleware. Zusätzlich muss jedoch «?wsdl» angehängt werden.

Es wird geprüft, ob alle Parameter einen Wert enthalten. Ist ein Parameter noch leer, kann nicht gespeichert werden. Sobald Sie alle Parameter angepasst haben, können Sie den Tomcat-Server starten.

2.1.2.5. Schritt 5: Middleware konfigurieren

Im gleichen Verzeichnis wie die «Middleware.jar» befindet sich die Datei «middleware.properties».

Der Abschnitt WebService muss folgendermassen angepasst werden:

«wsdlLocation» bezeichnet den Ort an welchem die Middleware auf eingehende Verbindungen wartet. Beispiel:

```
wsdlLocation = http://localhost:8081/webservice
```

«mabata» ist das Passwort, welches der WebService benötigt um eingehende Zugriffe authentifizieren zu können.

```
mabata = m@b@t@2009
```

Der Abschnitt CUCM enthält die Verbindungsparameter zum Communications Manager:

```
#******CUCM 6.0***********
CCM_HOST = 192.168.202.142    //IP-Adresse des CUCM
CCM_PORT = 8443               //Netzwerkport des CUCM
CCM_USER = admin              //Benutzername des Administrators
CCM_PASSWORD = +mabata1+      //Passwort des Administrators
```

2.1.2.6. Schritt 6: Middleware starten

Im Verzeichnis «Middleware» die BAT-Datei «Start Middleware.bat» ausführen und warten bis die Meldung *«Server up and running!»* erscheint.

Abbildung 2-6 Middleware läuft

Als Alternative zur Batch-Datei kann die Middleware auch als Windows-Dienst eingerichtet werden. Hierzu wird das JavaService-Tool verwendet, welches unter «http://forge.ow2.org/projects/javaservice/» verfügbar ist.

Um die Middleware als Dienst einzurichten, muss folgender Befehl in der CMD-Umgebung eingegeben werden:

```
JavaService.exe -install ServiceName %JAVA_HOME%\jre\bin\server\jvm.dll
-Djava.class.path=Middleware.jar
-start org.middleware.webservice.WebServiceServer
```

Wobei «ServiceName» für irgendeinen frei verfügbaren Namen des späteren Services steht. «JavaService.exe» und «Middleware.jar» müssen mit dem absoluten Pfad angegeben werden.

2.1.2.7. Schritt 7: Mabata starten / Datenbank initialisieren

Starten Sie Mabata, in dem Sie im Browser «http://localhost:8080/mabata» eingeben. Falls Sie unter Schritt 2 einen anderen Namen angegeben haben, müssen Sie die Adresse dementsprechend anpassen. Sollte Ihr Tomcat Server auf einen anderen Port eingestellt sein, müssen Sie auch dies berücksichtigen. Wenn die Startseite erscheint, kann die Mabata Datenbank über die Adresse «http://localhost:8080/mabata/setup.iface» initialisiert werden. Gleichzeitig muss das Passwort für den Administrator festgelegt werden.

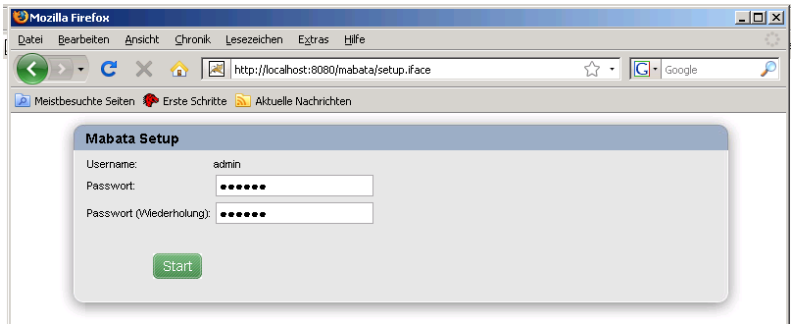


Abbildung 2-7 Initialisierung der Mabata Datenbank

Nach erfolgreicher Initialisierung erhalten Sie folgende Meldung:

Abbildung 2-8 Datenbank wurde erfolgreich initialisiert

Sie können Sich mittels der grafischen Datenbank-Oberfläche vergewissern, dass alle Tabellen erstellt wurden. Starten Sie dazu den pgAdmin III:

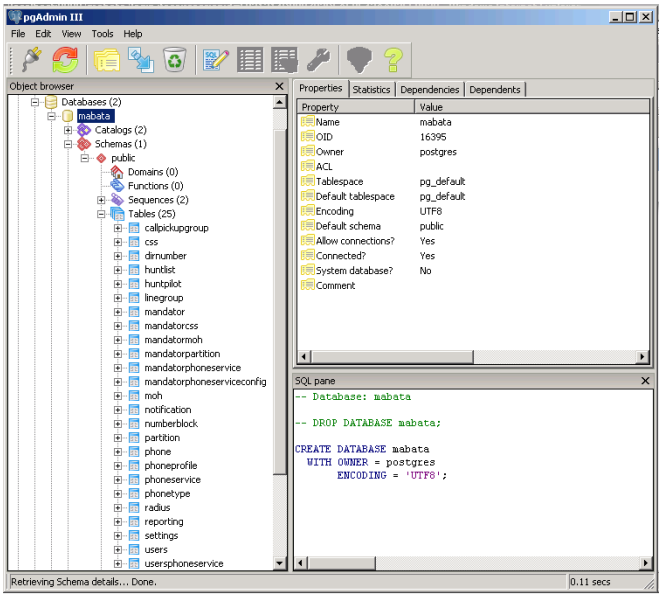


Abbildung 2-9 Alle Tabellen wurden angelegt

Falls Sie Probleme mit diesem Schritt haben, können Sie die Log-Dateien des Tomcat Servers überprüfen. Diese finden Sie unter: «C:\Program Files\Apache Software Foundation\Tomcat 6.0\logs»

2.1.2.8. Schritt 8: Erstmaliges Anmelden

Mabata ist nun fertig installiert und konfiguriert. Über den Button «Login» kann Mabata gestartet werden.

Falls sich auf Ihrem Communications Manager bereits Daten befinden, können Sie diese über die Importfunktion importieren. Vergessen Sie nicht, vorgängig einen Mandanten sowie einen dazugehörigen Nummernblock zu erstellen.

2.2. Installationsanleitung für Entwickler

2.2.1. Voraussetzungen

Webserver:	Apache Tomcat 6.0.x installiert
Datenbankserver:	PostgreSQL 3.8 installiert
Java Development Kit:	1.6 installiert
Communications Manager:	Version 6.x installiert und gemäss Kapitel 1.2.2 konfiguriert.
Eclipse:	Eclipse IDE for Java EE Developers 3.4.1 mit JDK 6 installiert inkl: Ajax-Framework für Java: ICEfaces Projektintegration Version 1.7.2 für Eclipse 3.4.
SVN-Plugin:	SVN-Plugin Subclipse (Version 1.4.5)

2.2.2. Integration des Projekts in Eclipse

Das bestehende Projekt kann auf zwei Arten in Eclipse integriert werden. Entweder von einem SVN-Server oder über die Archiv-Dateien, welche auch die Source-Dateien enthalten.

2.2.2.1. Integration per SVN

Mittels dem integrierten SVN Plugin das Repository öffnen und das gewünschte Projekt mittels «Checkout…» ins Eclipse importieren.

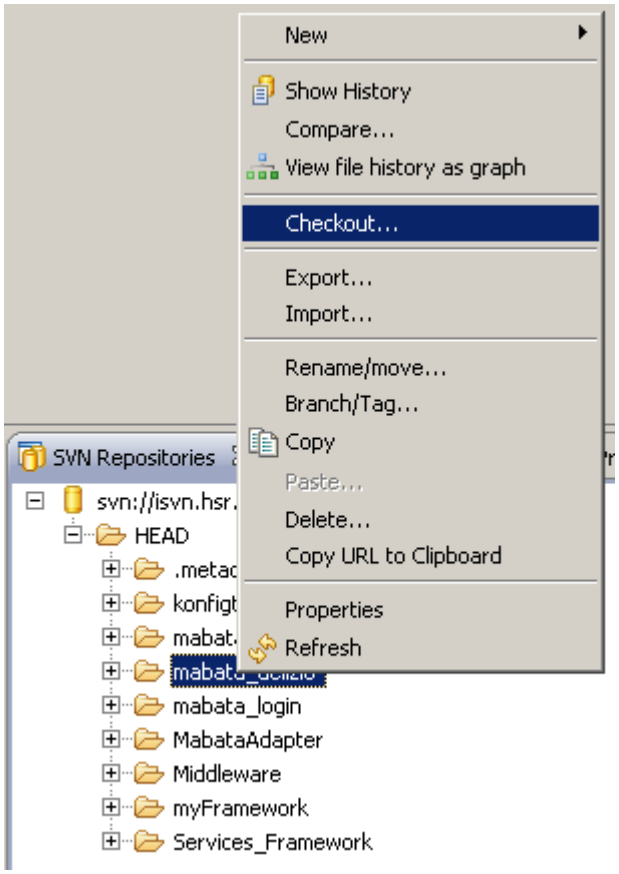


Abbildung 2-10 Checkout der Projekte

2.2.2.2. Integration aus der Archiv-Datei

In Eclipse über «Datei -> Import -> Web -> WAR» den Dialog für den Import von WAR-Dateien öffnen. Unter «War file:» die mitgelieferte WAR-Datei öffnen.

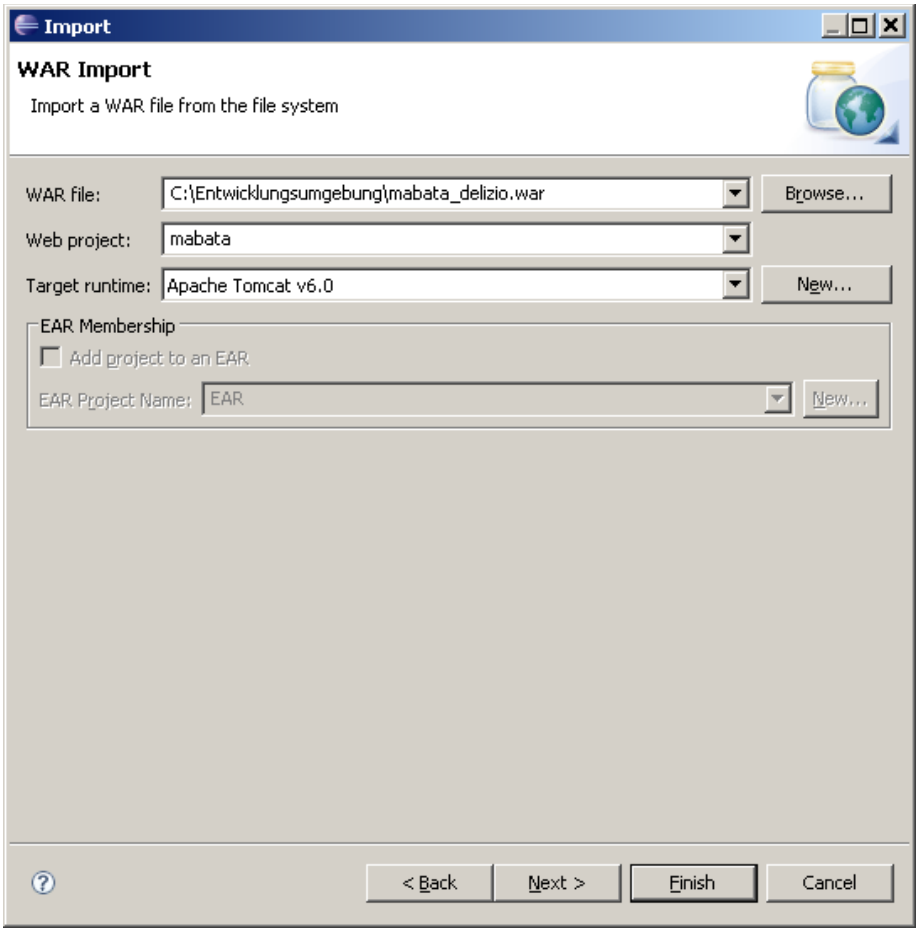


Abbildung 2-11 Import der Projekte

2.2.3. Konfiguration der Projekte

Werden die Projekte in Eclipse geöffnet, kann der Konfigurationsassistent nicht verwendet werden. Daher müssen die einzelnen Konfigurationsdateien manuell konfiguriert werden.

2.2.3.1. Mabata Loginmodul

Die Konfiguration für das Loginmodul befindet sich in diesem Ordner:

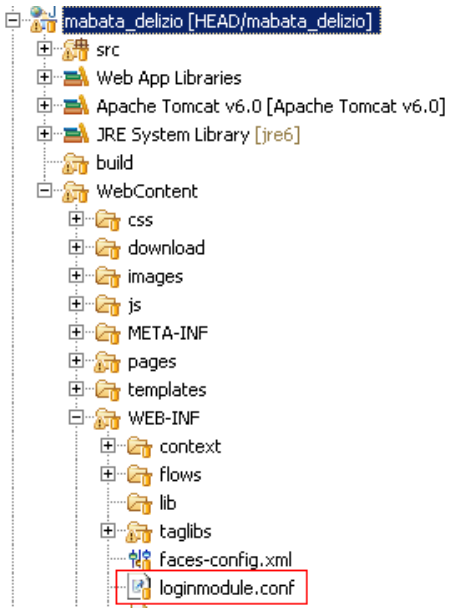


Abbildung 2-12 Ort der loginmodule.conf

Darin sind alle Parameter enthalten die benötigt werden, damit sich das Loginmodul an der Mabata Datenbank anmelden kann.

```
mabataLM {
      org.mabata.security.MabataLoginModule required
      debug="true"
      dbdriver="org.postgresql.Driver"
      db="jdbc:postgresql://127.0.0.1:5432/mabata"
      dbuser="postgres"
      dbpassword="Student_09"
      localdbradiusname="local"
      usertable="users"
      usernameattr="userid"
      userpasswordattr="password"
      userradiuskeyattr="radiusid"
      userrolesattr="userrole"
      radiustable="radius"
      radiustablekeyattr="radiusid"
      radiusnameattr="radiusname"
      radiushostattr="radiushost"
      radiushostsharedsecretattr="sharedsecret"
      radiusauthportattr="authport"
      radiusaccportattr="accport";
};
```

Abbildung 2-13 Inhalt der loginmodule.conf

2.2.3.2. Mabata Allgemein

Die restlichen drei Konfigurationsdateien («hibernate.cfg.xml», «log4j.properties», «mabata.properties») befinden sich hier:

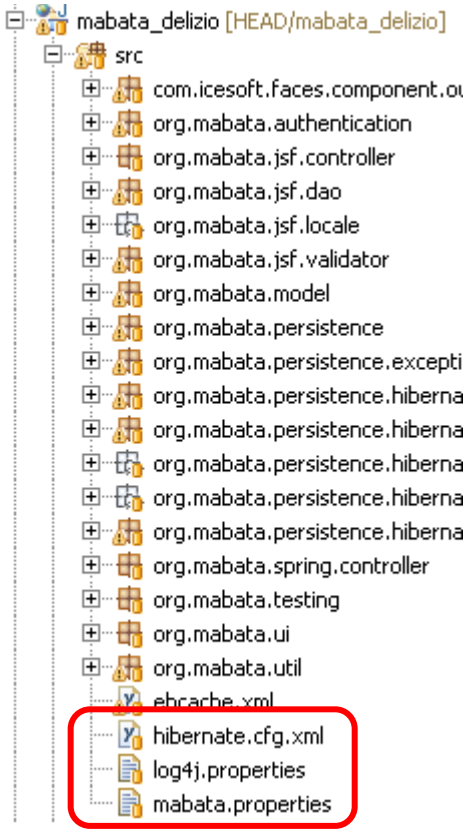


Abbildung 2-14 Übrige Konfigurationsdateien

Hibernate Konfiguration

Die Hibernate Konfiguration «hibernate.cfg.xml» enthält, wie der Namen schon sagt, alle Parameter damit Hibernate mit der zugrunde liegenden Datenbank kommunizieren kann.

Hier sind vor allem die Einstellungen für die lokale Datenbankverbindung anzupassen.

```
<property name="connection.username">postgres</property>
<property name="connection.url">
jdbc:postgresql://127.0.0.1:5432/mabata</property>
<property name="connection.password">m@b@t@2009</property>
```

Log4j Konfiguration

Die Log4j Konfiguration «log4j.properties» beinhaltet alle Parameter, welche für ein erfolgreiches Aufzeichnen benötigt werden.

```
# Der Root-Logger hat den Level FATAL
log4j.rootCategory=FATAL, SYSLOG, file

# Konfiguration des Syslog-Servers ######################
log4j.appender.SYSLOG=org.apache.log4j.net.SyslogAppender

# Pfad zum Syslog-Server
log4j.appender.SYSLOG.syslogHost=localhost
log4j.appender.SYSLOG.layout=org.apache.log4j.PatternLayout
log4j.appender.SYSLOG.layout.ConversionPattern=%m
log4j.appender.SYSLOG.Facility=USER

# Konfiguration der Log-Datei ###########################
log4j.appender.file=org.apache.log4j.RollingFileAppender
log4j.appender.file.layout=org.apache.log4j.PatternLayout
log4j.appender.file.layout.ConversionPattern=%m%n

# Pfad zur Log-Datei
log4j.appender.file.File=${catalina.home}/logs/mabata.log

# Maximale Grösse der Log-Datei
log4j.appender.file.MaxFileSize=1000KB
# Eine Backup-Datei behalten
log4j.appender.file.MaxBackupIndex=1

# Abspeicherung der Levels ##############################
level.error=true
level.notice=true
level.alert=true
level.debug=true
currentSyslogURL=localhost
```

Mabata Konfiguration

Die Mabata Konfiguration «mabata.properties» enthält das Standardpasswort, welches ein Benutzer bekommt, wenn er importiert wird. Zudem ist auch die Adapterkonfiguration darin enthalten. Diese wird benötigt um den Zugriff auf die Middleware herzustellen.

```
# Import Konfiguration ##################################
default_password=m@b@t@2009

# WebService ############################################
WebServiceURL = http://localhost:8081/webservice?wsdl
# Passwort
mabata = m@b@t@2009

#****** Extension Mobility ******
ISSERVICECHECKED = true
TELECASTERSERVICEUUID = {00830540-cfaa-e481-06b7-3a1df8e1ed82}
EXTENSIONMOBILITYNAME = ExtMob
EXTENSIONMOBILITYURL = http://192.168.202.141:8080/emapp/
                       EMAppServlet?device=#DEVICENAME#
```

2.2.4. Exportieren der einzelnen Projekte

Um die Projekte korrekt zu exportieren sind einige Punkte zu beachten. Diese sind nachfolgend beschrieben.

2.2.4.1. Middleware Projekt exportieren

Beim Exportieren der Middleware ist darauf zu achten, dass nur die Packages ausgewählt sind, nicht aber die Properties-Datei. Diese muss separat im gleichen Ordner wie die JAR-Datei bereitgestellt werden.

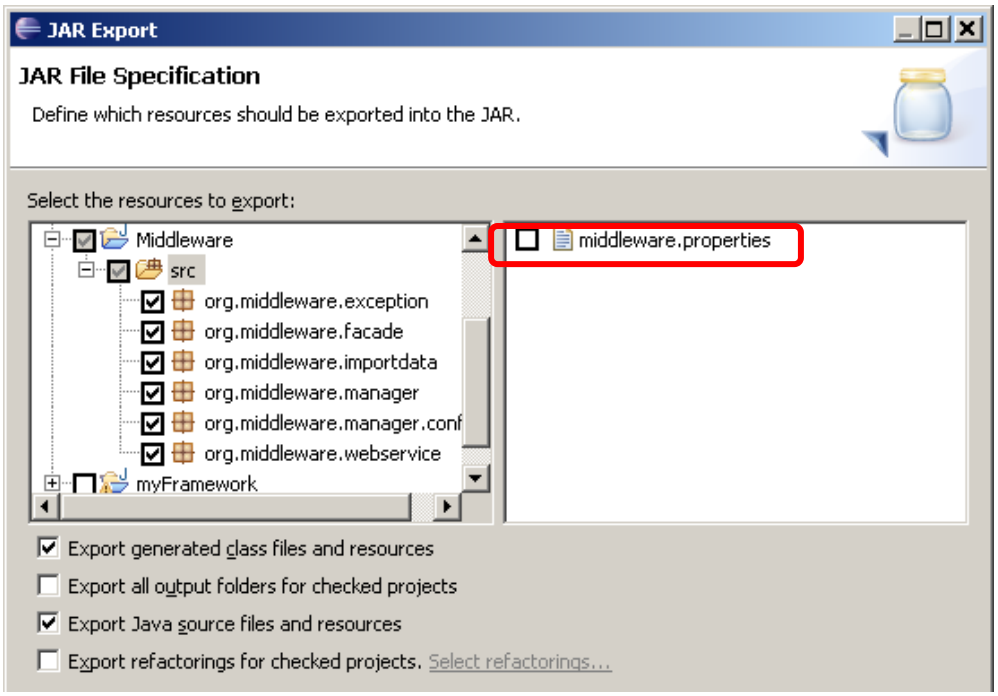


Abbildung 2-15 Exportieren der Middleware

Beim Exportieren ist darauf zu achten, dass das bereitgestellte Manifest verwendet wird.

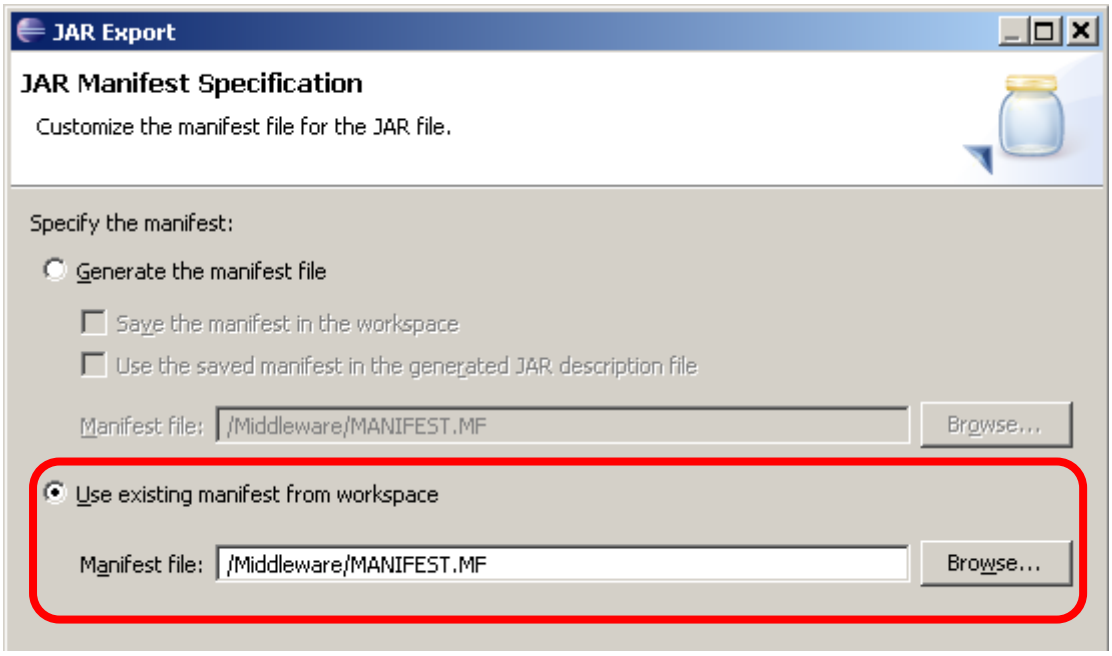


Abbildung 2-16 Auswählen des Manifestes

2.2.4.2. Konfigurationsassistent exportieren

Beim Exportieren ist darauf zu achten, dass das bereitgestellte Manifest verwendet wird.

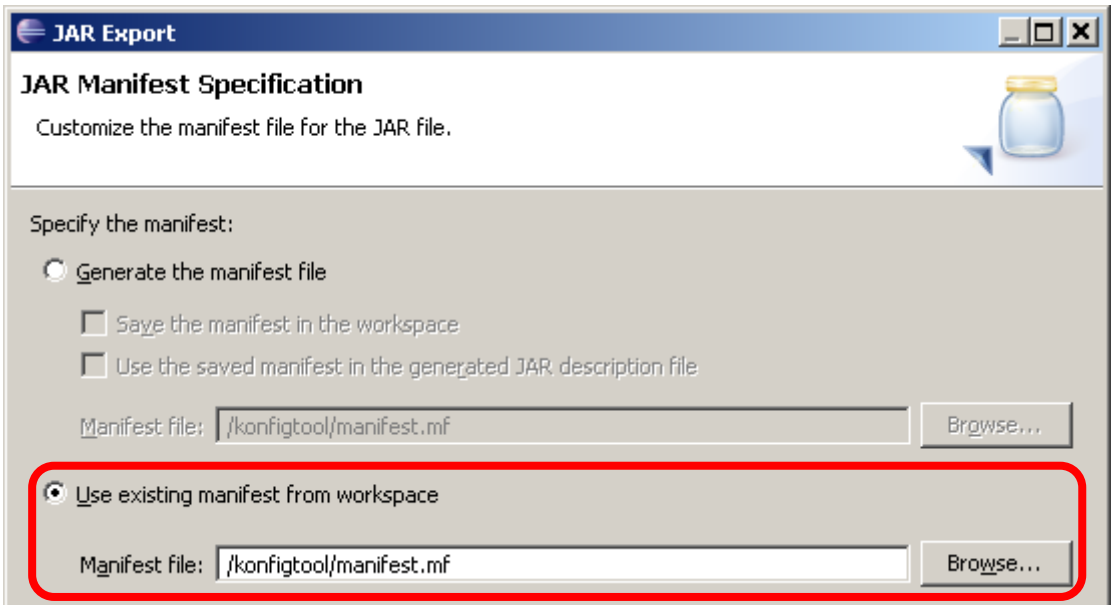


Abbildung 2-17 Exportieren des Konfigurationsassistenten

Weiter benötigt das Konfigurationstool vier Bibliotheken für den erfolgreichen Start.

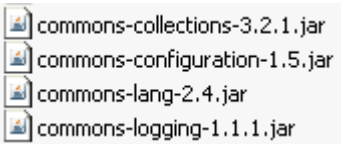


Abbildung 2-18 Zusätzliche Bibliotheken

2.2.4.3. Mabata exportieren

Mabata muss als WAR-Datei exportiert werden.

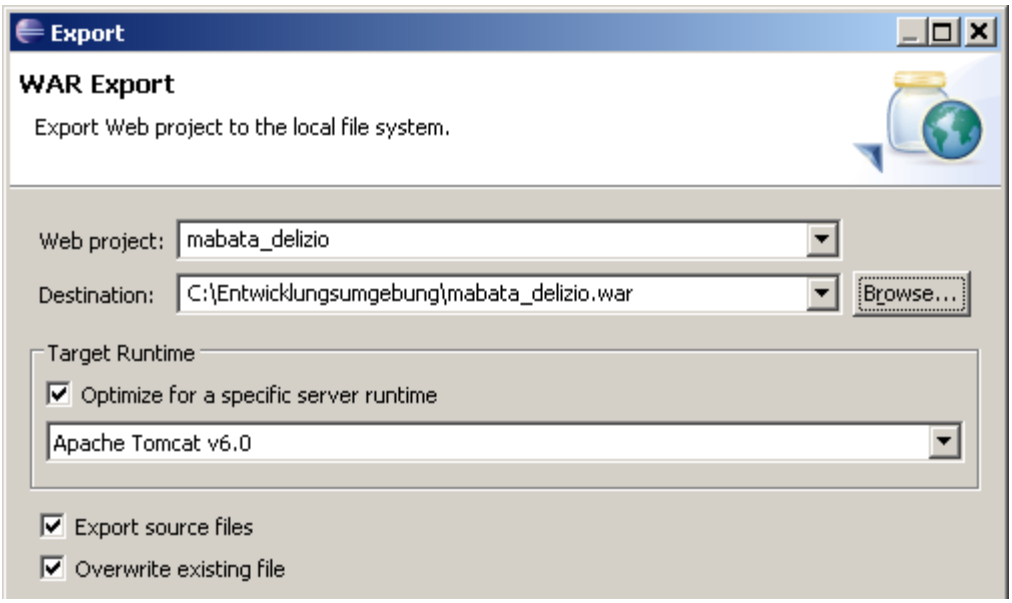


Abbildung 2-19 Mabata Exportieren

Analyse

Im nachfolgenden Teil der Dokumentation wird die Analyse behandelt, welche folgendermassen unterteilt ist:

- Übersicht
- New VoIP Source Architecture
- Mabata Business Case Vereinfachung
- Weitere Funktionen

3. Analyse

3.1. Übersicht

3.1.1. Technologien

Nachfolgend sind die von Mabata benutzten Technologien beschrieben.

3.1.1.1. Frontend

Das Frontend wurde mit Spring 2.5.2, JSF 1.1 basierend auf Facelets und ICEfaces 1.7.2-SP1 for Eclipse 3.4 erstellt.

3.1.1.2. Security

Als Security Framework wird Acegi Security 1.0.6 verwendet. Dieses ergänzt das Spring Framework. Das JAAS Login-Modul dient als Authentisierungs-Modul für Acegi Security.

3.1.1.3. Applikationsserver

Als Applikationsserver wird Apache Tomcat verwendet. Es wird die Version 6.0 verwendet.

3.1.1.4. Datenbank

Die Datenbank läuft auf PostgreSQL Version 8.3.

3.1.1.5. Problem Domain

Die Problem Domain von Mabata läuft auf der Java Version 6.0.

3.1.1.6. Neue Technologien

Da die Middleware selbständig auf einem eigenen Server laufen muss, werden dafür WebServices verwendet. Diese sind im JDK 6.0 enthalten. In Mabata liegt das Hauptaugenmerk darauf, das GUI benutzerfreundlicher zu gestalten sowie weitere Funktionen zu implementieren. Aus diesem Grund sind keine weiteren neuen Technologien nötig.

3.1.2. Architektur

Mabata ist in die folgenden logischen Schichten aufgeteilt. Dabei greift jede Schicht nur auf darunterliegende Schichten zu. Die Schichten werden nachfolgend erläutert.

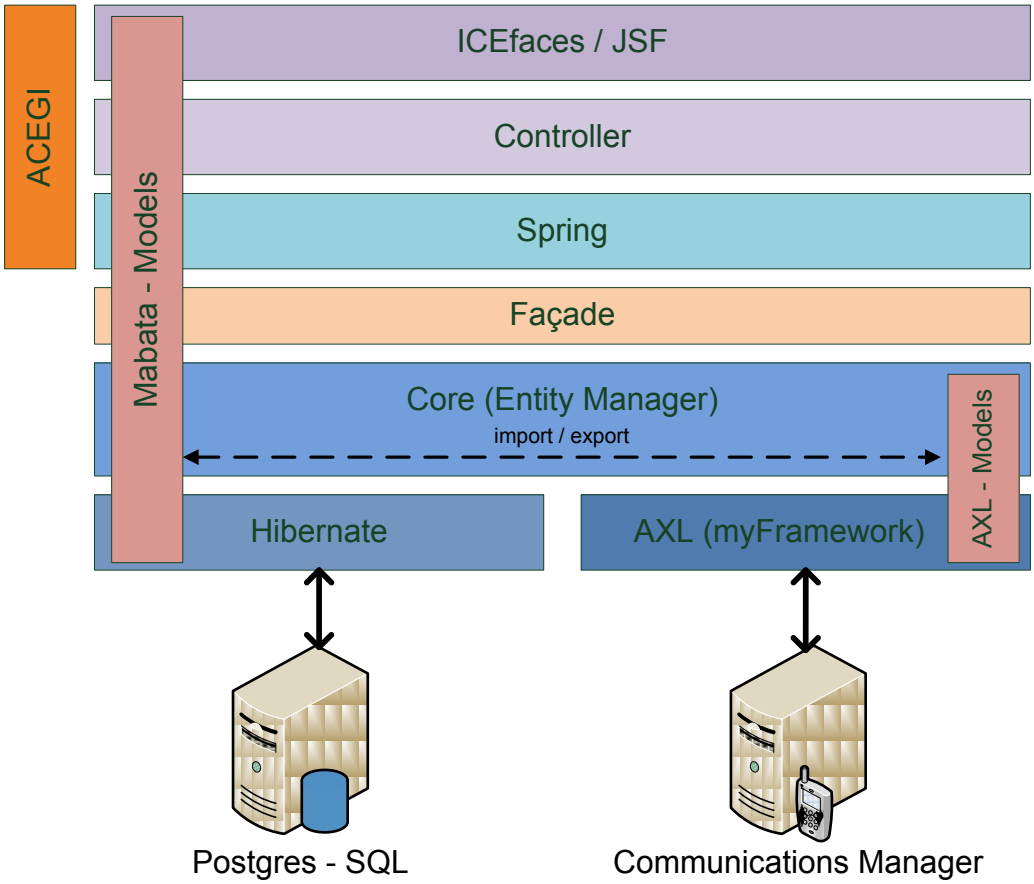


Abbildung 3-1 Architektur von Mabata

ACEGI	ACEGI ist ein Sicherheitsframework, welches für die Authentifikation und Authentisierung in Mabata verantwortlich ist.
ICEfaces / JSF	ICEfaces ist ein Framework, welches auf JSF (Java Server Faces) aufsetzt. Dieses wird in Mabata für das GUI verwendet.
Controller	Der Controller bestimmt, welche Daten im GUI angezeigt werden. Weiter ist er für die Weiterleitung von Methodenaufrufe zum Façade verantwortlich.
Spring	In Mabata wird das Spring Framework für die Dependency Injection der Façade verwendet.
Façade	Das Façade stellt ein zentraler Eintrittspunkt vom GUI zum Core dar. Alle Aufrufe vom Controller werden an das Façade weitergeleitet, welches die Methodenaufrufe im Core verteilt.

Core	Der Core besteht aus verschiedenen Entity Mangern, welche bestimmen, was ein Methodenaufruf im GUI bewirkt. Falls eine Funktion eine Speicherung von Daten mit sich zieht, ruft ein Entity Manager entweder das myFramework auf, welches die Daten auf dem Communications Manager persistiert, oder speichert über Hibernate die Daten lokal ab.
Hibernate	Hibernate ist ein Open Source Persistenz Framework welches das Objekt Relationale Mapping zwischen der Datenbank und den Java Objekten abstrahiert. Dieses Framework befreit den Entwickler vom Schreiben von SQL Abfragen und hält die Applikation unabhängig von dem verwendeten Datenbankmanagementsystem.
AXL (myFramework)	MyFramework ist ein selbst entwickeltes Framework, welches die Generierung der SOAP-Messages zum Communications Manager abstrahiert.
AXL-Models	AXL-Models sind Objekte, welche alle Datenfelder enthalten, welche in Mabata bearbeitet werden. Aus diesen Objekten erstellt das myFramework die SOAP-Messages.
Mabata-Models	Mabata-Models sind Java Objekte welche die Datenfelder enthalten, die in Mabata angezeigt werden. Sie enthalten eine import bzw. export Funktion, mit welchen die AXL-Models eingelesen bzw. erstellt werden.

3.1.3. Domainanalyse

3.1.3.1. Package Übersicht

Die vorhin beschriebenen logischen Schichten sind im nachstehenden Package Diagramm sichtbar.

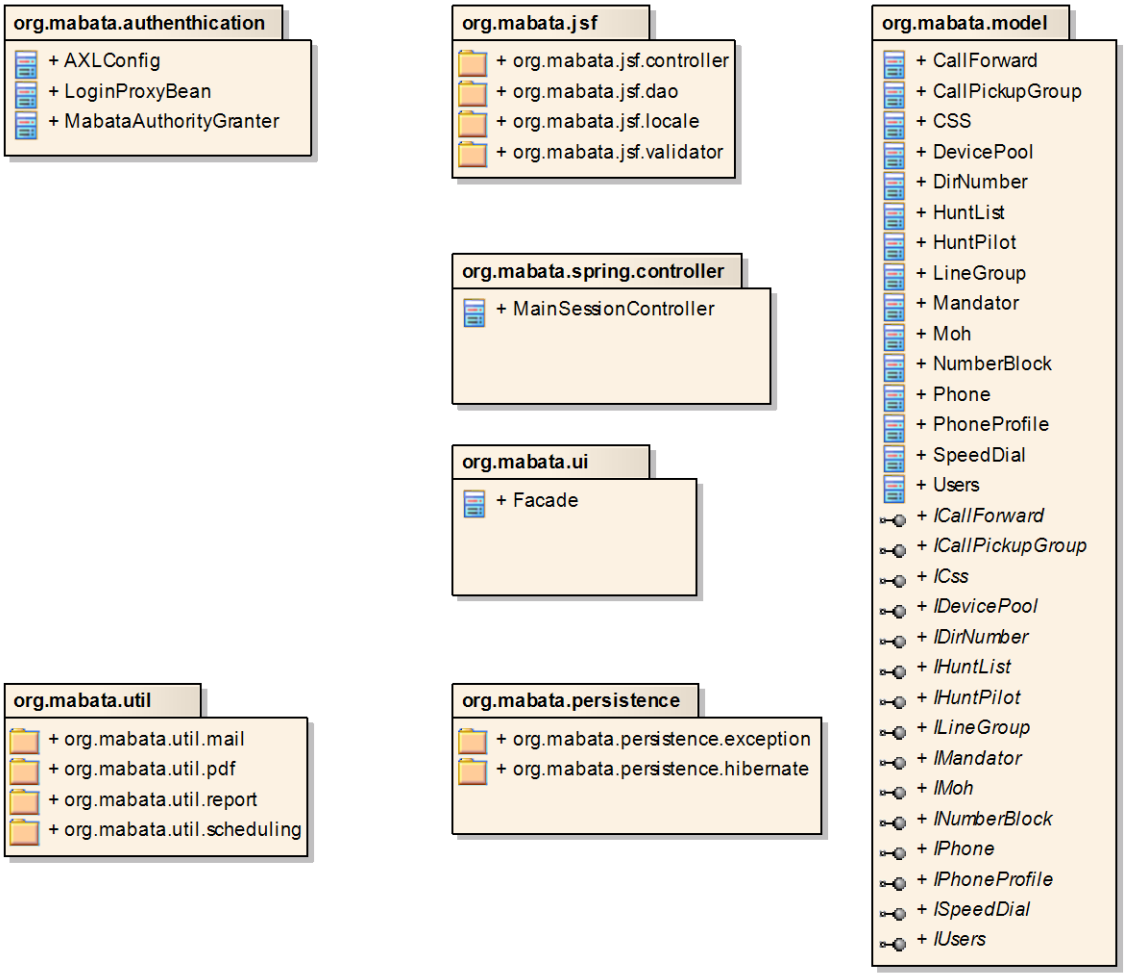


Abbildung 3-2 Package Diagramm Mabata

Package org.mabata.jsf

Für die Anzeige im GUI ist das Package «org.mabata.jsf» verantwortlich. Dieses ist wie folgt weiter unterteilt:

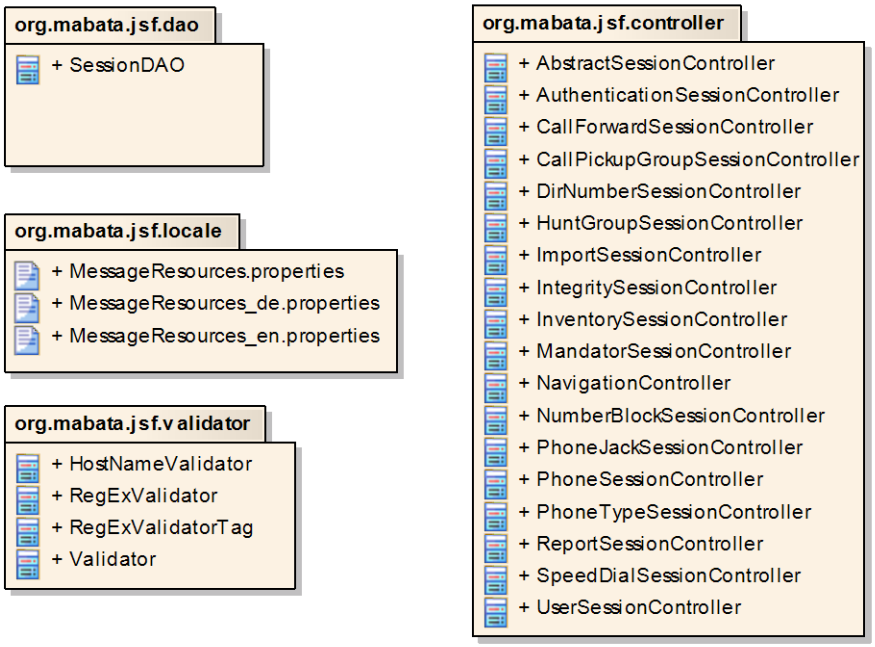


Abbildung 3-3 Package org.mabata.jsf

org.mabata.jsf.validator	Wird für die Validierung der Benutzereingaben verwendet.
org.mabata.jsf.locale	Aktuell sind alle Texte, welche ausgegeben werden, im «MessageResources.properties» abgespeichert. Dies ermöglicht eine einfache Anpassung der Texte sowie eine Internationalisierung.
org.mabata.jsf.controller	Ist für die Anzeige im GUI sowie für die Weiterleitung der Methodenaufrufe an den `SessionDAO` verantwortlich.
org.mabata.jsf.dao	Stellt die Façade für den Core bereit.

Package org.mabata.persistence.hibernate

Für die Anbindung der Datenbank wird Hibernate verwendet. Mabata verwendet hierfür zwei Packages:

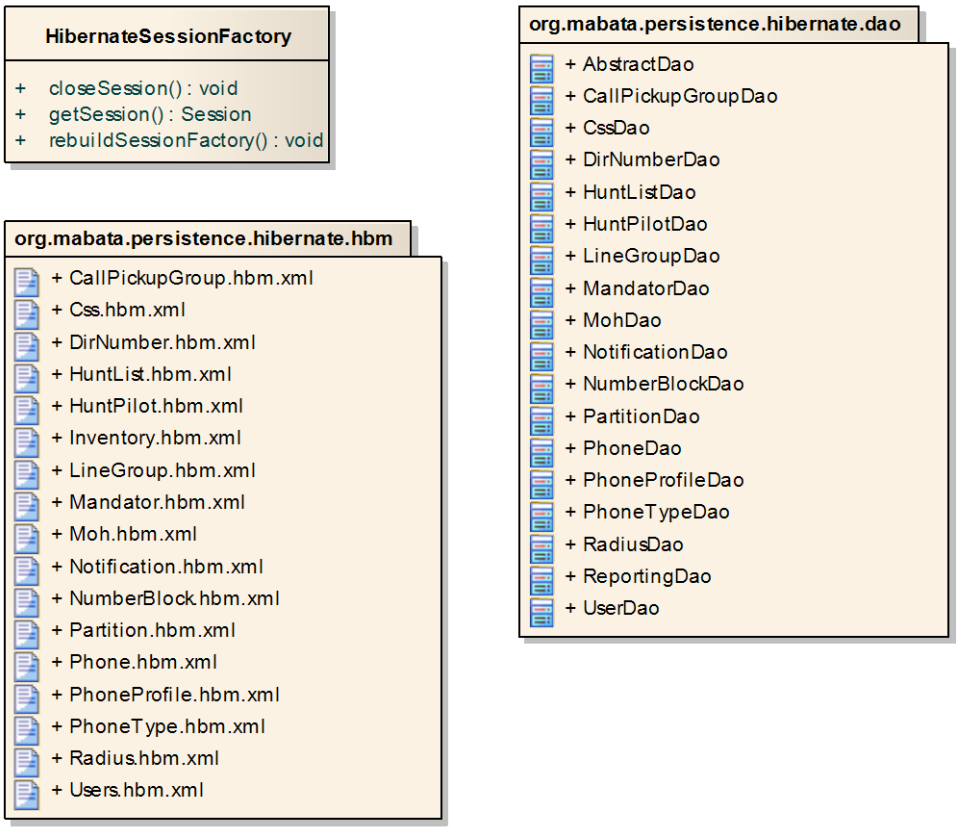


Abbildung 3-4 Package org.mabata.persistence.hibernate

org.mabata.persistence.hibernate.hbm	In diesem Package befinden sich die XML-Dateien, in welchen die einzelnen Tabellen definiert werden. Pro Tabelle existiert eine XML-Datei.
org.mabata.persistence.hibernate.dao	Pro Tabelle existiert eine DAO-Klasse, in welcher alle Datenbankabfragen definiert werden.
HibernateSessionFactory	Die `HibernateSessionFactory` stellt den DAOs ein Objekt zur Verfügung, mit welchem sie die Datenbankabfragen starten.

3.1.3.2. Domain Model

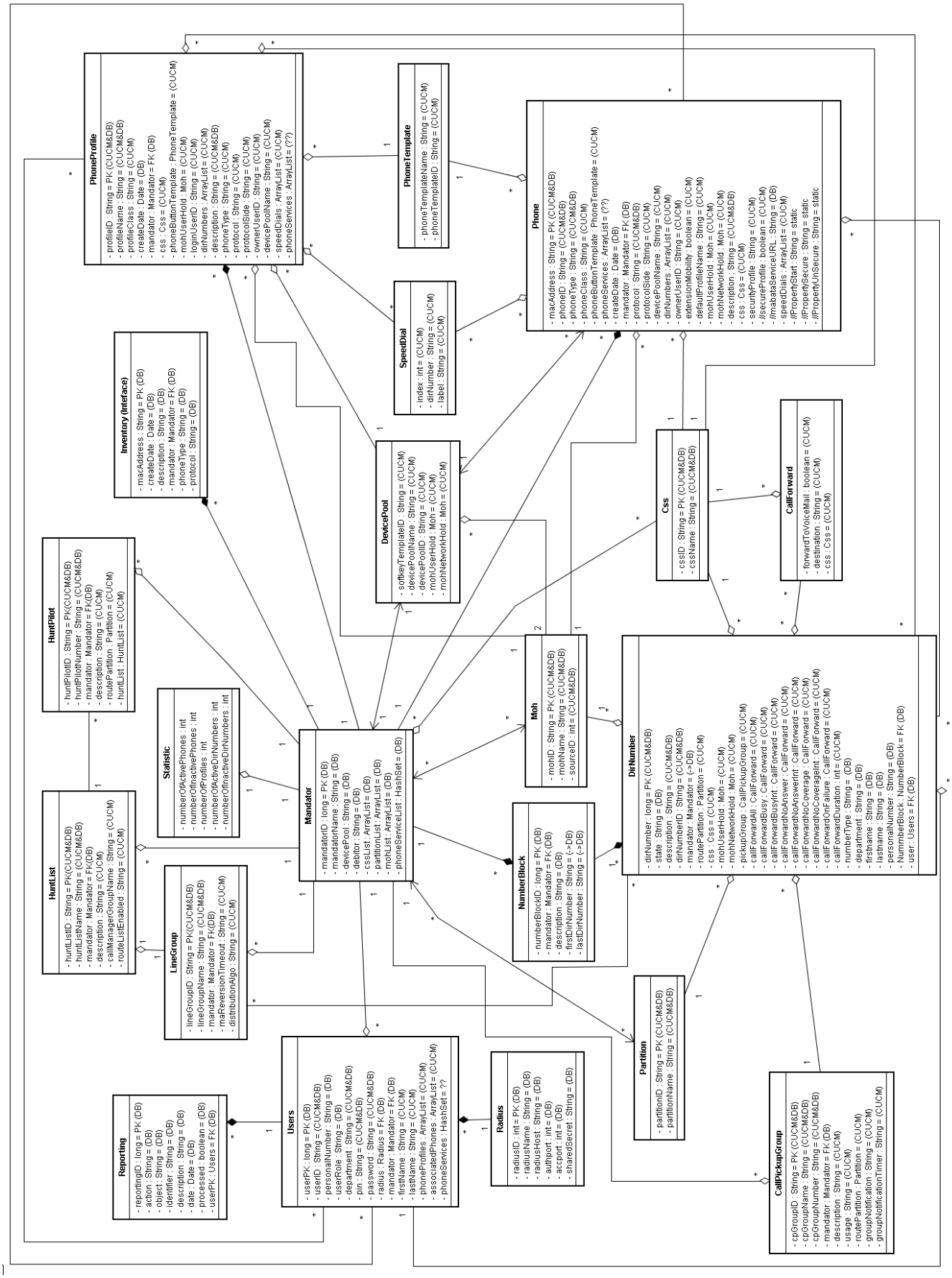


Abbildung 3-5 Domain Model

3.1.3.3. Wichtigste Klassen

Die im obigen Domainmodell abgebildeten Klassen sind Models, welche Mabata verwendet. Diese Models beinhalten jeweils alle Attribute, welche für die Datenbank, wie auch für den Communications Manager benötigt werden. Dies hat den Vorteil, dass mit den Models in jeder Software-Architektur-Schicht (von der Datenbank bis zum GUI) gearbeitet werden kann.

Nachfolgend werden die wichtigsten Models näher erläutert:

Mandator	Ein `Mandator` ist die zentrale Klasse in Mabata. Fast jedes Objekt ist einem Mandanten zugewiesen. In ihr sind alle für den Mandanten wichtigen Attribute enthalten. Persistiert wird der `Mandator` nur in der lokalen Mabata Datenbank.
User	Ein `User` ist immer einem `Mandator` zugewiesen. Wichtig für das `User`-Objekt ist, welcher Userrolle es angehört. Von der `User`-Klasse werden teils Attribute auf dem Communications Manager und teils auf der lokalen Mabata Datenbank abgespeichert.
DirNumber	Eine `DirNumber` ist immer über einen Nummernblock einem `Mandaten` zugewiesen. Dieser Nummernblock wird nur Mabata intern verwendet. Er existiert auf dem Communications Manager nicht. Eine `DirNumber` kann in Mabata verschiedene Status annehmen. Gespeichert wird die `DirNumber` auf dem Communications Manager sowohl auch einzelne Attribute in der Mabata Datenbank.

3.1.3.4. MyFramework

Das myFramework besteht aus mehreren Funktionen, welche Zugriffe auf verschiedene Services erlauben. Für Mabata wird nur ein kleiner Teil vom myFramework benötigt. Da dieses Framework aber in mehreren Projekten genutzt wird, macht es Sinn, dieses auch für Mabata zu verwenden. Dadurch muss in Zukunft nur noch ein Framework gewartet werden.

Nachfolgend eine Kurzbeschreibung der Klassen, welche in Mabata verwendet werden:

Socket	Stellt eine SSL Verbindung mit dem Server her.
AXLRequest	Erstellt aus dem Input der XML-Message die fertige SOAP-Message, welche dem Communications Manager zurückgeschickt wird. Empfängt auch den Response des Servers.
AXLModel	Generiert aus einem Model die XML-Tags, welche dem Request übergeben werden müssen.
XMLParser	Mittels dem `XMLParser` können aus einer Antwort des Servers alle Datenfelder heraus geparst werden.

Beispiel

Nachfolgendes Sequenzdiagramm stellt den Ablauf dar, wie Mabata auf den Communications Manager zugreift. Als Beispiel wird hier eine Partition abgerufen.

Als erstes muss ein `Partition`-Objekt erstellt werden. Bei dieser Partition muss nun der Name oder die ID gesetzt werden, damit die Partition anhand dieses Attributs vom Communications Manager geholt werden kann. Als nächstes wird ein Request erstellt. Diesem Request müssen die Verbindungsdaten und die soeben erstellte Partition übergeben werden. Mittels der übergebenen Partition kann der Request nun die XML-Meldung erstellen.
Dieser Request kann nun abgesendet werden. Der Request erhält daraufhin die Antwort vom Communications Manager. Diese Antwort muss er parsen und wieder in ein `Partition`-Objekt verpacken, welche dem Aufrufer zurückgegeben wird.

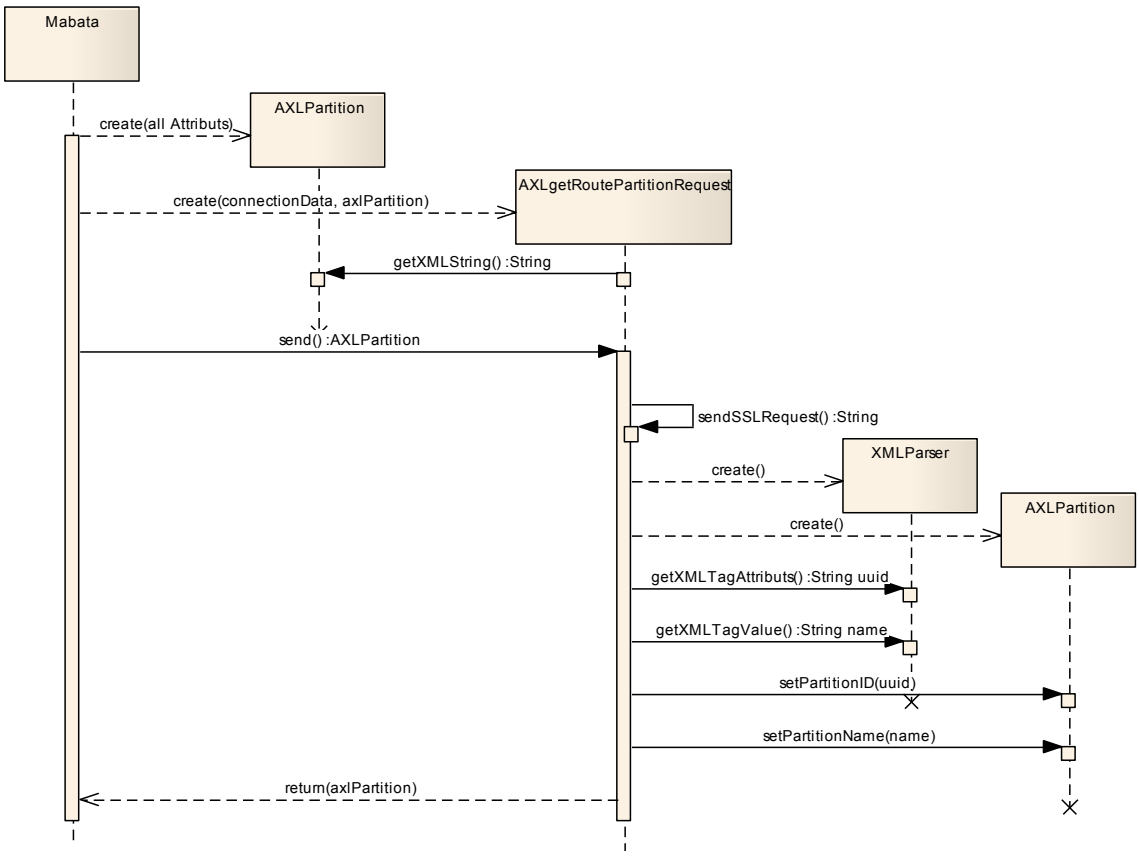


Abbildung 3-6 Zugriff Mabata auf Communications Manager

Package Diagramm

Die in Mabata relevanten Packages vom myFramework sind in der nachstehenden Grafik abgebildet:

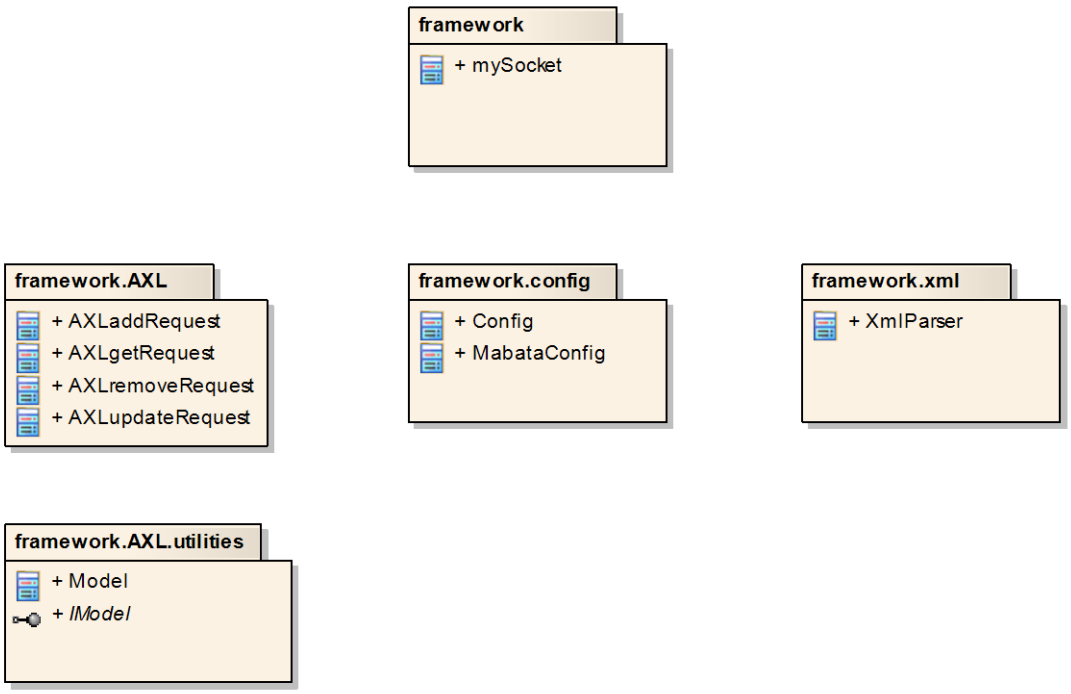


Abbildung 3-7 Package Diagramm myFramework

3.1.4. Mabata Datenbank

3.1.4.1. Bestehendes Datenbankschema

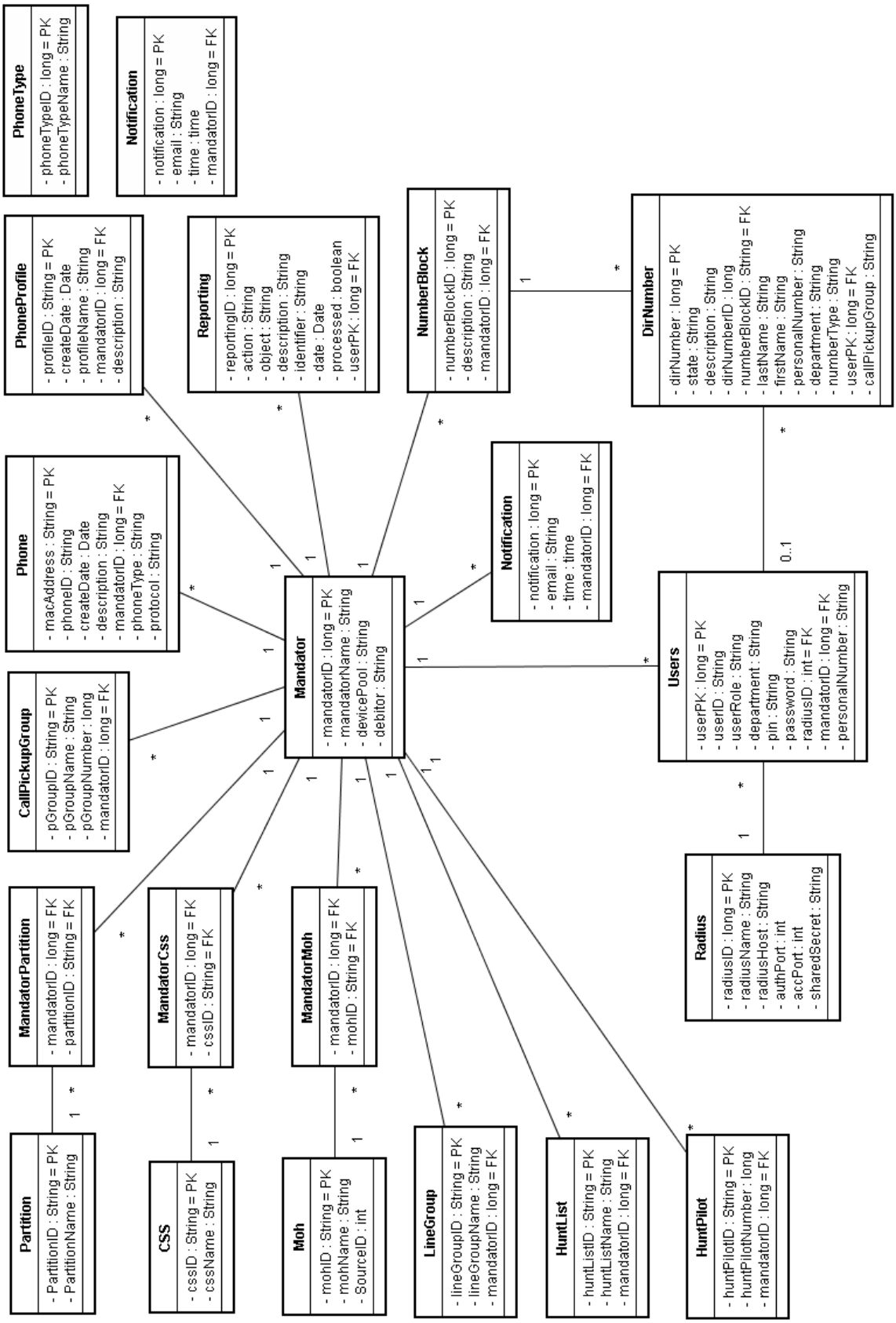


Abbildung 3-8 Mabata Datenbankschema

3.1.4.2. Wichtigste Tabellen

- Users
- Phone
- PhoneProfil
- Mandator

Diese Tabellen repräsentieren die einzelnen Objekte in Mabata. Die Beziehungen dieser Objekte sind nur über den Mandator geregelt. Untereinander bestehen keine Beziehungen.

Die Beziehungen untereinander (z.B. von einem Benutzer zu einem Telefon) werden nicht in Mabata gespeichert, sondern nur im Cisco Communications Manager. Werden diese Beziehungen in Mabata angezeigt, holt Mabata diese automatisch aus dem Communications Manager. Änderungen bei diesen Beziehungen werden direkt im Communications Manager gespeichert.

3.1.4.3. Beispielablauf einer Zuweisung

Für die Detailansicht eines Benutzers werden mit Hilfe der `getUser()` AXL-Funktion alle nötigen Informationen aus dem Communications Manager geholt:

```
<getUser>
      <userid>f1looser</userid>
</getUser>
```

Wenn dieser Benutzer bereits Telefone besitzt, werden diese in der Response zurückgegeben.

```
<associatedDevices>
      <device>SEP0015F9490734</device>
</associatedDevices>
```

Wird ein neues Telefon zugewiesen, werden mit `getPhone()` zusätzliche Informationen zum Telefon geholt.

```
<getPhone>
      <phoneName>SEP0015F9490734</phoneName>
</getPhone>
```

Nach dem Speichern wird folgende Nachricht an den Communications Manager gesendet. In dieser wird der User aktualisiert und das Phone zugewiesen:

```
<updateUser>
      <userid>f1looser</userid>
      <lastname>Looser</lastname>
      <associatedDevices>
            <device>SEP000D65707B8A</device>
      </associatedDevices>
</updateUser>
```

3.1.5. Benutzerrollen

Die folgende Grafik gibt einen Überblick über die verschiedenen Benutzerrollen sowie deren zur Verfügung stehenden Funktionen. (Diese Grafik wurde von einer vorgängigen Studienarbeit übernommen.)

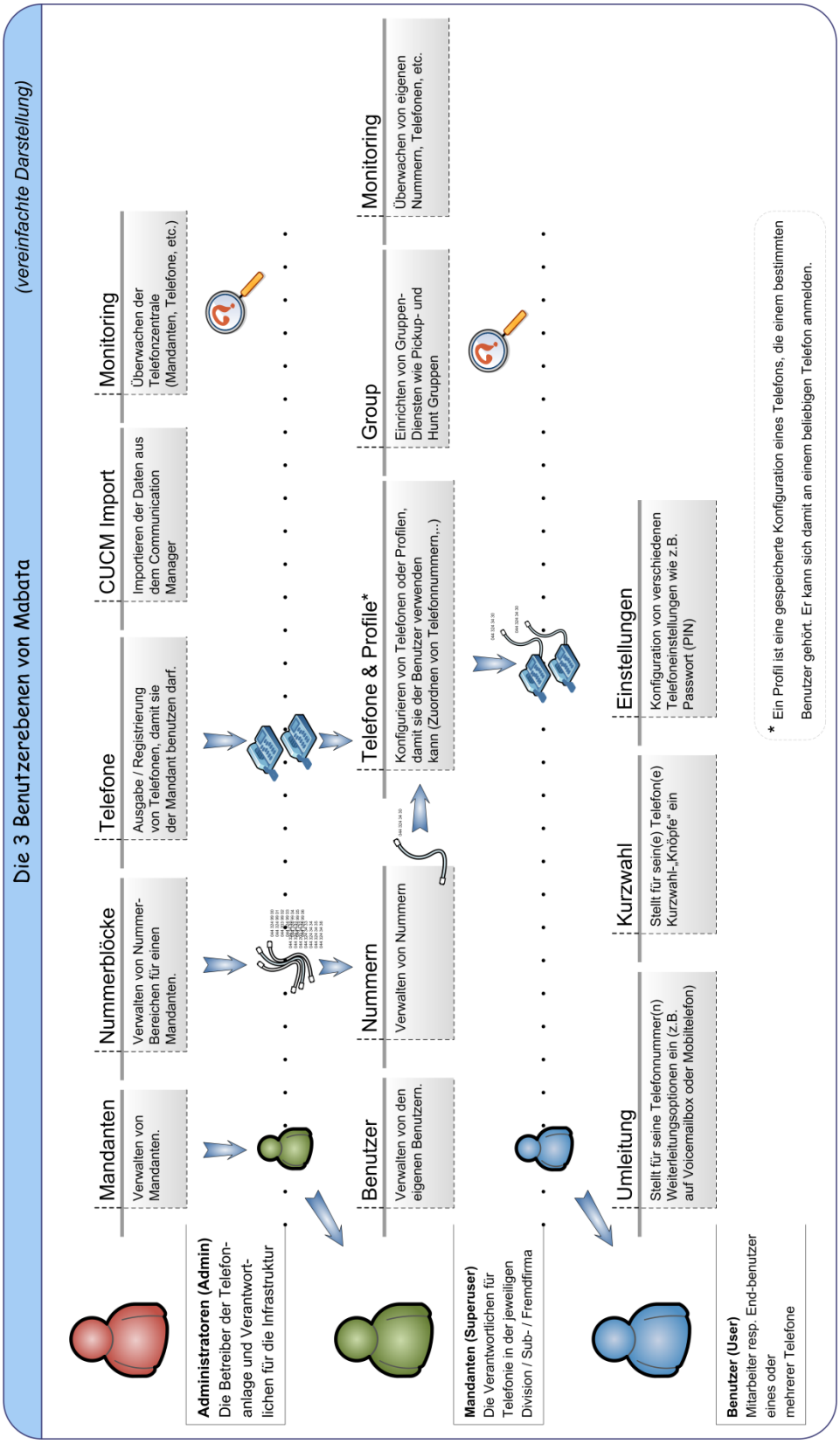


Abbildung 3-9 Mabata Benutzerrollen

3.1.6. Funktionen

Nachfolgend wird untersucht, welche Funktionen Mabata bereits unterstützt und welche neuen Funktionen in Mabata sinnvoll wären.

3.1.6.1. Bestehende Funktionen

Administrator Funktionen

Funktion	Beschreibung
Mandant	Neue Mandanten können erstellt, bearbeitet und gelöscht werden.
Benutzer	Superuser können erstellt, bearbeitet und gelöscht werden.
Inventar	Neue Telefone können manuell erfasst oder über eine Inventardatei eingelesen werden.
Nummernblock	Neue Nummernblöcke können erstellt werden. Bestehende Nummernblöcke können einem anderen Mandanten zugewiesen, aufgeteilt oder zusammengefügt werden. Beim Zusammenfügen zweier Nummernblöcke fehlt die Überprüfung, ob diese auch wirklich aufeinanderfolgend sind. Sind zwei Blöcke nicht aufeinanderfolgend, wird «Blöcke wurden zusammengefügt» ausgegeben, obwohl die Blöcke nicht zusammengefügt wurden.
Authentifizierung	Radiusserver können hinzugefügt, bearbeitet und entfernt werden.
Reporting	Die Reportingfunktion erlaubt es, die getätigten Aktivitäten in Mabata auszuwerten und falls erwünscht, können die Reports per E-Mail im PDF-Format versendet werden. Das Versenden der Reports funktioniert nicht. E-Mails werden zwar versendet, jedoch ist die PDF-Datei immer 0 Byte gross.
Statistik	Statistiken wie aktive/inaktive Telefone, Profile und aktive/inaktive Nummern werden für jeden Mandanten aufgelistet.
CUCM Import	Erlaubt das Importieren der Daten aus einem Cisco Communication Manager. Telefone, Profile, Nummern und Benutzer werden falls möglich anhand der zuvor definierten Nummernblöcke automatisch den entsprechenden Mandanten zugeordnet. Ist dies nicht möglich, können die Objekte manuell den einzelnen Mandanten zugeordnet werden. Der Import von Hunt Gruppen ist in der aktuellen Version von Mabata nicht möglich. Die Pickup Gruppen können nur importiert werden, falls diese einer Nummer zugewiesen sind.
Konsistenzprüfung	Inkonsistenzen zwischen Mabata und dem Communications Manager können mit dieser Funktion angezeigt und aufgelöst werden. Die Daten des Communications Manager gelten als Referenz. Kann die Mandanten-Zuweisung von in Mabata neu erstellten Objekten nicht automatisch vorgenommen werden, muss sie vom Administrator manuell durchgeführt werden. Hierzu erscheint die gleiche Ansicht wie beim Import-Assistenten. Die Konsistenzprüfung hat die gleichen Einschränkungen bezüglich Hunt- und Pickup Gruppen wie der CUCM Import.
Telefon Typen	Telefon Typen können erfasst sowie gelöscht werden. Leere Telefon Typen können erfasst werden.
Portierung	Benutzer können von einem Mandanten zum Anderen portiert werden. Sind diesem Benutzer Telefone sowie Nummern zugewiesen, können diese auch portiert werden.

Transcation Handling	Vor jeder Transaktion wird überprüft, ob die Mabata Datenbank funktionsfähig ist. Erst nach dieser Überprüfung wird das Update auf dem Communications Manager durchgeführt und anschliessend auf der Mabata Datenbank. Bei einer Fehltransaktion zum Communications Manager wird dem Benutzer eine Fehlermeldung angezeigt und die Transaktion zur Mabata Datenbank abgebrochen.
myFramework	Die Kommunikation zwischen Mabata und dem Communications Manager erfolgt über das myFramework, welches die AXL-Messages generiert.
Konfigurationstool	Mabata wird durch ein Konfigurationstool konfiguriert.

Tabelle 3-1 Administrator Funktionen

Superuser Funktionen

Funktion	Beschreibung
Anschluss	Hier werden alle Telefonnummern aufgelistet, welche vom Administrator zugewiesen wurden. Ebenfalls kann eine Nummer einem Benutzer zugeordnet werden und diese aktiviert werden.
Benutzer	Der Superuser kann Benutzer erstellen. Diesen werden Telefone und Profile zugewiesen, welche sie verwalten können.
Nummern	An dieser Stelle können Telefonnummern eines Mandanten bearbeitet, aktiviert und deaktiviert werden. Über die Filteroption kann die Auswahl der Telefonnummern auf inaktive oder aktive Nummern gesetzt werden.
Telefon	Vom Administrator zugewiesene Telefone können hier erfasst werden. Man kann ihnen mehrere Optionen zuweisen (unter anderem: Warte Musik, Tastaturbelegung und Telefonnummern).
Profil	Unter der Telefonprofilverwaltung können alle Profile des aktuellen Mandanten verwaltet werden. Ein Telefonprofil dient dazu, dass sich ein Benutzer auf einem beliebigen Telefon anmelden kann, welches für die Mehrfachbenutzung aktiviert ist. Dort findet er nach dem Login seine persönlichen Einstellungen wieder. Diese Einstellungen sind praktisch identisch zu denen eines Telefons. Unter diesem Punkt ist es zudem möglich, die Telefonnummern zu diesem Profil zu verwalten.
Statistik	Anzeige einer Übersicht über die Telefone, Nummern, Profile sowie Benutzer des Mandanten.
Pickup Gruppen	Pickup Gruppen können erstellt, bearbeitet und gelöscht werden. In der Detailansicht einer Pickup Gruppe können Nummern zu dieser Pickup Gruppe hinzugefügt werden.
Hunt Gruppen	Hunt Gruppen können erstellt, bearbeitet sowie gelöscht werden. Die Nummern, welche dieser Gruppe angehören können bearbeitet werden. Zudem können mehrere Gruppennummern zugewiesen werden.

Tabelle 3-2 Superuser Funktionen

User Funktionen

Funktion	Beschreibung
Weiterleitung	Der Benutzer kann für seine Telefone und Profile die Weiterleitungseinstellungen konfigurieren.
Kurzwahl	Die Kurzwahltasten können belegt werden.
PIN	Der PIN kann geändert werden.

Tabelle 3-3 User Funktionen

3.1.6.2. Neue Funktionen

Funktion	Beschreibung
Undo-Funktion	Vom Benutzer getätigte Änderungen sollen angezeigt und rückgängig gemacht werden können.
New VoIP Source Architecture	Trennung von Mabata und myFramework, damit das Framework eigenständig läuft und dadurch auch von anderen Applikationen verwendet werden kann.
Kostenabrechnung	Die verwendete Anzahl von Telefonen und Benutzern soll protokoliert und die Möglichkeit zur Rechnungserstellung geboten werden.
Nummerntyp	Erweiterung des Nummerntyps um die Pickup- und Hunt Gruppen Funktionen.
SoftKey Templates	Zuweisen von SoftKey Templates für die Benutzung der Pickup Funktion.
PhoneButton Template inkl. Busy Lamp	Die Kurzwahltasten sollen die Verfügbarkeit der entsprechenden Telefonlinie anzeigen.
Services Framework	Das bestehende Services Framework soll eigenständig von Mabata lauffähig sein.
Mabata Business Case Vereinfachung	Das Mabata-GUI soll einfacher und intuitiver werden. Momentan wird das GUI noch zu wenig vom Communications Manager abstrahiert.
Reporting/Syslog	Alle Operationen, welche in Mabata ausgeführt werden, sollen in eine Datenbank abgespeichert werden. Zusätzlich soll dem Administrator ermöglicht werden, diese Meldungen an einen externen Syslog-Server zu schicken.
Displayname für Line	Ermöglicht das Konfigurieren eines benutzerdefinierten Namen für jede vergebene Line.
Passwörter als Hash-Wert ablegen	Alle Benutzerpasswörter werden als Hash-Werte in der Datenbank abgespeichert.
Erweiterung des Nummerndatentyps	Um den neusten Anforderungen des Communications Manager 7.0 gerecht zu werden, soll neu möglich sein, der Nummer ein Präfix vorzuschalten.

Tabelle 3-4 Neue Funktionen

3.2. New VoIP Source Architecture

3.2.1. Problemstellung

Das INS hat mehrere eigene Softwareprodukte, welche verschiedene Frameworks nutzen. Wenn sich in einem Framework Änderungen ergeben, müssen alle Applikationen geändert werden, welche das Framework benutzen.

Um dies zu verhindern, soll eine neue Zwischenschicht entwickelt werden, in welcher die Änderungen eines Frameworks zentral angepasst werden können.

Zu beachten gilt, dass die Applikationen und Frameworks in verschiedenen Programmiersprachen geschrieben wurden. Microsoft Exchange zum Beispiel kann nur über .Net angesprochen werden. Die neu entwickelte Schicht muss daher die Kommunikation zu anderen Programmiersprachen zulassen.

Für jede Applikation kann konfiguriert werden, in welchen Frameworks die Änderungen eines Objektes vorgenommen werden sollen. Wenn zum Beispiel in Mabata ein neuer Benutzer erstellt wird, soll per Konfiguration vorgegeben werden, ob der Benutzer auch in anderen Frameworks erstellt werden soll.

Ein weiterer wichtiger Punkt ist, dass in den Applikationen Änderungen welche durch andere Applikationen getätigt wurden, nachgeführt werden. Wird zum Beispiel im OCS Connector ein neuer Benutzer erstellt, muss dieser Benutzer auch in Mabata erstellt werden.

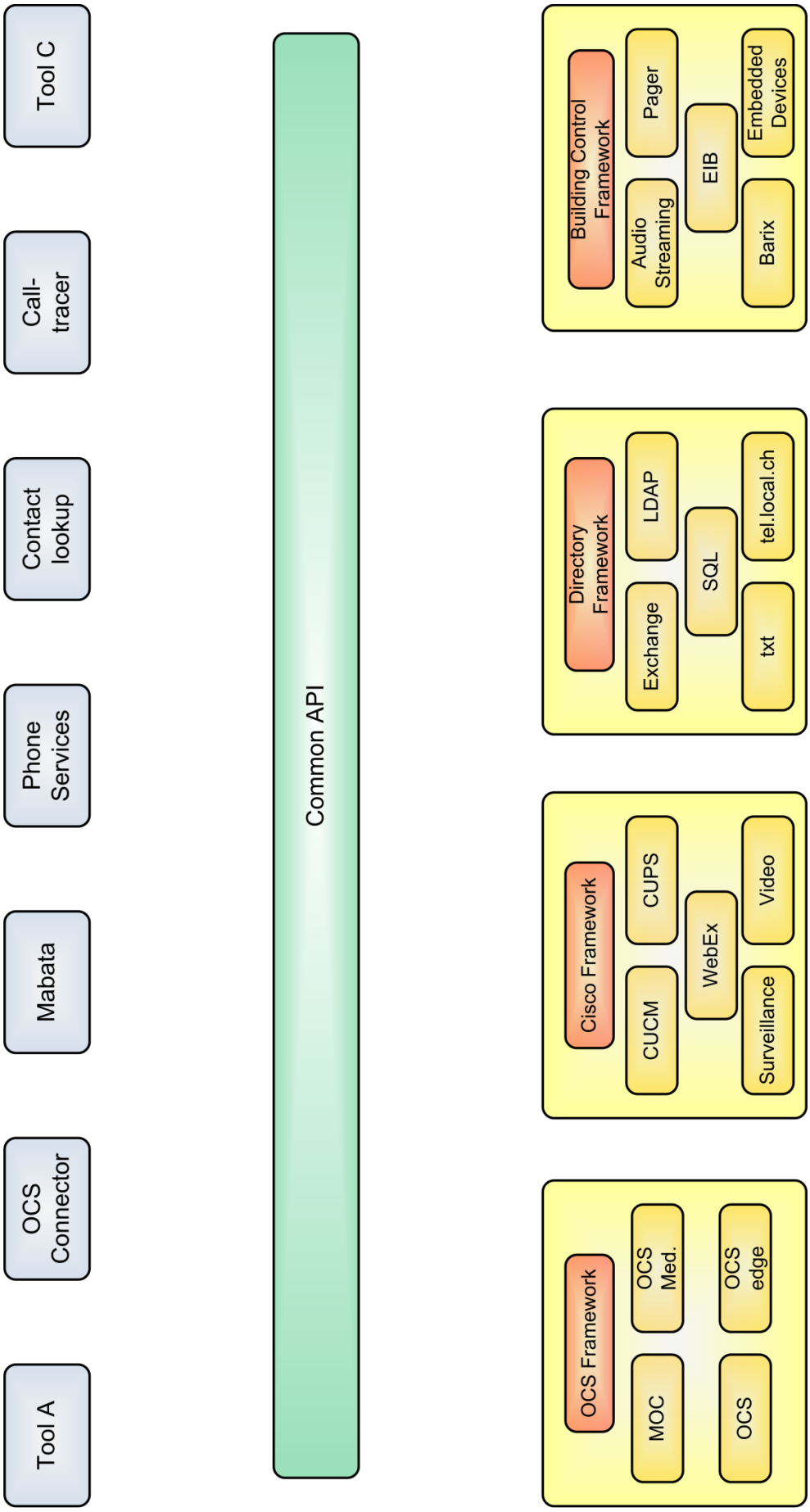


Abbildung 3-10 New VoIP Source Architecture

3.2.2. Anforderungen

Nachfolgend werden die verschiedenen Anforderungen aufgelistet und beschrieben.

3.2.2.1. Kommunikationsschicht

Die Applikationen kommunizieren momentan grösstenteils direkt mit dem Zielsystem. Diese Kommunikation muss später über die Middleware erfolgen. Um dies zu bewerkstelligen, müssen die Kommunikationsschichten der Applikationen mit einem Adapter, welcher mit der Middleware kommuniziert, ausgetauscht werden. Falls die Kommunikationsschicht nicht von der Applikation extrahiert werden kann, muss die Kommunikation vom Adapter mit anderen Mitteln abgefangen werden.

Der Adapter sollte die gleichen Schnittstellen wie das Zielsystem anbieten, damit in der Applikation keine Änderungen vorgenommen werden müssen.

3.2.2.2. Dezentrale Lösung

Es kann nicht davon ausgegangen werden, dass die Applikationen, die Middleware und die Frameworks auf dem gleichen Server ausgeführt werden. Daher muss eine dezentrale Lösung gefunden werden, welche die Kommunikation von der Applikation über die Middleware zum Framework regelt.

3.2.2.3. Plattform- und programmiersprachenunabhängig

Die verschiedenen Applikationen und Frameworks wurden mit unterschiedlichen Programmiersprachen entwickelt. Aus diesem Grund muss die entwickelte Lösung programmiersprachen- und plattformunabhängig sein.

3.2.2.4. Nebenläufigkeit

Da alle Applikationen die gleiche Middleware verwenden werden, muss die Nebenläufigkeit gewährleistet werden. Gleichzeitige Zugriffe auf die gleichen Methoden dürfen nicht erlaubt werden.

3.2.2.5. Transaction Handling

Falls eine Aktion auf mehreren Frameworks ausgeführt werden muss, soll diese Aktion entweder auf allen oder auf keinem Framework ausgeführt werden. Dabei muss eine Rückmeldung an die Applikation erfolgen. Falls in dieser keine Fehlerbehandlung existiert, kann der Benutzer nicht über das Fehlschlagen der Aktion informiert werden.

3.2.2.6. Kommunikation

Für die Kommunikation gibt es zwei Kriterien, welche für eine erfolgreiche Middleware beachtet werden sollten.

Der erste Punkt ist die Performance. Durch die zusätzliche Schicht in der Kommunikation zwischen Quell- und Zielsystem ist es unumgänglich, dass die Kommunikation verzögert wird. Das Ziel besteht darin, diese Verzögerung so klein wie möglich zu halten, um das Antwortverhalten der Quellapplikation nicht zu stark zu verlangsamen. Der grösste Teil der Verlangsamung entsteht sicher bei den zusätzlichen Netzwerkverbindungen, welche benötigt werden um die Middleware anzusprechen. Dabei muss ein möglichst performantes Kommunikationsprotokoll verwendet werden. Eine weitere Verzögerung tritt in der Business-Logik der Middleware auf. Diese Verzögerung ist vernachlässigbar, da die Business-Logik sich darauf beschränkt die Anfragen des Quellsystems an das entsprechende Zielsystem weiterzuleiten.

Der zweite Punkt ist die Sicherheit. Die Kommunikation zwischen allen beteiligten Systemen muss authentisiert erfolgen, damit ein Unberechtigter nicht mit den Zielsystemen kommunizieren kann. Dies wird vor allem wichtig, wenn die verschiedenen Systeme dezentral sind. In diesem Fall kann nicht davon ausgegangen werden, dass diese Netzwerke speziell gesichert sind. Daher muss die Applikation selber für eine Authentisierung sorgen.

3.2.2.7. Zuverlässigkeit

Auch auf die Stabilität und Zuverlässigkeit der Middleware muss geachtet werden. Ohne ein bestimmtes Mass an Zuverlässigkeit wird die Lösung nicht eingesetzt. Vor allem durch die dezentrale Kommunikation welche über das Netzwerk realisiert wird, muss sehr viel Wert auf Zuverlässigkeit gelegt werden. Ein Ausfall der Middleware betrifft alle Applikationen. Um diesen Fall vorzubeugen muss die Middleware redundant ausgelegt werden.

3.2.2.8. Skalierbarkeit

Eine möglichst einfache Erweiterung der Middleware mit zusätzlichen Applikationen muss ermöglicht werden. Dadurch kann die Middleware für alle in einem Unternehmen eingesetzten Applikationen, die gemeinsame Objekte besitzen, verwendet werden. Am einfachsten wird dies realisiert, indem der Aufbau der Middleware modular erfolgt und die einzelnen Applikationen durch eigene Adapter an die Middleware angebunden werden. Eine weitere Applikation kann somit einfach durch einen zusätzlichen Adapter angebunden werden.

3.2.2.9. Framework

Für die verschiedenen Zielsystem-Gruppen sind Frameworks vorhanden, somit besteht ein zentraler Zugangspunkt für mehrere Zielsysteme. Werden die Middleware und das Framework auf der selben Maschine betrieben, kann die Kommunikation zwischen Middleware und Framework direkt erfolgen. Ansonsten kommuniziert das Framework mit einem Adapter. Dieser wiederum stellt die Kommunikation über das Netzwerk mit der Middleware her.

Die Kommunikation zwischen Middleware und Framework-Adapter muss plattformunabhängig sein. Dadurch ist auch eine Kommunikation zwischen zwei Anwendungen, welche in unterschiedlichen Programmiersprachen entwickelt worden sind, möglich.

3.2.2.10. Konfigurierbarkeit

Für welche Transaktionen welche Frameworks miteinbezogen werden, soll für jede Applikation einzeln konfigurierbar sein.

3.2.2.11. Unterschiedliche Parameter

Da in den Applikationen teilweise die gleichen Aufgaben für unterschiedliche Zielsysteme gemacht werden müssen, sind die Parameter nicht immer identisch. Da die Middleware diese Aufgaben stark abstrahiert, dass sie unabhängig vom Zielsystem sind, müssen diese Parameter auf den gleichen Nenner gebracht werden.

3.2.3. Problemlösung

Das Gesamtproblem wurde in verschiedene Teilprobleme unterteilt, um damit die Komplexität zu verringern.

3.2.3.1. Kommunikation

Für die Kommunikation kommen verschiedene Lösungsvarianten in Frage. Untersucht wurden ein eigenes Verfahren über XML und Corba/Web Services.

XML

Ein Lösungsvorschlag ist die Kommunikation auf einer eher rudimentären Basis zu erstellen. Hierfür wird pro System ein Adapter in der jeweiligen Programmiersprache erstellt. Dieser Adapter nimmt den Methodenaufruf des Quellsystems entgegen und generiert daraus eine XML-Datei. Diese wird dann noch um die nötigen Angaben, welche Zielsysteme diese Methode aufrufen soll, ergänzt.

Der Adapter leitet diese XML-Datei dann über Sockets an die Middleware weiter. Die Middleware nimmt die XML-Datei entgegen und parst den Parameter, in welchem die betroffenen Zielsysteme angegeben sind. Weiter wird die XML-Datei an die Adapter der entsprechenden Zielsysteme weitergeleitet.

Der Adapter des Zielsystems nimmt die XML-Datei entgegen, parst diese und ruft die darin enthaltene Methode in der Programmiersprache auf, die das Zielsystem unterstützt.

Vorteile

- Programmiersprachenunabhängig, jede Programmiersprache kann mit XML-Dateien umgehen

Nachteile

- Grosser Aufwand in der Entwicklung im Vergleich zu WebServices.
- Für aussenstehende Entwickler schwer zu warten.

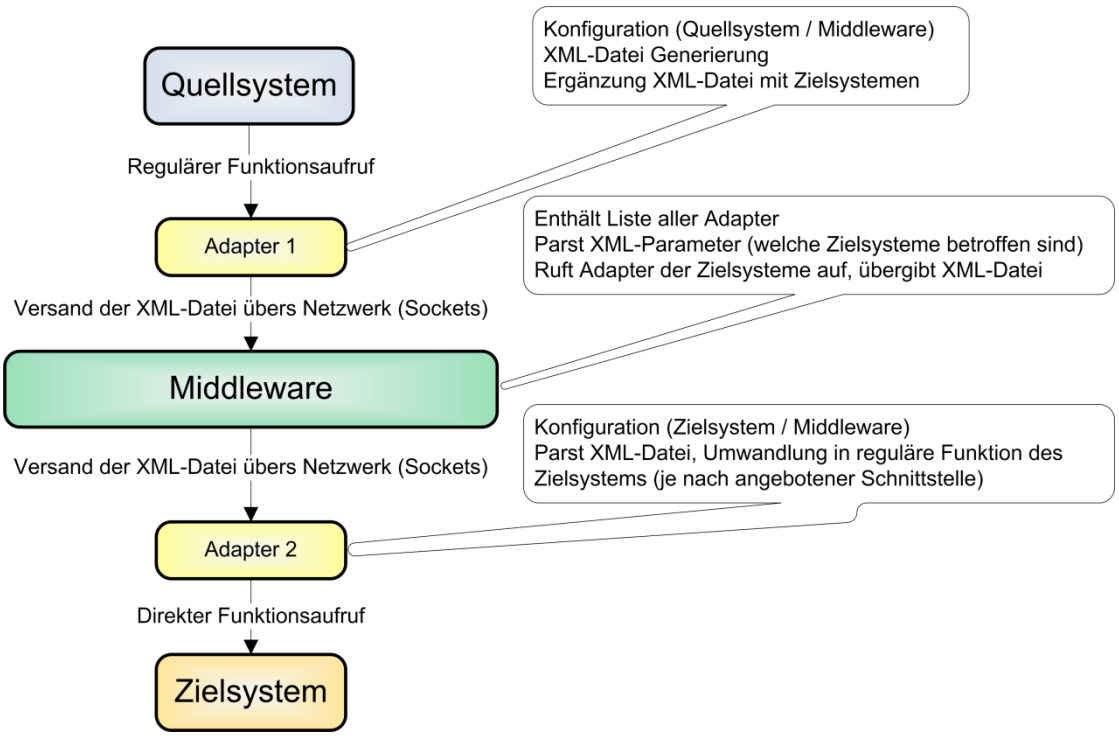


Abbildung 3-11 Kommunikation mittels XML

Corba

Die Kommunikationsschicht der Applikation wird ersetzt. Die Kommunikationsschicht wird als Adapter neu geschrieben, welcher über Corba mit der Middleware kommuniziert.

Corba ist nicht an eine bestimmte Programmiersprache gebunden und erlaubt dadurch die Kommunikation zwischen unterschiedlichen Plattformen und Programmiersprachen.

Für jedes Zielsystem wird ein eigener Adapter entwickelt, welcher die Schnittstelle des Zielsystems zur Verfügung stellt. Falls der Adapter in einer anderen Programmiersprache als die Middleware benötigt wird, erfolgt die Kommunikation zwischen diesem Adapter und der Middleware ebenfalls über Corba. Falls die gleiche Programmiersprache verwendet wird, kann direkt auf den Adapter zugegriffen werden.

Die Kommunikation zwischen dem zweiten Adapter und dem Zielsystem kann nicht standardisiert werden, da jeweils die Kommunikationsart des jeweiligen Zielsystems verwendet werden muss.

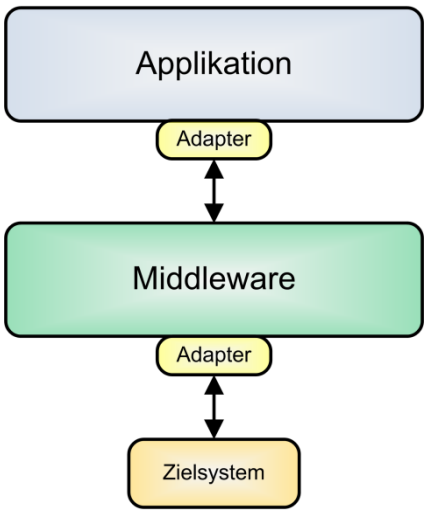


Abbildung 3-12 Kommunikation mittels Corba

WebServices

Durch die Verwendung von WebServices ist eine Kommunikation zwischen unterschiedlichen Plattformen und Programmiersprachen möglich. WebServices sind nicht an bestimmte Protokolle gebunden und können auf praktisch jedes Übertragungsprotokoll aufgesetzt werden. Normalerweise wird HTTP als Übertragungsprotokoll verwendet, es kann aber auch SMTP oder FTP verwendet werden.

In der Applikation wird die Kommunikationsschicht ersetzt. Die Kommunikationsschicht wird als Adapter neu geschrieben, welcher die Kommunikation zum WebService der Middleware herstellt.

Für jedes Zielsystem wird ein eigener Adapter entwickelt, welcher die Schnittstelle des Zielsystems zur Verfügung stellt. Falls der Adapter in einer anderen Programmiersprache als die Middleware benötigt wird, erfolgt die Kommunikation zwischen diesem Adapter und der Middleware ebenfalls über WebServices. Falls die gleiche Programmiersprache verwendet wird, kann direkt auf den Adapter zugegriffen werden.

Die Kommunikation zwischen dem zweiten Adapter und dem Zielsystem kann nicht standardisiert werden, da jeweils die Kommunikationsart des jeweiligen Zielsystems verwendet werden muss.

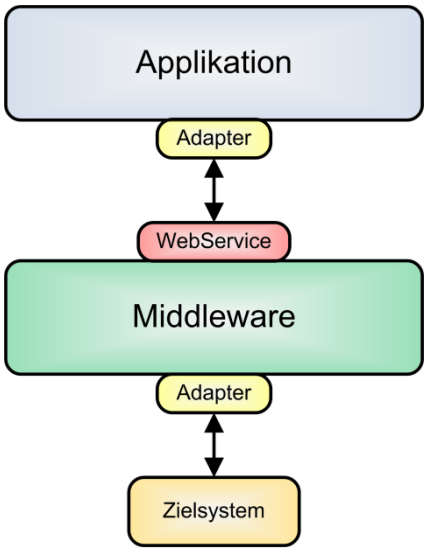


Abbildung 3-13 Kommunikation mittels WebServices

WebServices vs. Corba

Um die Kommunikation zwischen einem Adapter und der Middleware zu ermöglichen, wurden WebServices und Corba miteinander verglichen.

Corba bietet einen schnelleren Datentransfer gegenüber WebServices. Denn WebServices müssen die Daten in XML serialisieren und deserialisieren. Im Gegenteil zu WebServices werden die Daten mit Corba nicht im Klartext, sondern in binärer Form übertragen.

WebServices sind vollkommen plattform- und programmiersprachenunabhängig. Der Nachteil der Klartextübertragen kann durch die Verwendung von HTTPS ausgeglichen werden. Da die Kommunikation über HTTP/HTTPS hergestellt wird, müssen die Firewalls nicht umkonfiguriert werden. Falls gewünscht, kann der WebService auf einem selbst definierten Port veröffentlicht werden.

In unserem Projekt sind die WebServices besser geeignet, da damit jegliche Applikationen unterschiedlicher Art angebunden werden können. Weiter sind die WebServices weniger komplex zu erstellen, was auch die Weiterentwicklung für andere Entwickler stark vereinfacht. Die Performance-Einbussen, welche durch die WebServices entstehen, haben sich als vernachlässigbar gezeigt.

3.2.3.2. Konfiguration

Jeder Adapter besitzt eine Konfiguration. In dieser Konfiguration wird festgelegt, welche Operationen auf welchen Frameworks durchgeführt werden sollen. Somit kann z.B. im Adapter von Mabata festgelegt werden, dass neu erstellte Benutzer auch auf dem OCS Framework angelegt werden sollen.

Zusätzlich werden in dieser Konfiguration die Verbindungsinformationen (IP-Adresse, Port, Authentifizierung) zur Middleware gespeichert.

3.2.3.3. Middleware

Die Middleware selbst hat keine eigenen Funktionen. Sie dient nur dazu, die Methodenaufrufe weiterzuleiten. Dabei kommen die Aufrufe über die WebServices rein. Diese Aufrufe enthalten zusätzlich die Information, auf welchen Zielsystemen die Aufgabe durchgeführt werden soll. Die Aufgabe der Middleware besteht nun darin, diese Aufrufe an die richtigen Adapter der jeweiligen Zielsysteme weiterzuleiten.

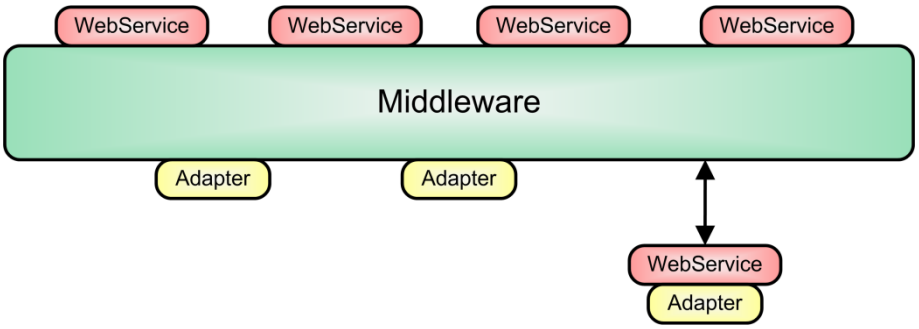


Abbildung 3-14 Middleware

3.2.3.4. Microsoft BizTalk Server

Als Alternative zur Entwicklung einer eigenen Middleware wurde eine bereits existierende Lösung von Microsoft analysiert.

Microsoft hat auch erkannt, dass es in Zukunft immer mehr darauf ankommt verschiedenste Systeme untereinander zu vernetzen. Ihr dafür erstelltes Produkt heisst BizTalk Server.

BizTalk Server 2006 R2 integriert heterogene Systeme (Enterprise Application Integration, EAI). Über Adapter oder standardisierte Schnittstellen wie WebServices werden diese an den BizTalk Server gekoppelt. BizTalk Server kann als zentraler Hub für praktisch beliebig viele Systeme eingesetzt werden (im folgenden Bild rechts). Damit entfällt der Aufwand für die sonst erforderliche Verwaltung und Pflege einer Vielzahl von Punkt-zu-Punkt-Verbindungen, bei der die Anzahl der Verbindungen exponentiell mit der Anzahl der Systeme steigt ("Spaghetti-Integration", im folgenden Bild links).

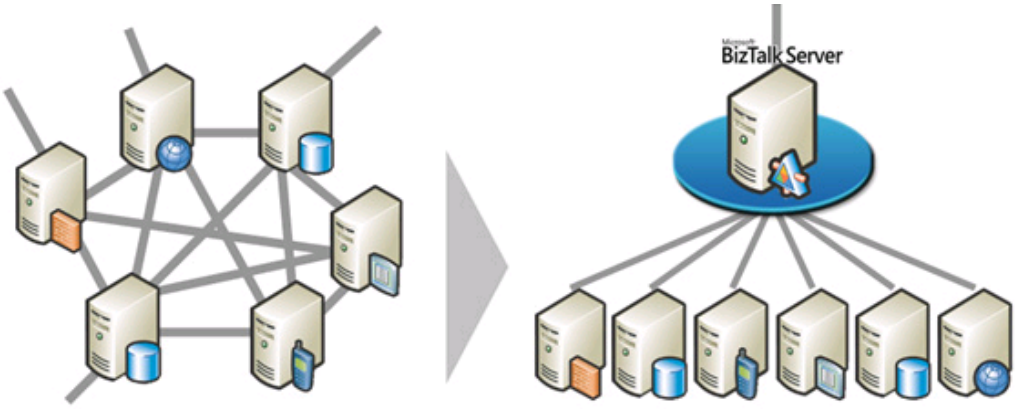


Abbildung 3-15 Übersicht BizTalk Server

Mit BizTalk Server 2006 R2 ist es beispielsweise möglich, Daten zwischen mehreren Anwendungen zu synchronisieren. Ein gängiges Szenario ist das Abgleichen von Stammdaten (Master Data Management) zwischen einem CRM- und einem ERP-System. Häufig wird BizTalk Server auch in sogenannten Hub & Spoke-Szenarien eingesetzt, in denen beispielsweise Kundendaten, die in mehreren Niederlassungen erfasst werden, in Echtzeit mit einem zentralen System abgestimmt werden. Die Anbindung der beteiligten Systeme erfolgt über Standards wie WebServices oder über BizTalk Adapter.
(Quelle «http://www.microsoft.com/germany/biztalk/einsatz/eai.mspx»)

Für die Anbindung der verschiedenen Systeme hat Microsoft wie oben erwähnt eine Vielzahl an unterschiedlichen Adaptern entwickelt. Zudem haben Dritt-Firmen für ihre Produkte entsprechende Adapter erstellt. Eine Vollständige Tabelle aller Adapter ist unter «http://www.microsoft.com/germany/biztalk/interop/adapter.mspx ersichtlich».
Der Vorteil einer standardisierten Lösung besteht darin, dass keine eigene Middleware entwickelt werden muss, sondern die bestehende genutzt werden kann. Somit kann die gesparte Zeit in die Entwicklung der Adapter investiert werden.

Nachteilig an dieser Lösungsvariante ist sicherlich, dass die Vorteile einer Open-Source Lösung durch den Einsatz von kommerziellen Produkten verloren gehen. Die Kosten für eine 1 Prozessor Lizenz vom BizTalk Server 2006 R2 Standard Edition belaufen sich auf ca. CHF 12000.-. Auch müsste Know-How aufgebaut werden, damit die Konfiguration und Erweiterung des BizTalk Servers angepackt werden könnte.

3.2.3.5. Parametermapping

Jedes der Quell- und Zielsysteme benötigt unterschiedliche Parameter, um beispielsweise einen Benutzer zu erstellen. Die übergebenen Parameter des Quellsystems müssen auf die Methoden der Zielsysteme gemappt, sowie die noch zusätzlich benötigten Parameter definiert werden.

Dadurch können Probleme entstehen, wenn Parameter im Zielsystem gesetzt werden müssen, die vom Quellsystem gar nicht geliefert werden. Am besten lässt sich dieses Problem anhand eines Beispiels erläutern:

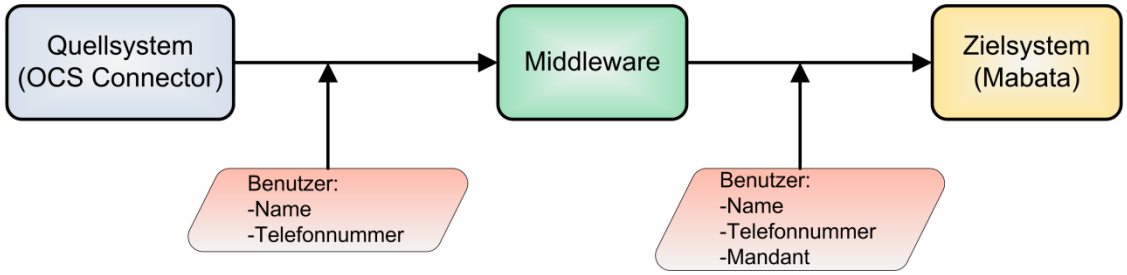


Abbildung 3-16 Parametermapping

Das Zielsystem, in diesem Beispiel Mabata, benötigt einen Mandator um einen Benutzer anlegen zu können. Das Quellsystem, OCS Connector, verfügt aber nicht über diese Informationen.

Voraussetzung für das erfolgreiche Erstellen eines Benutzers im Zielsystem ist, dass alle benötigten Parameter bekannt sind. Um dies zu ermöglichen, werden Parameter die vom Quellsystem nicht in Erfahrung gebracht werden können mit Standardwerten vorbelegt.

3.2.3.6. Adapter

Die Adapter, welche die Applikationen mit der Middleware verbinden, können nicht standardisiert werden. Sie hängen stark von der Implementierung der Applikation ab. Als Beispiel kann Mabata genannt werden. Hier müssen keinerlei Änderungen im Code von Mabata getätigt werden, da die Kommunikationsschicht im myFramework ist. Darum kann das myFramework mit dem Adapter ausgetauscht werden. Jedoch wird diese Kommunikationsschicht nicht in jeder Applikation einfach auszutauschen sein. Aus diesem Grund muss für jede Applikation ein individueller Adapter entwickelt werden.

Aus Performancegründen sollte das Framework direkt mit der Middleware kommunizieren. Dadurch erübrigt sich zudem die Entwicklung eines zweiten Adapters. Als Beispiel soll die Abbildung 3-18 im nächsten Kapitel dienen.

3.2.4. Auswirkungen

Durch diese neu konzipierte Middleware werden die Applikationen unabhängiger von ihren Frameworks. Falls sich eine Schnittstelle eines Servers ändert, muss nur die Middleware angepasst werden. Durch diese Anpassung in der Middleware können wieder alle Applikationen mit dem Server kommunizieren, ohne dass eine Änderung nötig wird.

Im Gegensatz zu Änderungen an den Serverschnittstellen ergibt eine Änderung in der Applikation einen Mehraufwand gegenüber der Architektur ohne Middleware. Um die Änderungen in der Applikation zu verwenden, muss diese Änderung auch in der Middleware implementiert und beachtet werden.

Als grossen Nachteil dieser Middleware ist die Performance zu erwähnen. Als Beispiel soll Mabata dienen. Vor der Integrierung der Middleware sah die Kommunikation wie folgt aus:

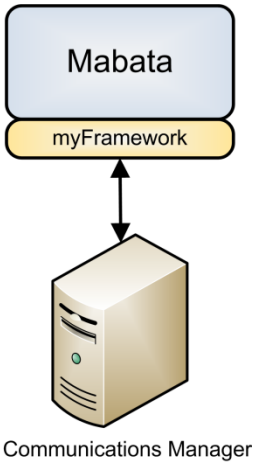


Abbildung 3-17 Alte Kommunikation

Die Kommunikation zwischen Mabata und dem myFramework erfolgt direkt. Das heisst, Mabata hat das myFramework als JAR-Datei eingebunden. Die Kommunikationszeit kann daher vernachlässigt werden. Die einzige Netzwerkverbindung besteht zwischen dem myFramework und dem Communications Manager.

Vergleicht man diese Architektur mit der Neuen, in welcher die Middleware integriert ist, wird deutlich, dass diese neue Architektur klare Performance-Nachteile aufweist.

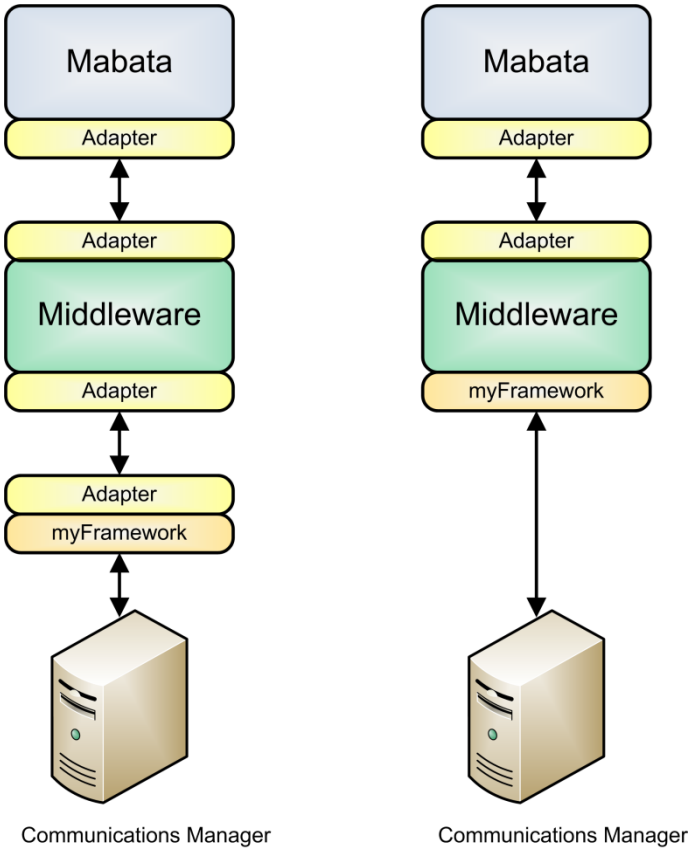


Abbildung 3-18 Neue Kommunikation

Abhängig davon, ob das Framework dezentral betrieben wird oder nicht, bestehen in der neuen Architektur zwei – drei Netzwerkverbindungen. Diese beeinflussen die Geschwindigkeit der verwendeten Applikation mit Sicherheit negativ.

3.2.5. Vergleich der verschiedenen Lösungen

Produkte	Kosten	Performance	Kommunikation	Sicherheit	Know-How	Zukunfts-sicherheit	Wartbarkeit
Web-Services	Keine	Schneller Netzwerkverkehr	Bestehende Kommunikationsschnitt-stelle kann verwendet werden	Versand der Nachrichten über HTTPS	Grundlagen vorhanden	Gross	Einfach
Corba	Keine	Schnelle Übertragung der Dateien da im Binärformat übertragen wird	Funktionsaufrufe wie lokal, durch generierte Stubs und Skeletons	Corba Security Services -> nur Schnittstellen und Architek-turelemente	Keine Vorkenntnisse vorhanden	Wird durch WebServices abgelöst	Mittel
XML über TCP	Keine	Schneller Netzwerkverkehr	Über selber programmierte Sockets (TCP)	Muss selbst implementiert werden.	Grundlagen vorhanden	Hängt von der eigenen Weiterentwicklung ab	Komplex
BizTalk Server	CHF 12000.- für eine Prozessorlizenz	Hängt vom eingesetzten Adapter ab	Verschiedene Adapter stehen zur Verfügung, welche für die Kommunikation verwendet werden können	Je nach Adapter	Keine Vorkenntnisse vorhanden	Gross	Sehr komplex

Tabelle 3-5 Vergleich der Lösungen

3.2.6. Fazit

Während der Analyse hat sich gezeigt, dass WebServices für die Kommunikation zwischen den verschiedenen Applikationen am besten geeignet sind. Durch die Sprachunabhängigkeit und die bereits bestehende Kommunikationsschnittstelle können WebServices effizient eingesetzt werden. Zudem wird nicht wie bei der Lösung «XML über TCP» viel Zeit für grundlegende Kommunikationsprobleme verbraucht. Auch kann bestehendes Wissen angewendet werden um gleich mit der Implementation starten zu können. Würde der BizTalk-Server oder CORBA eingesetzt, müsste zuerst viel Zeit in das Studium der Technologien eingesetzt werden.

3.3. Undo-Funktion

3.3.1. Problemstellung

Alle in Mabata durchgeführten Aktionen sollen protokolliert werden. Zusätzlich können einzelne oder mehrere Änderungen rückgängig gemacht und somit ein früherer Systemzustand wiederhergestellt werden.

3.3.2. Anforderungen

Die verschiedenen Anforderungen an die Undo-Funktion werden nachfolgend aufgelistet und beschrieben.

3.3.2.1. Protokollierung

Alle Transaktionen welche in Mabata getätigt werden, werden in einer Datenbank protokolliert. Aus dem Eintrag müssen folgende Informationen gelesen werden können:

- Benutzer
- Datum
- Transaktion
    - Art (create, update, delete)
    - Datenfelder des geänderten Objektes

3.3.2.2. Anzeige

Damit die einzelnen Transaktionen rückgängig gemacht werden können, wird eine Auflistung aller Transaktionen benötigt.

Dabei muss beachtet werden, dass alle Transaktionen eindeutig zugeordnet werden können. Auch wird eine gewisse Abstraktionsstufe der einzelnen Transaktionen benötigt. Dafür müssen zum Beispiel für das Hinzufügen eines Benutzers mehrere AXL-Abfragen gestartet werden. Diese einzelnen Abfragen sollen aber zusammengefasst als eine Transaktion «Benutzer hinzufügen» angezeigt werden.

3.3.2.3. Undo

Durchgeführte Änderungen sollen rückgängig gemacht werden können und das System dadurch auf einen früheren Stand gebracht werden. Dies ist vor allem nützlich, wenn mehrere Einstellungen getätigt wurden, diese schlussendlich jedoch nicht übernommen werden sollen.

3.3.3. Problemlösung

Nachfolgend werden die einzelnen Problembereiche behandelt.

3.3.3.1. Protokollierung

Das Problem der Protokollierung besteht darin, dass sowohl die Daten der Mabata Datenbank wie auch diese des Communications Managers aufgezeichnet werden müssen. In Mabata sind die Beziehungen zu einem Mandanten und im Communications Manager die Beziehungen der Objekte untereinander abgebildet.

Falls ein Objekt gelöscht wird, müssen zuerst alle Abhängigkeiten zu anderen Objekten abgespeichert werden. Diese Abhängigkeiten sind jedoch nur auf dem Communications Manager gespeichert und müssen demzufolge einzeln vom Communications Manager abgefragt werden. Gewisse Abhängigkeiten sind nur einseitig traversierbar. Das heisst, wird z.B. ein Telefon gelöscht, müssen

zuerst alle Benutzer einzeln vom Communications Manager abgefragt werden, um zu überprüfen, ob eine Abhängigkeit zwischen dem Benutzer und dem Telefon besteht. Die Abfrage eines Objektes vom Communications Manager dauert zwischen 100 – 120 ms. Daraus ergibt sich bei 20 Benutzern eine zusätzliche Verzögerung von über 2 Sekunden um ein Telefon zu löschen. Darin nicht eingerechnet sind die Überprüfung der Objekte sowie weitere Abhängigkeiten, welche aufgelöst werden müssen. Um diese Abhängigkeiten abzuspeichern, müsste die gesamte Communications Manager Datenbank sowie die Mabata Datenbank nochmals nachgebildet werden.

3.3.3.2. Anzeige

Die protokollierten Transaktionen werden der Reihe nach aufgelistet. Da immer nur die letzte getätigte Transaktion rückgängig gemacht werden kann, wird diese unten angezeigt. Es werden nur diejenigen Transaktionen angezeigt, die effektiv rückgängig gemacht werden können. Wenn ein Superuser Transaktionen rückgängig machen will, kann er dies nur bis zur letzten Transaktion die er getätigt hat. Wen der Administrator dazwischen eine Transaktion gemacht hat, die den Superuser oder dessen Mandant betreffen, kann ein Superuser nur bis dorthin die Transaktionen rückgängig machen.

3.3.3.3. Undo

Die Transaktionen können nur in der umgekehrten Reihenfolge rückgängig gemacht werden. Ansonsten kann das System inkonsistent werden.

Transaktionen können nur so weit rückgängig gemacht werden, wie sie innerhalb der gleichen Hierarchiestufe abgelaufen sind. Löscht z.B. der Superuser ein Telefon und gibt dem Administrator den Auftrag, dieses Telefon auch aus dem Inventar zu löschen, kann diese Aktion nicht mehr rückgängig gemacht werden. Denn der Superuser darf keine Aktionen vom Administrator und der Administrator keine Aktionen vom Superuser rückgängig machen.

3.3.4. Machbarkeit

Anhand der beschriebenen Problemlösung wurde die Machbarkeit der Funktionen Create, Update und Delete untersucht.

3.3.4.1. Create

Ein neu erstelltes Objekt wieder zu löschen, ist die einfachste Undo-Funktion. Allerdings ist es sehr fraglich, ob diese Aktion per Undo-Funktion rückgängig gemacht werden soll. Denn der Benutzer kann die getätigte Aktion durch löschen des soeben erstellten Objektes selbst rückgängig machen.

3.3.4.2. Update

Eine Änderung auf ein Objekt rückgängig zu machen, macht sehr viel mehr Sinn wie das Rückgängigmachen eines neu erstellten Objektes. Dazu müsste jeweils der alte Zustand eines Objektes abgespeichert werden. Hierzu müsste für alle Objekte, welche in Mabata bearbeitet werden können, eine Historytabelle angelegt werden.

3.3.4.3. Delete

Um ein Löschen eines Objektes rückgängig zu machen, müssten wie oben beschrieben alle Abhängigkeiten abgespeichert werden. Dies ist zwar möglich, jedoch nur mit einem sehr grossen Aufwand. Ausserdem wird die Applikation durch die Protokollierung dieser Funktionen extrem langsam.

3.3.5. Entscheid

Aufgrund der durchgeführten Machbarkeitsanalyse wird festgehalten, dass eine Undo-Funktion nicht realisiert wird. Um die Create-Funktion rückgängig zu machen besteht schon die Möglichkeit, das Objekt wieder zu löschen. Der Aufwand für die Delete-Funktion würde bei weitem grösser werden als der schlussendliche Nutzen. Einzig für die Update Funktion würde es beschränkt Sinn ergeben eine Undo-Funktion einzubauen. Dies wird dadurch abgefangen, in dem das Reporting soweit erweitert wird, dass bei einem Update immer der alte und neue Zustand eines Objektes aufgezeichnet wird.

3.4. Kostenabrechnung

3.4.1. Problemstellung

Die Unique möchte die Möglichkeit haben, den verschiedenen Mandaten die Geräte in Rechnung zu stellen. Dabei sollen Anzahl Telefone, Benutzer und Nummern in die Rechnung einfliessen.

3.4.2. Anforderungen

Softphones sollen von den normalen Telefonen unterschieden werden, da diese keine Hardwarekosten verursachen.
Ein wichtiger Bestandteil dieser Funktion ist der Erfassungszeitpunkt der Objekte. Der Mandant soll nicht die Möglichkeit haben, durch entfernen von Objekte die Kosten zu senken.

3.4.3. Problemlösung

Mittels des Reportings werden Datensätze abgespeichert, mit welchen festgestellt werden kann, über welche Zeitdauer ein Mandant wie viele Objekte (Benutzer, Nummern, Telefone) besass. Aus diesen Angaben kann berechnet werden, welche Kosten der Mandant verursacht hat.

3.5. Reporting

3.5.1. Problemstellung

Damit die getätigten Änderungen angezeigt werden können, sollen alle von den Benutzern durchgeführten Änderungen protokolliert werden. Diese protokollierten Daten sollen zudem den Benutzer unterstützen, Konfigurationsänderungen rückgängig zu machen, in dem für geänderte oder gelöschte Objekte die aktuellen und vorherigen Attribute angezeigt werden.

3.5.2. Anforderung

Alle Änderungen durch die Benutzer werden protokolliert. Bei Änderungen an Objekten werden die vorher-/nachher Attribute aufgezeichnet. Zusätzlich besteht die Möglichkeit, die protokollierten Daten an einen Syslog-Server zu übertragen.

3.5.3. Problemlösung

Die protokollierten Daten werden in der Datenbank gespeichert und falls konfiguriert, an einen Syslog-Server übermittelt. Bei der Anzeige des Reportes sind bei Objektänderungen die vorherigen Attribute angegeben.

3.6. Nummerntyp

3.6.1. Problemstellung

In Mabata ist eine Telefonnummer vom Typ persönlich oder unpersönlich. In der vorgängigen Studienarbeit wurden die Pickup- und Hunt Gruppen eingeführt, welche ebenfalls Telefonnummern verwenden. Damit keine Konflikte entstehen, können für die Gruppen-Funktionen nur inaktive Nummern verwendet werden. Zugleich musste das Aktivieren einer inaktiven Nummer, welche für eine Gruppen-Funktion benutzt wird, unterbunden werden. Für den Benutzer ist demnach nicht ersichtlich, ob eine inaktive Nummer bereits für eine Gruppen-Funktion verwendet wird oder nicht.

3.6.2. Anforderung

Bei der Übersicht der Telefonnummer muss erkennbar sein, ob eine Telefonnummer für eine Hunt- oder Pickup Gruppe verwendet wird.

3.6.3. Problemlösung

Der Typ der Telefonnummern wird um «PICKUP» und «HUNT» erweitert. Dadurch ist in der Übersichtstabelle auf den ersten Blick ersichtlich, für was eine Telefonnummer verwendet wird.

3.7. SoftKey Template

3.7.1. Problemstellung

In der vorgängigen Studienarbeit wurde die Pickup Gruppen Funktion hinzugefügt. Diese Funktion erlaubt es den Benutzern, eingehende Anrufe, welche nicht auf die eigene Telefonnummer kommen, zu übernehmen. Ist z.B. ein Mitarbeiter abwesend, können seine Arbeitskollegen im selben Büro die Anrufe über eine spezielle Taste auf ihren eigenen Telefonen entgegennehmen. Diese Tasten werden mittels den SoftKey Templates definiert.

3.7.2. Anforderung

Die neu für Pickup Gruppen benötigten SoftKey Tasten «Pickup» (Entgegennahme eines Anrufes innerhalb derselben Gruppe) und «GPickup» (Entgegennahme eines Anrufes ausserhalb der eigenen Gruppe) können den Telefonen zugewiesen werden.

3.7.3. Problemlösung

Diese Funktion kann nicht über AXL realisiert werden. In der [AXL-API] bestehen zwar die `addSoftKeyTemplate` und `getSoftKeyTemplate` Methoden, diese können aber nicht verwendet werden um das Layout der SoftKeys zu konfigurieren. Diese Templates müssen demnach vorgängig auf dem Communications Manager erstellt werden. Der Superuser kann dann diese Templates den entsprechenden Telefonen zuweisen.

3.8. PhoneButton Template inkl. Busy Lamp

3.8.1. Problemstellung

In der Vorgängigen Studienarbeit wurde die Funktion der Kurzwahltasten eingefügt. Dies ermöglicht es, auf dem Telefon Tasten zu definieren mit denen direkt Jemand angerufen werden kann. Zusätzlich zur Kurzwahl soll nun auch der Status (frei/besetzt) der jeweiligen Telefonlinie angezeigt werden. Dies kann mittels der Konfiguration der Busy Lamp Fields erreicht werden. Bei entsprechender Konfiguration leuchtet die Taste neben dem Teilnehmer rot, wenn dieser gerade am telefonieren ist.

3.8.2. Anforderung

Der Benutzer kann seine Kurzwahltasten auch mit der Busy Lamp Funktion nutzen, sofern der Superuser ein entsprechendes Template konfiguriert hat.

3.8.3. Problemlösung

Diese Funktion kann nicht vollständig via AXL realisiert werden. In der [AXL-API] fehlt die Möglichkeit neue PhoneButton Templates zu erstellen. Diese müssen vorgängig auf dem Communications Manager definiert werden. Der Superuser kann diese dann seinen Telefonen zuweisen.

Für den Benutzer gibt es keine Änderungen, denn er kann die vorhandene Kurzwahl Administrationsoberfläche verwenden. Hat der Superuser ein Busy Lamp fähiges Template konfiguriert, steht dem Benutzer diese neue Funktion zur Verfügung, ansonsten nicht.

3.8.4. Einschränkungen

Die Funktion der Busy Lamps kann nur generell aktiviert oder deaktiviert werden. Könnte der Benutzer jeden einzelnen Kurzwahl-Eintrag separat ein- oder ausschalten, würde dies für jede mögliche Kombination ein eigenes Template auf dem Communications Manager benötigen.

3.9. Services Framework

3.9.1. Problemstellung

In einer vorgängigen Diplomarbeit wurde ein Services Framework erarbeitet, welches ermöglicht auf VoIP-Telefonen zusätzliche Services anzubieten, diese mandantenfähig zu machen und über Mabata zu konfigurieren. Der Benutzer kann durch das Aufrufen des Services-Knopfes alle von seinem Mandanten angebotenen Services auflisten und den gewünschten Service auswählen.

Die angebotenen Services werden bereitgestellt in dem die Service-URL und die zu übergebenden Parameter dem Services Framework mitgeteilt werden. Was der Service schlussendlich ist und welche Funktionen er hat ist nicht Bestandteil des Services Framework.

In dieser Bachelorarbeit geht es darum, das Services Framework unabhängig von Mabata bereitstellen zu können.

3.9.2. Bestehende Probleme

Einige Anforderungen, welche für das Services Framework bereits hätten erfüllt sein müssen, sind noch nicht implementiert oder unbefriedigend gelöst:

- Die Performance vor dem Anmelden ist sehr langsam.
- Die Berechtigungsstufe, in welcher der Superuser festlegen kann, welcher Benutzer auf den Service zugreifen kann, fehlt komplett. Momentan haben alle Benutzer innerhalb eines Mandanten Zugriff auf alle Services.
- Ein globaler Service, der für alle Mandanten verfügbar ist, kann nicht konfiguriert werden. Diese Möglichkeit fehlt sowohl im GUI als auch im Datenbankschema.

Abbildung 3-19 zeigt das Datenbankschema des Services Frameworks. Es fehlt die Möglichkeit einen Service global zu konfigurieren und Berechtigungen für einzelne Benutzer festzulegen.

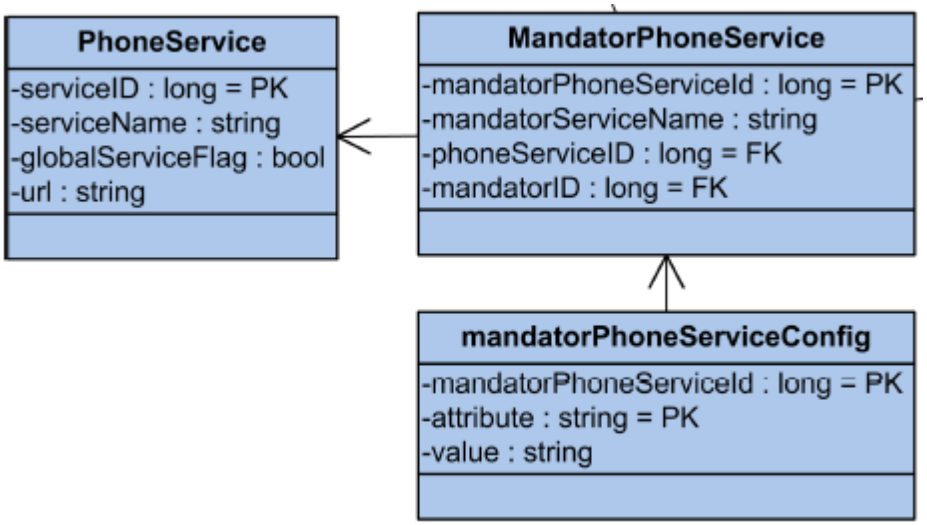


Abbildung 3-19 Datenbankschema Services Framework

3.9.3. Anforderung

3.9.3.1. Eigenständigkeit

Das Services Framework muss eigenständig lauffähig sein. Es dürfen keine Abhängigkeiten vom Communications Manager zu Mabata bestehen, denn Mabata soll weiterhin nur für die Konfiguration verwendet werden und nicht um irgendwelche Dienste bereit zu stellen.

3.9.3.2. Konfiguration

Wie bis anhin soll das Services Framework über eine Weboberfläche konfiguriert werden können.

3.9.3.3. Funktionen

Alle bestehenden Funktionen sowie die unter Kapitel 3.9.2 erwähnten nicht funktionierenden Funktionen sollen zur Verfügung stehen.

3.9.4. Problemlösung

3.9.4.1. Eigenständigkeit

Das bestehende Services Framework ist vollkommen in eine alte Mabata Version integriert. Durch ein Loslösen von Mabata wird die Eigenständigkeit erreicht.

3.9.4.2. Konfiguration

Durch das Loslösen des Services Framework von Mabata müssen in die neue Mabata Version die Konfigurations-Elemente der alten Mabata Version übernommen werden.

3.9.4.3. Funktionen

Die bestehenden Funktionen werden auf Schwachstellen analysiert und wenn möglich verbessert sowie die neuen Funktionen integriert.

3.9.5. Fazit

Das Services Framework kann in der jetzigen Form nicht weiter benutzt werden. Durch diese Generalüberholung entsteht wieder ein nutzbares Produkt, welches Mabata und die VoIP-Telefone sinnvoll um weitere Funktionen erweitern kann.

3.10. Mabata Business Case Vereinfachung

3.10.1. Problemstellung

Die bestehende Business Logik für die Konfiguration des Communications Manager wurde für Mabata nicht abstrahiert sondern von diesem übernommen. Aus diesem Grund ist es für eine Person, welche mit dem Communications Manager nicht vertraut ist schwierig, sich in Mabata zu Recht zu finden.

3.10.2. Problemlösung

Um dieses Problem lösen zu können muss eine Abstraktionsschicht im GUI eingebaut werden, welche die Konfigurationsabläufe des Communications Managers abstrahiert. Um dies zu ermöglichen werden die einzelnen Business Case für die Konfiguration zusammengefasst in einer einzigen Oberfläche dargestellt. Somit muss ein Benutzer nicht mehr wissen wie der Communications Manager konfiguriert werden muss, sondern kann alle benötigten Daten in einem Fenster eingeben. Mabata leitet diesen Business Case dann an die Middleware weiter. Diese übernimmt anschliessend die Aufgabe, die einzelnen Transaktionen getrennt durchzuführen.

3.11. Modularisierung

3.11.1. Problemstellung

Durch die Vereinfachung der Mabata Business Case wurde die Konfiguration in einer zentralen Oberfläche zusammengefasst. Werden nun weitere Zielsysteme an die Middleware angebunden, müssen für die Konfigurationen dieser Zielsysteme zusätzliche Felder zur Verfügung stehen. Damit nicht jede Erweiterung manuell eingebunden werden muss, wird die Benutzeroberfläche modularisiert und die Erweiterungen können automatisch in die bestehende Oberfläche integriert werden.

3.11.2. Problemlösung

Um die gewünschten Erweiterungen in Mabata zu integrieren müssen die JSPX-Dateien, welche für die Benutzeroberfläche verwendet werden, erweitert sowie zusätzliche Java-Klassen erstellt werden. Damit die neu erstellten Klassen aus dem GUI angesprochen werden können, müssen zusätzlich die neuen Klassen in der ICEfaces Konfiguration «faces-config.xml» eingetragen werden. Um ein exklusiver Zugriff zu gewährleisten, wurde entschlossen, die Erweiterungen durch eine externe Applikation vorzunehmen.

Die Erweiterungen können entweder in einer neuen WAR-Datei oder im bestehenden Tomcat Verzeichnis vorgenommen werden.

3.11.2.1. Variante WAR-Datei

In dieser Lösungsvariante werden durch die oben beschriebene Applikation die Erweiterungen durchgeführt und eine neue WAR-Datei erstellt. Der Vorteil dieser Variante besteht darin, dass die Erweiterungen nicht auf dem Server erfolgen müssen. Nachdem die Erweiterungen vorgenommen wurden, kann die neue WAR-Datei auf den Server kopiert und deployed werden. Der einzige Nachteil bei diesem Vorgehen besteht darin, dass die Konfiguration nach dem deployen neu gemacht werden muss. Denn das Konfigurationstool ändert die Konfiguration in den aus der WAR-Datei extrahierten Dateien direkt im Tomcat Verzeichnis.

3.11.2.2. Variante Tomcat Verzeichnis

Der Nachteil der ersten Lösungsvariante kann dadurch eliminiert werden, in dem die Applikation, welche die Erweiterung vornimmt, direkt auf dem Server ausgeführt wird. Dadurch können die Dateien direkt im Tomcat Verzeichnis geändert und erstellt werden. Die Konfiguration bleibt folglich bestehen und muss nicht neu erstellt werden. Zusätzlich könnte bei dieser Variante die Funktion für die Erweiterung von Mabata in das Konfigurationstool integriert werden. Infolge dessen würde es bei einer einzigen externen Applikation für die Verwaltung von Mabata bleiben.

3.11.3. Fazit

An der Sitzung wurde entschieden, dass die Variante WAR-Datei weiterverfolgt werden soll. Der Nachteil dieser Variante, dass die Konfiguration neu erstellt werden muss, wird durch eine Anpassung des Konfigurationstools eliminiert. Das Konfigurationstool wird erweitert, sodass die Einstellungen in eine XML-Datei exportiert und folge dessen wieder importiert werden können. Dadurch kann vor dem Einfügen der neuen WAR-Datei die Konfiguration exportiert und später wieder importiert werden.

3.12. Displayname für Line

3.12.1. Problemstellung

Telefonnummern können in Mabata Telefonen und Profilen zugewiesen werden. Ruft ein Telefon A ein Telefon B an, wird die Nummer des Telefons A angezeigt. Allerdings wäre es wünschenswert, wenn ein zu definierender Text anstelle der Nummer angezeigt würde, um den Anrufer zu identifizieren.

3.12.2. Anforderung

Zu jeder Nummer, welche einem Profil oder einem Telefon zugewiesen wird, kann ein Text angegeben werden, welcher dann anstelle der Nummer auf dem Display angezeigt wird. Zu beachten gilt, dass dieser Text nicht direkt einer Nummer zugewiesen wird, sondern einer Line (eine Line verbindet eine Nummer mit einem Telefon/Profil).

3.12.3. Problemlösung

Um diese Funktion abzubilden, müssen die Phone/Profile Requests im myFramework angepasst werden. Hierzu steht jeweils im Line-Tag ein Tag «Display» zur Verfügung.

3.13. Passwörter als Hash-Werte ablegen

3.13.1. Problemstellung

Die Passwörter aller Mabata Benutzer werden im Klartext in der Mabata Datenbank abgespeichert. Damit ist das Auslesen aller Passwörter leicht möglich.

3.13.2. Anforderung

Die Passwörter sollen unkenntlich in der Datenbank abgespeichert werden, damit auch der Administrator die Passwörter der Mabata Benutzer nicht kennt.

3.13.3. Problemlösung

Nachdem ein Benutzer sein Passwort eingegeben hat, wird darüber ein MD5-Hash gebildet. Danach wird ausschliesslich mit diesem Hash-Wert gearbeitet.

3.14. Erweiterung des Nummerndatentyps

3.14.1. Problemstellung

In der neuen Version des Cisco Communications Managers ist es möglich, ein Präfix für eine Nummer einzugeben (z.B. +). Da in Mabata der Datentyp der Nummer ein `long` ist, unterstütz Mabata diesen neuen Nummerndatentypen nicht.

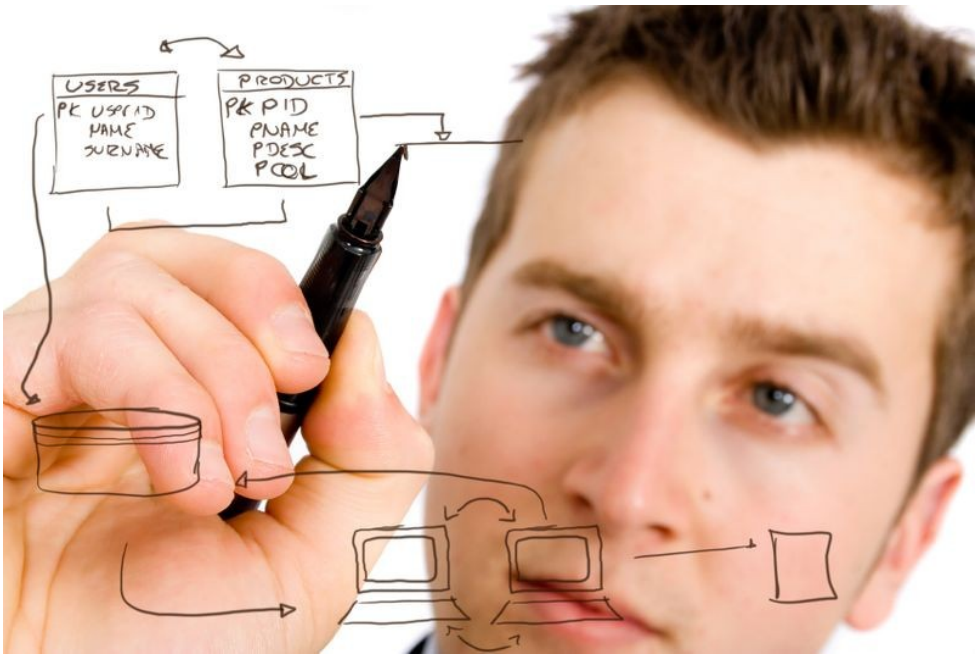
3.14.2. Anforderung

Mabata muss erweitert oder angepasst werden, damit eine Nummer nicht nur aus Zahlen bestehen kann.

3.14.3. Problemlösung

Da die Nummer in Mabata häufig für Vergleiche verwendet wird, ist eine Typenänderung von `long` nach `String` nur mit einem Performanceverlust durchführbar, da danach bei jedem Vergleich ein Cast von `String` auf `long` erfolgen muss.

Aus diesem Grund wird für den Präfix ein neues Datenfeld vom Typ `String` eingeführt, in welchem der Präfix abgespeichert werden kann.



Anforderungsspezifikation

Im nachfolgenden Teil der Dokumentation wird die Anforderungsspezifikation behandelt, welche folgendermassen unterteilt ist:

- Allgemeine Beschreibung
- Spezifische Anforderungen
- Use Cases Diagramm
- Elementary Use Cases

4. Anforderungsspezifikation

4.1. Allgemeine Beschreibung

4.1.1. Ist-Zustand

Die Applikation abstrahiert die Komplexität des Communications Managers und erleichtert die Administration. Nebst der einfacheren Handhabung erzielt die Applikation auch die Mandantenfähigkeit des Communications Managers.

4.1.2. Produkt Perspektiven

Ziel dieser Bachelorarbeit ist es, eine Zwischenschicht einzuführen die es ermöglicht, Daten von einer Applikation in eine oder mehrere andere Applikationen zu verteilen und innerhalb diesen Applikationen Aktionen durchzuführen. Danach soll Mabata erweitert werden, damit die Kommunikation zwischen Mabata und dem Communications Manager über die neue Zwischenschicht erfolgt.

Auch eine Vereinfachung des GUIs ist geplant, damit wird die Konfiguration von der jetzt an den Commuincations Manager angelehnten Sichtweise abstrahiert und dadurch die Benutzerfreundlichkeit gesteigert.

Zusätzlich sollen weitere Funktionen zu Mabata hinzugefügt, sowie auch das Services Framework als eigenständige Applikation zur Verfügung gestellt werden.

In dieser Bachelorarbeit werden:

- Eine Zwischenschicht für die Kommunikation mit mehreren Applikationen erstellt
- Mabata an diese Zwischenschicht angebunden
- Die Business Case Logik von Mabata vereinfacht
- Zusätzliche Funktionen für Mabata erstellt
    - PhoneButton Template inkl. Busy Lamp
    - SoftKey Template
    - Nummerntyp
    - Reporting/Syslog
    - Displayname für Line
    - Kostenabrechnung
    - Erweiterung des Nummerndatentyps
    - Passwörter als Hash-Werte ablegen
- Services Framework

Die erwähnten Punkte werden im Kapitel 4.2 spezifische Anforderungen detaillierter beschrieben.

4.1.3. Abhängigkeiten/Einschränkungen

Durch die Benutzung neuster AXL-Funktionen werden die Versionen 6.x und 7.x vom Cisco Communications Manager unterstützt, mit einer hohen Wahrscheinlichkeit zur Aufwärtskompatibilität.

4.1.4. Benutzercharakteristik

4.1.4.1. Administrator der Applikation (Role: ADMIN)

Die Administratoren betreiben den Communications Manager und die dazugehörigen Telefone. Die Benutzer haben Erfahrung mit der Administration des Communications Managers und sind mit der ganzen Thematik vertraut.

4.1.4.2. Verwalter eines Mandanten (Role: SUPERUSER)

Ein Mandant ist für den Betrieb und die Konfiguration der Telefone zuständig. Die Benutzer haben gute IT Kenntnisse und beherrschen die Bedienung und Konfiguration der Telefone sowohl in Mabata als auch am Telefon selbst.

4.1.4.3. Normaler Telefonbenutzer (Role: USER)

Die User sind die Benutzer der Telefone. Die Benutzer haben durchschnittliche IT Kenntnisse und können die Funktionen des Telefons in Mabata und am Telefon nutzen.

4.2. Spezifische Anforderung

4.2.1. New VoIP Source Architecture

4.2.1.1. Designvorschlag 1

Übersicht

Die in der Analyse festgehaltene Architektur und deren Anforderungen werden auf die für Mabata relevanten Bestandteile abgegrenzt.

Die Analyse beinhaltet eine Architektur zur Anbindung mehrerer Anwendungen über eine zentrale Stelle an diverse Frameworks. Diese Arbeit beschränkt sich darauf, das myFramework von Mabata zu trennen und über WebServices unterschiedlichen Applikationen verfügbar zu machen.

Anforderungen

Nachfolgende Abbildung dient als Grundlage für die Anforderungen. Es zeigt die zu realisierende Architektur. Dabei kommuniziert Mabata über WebServices mit dem myFramework. Die in der Analyse beschriebene Middleware wird dadurch nicht mehr benötigt.

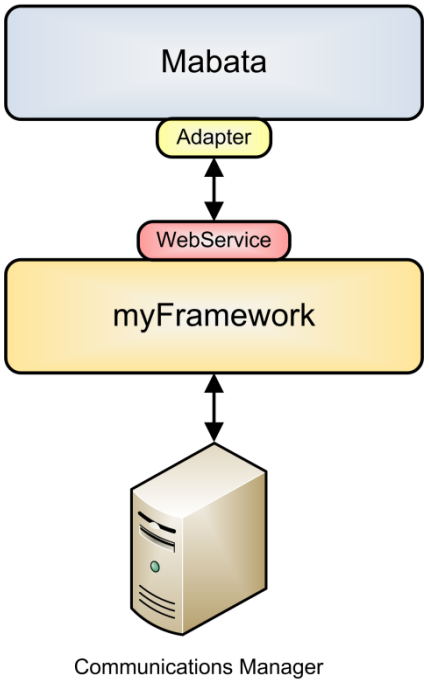


Abbildung 4-1 Kommunikation über eine Zwischenschicht

Daraus werden die folgenden Anforderungen definiert:

- Das myFramework ist aus Mabata herausgelöst und selbstständig lauffähig.
- Mabata kommuniziert über einen Adapter mit dem herausgelösten myFramework.
- Die Kommunikation erfolgt über WebServices.
- Das myFramework ist plattform- und programmiersprachenunabhängig.

4.2.1.2. Designvorschlag 2

Übersicht

Die im Designvorschlag 1 getroffene Entscheidung wurde überdacht und die Einführung einer Middleware wurde doch als nötig erachtet. Vor allem im Zusammenhang mit der Vereinfachung der Business Case Logik in Mabata wird eine Middleware benötigt, die eine gewisse Logik beinhaltet.
Auch kann nun das Services Framework die Middleware nutzen um auf den Communications Manager zugreifen zu können.

Anforderungen

Nachfolgende Abbildung dient als Grundlage für die Anforderungen. Es zeigt die zu realisierende Architektur. Dabei kommuniziert Mabata über WebServices mit der Middleware, die Middleware leitet die Aufrufe an das myFramework weiter. Dadurch können auch andere Systeme die Middleware nutzen. Zudem kann die benötigte Logik, um einen Business Case wieder in einzelne Transaktionen aufzuteilen, in die Middleware eingebaut werden.

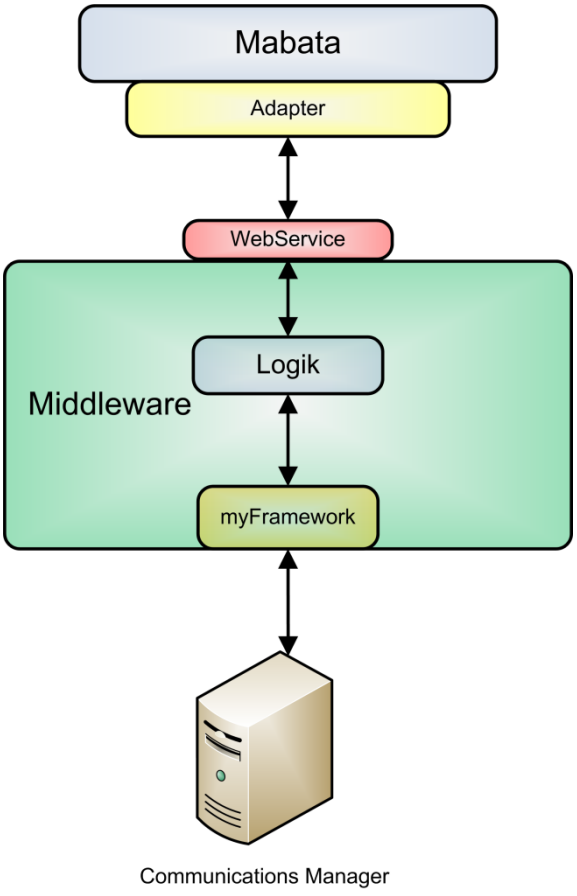


Abbildung 4-2 Kommunikation über eine Middleware

Daraus werden die folgenden Anforderungen definiert:

- Das myFramework ist aus Mabata herausgelöst und selbstständig in der Middleware lauffähig.
- Mabata kommuniziert über einen Adapter mit der Middleware.
- Die Middleware besitzt die Fähigkeit, logische Abläufe in einzelne Transaktionen für die Frameworks aufzuteilen.
- Die Kommunikation erfolgt über WebServices.
- Die Middleware ist plattform- und programmiersprachenunabhängig.
- Mabata wird als einziges Quellsystem an die Middleware angebunden.
- Der Cisco Communications Manager wird als einziges Zielsystem an die Middleware angebunden.

4.2.2. Reporting/syslog

4.2.2.1. Übersicht

Damit die getätigten Änderungen angezeigt werden können, sollen alle von den Benutzern durchgeführten Änderungen protokolliert werden. Diese protokollierten Daten sollen zudem den Benutzer unterstützen, Konfigurationsänderungen manuell rückgängig zu machen, in dem für geänderte oder gelöschte Objekte die aktuellen und vorherigen Attribute angezeigt werden.
Über Syslog wird die Möglichkeit geboten, diese Protokolldaten an einen externen Server zu übermitteln.

4.2.2.2. Anforderungen

Die von Mabata generierten Meldungen werden in die vier folgenden Kategorien eingeteilt:

Kategorie	Beschreibung
1 - Alert	Kritische Fehler, welche ein Eingreifen des Administrators benötigen
3 - Error	Fehler in der Applikation, beispielsweise durch eine nicht erfolgreiche Transaktion
5 - Notice	Vom Benutzer durchgeführte Aktionen (z.B. Telefon erstellen, editieren, löschen)
7 - Debug	Für den Betrieb unwichtige Informationen über alle Aktionen (z.B. gesamte Kommunikation zwischen Mabata und Communications Manager)

Tabelle 4-1 Reportingkategorien

Diese Kategorien wiederspiegeln die für Syslog verwendeten Message-Levels. Anhand dieser Kategorien können die Meldungen gefiltert werden. Zudem kann konfiguriert werden, welche Kategorien an den Syslog-Server weitergereicht werden sollen.

Kategorie 1 – Alert und Kategorie 3 - Error

Meldungen der Kategorie 1 und 3 werden in einer Textdatei gespeichert.

Kategorie 5 - Notice

Diese Meldungen werden in der Mabata Datenbank abgelegt. Hierzu werden jeweils die Attribute des entsprechenden Objektes in einem einzigen String abgespeichert. Die Meldungen dieser Kategorie könnten als Grundlage für die Kostenabrechnung verwendet werden.

Kategorie 7 – Debug

Meldungen dieser Kategorie können nur über Syslog auf einen Syslog-Server protokolliert werden. Debug Meldungen werden nur bei aktiviertem Debugmodus erstellt. Dieser Modus kann vom Administrator aktiviert werden.

4.2.3. Nummerntyp

4.2.3.1. Übersicht

In Mabata ist eine Telefonnummer vom Typ persönlich oder unpersönlich. In der vorgängigen Studienarbeit wurden die Pickup- und Hunt Gruppen eingeführt, welche ebenfalls Telefonnummern verwenden. Damit keine Konflikte entstehen, können für die Gruppen-Funktionen nur inaktive Nummern verwendet werden. Zugleich musste das Aktivieren einer inaktiven Nummer, welche für eine Gruppen-Funktion benutzt wird, unterbunden werden. Für den Benutzer ist demnach nicht ersichtlich, ob eine inaktive Nummer bereits für eine Gruppen-Funktion verwendet wird oder nicht.

Der Typ der Telefonnummern soll erweitert werden, damit klar ersichtlich wird, ob und wofür eine Telefonnummer verwendet wird.

4.2.3.2. Anforderungen

Für den Superuser soll ersichtlich sein, ob und für welche Funktion eine Telefonnummer verwendet wird. Es existieren die folgenden vier Typen:

- persönlich: Telefonnummer ist einem Benutzer zugewiesen
- unpersönlich: Telefonnummer ist keinem Benutzer zugewiesen
- pickup: Telefonnummer wird als Pickup Gruppen Nummer verwendet
- hunt: Telefonnummer wird als Hunt Pilot Nummer verwendet

Der Typ Pickup- und Hunt Gruppe wird automatisch gesetzt, sobald eine inaktive Nummer für die Hunt- oder Pickup-Funktion verwendet wird. Zusätzlich können wie bis anhin nur inaktive Nummern für die Gruppen-Funktion verwendet werden.

4.2.4. SoftKey Template

4.2.4.1. Übersicht

Die Pickup Gruppen erlauben es den Benutzern, eingehende Anrufe, welche nicht auf die eigene Telefonnummer kommen, zu übernehmen. Ist z.B. ein Mitarbeiter abwesend, können seine Arbeitskollegen im selben Büro die Anrufe über eine spezielle Taste auf ihren eigenen Telefonen entgegennehmen. Diese Tasten werden mittels den SoftKey Templates definiert. In Abbildung 4.3 ist ersichtlich welche Tasten über das SoftKey Template konfiguriert werden.

Abbildung 4-3 Tasten des SoftKey Templates

Die SoftKey Templates können nicht über die AXL-Schnittstelle konfiguriert werden und müssen demzufolge vorgängig auf dem Communications Manager erstellt werden. Nach dem die Templates auf dem Communications Manager erstellt worden sind, können sie über AXL den Telefonen zugewiesen werden.

4.2.4.2. Anforderungen

Aus der Übersicht kann die folgende Anforderung für den Superuser abgeleitet werden.

- Kann vorgängig auf dem Communications Manager definierte SoftKey Templates seinen Telefonen/Profilen zuweisen.

4.2.5. PhoneButton Template inkl. Busy Lamp

4.2.5.1. Übersicht

Die Kurzwahltasten ermöglichen es, auf dem Telefon Tasten zu definieren mit denen direkt Jemand angerufen werden kann. Zusätzlich zur Kurzwahl soll auch der Status (frei/besetzt) der jeweiligen Telefonlinie angezeigt werden. Dies kann mittels der Konfiguration der Busy Lamp Fields erreicht werden. Bei entsprechender Konfiguration leuchtet die Taste neben dem Teilnehmer rot, wenn dieser gerade am Telefonieren ist.

Für diese Funktion werden neue PhoneButton Templates benötigt. Diese Templates regeln die Funktion der Tasten gemäss Abbildung 4-4. Die Standardtemplates der jeweiligen Telefonmodelle bieten nur die Funktion für Kurzwahl an. Für die Verwendung der Busy Lamp Funktion müssen demnach neue Templates erstellt werden.

Abbildung 4-4 Funktion des PhoneButton Templates

Diese Templates können nicht über AXL erstellt werden, daher müssen diese vorgängig auf dem Communications Manager erstellt werden.

4.2.5.2. Anforderungen

Für den Superuser entstehen daraus folgende Anforderungen:

- Kann Telefonen oder Profilen die vorgängig erstellten PhoneButton Templates zuweisen.
- Kann nur die mit dem Telefon kompatiblen PhoneButton Templates zuweisen.

Der User stellt folgende Anforderungen an die Lösung:

- Kann wie gewohnt die Kurzwahltasten konfigurieren. Neu erhalten diese zusätzlich die Funktion für Busy Lamp, sofern der Superuser ein entsprechendes Template zugewiesen hat.

4.2.6. Services Framework

4.2.6.1. Übersicht

In einer vorhergehenden Diplomarbeit wurde ein Services Framework in eine alte Mabata Version integriert. Da in der aktuelle Version dieses Framework nicht zur Verfügung steht, soll dieses wieder in Mabata verfügbar gemacht werden.

Das Services Framework dient dazu, externe PhoneServices einzubinden und diese mandantenfähig zu machen.

4.2.6.2. Anforderungen

Für den Administrator sind folgende Funktionen möglich:

- Services können erstellt, gelöscht und bearbeitet werden
- Ein Service kann entweder global für alle Benutzer zur Verfügung stehen oder per Mandant beschränkt werden
- Servicespezifische Einstellungen können pro Mandant vorgenommen werden
- Festlegen, ob ein Parameter ein Passwortfeld oder ein benötigter Parameter ist

Der Superuser kann folgende Einstellungen vornehmen:

- Einen Service nur einzelnen Benutzern zur Verfügung stellen
- Die Serviceeinstellungen ändern
- Die Parameter mit Standardwerten belegen

Der User kann folgende Einstellungen vornehmen:

- Die Werte der Parameter anpassen

Als direkte Anforderung an das Services Framework wird gestellt, dass dieses selbstständig lauffähig ist.

4.2.7. Mabata Business Case Vereinfachung

4.2.7.1. Übersicht

Die Mabata Business Case Vereinfachung dient dazu, Mabata vom Communications Manager zu abstrahieren. Momentan ist vor allem für den Superuser das Verständnis für den Communications Manager wichtig, um mit Mabata umgehen zu können.

4.2.7.2. Anforderungen

Für den Administrator ergeben sich die folgenden Anforderungen:

- Mandant erstellen/ändern beinhaltet die folgenden Aktionen auf einmal:
  - Mandant selbst wird erstellt/geändert.
  - Zusätzliche Felder für den Superuser werden erfasst (PIN fällt weg, da Superuser nicht auf dem Communications Manager erstellt wird).
  - Nummernblock wird dem Mandanten zugewiesen und erstellt.
  - Telefone werden erfasst und dem Mandanten zugewiesen.

  Nachfolgendes Ablaufdiagramm stellt den ganzen Prozess grafisch dar:

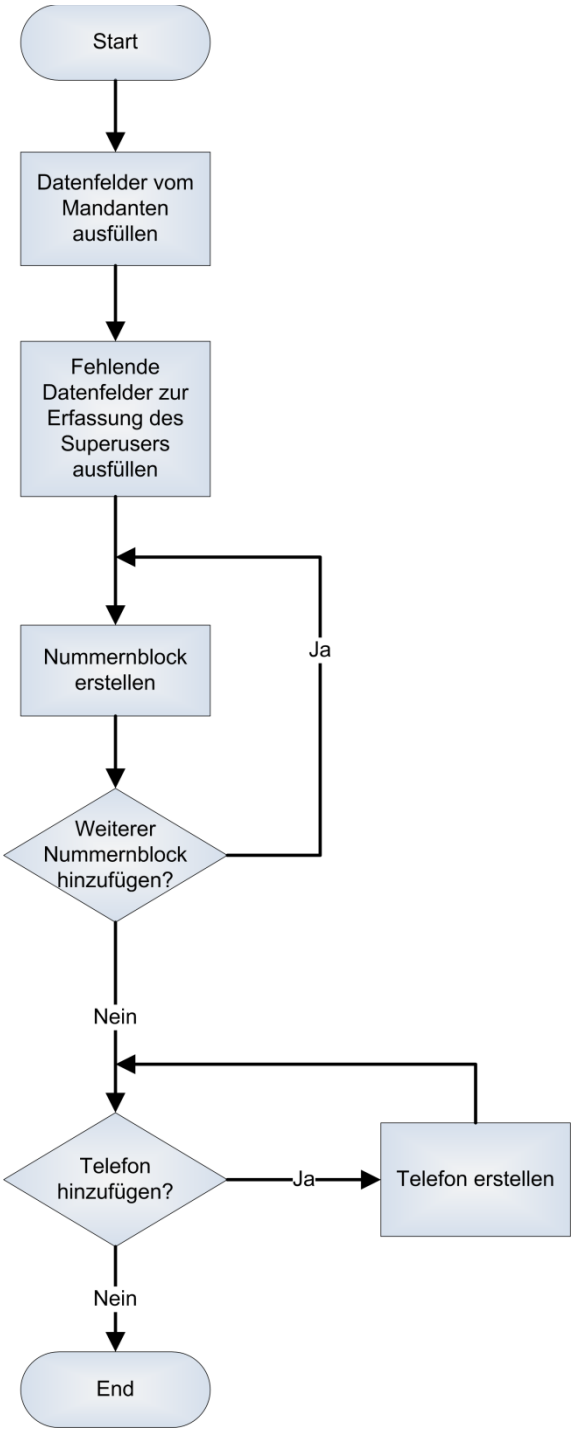


Abbildung 4-5 Ablauf des Mandanten Case

- Obiger beschriebener Ablauf ergibt die folgenden Änderungen:
  - Der Menüpunkt «Benutzer» fällt ganz weg.
  - Unter dem Menüpunkt «Funktionen» sind nur noch die Punkte «Authentisierung» und «Inventar» vorhanden.

Der Superuser profitiert am meisten von der Vereinfachung. Neu steht ein Benutzer im Zentrum der gesamten Konfiguration welche der Superuser betätigen kann:

- Benutzer erfassen gestaltet sich neu wie folgt:
  - Datenfelder für den Benutzer müssen ausgefüllt werden.
  - Telefon wird mitsamt Nummern dem Benutzer zugewiesen.
  - Profil wird inklusiv Nummern dem Benutzer zugewiesen.

  Dieser Business Case ist im nachfolgenden Diagramm veranschaulicht:

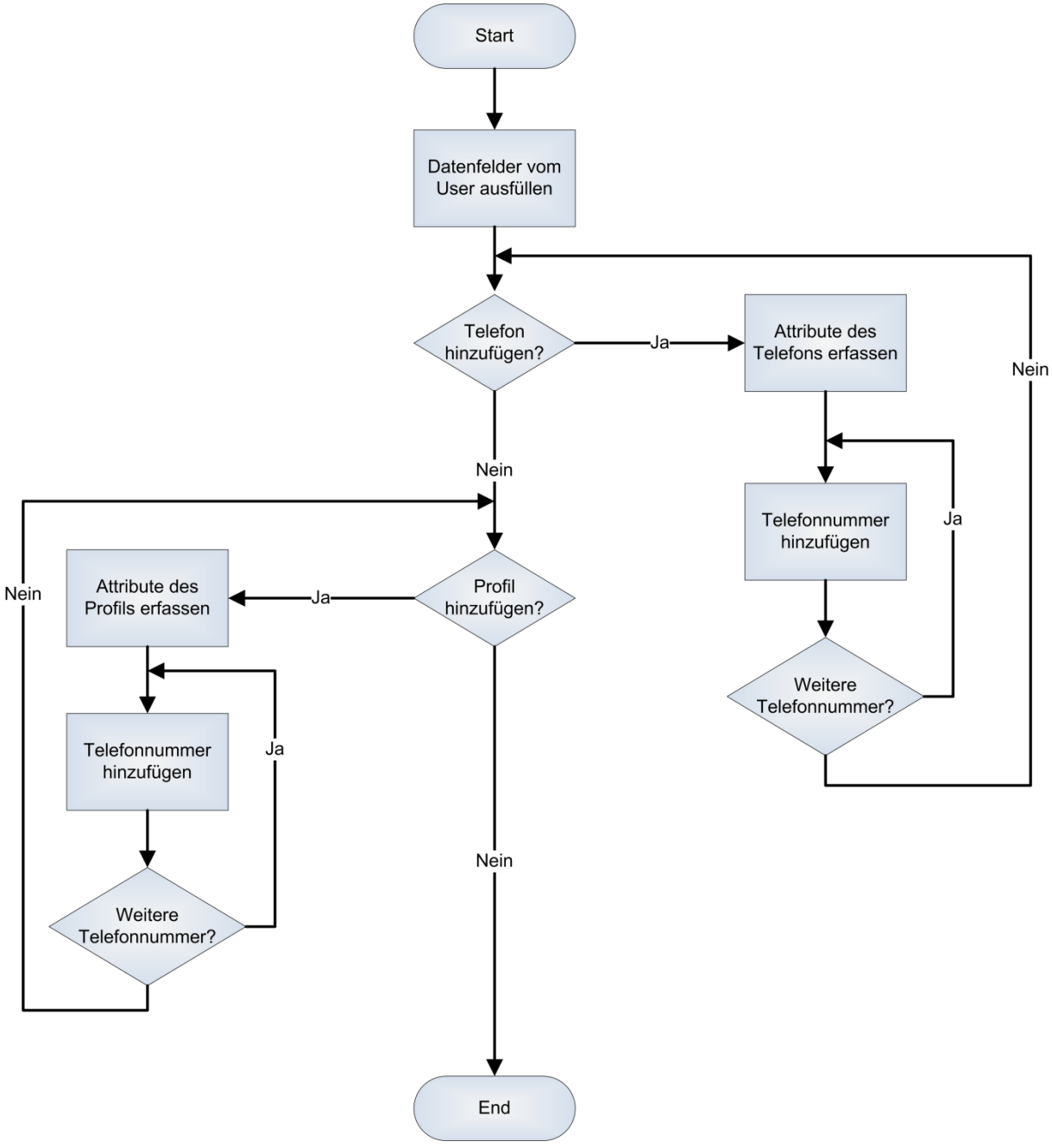


Abbildung 4-6 Ablauf des Benutzer Case

- Aus dieser Änderung folgt, dass die Menüpunkte «Anschluss», «Telefon» und «Profil» ganz wegfallen.
- Eine Übersicht über alle Objekte hat der Superuser wie bisher unter «Monitoring → Statistik».
- Der Menüpunkt «Group» bleibt wie bisher bestehen. Die Hunt- und Pickup Gruppen sind in Mabata schon wesentlich vereinfacht.
- Der Menüpunkt «Nummern» bleibt bestehen, in diesem werden die Nummern aktiviert/deaktiviert sowie die Attribute der Nummern geändert.

Für den User haben die Mabata Business Case Vereinfachungen keinen Einfluss, da das GUI für ihn schon sehr übersichtlich und einfach strukturiert ist.

4.2.8. Displayname für Line

4.2.8.1. Übersicht

Ein Displayname dient dazu, dass bei Anrufen zweier auf dem gleichen Communications Manager befindlichen Benutzern nicht die Nummer sondern ein definierter Text angezeigt wird. Dies hat den Vorteil, dass die Benutzer die Nummer des Anrufers nicht kennen müssen um dennoch zu wissen, wer sie anruft.

4.2.8.2. Anforderungen

Dem Superuser steht bei der Zuweisung der Nummer ein Textfeld zur Verfügung, in welchem er den Displaynamen eingeben kann. Standardmässig wird der Displayname mit Vor- und Nachnamen des jeweiligen Benutzers von dem entsprechenden Telefon/Profil vorgeschlagen. Der Displayname wird jeweils in der Nummerntabelle des Telefons/Profils angezeigt.

Eine Nummer kann mehreren Telefonen/Profilen zugewiesen werden. Bei jeder Zuweisung kann ein anderer Displayname angegeben werden.

4.2.9. Kostenabrechnung

4.2.9.1. Übersicht

Die Unique möchte die Möglichkeit haben, den verschiedenen Mandaten die Geräte in Rechnung zu stellen. Dabei sollen die Anzahl Telefone, Benutzer und Nummern in die Rechnung einfliessen.

4.2.9.2. Anforderungen

Es können Abrechnungen über selbst definierte Zeitperioden generiert werden, in welchen die vom Mandanten verwendeten Telefone, Benutzer und Nummern aufgelistet sind. In dieser Abrechnung ist jeweils enthalten, wann dieses Objekt erstellt wurde sowie der Zeitraum, in dem dieses Objekt aktiv war. Weiter wird anhand der Zeitdauer ein Betrag berechnet, welcher sich aus den vom Administrator vordefinierten Tagessätzen zusammensetzt.

Anforderungen für den Administrator:

- Kann Installationspauschale für Telefone auf Mandanten-Ebene festlegen
- Kann Tagespauschale für Benutzer, Telefone und Nummern auf Mandanten-Ebene festlegen
- Kann Abrechnungen für jeden Mandanten erstellen

4.2.10. Erweiterung des Nummerndatentyps

4.2.10.1. Übersicht

Eine Nummer wird in Mabata als `long` Datentyp behandelt. Somit ist es nicht möglich, Nummern nach dem internationalen Standard [E.164] einzufügen.

4.2.10.2. Anforderungen

Mabata kann mit Nummern, welche mit 00 oder + beginnen korrekt umgehen.

4.2.11. Passwörter als Hash-Werte ablegen

4.2.11.1. Übersicht

Alle verwendeten Benutzerpasswörter in Mabata werden als Klartext in der Mabata Datenbank gespeichert. Dies ist nicht zeitgemäss und sollte verhindert werden.

4.2.11.2. Anforderungen

Alle Passwörter werden in der Mabata Datenbank mittels des Hash-Wertes abgelegt.

4.3. Use Case Diagramm

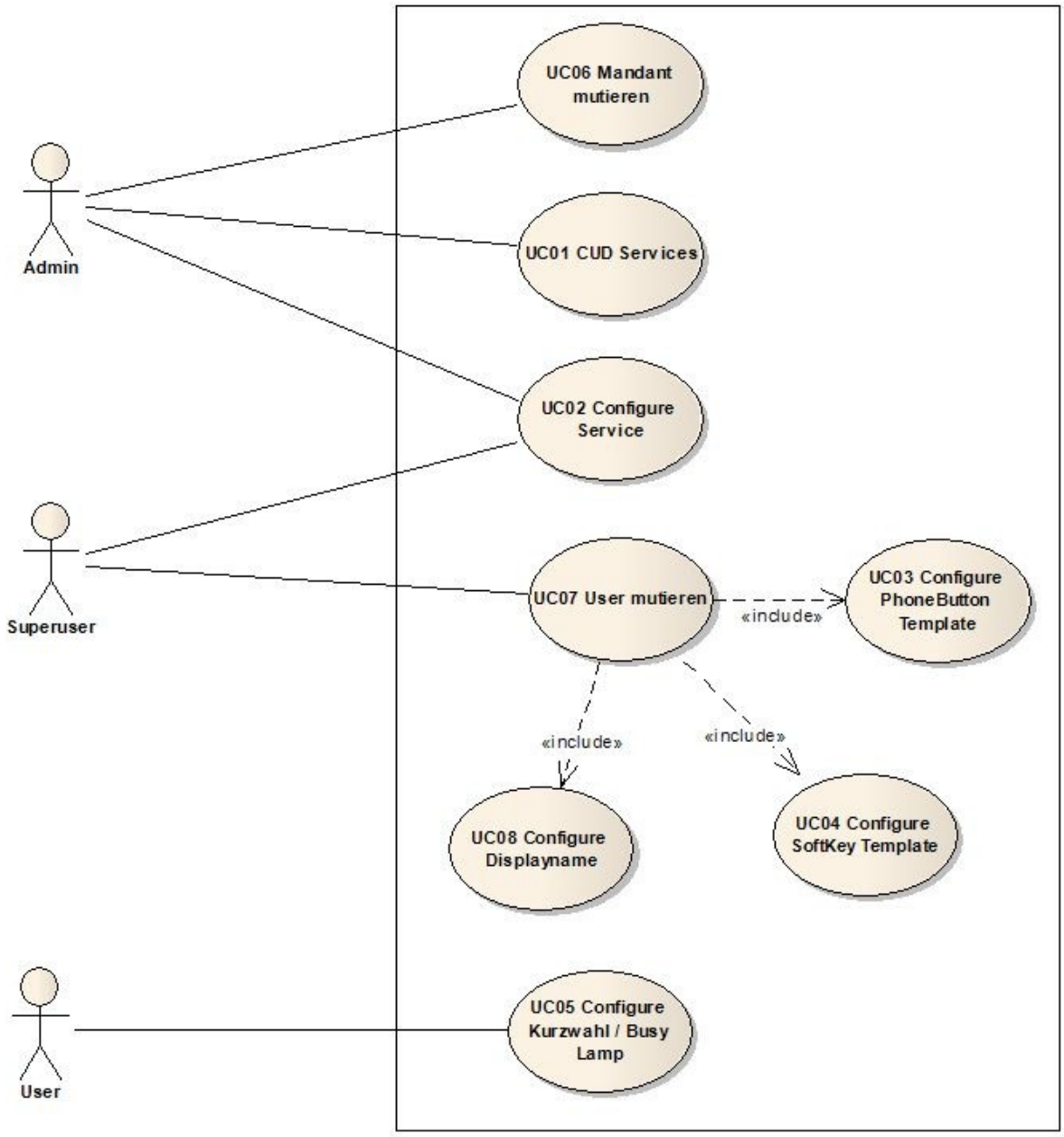


Abbildung 4-7 Use Case Diagramm

4.4. Elementary Use Cases

4.4.1. UC01 CUD Services

4.4.1.1. UC01a Create Service

Name	UC01a Create Service	
Description	Der Administrator hat die Möglichkeit einen Service zu erstellen	
Primary Actor	Administrator	
Frequenzy	1 mal pro Monat	
Preconditions	Admin ist angemeldet	
Postconditions	Neuer Service ist erstellt	
Process	User Intention	System Intention
	1. Erstellt einen neuen Service	
		2. Zeigt Eingabemaske an
	3. Gibt die dafür benötigten Angaben ein • PhoneService Name • URL • Global oder Mandanten	
		4. Validiert die Angaben und speichert diese ab

Tabelle 4-2 UC01a Create Service

4.4.1.2. UC01b Update Service

Name	UC01b Update Service	
Description	Der Administrator hat die Möglichkeit einen Service zu bearbeiten	
Primary Actor	Administrator	
Frequenzy	1 mal pro Monat	
Preconditions	Existierender Service vorhanden	
Postconditions	Service wurde aktualisiert	
Process	User Intention	System Intention
	1. Wählt gewünschten Service aus	
		2. Zeigt Details an
	3. Editiert Service	
		4. Speichert Daten ab

Tabelle 4-3 UC01b Update Service

4.4.1.3. UC01c Delete Service

Name	UC01c Delete Service	
Description	Der Administrator hat die Möglichkeit einen Service zu löschen	
Primary Actor	Administrator	
Frequenzy	1 mal pro Monat	
Preconditions	Existierender Service vorhanden	
Postconditions	Service wurde gelöscht	
Process	User Intention	System Intention
	1. Wählt gewünschten Service aus	
		2. Zeigt Details an
	3. Löscht Service	
		4. Löscht alle Servicedaten

Tabelle 4-4 UC01c Delete Service

4.4.2. UC02 Configure Service

Name	UC02 Configure Service	
Description	Pro erstellten Service kann für jeden Mandanten eine eigene Konfiguration eingegeben werden	
Primary Actor	Administrator / Superuser	
Frequenzy	1 mal pro Woche	
Preconditions	Existierender Service vorhanden	
Postconditions	Servicekonfiguration wurde abgespeichert	
Process	User Intention	System Intention
	1. Wählt gewünschten Service aus	
		2. Zeigt Details an
	3. Definiert Menu Text Gib alle gewünschten Attribute/Value Paare ein oder löscht nicht mehr gebrauchte Paare	
		4. Speichert die Eingaben ab

Tabelle 4-5 UC02 Configure Service

4.4.3. UC03 Configure PhoneButton Templates

Name	UC03 Configure PhoneButton Template	
Description	Für jedes Telefon kann der Superuser das PhoneButton Template festlegen	
Primary Actor	Superuser	
Frequenzy	Mehrmals wöchentlich	
Preconditions	Auf dem Communications Manager wurde ein entsprechendes PhoneButton Template eingerichtet	
Postconditions	Das neue PhoneButton Template des Telefons wurde abgespeichert	
Process	User Intention	System Intention
	1.Wählt das Drop-Down Menü für die PhoneButton Templates aus	
		2. Zeigt alle für dieses Telefon verfügbaren PhoneButton Templates an
	3. Wählt ein PhoneButton Template aus	

Tabelle 4-6 UC03 Configure PhoneButton Templates

4.4.4. UC04 Configure SoftKey Templates

Name	UC04 Configure SoftKey Template	
Description	Für jedes Telefon kann der Superuser das SoftKey Template festlegen	
Primary Actor	Superuser	
Frequenzy	Mehrmals wöchentlich	
Preconditions	Auf dem Communications Manager wurde ein entsprechendes SoftKey Template eingerichtet	
Postconditions	Das neue SoftKey Template des Telefons wurde abgespeichert	
Process	User Intention	System Intention
	1. Wählt das Drop-Down Menü für die SoftKey Templates aus	
		2. Zeigt alle für dieses Telefon verfügbaren SoftKey Templates an
	3. Wählt ein SoftKey Template aus	

Tabelle 4-7 UC04 Configure SoftKey Templates

4.4.5. UC05 Configure Kurzwahl / Busy Lamp

Name	UC05 Configure Kurzwahl / Busy Lamp	
Description	Für jedes seiner Telefone/Telefonprofile kann der Benutzer die Kurzwahltasten mit Busy Lamp Funktion konfigurieren	
Primary Actor	User	
Frequenzy	1 mal pro Monat	
Preconditions	Der Superuser hat dem Telefon/Profil des Benutzers ein PhoneButton Template mit Busy Lamp Funktion zugewiesen	
Postconditions	Die neue Kurzwahl / Busy Lamp Konfiguration wurde abgespeichert Die neuen Einstellungen werden auf dem Telefon des Benutzers angezeigt	
Process	User Intention	System Intention
	1. Wählt die Schaltfläche Kurzwahl an	
		2. Zeigt alle verfügbaren Telefone/Profile des Benutzers an
	3. Wählt gewünschtes Telefon/Telefonprofil aus	
		4. Zeigt die alte Kurzwahlkonfiguration an
	5. Ändert die Kurzwahlkonfiguration	
		6. Speichert die Eingabe ab

Tabelle 4-8 UC05 Configure Kurzwahl / Busy Lamp

4.4.6. UC06 Mandant mutieren

Name	UC06 Mandant mutieren	
Description	Der Administrator kann einen Mandanten erstellen/mutieren	
Primary Actor	Administrator	
Frequenzy	1 mal pro Woche	
Preconditions	Der Administrator ist am System angemeldet	
Postconditions	Ein neuer Mandant wurde erstellt. Ein Nummernblock wurde ihm zugewiesen, sowie je nach Bedarf Telefone	
Process	User Intention	System Intention
	1. Wählt die Schaltfläche Neuer Mandant aus	
		2. Holt benötigten Daten vom Communications Manager ab und zeigt diese an
	3. Füllt alle benötigten Felder für den Mandant aus	
	4. Erstellt einen Nummernblock für den Mandant	
	Wiederholt Schritt 4 so oft wie nötig	
	5a. Weist dem Mandant ein bestehendes Telefon zu 5b. Erstellt ein neues Telefon und weißt es dem Mandanten zu	
	Wiederholt Schritt 5a/5b so oft wie nötig	
	6. Speichert die Eingaben ab	
		7. Wertet die Antwort aus und zeigt diese dem Benutzer an

Tabelle 4-9 UC06 Mandant mutieren

4.4.7. UC07 User mutieren

Name	UC07 User mutieren	
Description	Der Superuser kann einen User erstellen/mutieren	
Primary Actor	Superuser	
Frequenzy	Mehrmals wöchentlich	
Preconditions	Der Superuser ist am System angemeldet	
Postconditions	Ein neuer User wurde erstellt	
Process	User Intention	System Intention
	1. Wählt die Schaltfläche Neuer Benutzer aus	
		2. Zeigt benötigte Daten an
	3. Füllt alle benötigten Felder für den Benutzer aus	
	4a1. Weist dem Benutzer ein bestehendes Telefon zu. Include UC03/UC04 4a2.Weisst dem Telefon zusätzliche Nummern zu. Include UC08 4b1. Erstellt ein neues Telefon und weisst es dem Benutzer zu. Include UC03/UC04 4b2. Weisst dem Telefon zusätzliche Nummern zu. Include UC08	
	Wiederholt Schritt 4a/4b so oft wie nötig	
	5a1. Weist dem Benutzer ein bestehendes Telefonprofil zu. Include UC03/UC04 5a2. Weisst dem Telefonprofil zusätzliche Nummern zu. Include UC08 5b1. Erstellt ein neues Telefonprofil und weisst es dem Benutzer zu. Include UC03/UC04 5b2. Weisst dem Telefonprofil zusätzliche Nummern zu. Include UC08	
	Wiederholt Schritt 5a/5b so oft wie nötig	
	6. Speichert die Eingaben ab.	
		7. Wertet die Antwort aus und zeigt diese dem Benutzer an.

Tabelle 4-10 UC07 User mutieren

4.4.8. UC08 Configure Displayname

Name	UC04 Configure Displayname	
Description	Für jede Line welche einen Telefon/Profil zugewiesen ist, kann ein Displayname konfiguriert werden	
Primary Actor	Superuser	
Frequenzy	Mehrmals wöchentlich	
Preconditions	Der Superuser ist am System angemeldet	
Postconditions	Auf dem Telefon erscheint beim Anruf nicht mehr die Nummer des Anrufers sondern der Displayname	
Process	User Intention	System Intention
	1. Wählt Nummer aus und gib Displaynamen ein. Bestätig die Eingabe	
		2. Kopiert die Nummer und speichert den Displaynamen in der Line

Tabelle 4-11 UC08 Configure Displayname

Design

Im nachfolgenden Teil der Dokumentation wird das Design behandelt, welches folgendermassen unterteilt ist:

- Mabata Übersicht
- Business Case Vereinfachung
- New VoIP S
- ource Architecture
- Weitere neue Funktionen
- Core Verbesserungen

5. Design

Das folgende Kapitel 5.1 Mabata Übersicht wurde aus der vorgängigen Studienarbeit übernommen und auf die neuen Anforderungen angepasst. Nicht alle in diesem Dokument beschriebenen und erarbeiteten Lösungen wurden umgesetzt. Welche Funktionen und Erweiterungen umgesetzt wurden, ist im Dokument Bachelorarbeit genauer erläutert.

5.1. Mabata Übersicht

Dieses Kapitel dient zur Übersicht von Mabata. Es soll erklären, wie Mabata aufgebaut ist. Dies dient zur Grundlagen der neuen Funktionen und Änderungen, welche in dieser Bachelorarbeit entwickelt wurden.

5.1.1. Physische Architektur

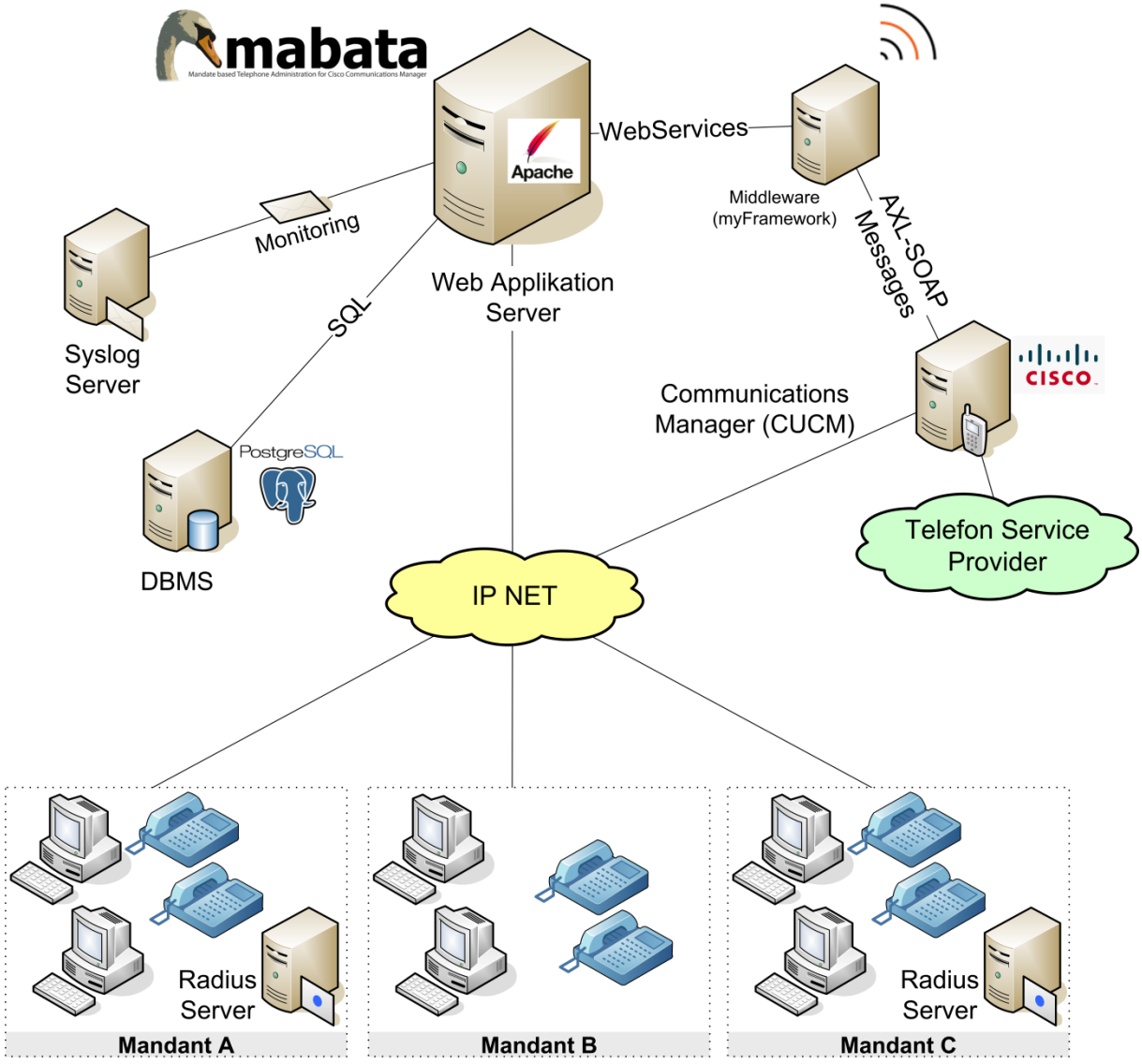


Abbildung 5-1 Physische Übersicht

5.1.1.1. Web Applikation Server

Mabata wird auf dem Apache Tomcat Version 6.0 betrieben. Dieser Server ist ein Open-Source Produkt. Die Aufwärtskompatibilität muss vor der Verwendung einer anderen Version zuerst geprüft werden, da ICEfaces verwendet wird.
Apache Tomcat ist betriebssystemunabhängig und deshalb sowohl auf Windows wie auch auf Unix/Linux Betriebssystemen lauffähig.

5.1.1.2. Syslog Server

Der Syslog Server wird verwendet um die Log-Daten von Mabata zu protokollieren. Er wird benötigt, um Debug Daten von Mabata anzuzeigen. Zudem kann er zum Archivieren der Protokolldaten verwendet werden.

5.1.1.3. DBMS – Datenbank Management System

PostgreSQL Server wird als Datenbank Management System verwendet. Da aber die Persistenz Logik mittels Hibernate realisiert wurde, sollte ein anderes Datenbanksystem ohne grössere Probleme verwendet werden können. Zu beachten gilt, dass die Kommunikation zur Datenbank unverschlüsselt erfolgt und daher die Übertragung der Daten über eine gesicherte Verbindung erfolgen sollten.

5.1.1.4. Middleware (WebServices)

Die Middleware nimmt Methodenaufrufe von Mabata entgegen, verarbeitet diese und leitet sie an den Communications Manager weiter.

5.1.1.5. Communications Manager

Mabata arbeitet zurzeit mit dem Cisco Communications Manager Version 6.x und 7.x. Die Kommunikation mit diesem System erfolgt über die AXL-Schnittstelle mittels SOAP-Messages. Da die AXL-Schnittstelle ständig erweitert wird und Mabata einige der neusten Funktionen verwendet, ist eine Abwärtskompatibilität zum Communications Manager 5.x nicht gewährleistet.

5.1.1.6. Radius Server

Die Benutzerauthentifizierung (wer) kann über zwei verschiedene Wege erfolgen. Entweder wird das Benutzer-Passwort auf der oben erwähnten Datenbank gespeichert und folglich beim Anmeldevorgang über diese geprüft, oder es können verschiedene Radius-Server definiert - und den Benutzern zugeteilt werden. In diesem Fall geschieht die Authentifizierung über den konfigurierten Radius Server. Die Autorisierung (was) geschieht in jedem Fall über die Datenbank. Dies kann pro Benutzer definiert werden.

5.1.1.7. Client

Die Clients kommunizieren über das IP-Netz (per (S)HTTP) mit Mabata. Für die Nutzung benötigt ein Client lediglich einen Web Browser der aktuellen Generation (IE 7, Firefox 3). Mozilla Firefox 3 wird für die Verwendung empfohlen. Spezielle Plugins sind dabei nicht notwendig.

5.1.2. Logische Architektur

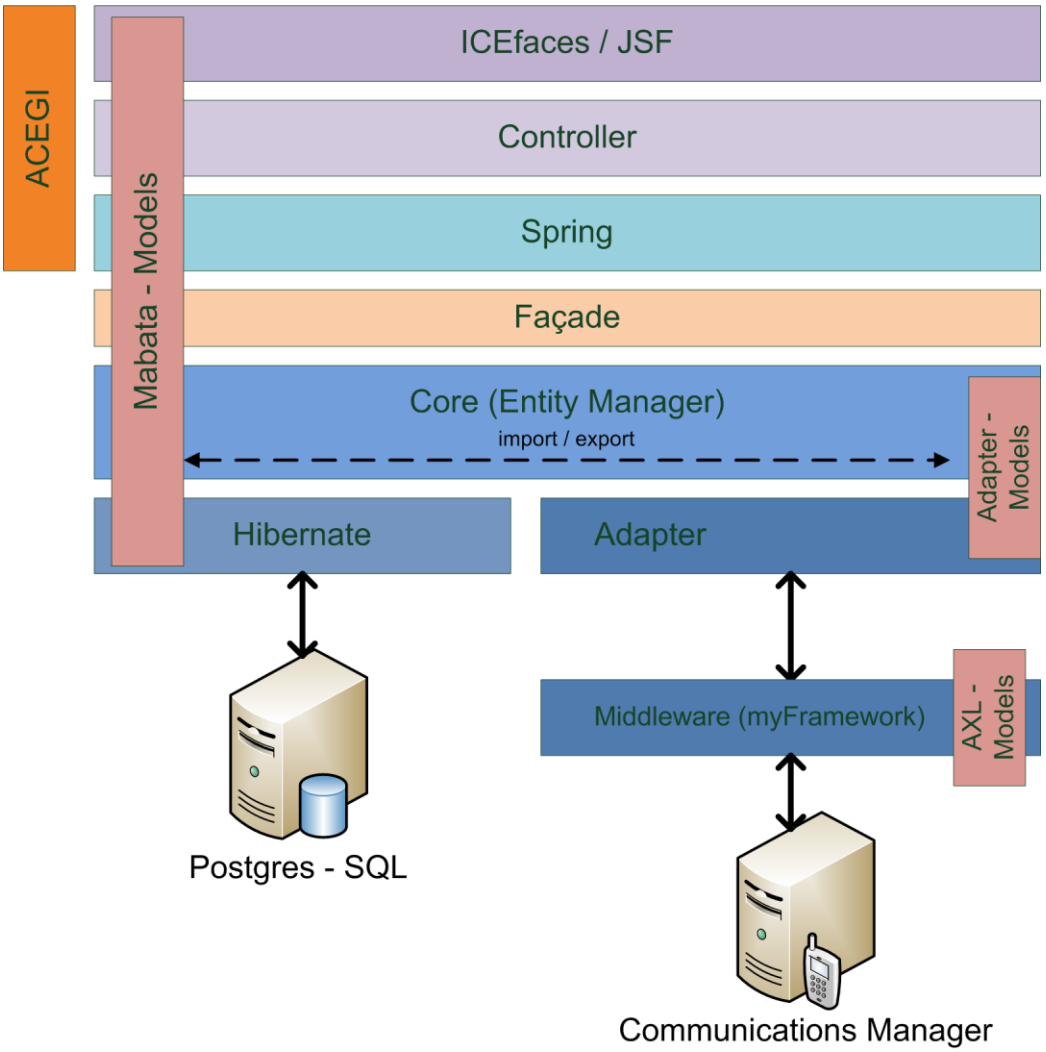


Abbildung 5-2 Logische Übersicht

In der oben stehenden Grafik ist eine Übersicht aller logischen Schichten aufgezeigt. Das Design wurde dementsprechend gewählt, dass eine möglichst klare Struktur entsteht. Dabei erfolgt eine Kommunikation nur mit den direkten Nachbarn und nie über eine Schicht hinweg. Die genaue Funktionsweise und weitere Details sind in den folgenden Abschnitten beschrieben.

5.1.2.1. ICEfaces

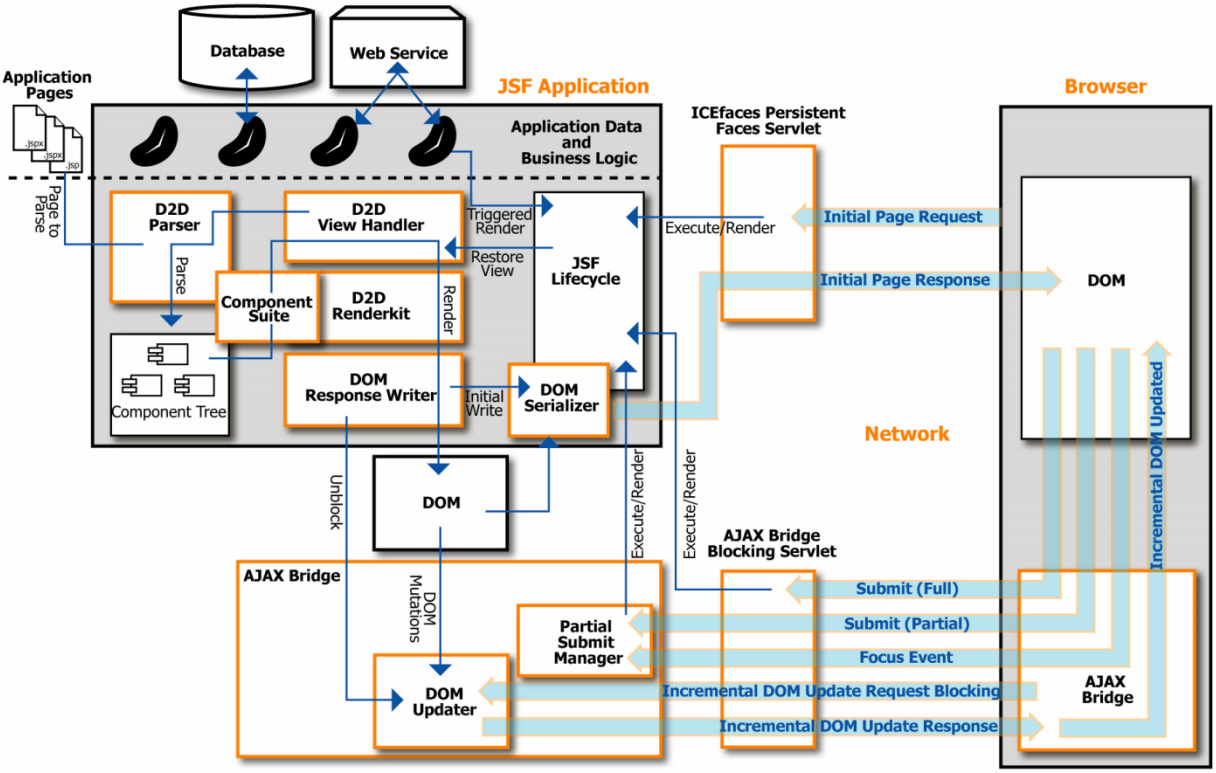


Abbildung 5-3 ICEfaces Übersicht

ICEfaces ist ein auf Java Server Faces (JSF) basierendes Ajax Framework für Java, welches durch ICEsoft entwickelt wird. Seit November 2006 steht das Projekt neben einer kommerziellen Lizenz auch unter der Mozilla Public License (MPL) zur Verfügung. ICEfaces verfügt über eine umfangreiche Bibliothek an Komponenten für die Benutzerschnittstelle mit eingebauter Ajax-Funktionalität. Dazu gehören beispielsweise Bäume, Registerkarten (Tabs) und Menüs. Diese Komponenten können in einer standardkonformen JSF-Seite verwendet werden. Im Gegensatz zu den üblichen Komponenten einer Webseite können diese jedoch über Ajax mit dem Webcontainer kommunizieren und bei Bedarf Daten nachladen ohne dass die gesamte Webseite neu geladen wird. Der Entwickler muss sich dabei nicht um die Details dieser Kommunikation kümmern, da diese im Hintergrund transparent abläuft. Mit ICEfaces lassen sich somit Rich Internet Application (RIA) entwickeln ohne JavaScript programmieren zu müssen. Der schlussendlich im Browser ausgeführte JavaScript-Code wird durch das Framework zur Laufzeit generiert. (Quelle: Wikipedia)

5.1.2.2. Navigation

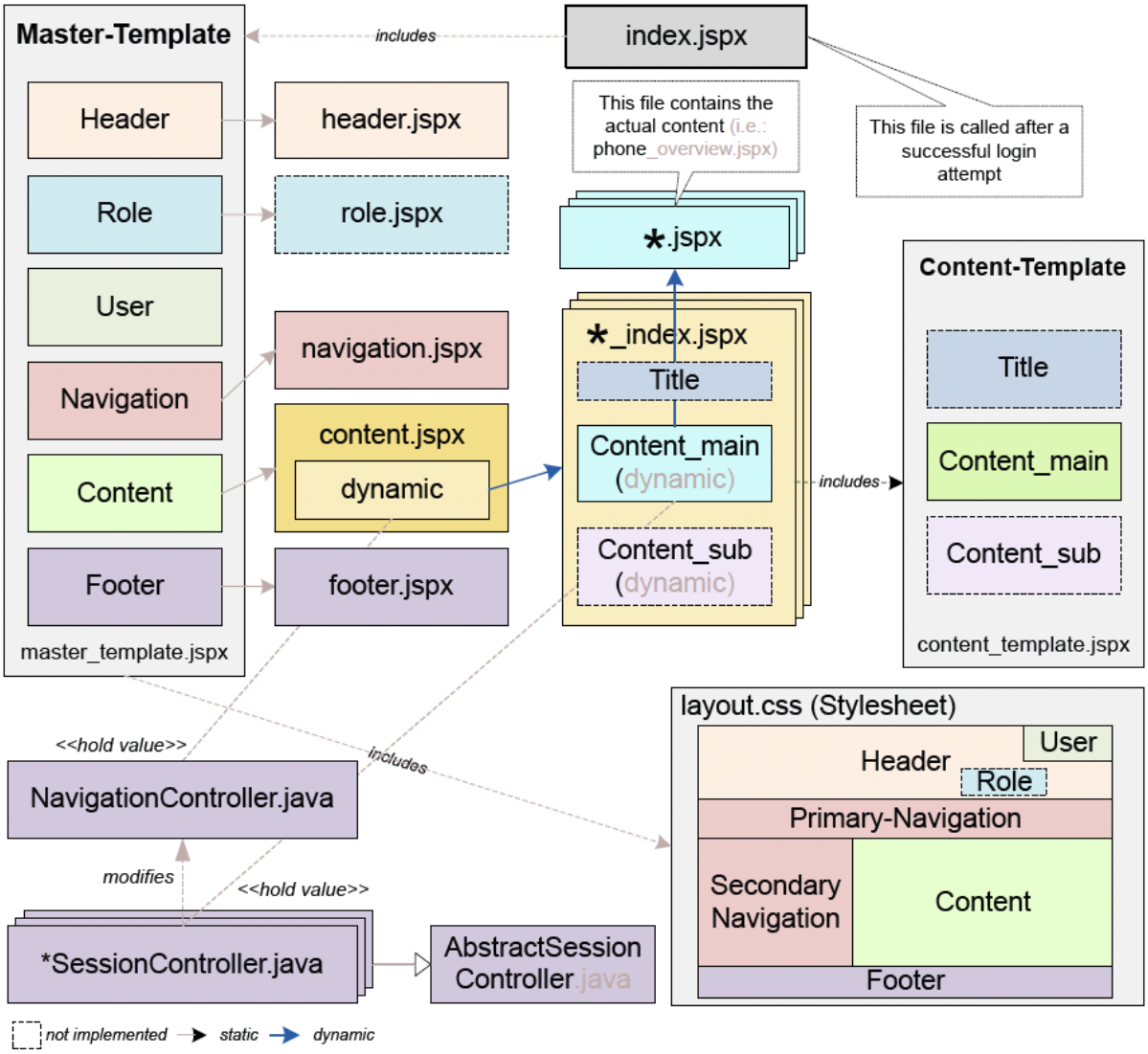


Abbildung 5-4 Navigationsübersicht

Die Navigation und das Page-Design sind mittels Facelets, Cascading Stylesheets und ICEfaces Komponenten implementiert. Zum besseren Verständnis von Facelets und Dynamic-Includes wird auf folgende Referenzen verwiesen:

Aufzählung:

- Facelets fits JSF like a glove «http://www.ibm.com/developerworks/java/library/j-facelets/»
- How to Use Facelets Dynamic Includes «http://facestutorials.icefaces.org/tutorial/dynamic-includestutorial.html»

Bis auf zwei dynamische Includes, ist der gesamte Aufbau statisch d.h. das Ein- und Ausblenden der Sekundär-Navigation wird mittels ICEfaces Kompetenten bewerkstelligt. Für das Austauschen der dynamischen Includes ist der `NavigationController` und der jeweilige `SessionController` verantwortlich. Den Hauptpfad resp. Use Case steuert der `NavigationController`, der

`SessionController` steuert dann die jeweilige Ansicht (detail, overview, upload, etc.). Für weitere Informationen sei auf die JavaDoc verwiesen.

Master-Template

Das Master-Template beinhaltet alle für die Anzeige einer Seite nötigen Teildateien. Eine Webseite wurde dazu in die 6 verschiedenen Teilseiten unterteilt: Header (Kopf der Seite), Role (erkennt die Rolle, welcher der Benutzer angehört), User (zeigt den angemeldeten Benutzer an), Navigation (beinhaltet das Menü), Content (beinhaltet den eigentlichen Inhalt der Seite, der je nach aktueller Seite dynamisch geladen wird), Footer (Fusszeile der Seite).

Das Master-Template fügt nun alle diese Teilseiten zu einer ganzen Seite zusammen. Das Layout der Seite wird durch CSS-Dateien gesteuert.

Content-Template

Das Content-Template ist für die Bereitstellung des dynamischen Inhalts zuständig.

Navigation

Für die Navigation ist der `SessionController` zuständig. Er erhält Befehle um das Menü zu wechseln und leitet diese an den `NavigationController` weiter. Der `NavigationController` gibt den Aufruf an den Content weiter, welcher jetzt dynamisch den für den neuen Menüpunkt benötigten Inhalt bereitstellt.

5.1.2.3. JSF

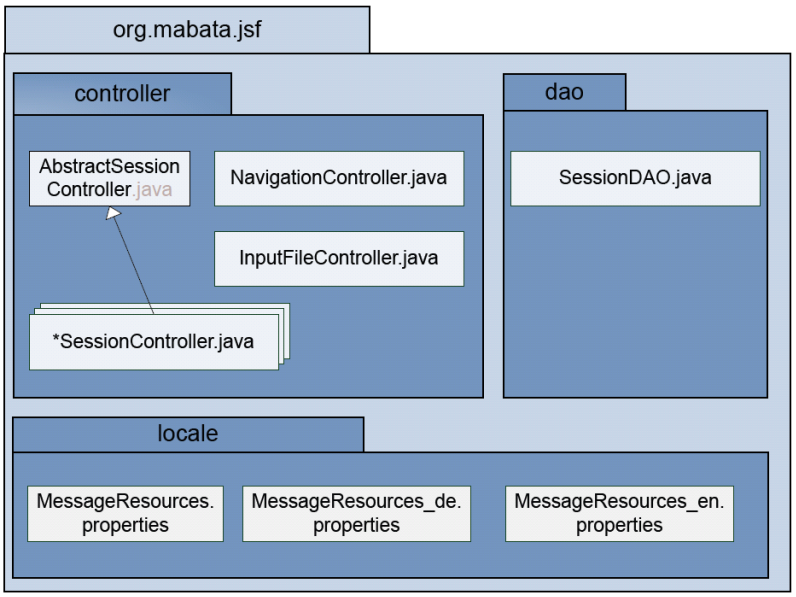


Abbildung 5-5 JSF Übersicht

Die JSF Logik befindet sich hauptsächlich in den einzelnen `SessionControllern`. Sie werden von den jeweiligen Facelets angesprochen und dienen als Stateholder für die momentan bearbeitenden Objekte. Zur Kommunikation mit der Business Logik wird ausschliesslich das `SessionDAO` verwendet. Es dient als Caching Objekt, in dem es vor allem Listen und Objekte, welche sich nicht bei jeder Aktion ändern, zwischenspeichert. Dies bedeutet, dass für die Präsentation der Daten auf der Benutzerschicht weniger häufig Datenbank- oder Communications Manager Zugriffe notwendig sind. Alle Labels, Buttons, etc. wurden mit Message-Properties eingebunden, so können einfach und direkt verschiedene Sprachen implementiert werden. Die Erkennung der jeweiligen Sprache läuft über die jeweilige Browser-Einstellung. Zurzeit wird nur Deutsch als Sprache unterstützt. Die Konfiguration von JSF befindet sich unter «/WEB-INF/faces-config.xml». Für weitere Informationen sei auf die JavaDoc verwiesen.

5.1.2.4. ACEGI Security Modul

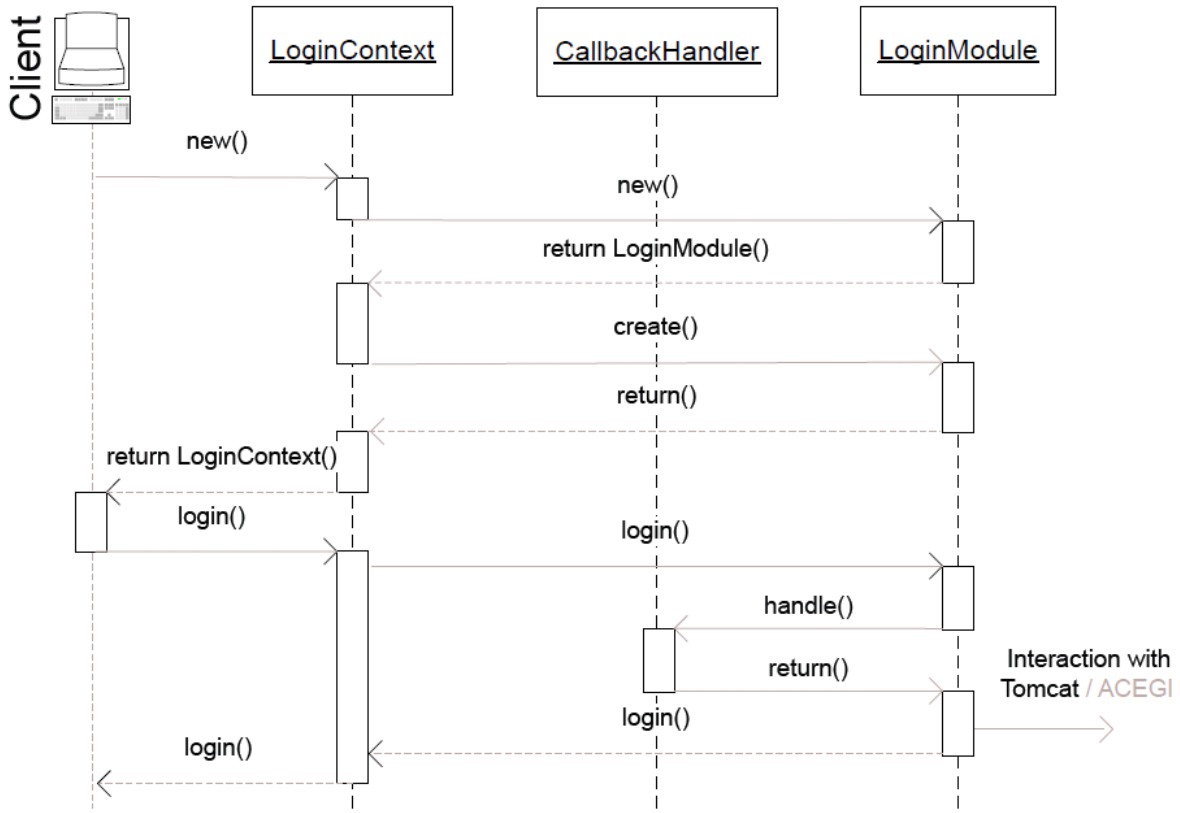


Abbildung 5-6 ACEGI-Security Modul

Die Security wurde innerhalb des Web-Projektes mit ACEGI von Spring realisiert. Das Authentifizieren wird über das eigene LoginModul abgehandelt. Die ACEGI Konfiguration befindet sich unter «/WEB-INF/context/acegi.ctx.xml». Für weitere Informationen sei auf die JAASLogin-Modul-Dokumentation, das ACEGI Security Framework und die JavaDoc von Mabata verwiesen.

5.1.2.5. Spring

Der MainSessionController initialisiert das oben erwähnte `SessionDAO`, abhängig von der entsprechenden Benutzer-Rolle. Die Konfiguration von Spring befindet sich unter «/WEB-INF/context/*». Für weitere Informationen sei auf die JavaDoc verwiesen.

5.1.3. Core

5.1.3.1. Models

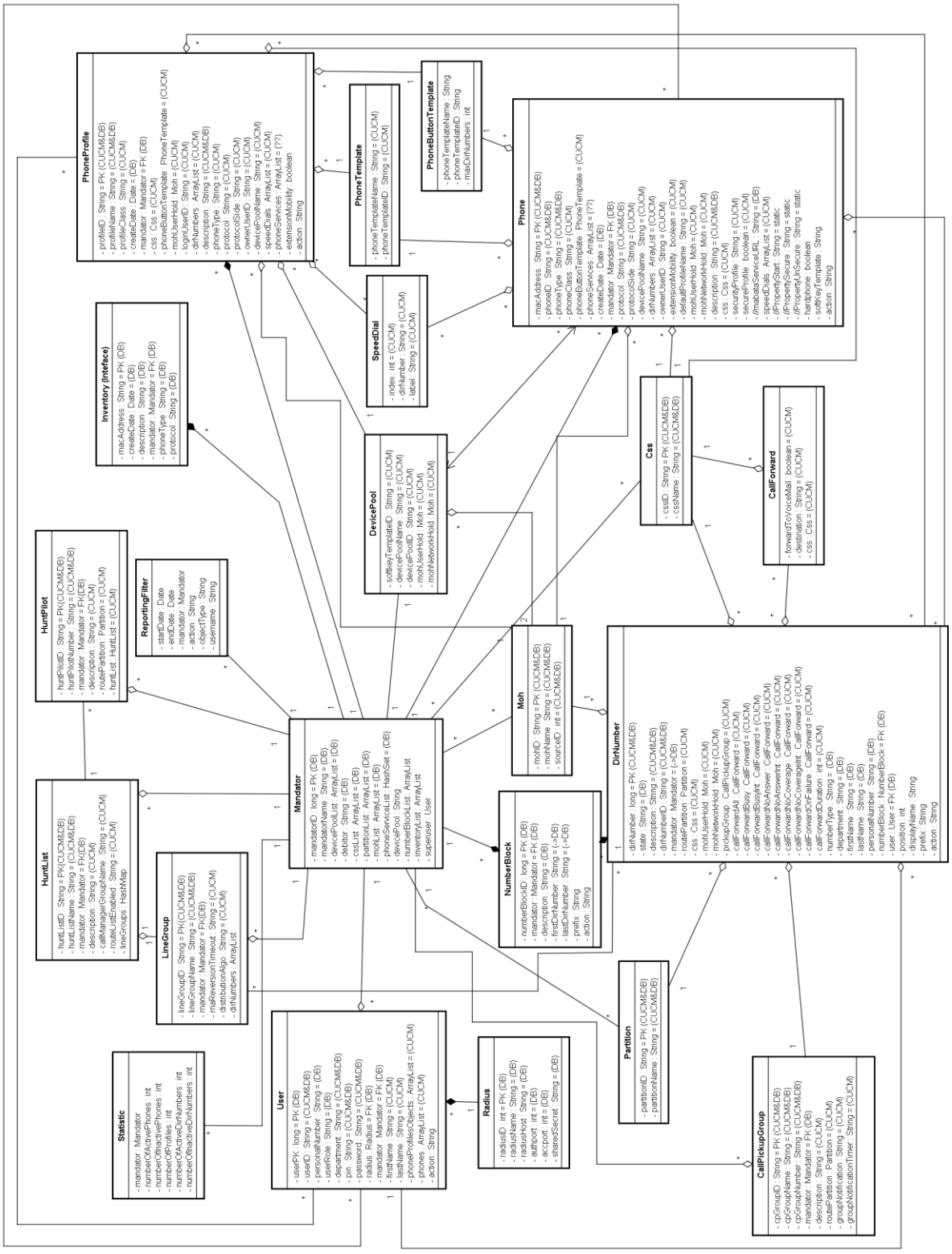


Abbildung 5-7 Mabata Models

Da Mabata mit zwei verschiedenen Systemen (Communications Manager und lokale Datenbank) zusammenarbeitet, muss eine verteilte Persistenz gepflegt werden. Dies hat zur Folge, dass verschiedene Typen von Models benötigt werden, um das grundlegende Prinzip des Low-coupling sicherzustellen.
Im Package «org.mabata.model» befinden sich alle Mabata-Models mit den dazugehörigen Interfaces. Diese Models beinhalten jeweils alle Attribute, welche für die Datenbank, wie auch für den Communications Manager benötigt werden. Dies hat den Vorteil, dass mit den Mabata-Models in jeder Software-Architektur-Schicht (von der Datenbank bis zum GUI) gearbeitet werden kann. Für die Kommunikation mit dem Communications Manager wurden eigene Adapter-Models geschrieben, welche nur die Felder beinhalten, die via AXL auch angesprochen werden können.
Um eine verteilte Persistenz zu gewährleisten, müssen die beiden Modeltypen in der Problem Domain miteinander interagieren. Die Interaktion geschieht nicht direkt sondern über den Adapter und die WebServices. Der Adapter erstellt aus allen benötigten Attributen mitsamt den Werten einen XML-String. Dieser XML-String wird vom Adapter zum WebService geschickt, welcher aus dem XML-String ein AXL-Objekt erstellt.

5.1.3.2. Persistenz Management

Das Management für die verteilte Persistenz wird in der Problem-Domain von den `EntityManagern` geführt. Pro Model existiert ein `EntityManager`, der das nötige Know How besitzt, welche Daten in welches System persistiert werden sollen.
Wenn nun ein Model in beiden Systemen gespeichert werden muss, kommt die angesprochene Aufteilung der zwei Modeltypen zum Einsatz. Danach muss dafür gesorgt werden, dass die Transaktionen zur Datenbank und zum Communications Manager konsistent ablaufen.
Eine weitere Aufgabe des Persistenz Managements ist das Protokollieren der wichtigsten Operationen (wie z.B. Add, Delete und Update von `Phone`, `PhoneProfile`, `User` und `DirNumber`), welche von einem Benutzer über Mabata getätigt wurden. Dank diesem Reporting kann später z.B. nachvollzogen werden, welcher Benutzer ein bestimmtes Telefon neu registriert hat.

5.1.3.3. Façade

Um eine möglichst lose Kopplung zwischen dem GUI und der Problem-Domain zu erreichen, wurde das Structural Pattern Façade eingesetzt. Diese Façade bietet dem GUI eine einheitliche Schnittstelle und einen einzigen Einstiegspunkt zur Problem-Domain.
Jeder User-Session wird eine Instanz der Façade zugewiesen, worüber nun alle Funktionalitäten den entsprechenden Persistenz Managern weiterdelegiert werden. Das eingesetzte Instanzenmanagement (Factory-Pattern) sorgt dafür, dass beim Aufruf einer Funktion nur gerade der benötigte Persistenz Manager (`EntityManager`) instanziiert wird, wodurch Ressourcen eingespart werden. Zudem wird bei jedem Zugriff überprüft, ob nicht bereits eine vorhandene Instanz dieses Managers besteht.

5.1.4. Datenbank

Alle Felder welche für die Mandantenfähigkeit von Mabata notwendig sind und nicht im Communications Manager bestehen, werden in der Datenbank von Mabata abgebildet. Um gewisse AXL-Abfragen einzusparen, werden einige Felder in beiden Systemen geführt. Auf dem Datenbankschema sind alle Tabellen inkl. Zwischentabellen mit ihren Attributen abgebildet. Bei der Namensvergabe der Attribute wurde darauf geachtet, dass sie mit den Attributen der Mabata-Models identisch sind. Die jeweiligen Assoziationen zwischen den Tabellen sind ebenfalls ersichtlich.

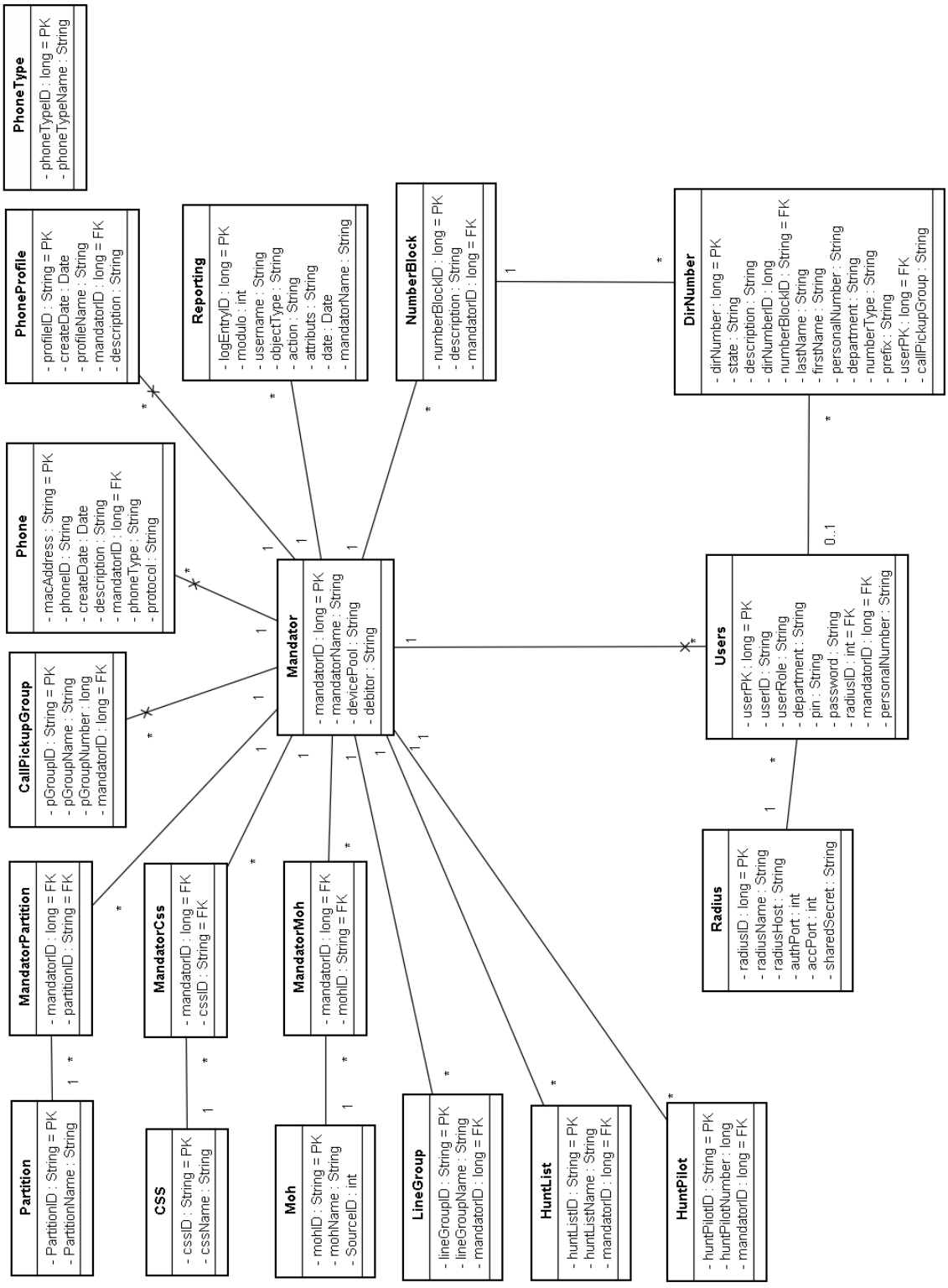


Abbildung 5-8 Mabata Datenbank

5.1.4.1. Hibernate

Data Access Object

Für jedes Mabata-Model, welches persistiert werden soll, besteht ein Data Access Object (DAO), welches die Transaktion zur Datenbank steuert. Alle DAOs sind von einem abstrakten DAO abgeleitet, welches alle Standardoperationen zur Verfügung stellt. Die wichtigste Funktion dieser Klasse ist `getSupportedClass()`, welche dafür sorgt, dass alle diese CRUD-Operationen generisch benutzt werden können. Komplexere SQL-Abfragen wurden in den jeweiligen DAOs implementiert.

Hibernate Mapping

Das Mapping zwischen den Mabata-Models und den Datenbanktabellen wurde jeweils über eine XML-Datei spezifiziert. In einer solchen Hibernate Mapping-Datei wird genau definiert, welche Attribute des Models in welche Spalte der Tabelle persistiert werden sollen.
Gewisse Primary-Keys werden im Hibernate-Mapping über einen Generator automatisch erzeugt. Mit der Einstellung `class=native` wird der Datenbank überlassen, welchen Mechanismus für die Generierung des Primary-Keys benutzt werden soll.

Hibernate Config

Die Hibernate Config ist notwendig, um der `SessionFactory` die Verbindungsattribute wie Treiber, Connection URL, Benutzer und Passwort der Datenbank bekannt zu geben. Zudem müssen alle Hibernate-Mapping-Dateien in dieser Konfiguration registriert sein, um sie benutzen zu können. Bei der Erstellung der `SessionFactory` kommt HBM2DDL zum Zug, welches dafür sorgt, dass das Schema DDL (Data Definition Language) direkt auf die Datenbank exportiert wird.

Communications Manager Schnittstelle

Bei der Communications Manager Schnittstelle handelt es sich um eine AXL-API. Diese API stellt den Mechanismus zur Verfügung, Daten auf dem Communications Manager hinzuzufügen, zu ändern oder zu löschen. Diese Kommunikation basiert auf SOAP-Messages.

5.2. WebServices

Die WebServices sind in den folgenden Designüberlegungen ein wichtiger Bestandteil. Aus diesem Grund wird in diesem Kapitel erläutert, wie WebServices zu erstellen sind und die Benutzerauthentifizierung vonstatten geht.

5.2.1. WebServices publizieren

Ein WebService wird mittels Java Annotations erstellt. Diese Annotations sind direkt in der Klasse, welche den Service erbringt, einzufügen. Dabei sind die Annotations `@WebService` und `@SOAPBindung(style=Style.RPC)` dazu da, um die Klasse als WebService zu deklarieren. In dieser Klasse müssen die Methoden, welche im WebService verfügbar sein sollen, mit der Annotation `@WebMethod` markiert werden.

```
@WebService
@SOAPBindung(style=Style.RPC)
public class WebService {
      @WebMethod
      public String doAnything(){
      …
      }
}
```

Diese `WebService` Klasse kann mit den folgenden Code-Zeilen unter der Adresse «http://beispieladresse.ch:8080/webservice» publiziert werden.

```
WebService service = new WebService();
Endpoint.publish("http://beispieladresse.ch:8080/webservice", \
service);
```

5.2.2. WebService Client erstellen

Um den WebService Client zu erstellen, muss der WebService bereits implementiert und publiziert sein. Aus dem laufenden WebService können die für den Client gebrauchten Service-Klassen generiert werden. Dies geschieht mit Hilfe der Java-Shell:

```
wsimport -keep http://beispieladresse.ch:8080/webservice?wsdl
```

Dabei werden zwei Service-Klassen erstellt (z.B. `Test` und `TestService`), welche in die Clientapplikation eingebunden werden müssen. Mithilfe dieser beiden Klassen kann der Client auf den WebService zugreifen:

```
Test test = new TestService().getTestPort();
test.doSomething();
```

Mithilfe der `TestService` Klasse kann ein Objekt `Test` vom WebService geholt werden. Das `Test` Objekt verhält sich daraufhin wie ein lokales Objekt.

5.2.3. WebService Security

Die WebServices sollen nicht jede Clientverbindung zulassen. Um zu verhindern, dass jeder Client die WebServices Methoden aufrufen kann, wurden verschiedene Möglichkeiten miteinander verglichen.

Die Kommunikation zwischen WebService Client und dem WebService darf in Klartext erfolgen.

5.2.3.1. Authentifizierung mit Benutzername/Passwort

Beim Aufruf des WebServices werden Benutzername und Passwort mit gesendet. Der WebService prüft daraufhin, ob die angegebenen Credentials korrekt sind.

Das Problem der Authentifizierung mit Benutzername und Passwort besteht darin, dass die verschickten Messages verschlüsselt werden müssen, da ansonsten mit einem Sniffer die Anmeldeinformationen herausgelesen werden können.

Bei der Analyse der Verschlüsselung der SOAP-Messages stellt sich schnell heraus, dass ein riesiger Overhead entsteht. Aus einer unverschlüsselten Nachricht von 405 Zeichen entsteht eine verschlüsselte Nachricht mit 4110 Zeichen.

5.2.3.2. Authentifizierung mit Hash-Wert

Da die Verschlüsselung einen sehr grossen Overhead generiert, wurde nach einer unverschlüsselten Lösung gesucht.

Als Ergebnis resultierte ein Algorithmus, welcher von dem zu übertragenen String und vom Benutzernamen mit Passwort einen Hash-Wert bildet. Dieser Hash-Wert wird zusammen mit der Nachricht und dem Benutzernamen übertragen. Da das Passwort nur im Hash-Wert enthalten ist, kann die Übermittlung der Nachricht im Klartext erfolgen.

Der WebService kann diese Nachricht wieder in die drei Teile zerlegen: String, Hash und Benutzername. Mittels des Benutzernamens kann das Passwort ermittelt werden, welches zuvor auf beiden Kommunikationsendpunkten konfiguriert wurde. Über den String, den Benutzernamen und das Passwort kann nun der WebService den Hash berechnen und mit dem übertragenen vergleichen. Falls diese identisch sind, hat der WebService die Gewissheit, dass der Kommunikationspartner das Passwort des angegebenen Benutzernamens kennt, und die Nachricht nicht verändert wurde.

Nachstehendes Diagramm stellt den Algorithmus bildlich dar:

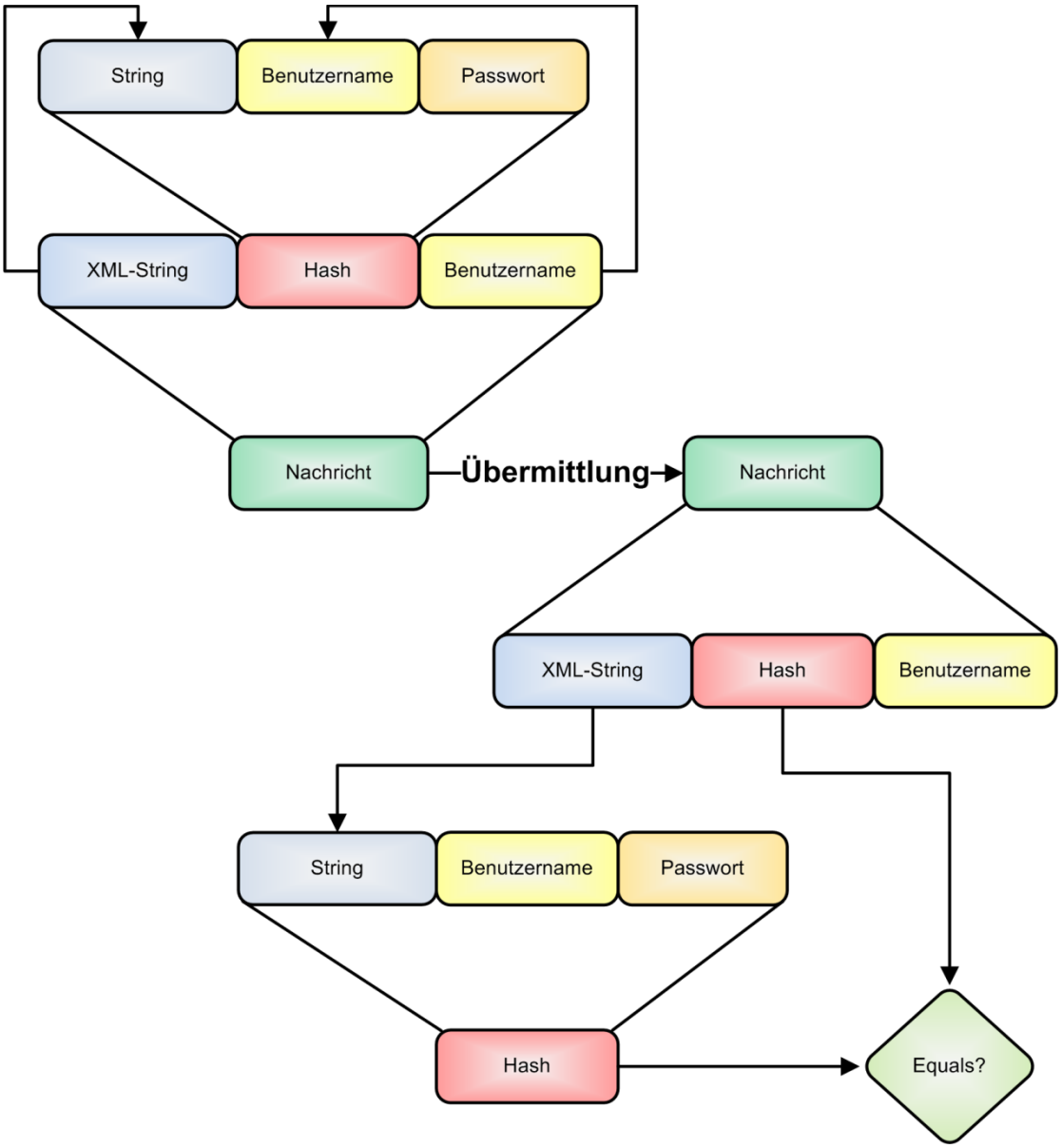


Abbildung 5-9 Authentifizierungsablauf

5.3. Mabata Business Case Vereinfachung

In diesem Kapitel wird beschrieben wie die Business Case in Mabata zusammengefasst und vereinfacht werden können. Daraus ändern sich die Benutzerrollen wie in folgendem Bild dargestellt. (Dieses Bild wurde von einer vorgängigen Studienarbeit übernommen und angepasst)

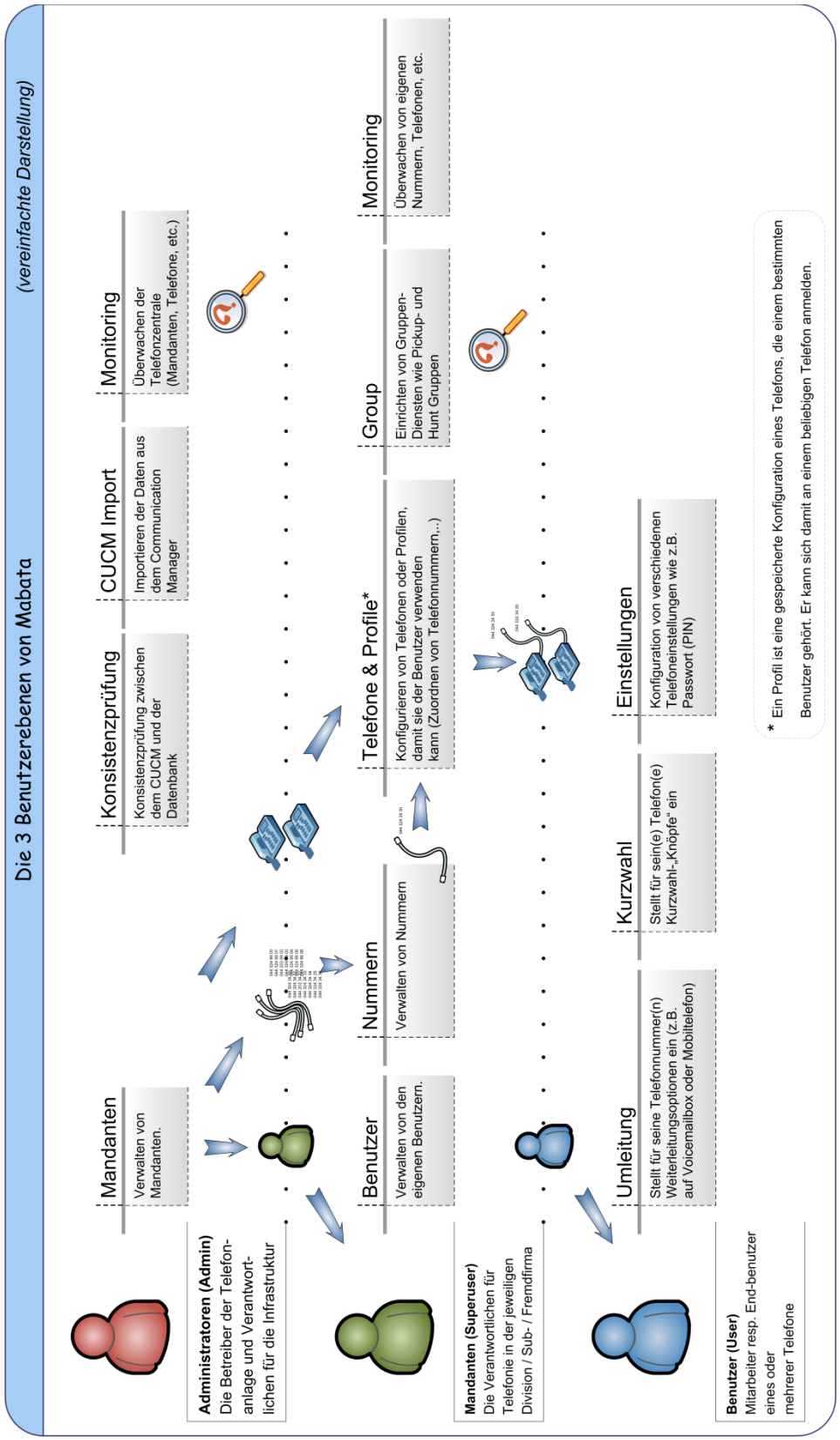


Abbildung 5-10 Benutzerrollen nach der Umstellung

5.3.1. Paper Prototype

Entwurf 1: Benutzer mutieren (Mandator-Ansicht)

In diesem Fenster ist die neue Oberfläche von Mabata ersichtlich. Alle Tätigkeiten die zur Erstellung eines Benutzers gehören sind unter einer einheitlichen Maske zusammengefasst. Dadurch kann die Konfiguration an einem Ort abgehandelt werden. Dies hilft unter anderem dem Benutzer, da dieser nicht mehr wissen muss, wie der Communications Manager zu konfigurieren ist.

Zwischen einem Telefon und einem Profil wird nicht mehr unterschieden. In der Drop-Down Liste, in welcher die Telefone aufgelistet sind, wird neu eine Auswahl «Profil» möglich sein. Danach wird je nach Auswahl ein Telefon oder ein Profil erzeugt. Der Superuser muss jedoch die gleichen Angaben betätigen und merkt von der Unterscheidung nichts.

Damit das Fenster nicht zu gross wird, werden die momentan nicht aktiven Komponenten ausgeblendet. Diese können über die Schaltfläche «Konfigurieren» eingeblendet werden.

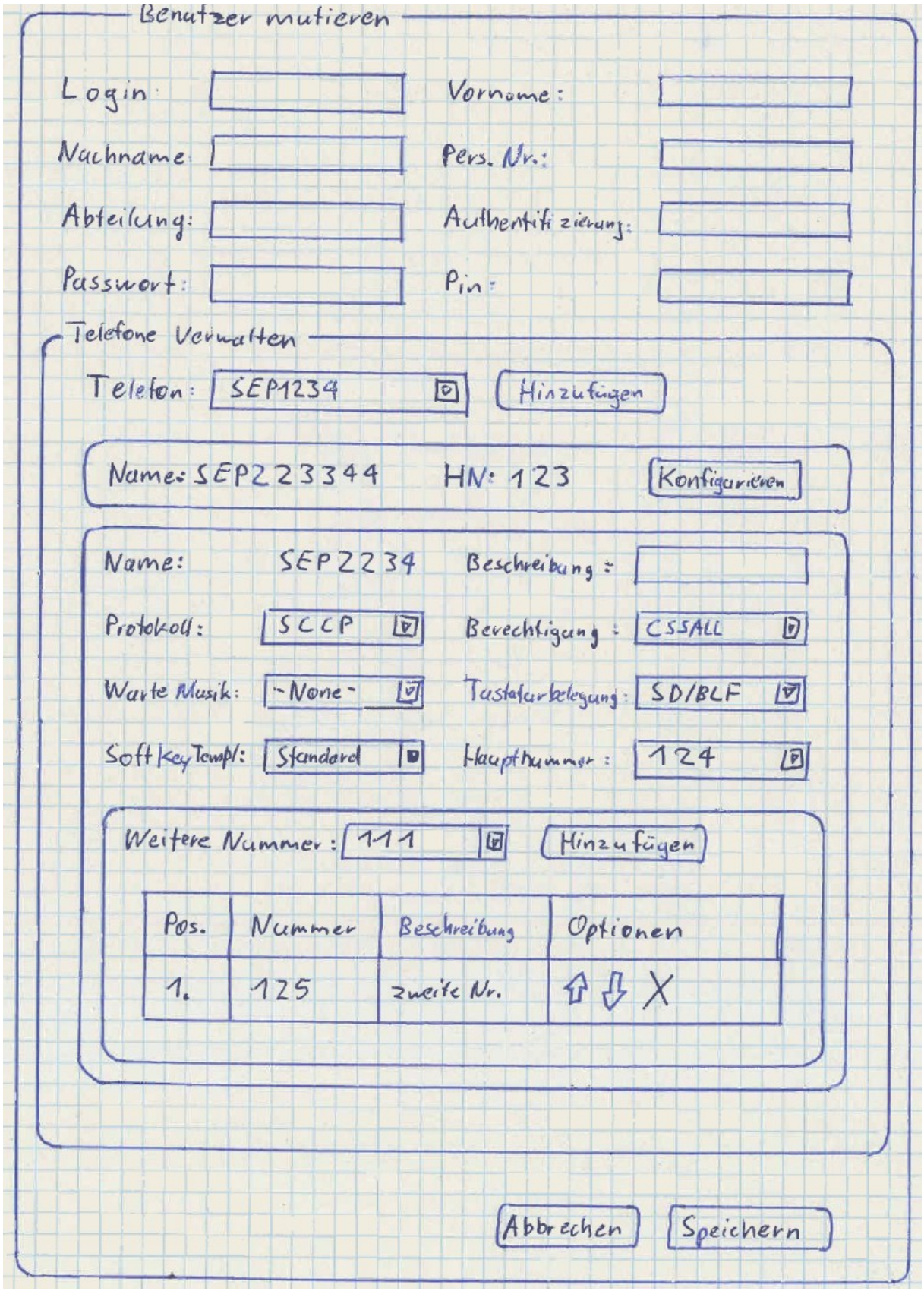


Abbildung 5-11 Paper Prototype Benutzer mutieren

5.3.2. Benutzeroberfläche

5.3.2.1. Administrator-Ansicht

Fenster 1: neuer Mandant erstellen

In diesem Fenster kann ein neuer Mandant erstellt werden. Zudem können Nummernblöcke und Telefone erstellt, bearbeitet, gelöscht sowie dem Mandanten zugewiesen werden.
Die vom Benutzer geänderten Objekte (Nummernblock/Telefon hinzufügen, bearbeiten und löschen) werden erste übernommen, wenn die «Speichern»/«Übernehmen» Schaltfläche in diesem Fenster gedrückt wird.

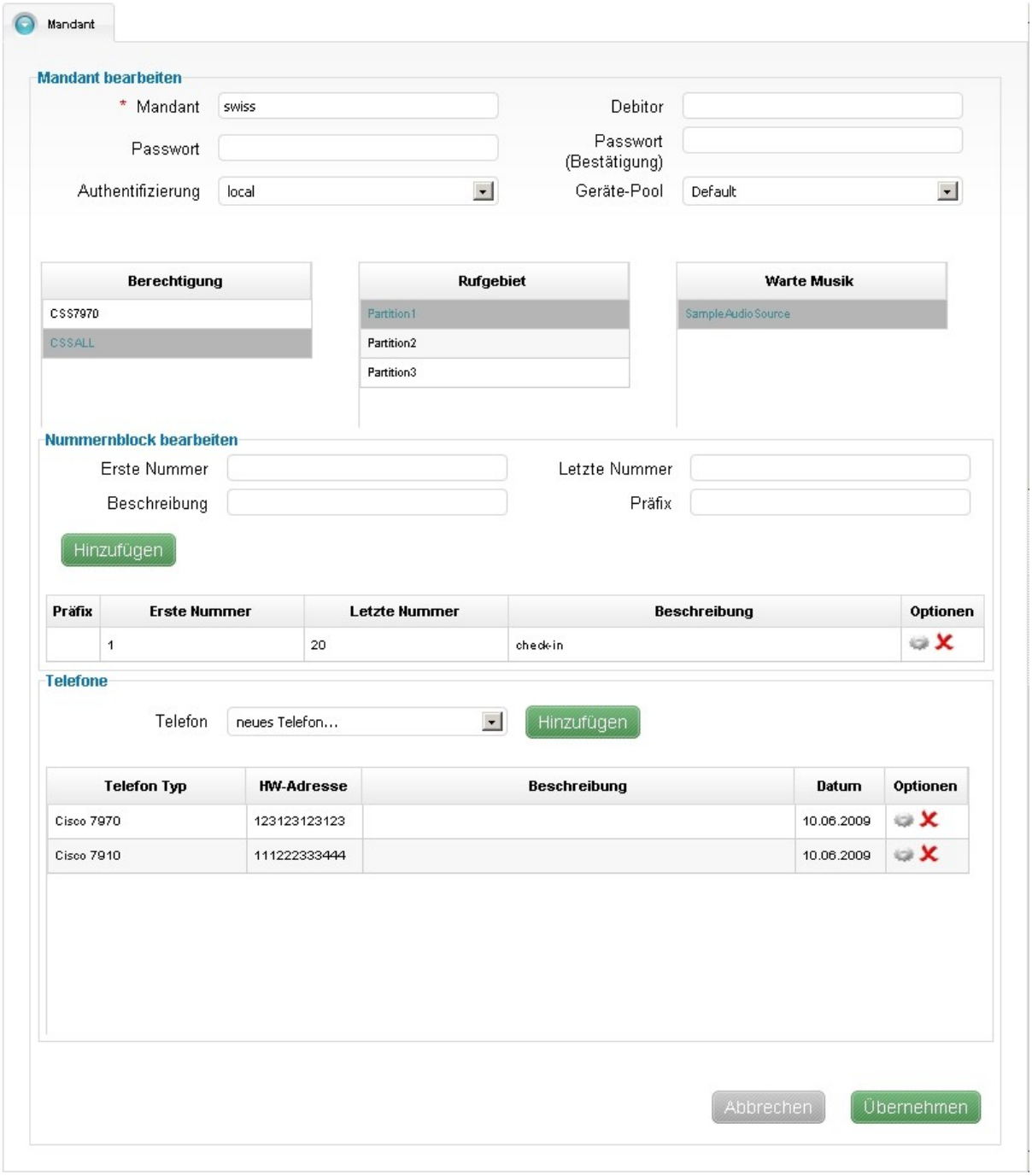


Abbildung 5-12 Benutzeroberfläche neuer Mandant erstellen

Fenster 2: Nummernblock bearbeiten

In dieser Oberfläche kann ein Nummernblock bearbeitet werden. Zudem ist ersichtlich, welche Nummern in diesem Nummernblock aktiv bzw. inaktiv sind. Ein Nummernblock kann nur verkleinert werden, wenn alle zu entfernenden Nummern inaktiv sind.

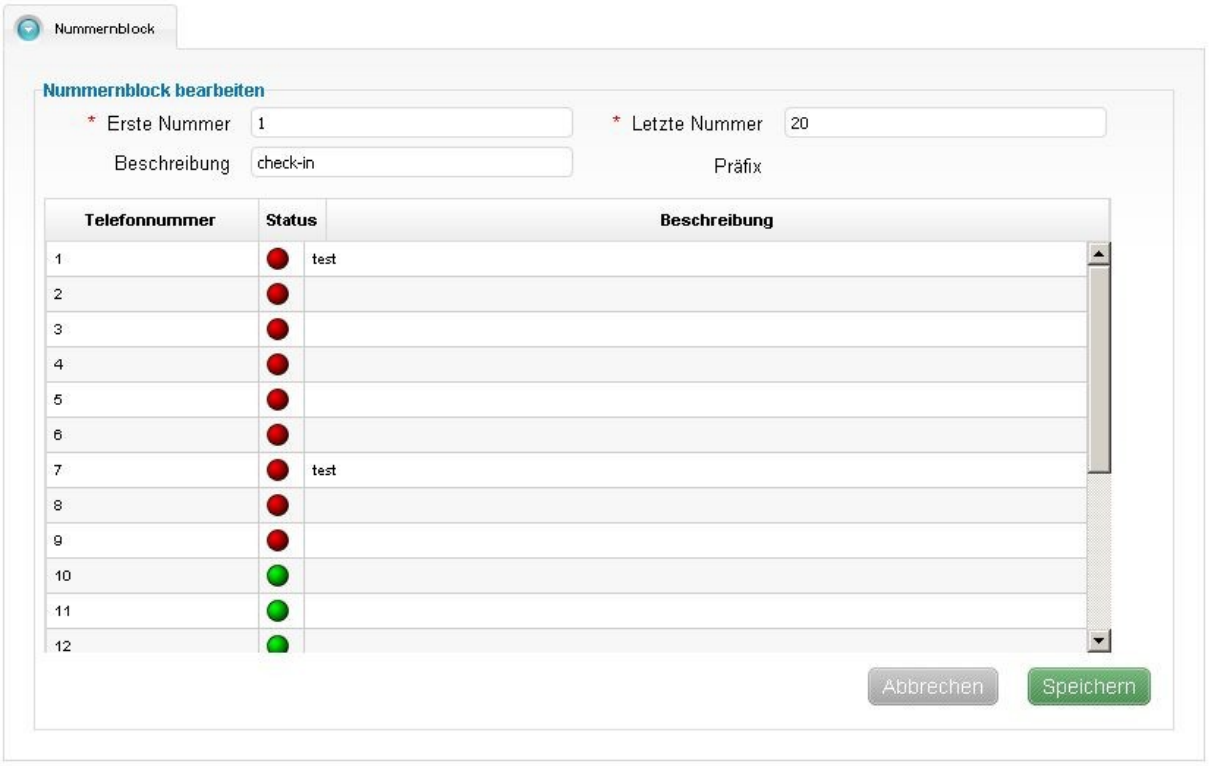


Abbildung 5-13 Benutzeroberfläche Nummernblock bearbeiten

Fenster 3: Telefon hinzufügen

In dieser Oberfläche können neue Telefone erstellt werden. Diese stehen anschliessend in der Mandator-Ansicht zur Verfügung.

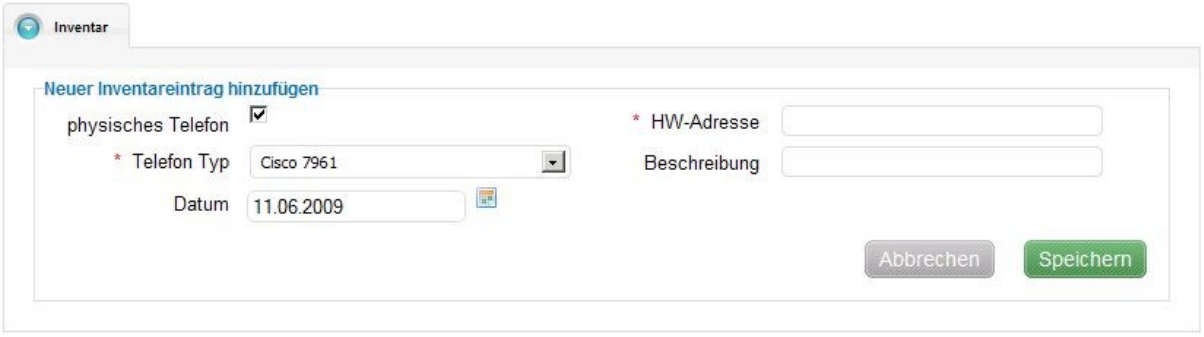


Abbildung 5-14 Benutzeroberfläche Telefon hinzufügen

5.3.2.2. Mandator-Ansicht

Fenster 1: neuer Benutzer erstellen

Der obere Bereich des Fensters wurde wie im Paper Prototype beschrieben umgesetzt. Der untere Bereich (Telefone verwalten) konnte nicht wie im Paper Prototype beschrieben implementiert werden. Da durch das Hinzufügen von neuen Geräten (Telefone/Profile) die Ansicht erweitert werden muss, wurde für diesen Zweck eine Tabelle eingesetzt. Über das Drop-Down Menü «Telefon» können die vom Administrator erfassten Telefone ausgewählt, oder über den Eintrag «neues Profil…» ein neues Profil hinzugefügt werden. Mithilfe der Icons der Tabellenspalte «Optionen» können die Geräte bearbeitet oder von dem Benutzer entfernt werden. Wird ein Gerät über dieses Icon entfernt, wird lediglich die Verknüpfung zwischen Benutzer und Gerät entfernt, das Gerät selbst bleibt in Mabata bestehen.

Die vom Benutzer geänderten Objekte (Telefon/Profil hinzufügen, bearbeiten und löschen) werden erste übernommen, wenn die «Speichern»/«Übernehmen» Schaltfläche in diesem Fenster gedrückt wird.

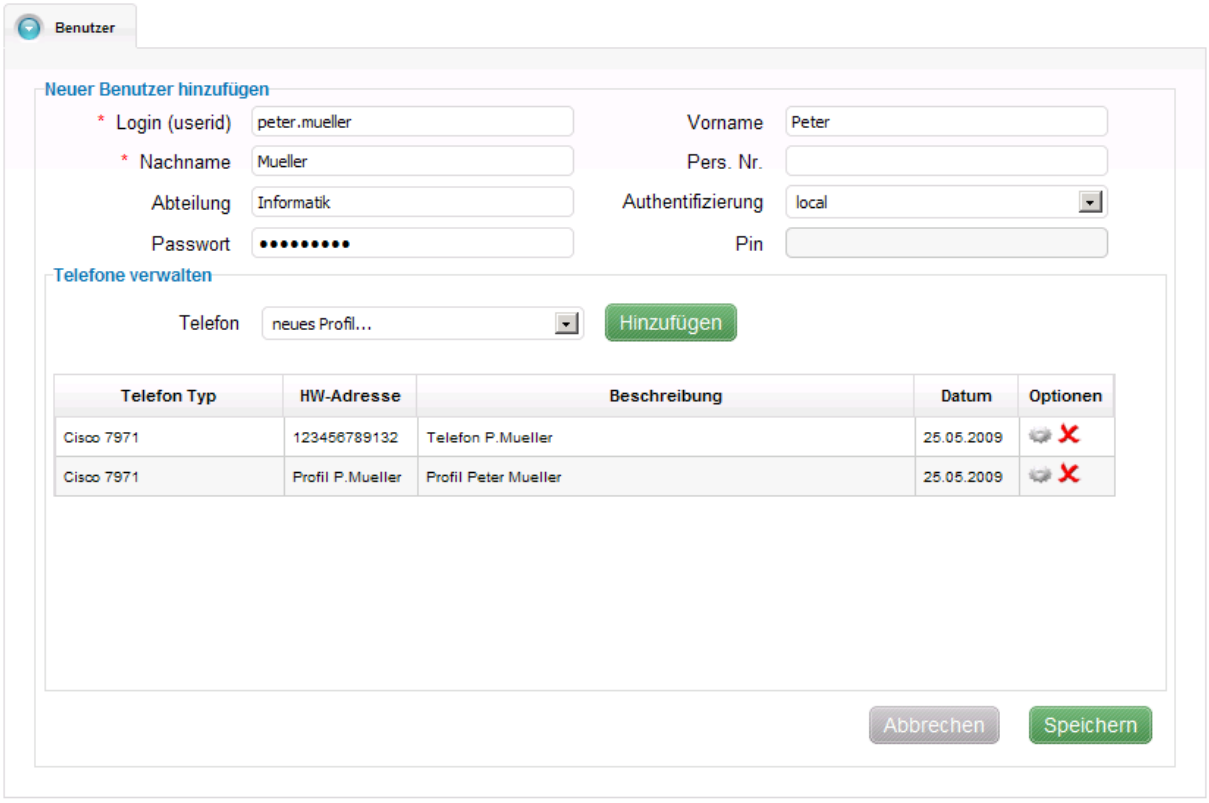


Abbildung 5-15 Benutzeroberfläche Benutzer erstellen

Fenster 2: neues Profil erstellen

In dieser Oberfläche kann ein neues Telefonprofil hinzugefügt werden.

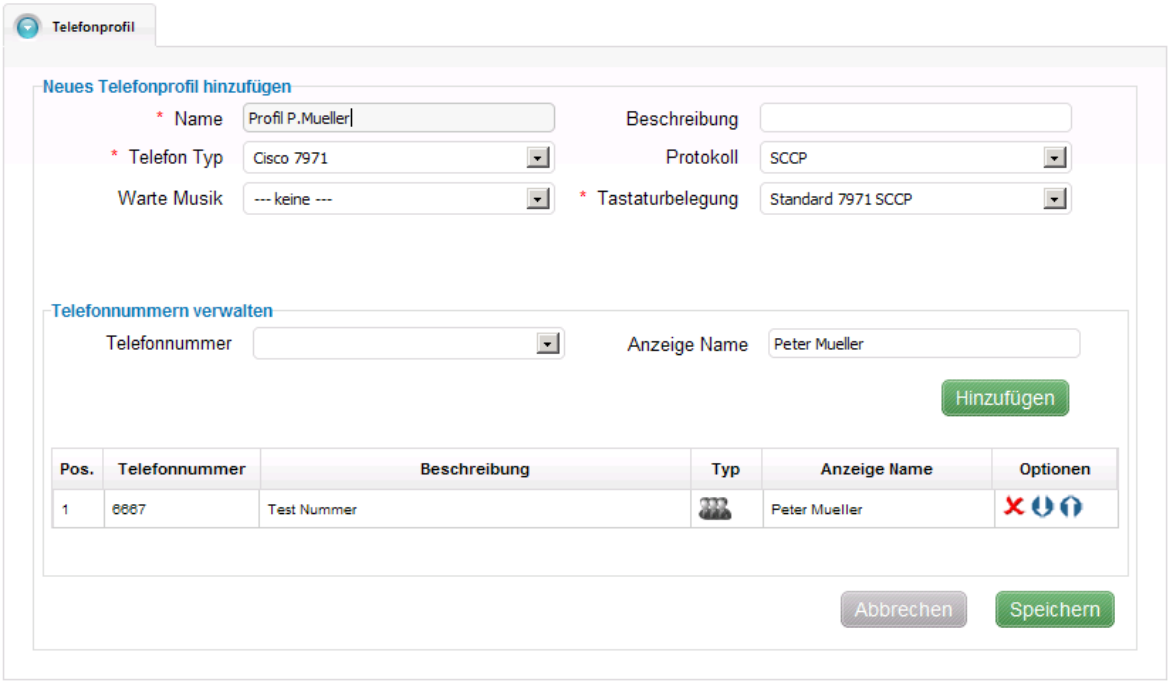


Abbildung 5-16 Benutzeroberfläche Profil erstellen

Fenster 3: neues Telefon hinzufügen

In dieser Oberfläche kann ein neues Telefon hinzugefügt werden.

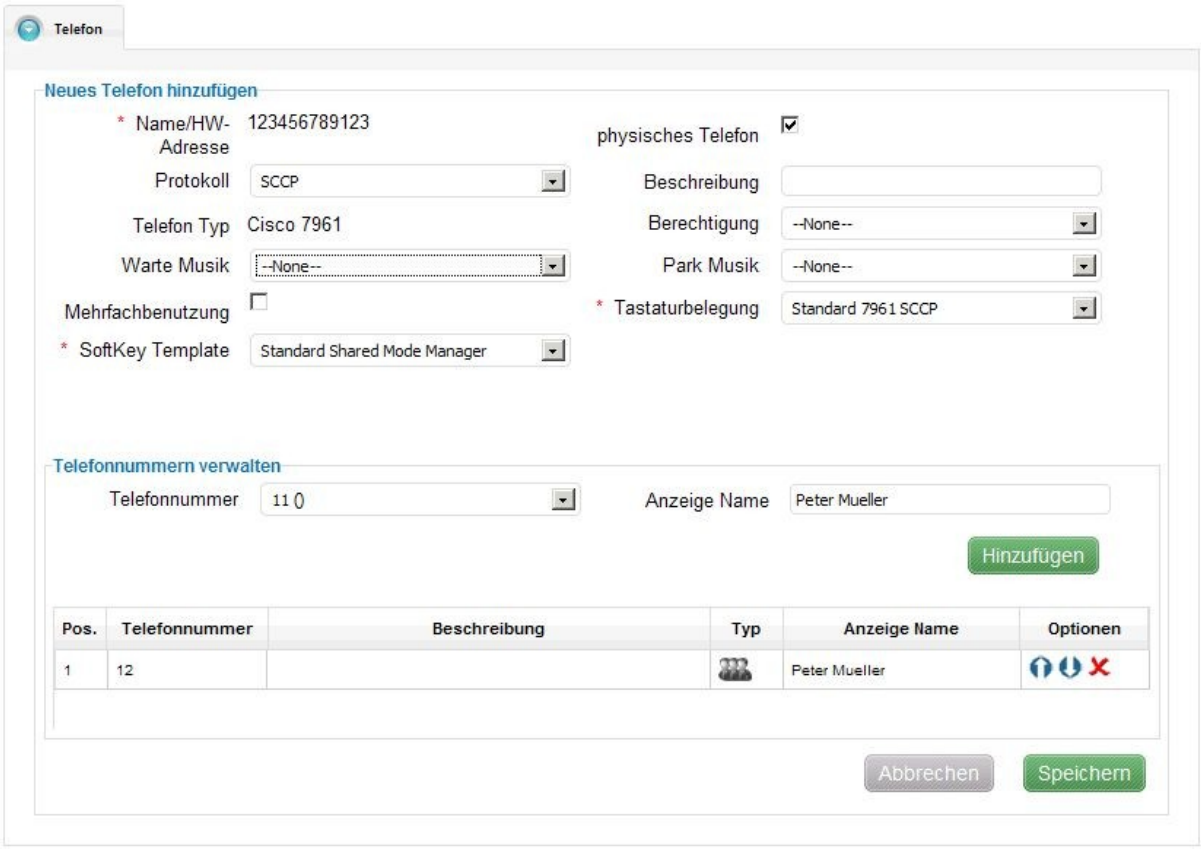


Abbildung 5-17 Benutzeroberfläche Telefon erstellen

5.3.3. Implementierung

5.3.3.1. User Model

Durch die Zusammenfassung der Objekte in der Benutzeroberfläche mussten auch die Models angepasst werden. Das `User` Model wurde um zwei Listen, in welchen die dem Benutzer zugewiesenen Profile und Telefone enthalten sind ergänzt. Dank diesen zwei Listen können alle Beziehungen zwischen Benutzer, Profilen, Telefonen und Nummern in einem Benutzer-Objekt gespeichert werden. Nach dem Konfigurieren muss demnach nur noch ein Benutzer-Objekt an die Middleware gesendet werden.

5.3.3.2. Mandator Model

Wie das `User`-Model wurde auch das `Mandator`-Model um zwei Listen, in welchen die Nummernblöcke sowie die Telefone des Mandanten enthalten sind, erweitert. Dank diesen zwei Listen können alle Beziehungen zwischen Mandant, Nummernblöcke und Telefone in einem Mandanten-Objekt gespeichert werden.

5.3.3.3. GUI Controller

In Mabata werden im GUI durchgeführte Änderungen direkt im entsprechenden Objekt gespeichert. Wird z.B. ein Benutzer geöffnet und dessen Vornamen geändert, wird der neue Vorname direkt ins Objekt geschrieben. Wenn nun die «Abbrechen» Schaltfläche gedrückt wird, wird das Objekt zwar nicht in die Datenbank oder auf den Communications Manager geschrieben, jedoch wird beim nächsten öffnen des Benutzers immer noch der geänderte Vorname angezeigt.

Die Lösung dieses Problems besteht darin, vor jedem öffnen eines Objektes eine Kopie dessen zu erstellen. Die Kopie kann anschliessend geändert werden ohne dass die bisherigen Werte verloren gehen. Erst nach dem Drücken der «Speichern» Schaltfläche werden die Werte aus der Objekt-Kopie in das Original Objekt übertragen.

5.4. Modularisierung

In diesem Kapitel wird beschrieben, wie die Modularisierung von Mabata umgesetzt werden kann.

5.4.1. Benutzeroberfläche

Die Benutzeroberfläche von Mabata ist durch ICEfaces realisiert. ICEfaces verwendet zur Darstellung der Inhalte JSPX-Dateien. Damit neue Inhalte angezeigt werden können, müssen demnach die bestehenden JSPX-Dateien erweitert werden.

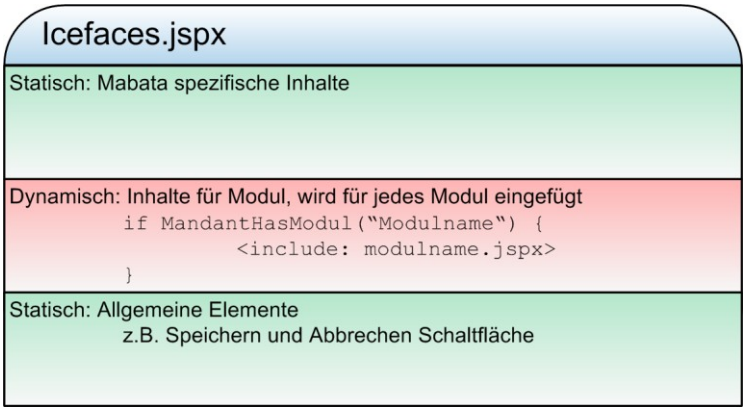


Abbildung 5-18 Schema der Modularisierung

Für jedes Modul wird eine neue JSPX-Datei erstellt, welche die Inhalte der entsprechenden Erweiterung beinhaltet. Damit dieser Inhalt in Mabata angezeigt wird, wird diese Datei in der Hauptanzeige mittels einem Include-Statement eingefügt. Vor jedem Include wird zudem überprüft, ob für den aktuellen Mandanten dieses Modul aktiviert ist. Falls das Modul für diesen Mandanten nicht aktiviert ist, wird es nicht angezeigt.

5.4.1.1. Controller

Damit die in der Anzeige verwendeten Attribute weiterverarbeitet werden können, wird ein Controller für jedes Modul benötigt. Der Controller besteht aus einer Java-Klasse und beinhaltet für jedes Attribut eine entsprechende Variable mit getter und setter Methode. Neben diesen Methoden beinhaltet der Controller weitere Methoden, welche z.B. aufgerufen werden, wenn der Benutzer «Speichern» in der Benutzeroberfläche drückt. Klickt der Benutzer in dem GUI auf Schaltflächen wie z.B. «Speichern», wird die entsprechende Methode aus dem Maincontroller aufgerufen. Dieser leitet den Aufruf anschliessend an alle beteiligten Module weiter. Damit der Maincontroller weiss, welche Controller von der aktuellen Transaktion betroffen sind, melden sich die Controller nach der Initialisierung der Seite bei dem Maincontroller an. Dieser Ablauf ist im folgenden Sequenzdiagramm bildlich dargestellt.

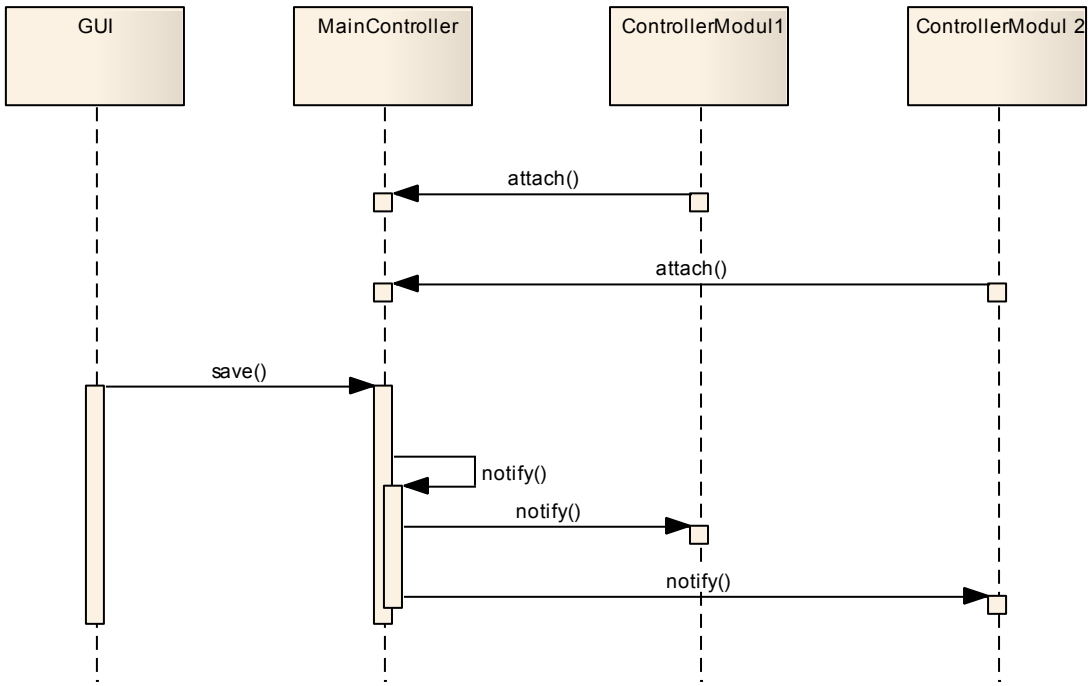


Abbildung 5-19 Ablauf des Controllers

Damit die Controller in ICEfaces verwendet werden können, müssen diese in der Konfigurationsdatei «faces-config.xml» definiert werden.

5.4.2. Datenbank

Für die Zuweisung zwischen Modul und Mandant muss die Datenbank um die Tabellen «Modul» und «Mandantmodule» erweitert werden.

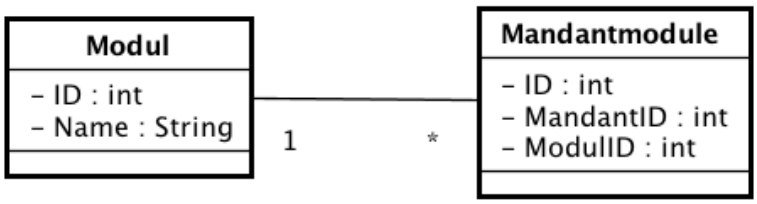


Abbildung 5-20 Datenbank der Modularisierung

Der Eintrag in der «Modul» Tabelle wird von Mabata automatisch für jedes installierte Modul vorgenommen. Die Einträge in der «Mandatmodule» Tabelle werden erstellt, wenn der Administrator die Module für die Mandanten aktiviert. Diese Datensätze werden vom Maincontroller abgerufen, um zu bestimmen ob ein Modul für diesen Mandanten angezeigt wird oder nicht.

5.4.3. Codegenerierung

Damit die soeben beschriebenen Elemente (JSPX-Dateien, Java-Klassen) nicht manuell erstellt werden müssen, wird eine Applikation zur Codegenerierung entwickelt. Über eine XML-Datei kann angegeben werden, welche Attribute und somit welche Felder in der Benutzeroberfläche zur Verfügung stehen sollen.

5.4.3.1. XML Moduldefinition

Die XML-Datei für die Definition neuer Module ist wie folgt aufgebaut:

```
<modulname>
    <fields>
        <field>
            <name>Attributname</name>
            <displayname>Attribut Display Name</displayname>
            <typ>Typ des Attributes</typ>
            <element>GUI Element</element>
            <position>Position des Elementes</position>
        </field>
    </fields>
</modulname>
```

Aus dieser XML-Datei werden die benötigten Elemente erstellt und in Mabata integriert.

5.5. New VoIP Source Architecture

Die new VoIP Source Architecture beschreibt eine Middleware, welche den Zugriff von verschiedenen Applikation auf diverse Zielsysteme regelt. Dabei soll diese Middleware plattform- und programmiersprachenunabhängig sein.

In einem ersten Teil wurde das Design anhand der Anforderungsspezifikation für den Designvorschlag 2 ausgearbeitet. Im späteren Verlauf des Projektes wurde die Mabata Business Case Vereinfachung eingeführt. Durch diese Änderung wird die gesamte Logik in die Middleware verschoben. Diese sind im Kapitel 5.5.7 dokumentiert.

Zuerst wurden zwei verschiedene Lösungsvarianten aufgezeigt, welche nachfolgend einander gegenübergestellt werden. Die bevorzugte Lösungsvariante wurde daraufhin detailliert ausgearbeitet.

5.5.1. WebService pro Applikation

Für jede Applikation wird ein Adapter erstellt, welcher die Methoden der Middleware zur Verfügung stellt. Dieser Adapter kommuniziert mit einem WebService. Dieser WebService wird nur von dieser einen Applikation verwendet. Der WebService bildet die Parameter auf jene der Middleware ab. Die Methoden in der Middleware werden von jeder Applikation verwendet und enthalten die Parameter aller Applikationen. In diesen Methoden werden schlussendlich die Methoden des jeweiligen Frameworks aufgerufen. Dabei werden die benötigten Parameter übergeben.

Untenstehendes Diagramm bildet den ganzen Vorgang anhand der Methode `addUser` ab.

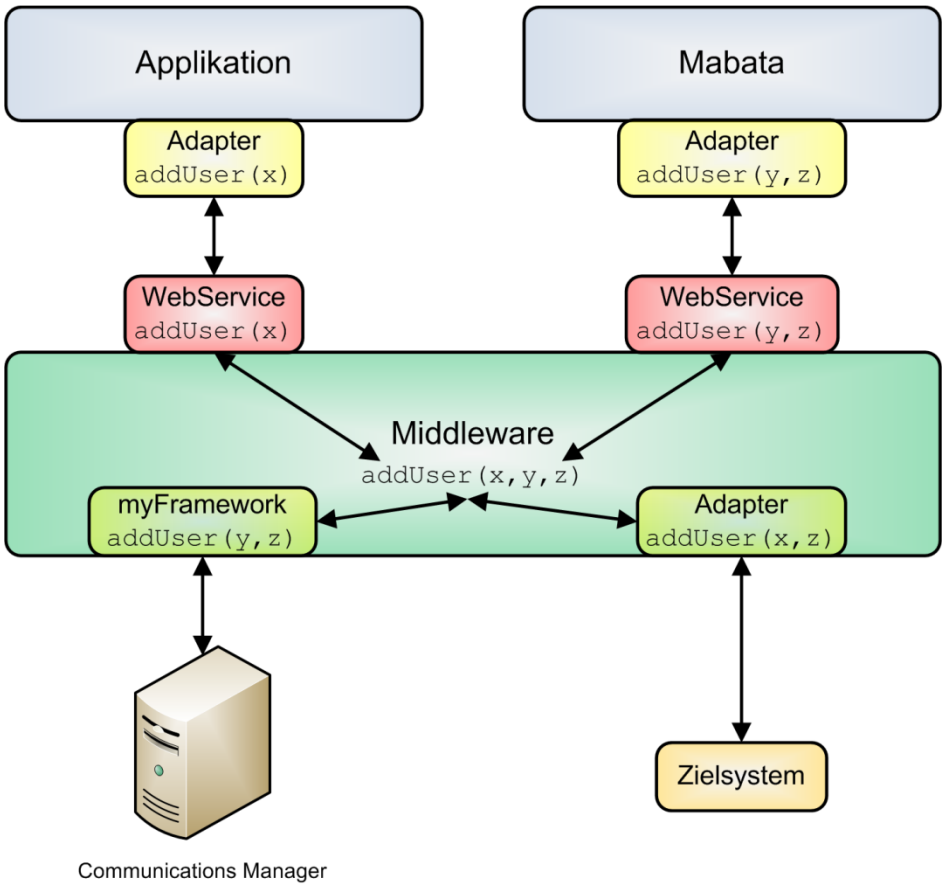


Abbildung 5-21 WebService pro Applikation

5.5.2. WebService pro Funktion

Bei diesem Lösungsvorschlag wird pro Funktion (add, update, remove, get) ein WebService zur Verfügung gestellt. Diese WebServices leiten die Methodenaufrufe an die entsprechenden Manager (z.B. `PhoneManager`) weiter. Damit dies erfolgen kann, erstellen die Applikationsadapter einen Methodenaufruf nach dem Schema:

```
funktion(String)
```

zum Beispiel wie folgt:

```
add("<user><username>ukuratli</username><lastname>Kuratli</lastname></user>")
```

Die Objekte greifen direkt auf die jeweiligen Adapter zu, diese sind im ersten Schritt in die Middleware integriert, können aber in einem späteren Schritt aus dieser extrahiert werden.

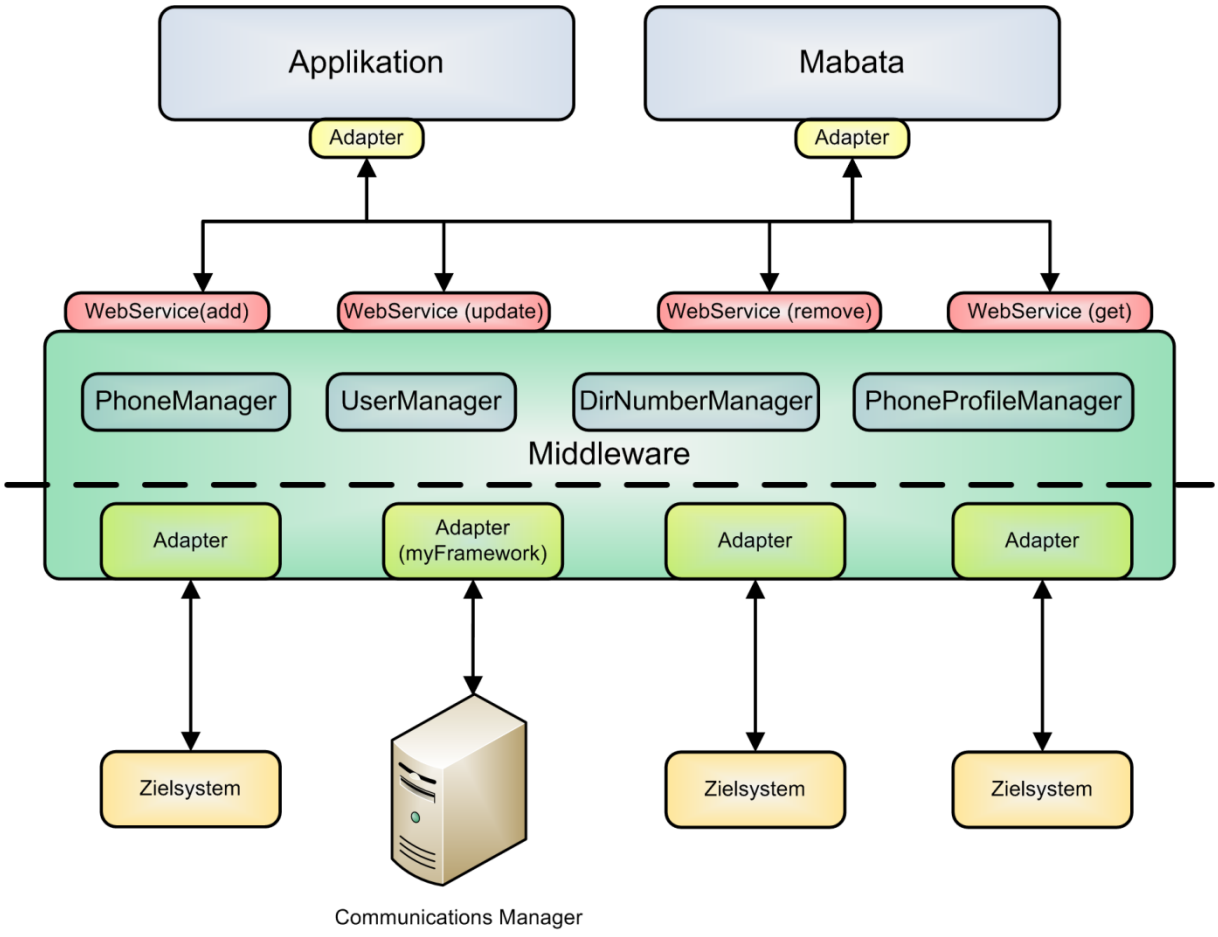


Abbildung 5-22 WebService pro Funktion

5.5.2.1. Beispielablauf

Das Quellsystem Mabata möchte einen User hinzufügen und ruft die Methode `addUser(User)` auf. Der Adapter generiert daraus einen neuen Methodenaufruf `add(String)`. Diesen Aufruf leitet der Adapter an den WebService (`add`) weiter. Dieser WebService untersucht anhand des Root-Tags welches Objekt angesprochen werden soll und leitet es an dieses weiter.

Das User Objekt generiert anschliessend aus dem übermittelten XML-String ein für das Framework benötigtes AXL-Objekt und übergibt dies an das Framework. Das myFramework generiert daraus eine SOAP-Message und leitet diese an den Communications Manager weiter.

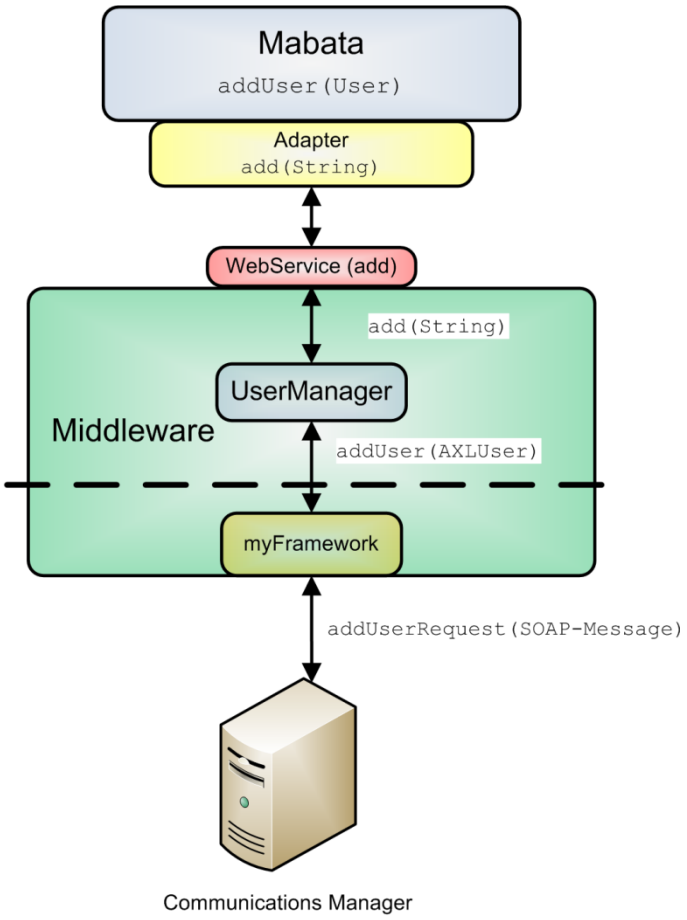


Abbildung 5-23 Beispielablauf Kommunikation über Middleware

5.5.3. Vergleich der Lösungen

Vergleich der Lösungen	WebService pro Applikation	WebService pro Funktion
Anzahl benötigter WebServices	Pro Applikation ein WebService	Pro Funktion ein WebService
Typenprüfung	Möglich	Nicht möglich
Aufwand der Adapterprogrammierung	Klein Nur Methodenaufruf weiterleiten	Gross Parsing der Attribute welche übergeben werden
Skalierbarkeit neue Applikation	Neuer WebService muss erstellt werden	Keine Änderungen nötig
Skalierbarkeit neues Zielsystem	Änderungen in der Middleware und neuer Adapter erstellen für die Anbindung des Zielsystems	Änderungen in der Middleware und neuer Adapter erstellen für die Anbindung des Zielsystems
Skalierbarkeit Änderungen Methodenparameter	Durchgehend an neue Attribute anpassen	Änderungen nur in der Middleware
Performance	Keine zusätzliche Rechenzeit nötig, reine Weiterleitung	Rechenzeit für Parsing nötig
Komplexität	Gering, keine Funktionalität in Adapter und WebService, reine Weiterleitung	Hoch, WebService muss entscheiden welche Methode aufgerufen wird. Applikationsadapter benötigt ein generisches Parsing

Tabelle 5-1 Vergleich WebService pro Funktion/Applikation

5.5.3.1. Fazit

Die zum Entwickeln einfachere Lösungsvariante ist die WebService pro Applikation Architektur. Der WebService und die Adapter müssen nur mit der jeweiligen Applikation kompatibel sein. Aus diesem Grund reicht ein minimales Wissen über die WebServices Technologie aus, um eine neue Applikation anzubinden.

Im Vergleich zu dieser ersten Lösung ist die Entwicklung der WebService pro Funktion Architektur komplexer, da die Zugriffe auf die WebServices von allen Applikation aus erfolgen. Zudem muss ein Mechanismus entwickelt werden, welcher die verschiedenen Parameter in den Methodenaufrufen auf eine Methode mit festen Parametern abbildet. Allerdings ist der Unterhaltsaufwand der zweiten Lösung viel geringer, da bei einer Schnittstellenänderung nur die Middleware angepasst werden muss. Bei der Architektur mit einem WebService pro Applikation müssten zusätzlich zur Middleware alle WebServices angepasst werden. Aus diesem Grund lohnt sich ein grösserer Aufwand bei der Entwicklung, da später ein viel geringerer Wartungsaufwand entsteht.

Aus diesen Gründen wurde beschlossen, die Variante WebServices pro Funktion zu realisieren.

5.5.4. Adapter

Ein Adapter wird benötigt, um die Aufrufe der Applikation an einen WebService umzuleiten.

Die Hauptaufgabe des Adapters besteht darin, eine Verbindung zum richtigen WebService herzustellen sowie den XML-String für die Middleware aus dem Adapter-Model zu generieren.

Hierzu wird der XML-String als Methodenparameter übertragen. Anhand des Root-Tags in dem XML-String kann der WebService entscheiden, wohin der Methodenaufruf weitergeleitet werden muss.

5.5.5. WebService Schnittstellen

Die Middleware stellt ihre Dienste per WebServices zur Verfügung. Dabei wird nach Funktionalität (`add`, `update`, `remove`, `get`) unterschieden.

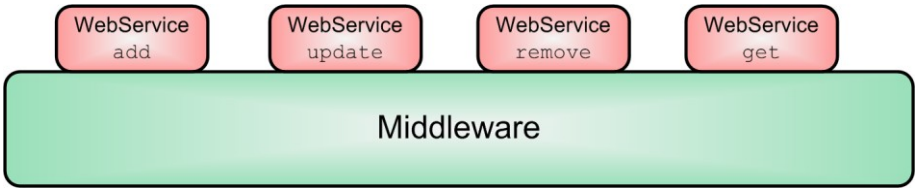


Abbildung 5-24 Middleware mit WebServices

Die Schnittstelle, welche der WebService zur Verfügung stellt, sieht folgendermassen aus:

```
funktion(String)
```

Die Attribute stehen dabei für:

funktion	Funktionsname der vom WebService angebotenen Funktion. Ein WebService bietet jeweils nur eine Funktion an (`add`, `update`, `remove` oder `get`).
String	Enthält einen XML-String in welchem alle Parameter enthalten sind.

Anhand des XML-Strings überprüft der WebService, für welchen Objekttyp die Funktion durchgeführt werden soll und gibt den Aufruf anschliessend an die entsprechende Methode der Middleware weiter. Z.B. Hinzufügen eines Benutzers: Der Adapter der Applikation ruft den WebService `add` auf, welchem er einen XML-String mit den benötigten Parametern übergibt. Der WebService erkennt anhand des Root-Tags <user> im XML-String, dass ein neuer Benutzer erstellt werden muss und ruft die entsprechende Methode der Middleware auf.

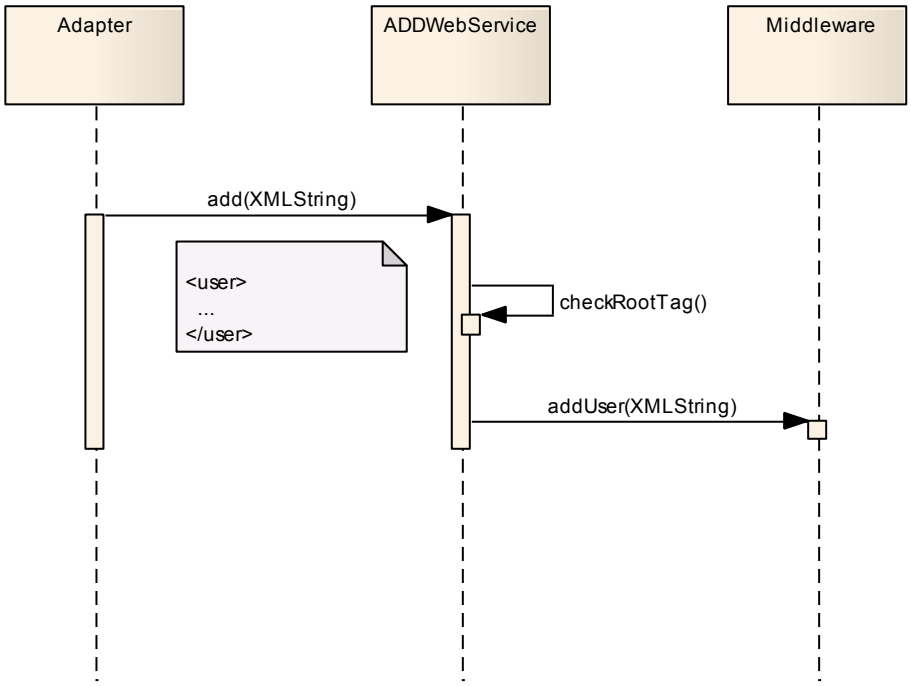


Abbildung 5-25 Ablauf Adapter zu Middleware

5.5.6. Middleware

Die Middleware beinhaltet pro Objekt eine Anlaufstelle. Der WebService untersucht welches Objekt angesprochen werden soll und leitet den Aufruf direkt weiter. In einem ersten Schritt werden die Ziel Frameworks noch direkt angesprochen. Sie sind daher in die Middleware integriert. Diese Integrierung könnte zu einem späteren Zeitpunkt aufgehoben werden, in dem die Frameworks ausgelagert und über WebServices angesprochen werden. Dadurch könnten mehrere Zielsysteme angesprochen werden.

5.5.6.1. Nebenläufigkeit

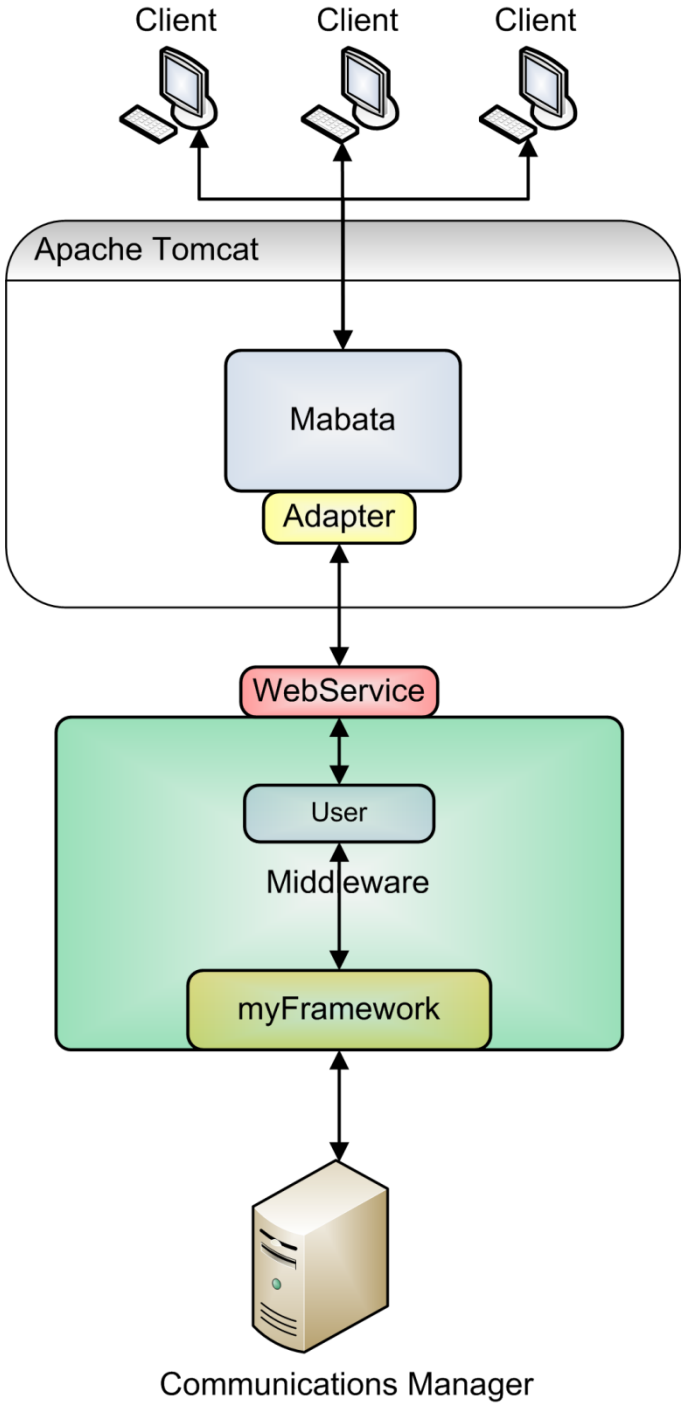


Abbildung 5-26 Nebenläufigkeit

Die parallelen Zugriffe der Clients werden durch den Apache Tomcat Server geregelt. Demnach muss in Mabata nicht auf die Nebenläufigkeit geachtet werden. Die Zugriffe vom Adapter zum WebService werden synchron vom Tomcat Server durchgeführt. Aus diesem Grund erfolgen keine parallelen Zugriffe auf die Middleware. Da in dieser Arbeit nur eine Applikation über WebServices angebunden wird, treten keine gleichzeitigen Zugriffe auf die Middleware auf. Werden in Zukunft weitere Applikationen mittels den WebServices angebunden, müssen die WebServices erweitert werden damit keine Nebenläufigkeiten entstehen können.

5.5.6.2. Plattform- und programmiersprachenunabhängig

Die Plattformunabhängigkeit wird durch die Verwendung von Java als Programmiersprache sichergestellt. Die Programmiersprachenunabhängigkeit bieten die WebServices.

5.5.6.3. Methodenbeschreibungen

Folgende Methoden werden den WebServices durch die Middleware zur Verfügung gestellt. Die WebServices erhalten die Aufrufe von den Adaptern und leiten diese an die unten definierten Methoden der Middleware weiter.

Methode	Beschreibung
getUser(String)	Liefert den Benutzer inkl. Telefone, Profile und Nummern
addUser(String)	Fügt einen Benutzer inkl. Telefone, Profile und Nummern hinzu
deleteUser(String)	Löscht einen Benutzer
updateUser(String)	Aktualisiert ein Benutzer inkl. Telefone, Profile und Nummern
portUser(String)	Portiert ein Benutzer
getGroup(String)	Liefert die Attribute einer Gruppe (Pickup, Hunt)
addGroup(String)	Fügt eine neue Gruppe hinzu
deleteGroup(String)	Löscht eine Gruppe (Pickup, Hunt)
updateGroup(String)	Aktualisiert eine Gruppe
getEnvironment(String)	Gibt systemweite Objekte wie CSS, Partition, MOH zurück
getImportData(String)	Liefert alle Objekte des CUCM welche für den Import benötigt werden
getDirNumber(String)	Liefert eine Nummer zurück. Wird gebraucht, da eine Nummer unabhängig von einem Telefon oder Profil existieren kann
addDirNumber(String)	Fügt auf dem Communications Manager eine neue Nummer hinzu
deleteDirNumber(String)	Löscht eine Nummer auf dem Communications Manager
updateDirNumber(String)	Aktualisiert eine Nummer
getPhone(String)	Liefert ein Phone Objekt zurück. Wird gebraucht, da ein Telefon nicht immer einem Benutzer zugewiesen sein muss
getProfile(String)	Liefert ein Profil Objekt zurück. Wird gebraucht, da ein Profil nicht immer einem Benutzer zugewiesen sein muss

Tabelle 5-2 Methodenbeschreibung

Die Methoden werden in den folgenden Kapiteln genauer spezifiziert. Zudem sind die Parameter sowie Rückgabewerte im JavaDoc dokumentiert.

getUser

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<user>
  <userId>String</userId>
</user>
```

Im `userId`-Tag kann entweder ein Benutzername – um einen definierten Benutzer zu bekommen – oder ein «*» für alle Benutzer angegeben werden.

Die Rückgabe der Middleware sieht wie folgt aus:

```
<users>
  <user>
    <firstName>String</firstName>
    <lastName>String</lastName>
    <userId>int</userId>
    <department>String</department>
    <phoneprofiles>
      <profileName>String</profileName>
      <profileName>…</profileName>
    </phoneprofiles>
    <associatedDevices>
      <device>String</device>
      <device>…</device>
    </associatedDevices>
    <phones>
      <phone>
        <name>String</name>
        <description>String</description>
        <product>String</product>
        <model>String</model>
        <MAC>String</MAC>
        <class>String</class>
        <protocol>String</protocol>
        <protocolSide>String</protocolSide>
        <devicePoolName>String</devicePoolName>
        <securityprofileName>String</securityProfileName>
        <networkHoldMOHAudioSourceId>int</networkHoldMOHAudioSourceId>
        <userHoldMOHAudioSourceId>int</userHoldMOHAudioSourceId>
        <callingSearchSpaceName>String</callingSearchSpaceName>
        <enableExtensionMobility>String</enableExtensionMobility>
        <servicesURL>String</servicesURL>
        <ownerUserId>int</ownerUserId>
        <defaultProfileName>String</defaultProfileName>
        <lines>
          <line index="">
            <dirn uuid="">int</dirn>
            <pattern>int</pattern>
          </line>
          <line index="">
            …
          </line>
        </lines>
      </phone>
      <phone>
        …
      </phone>
    </phones>
```

```
    <profiles>
      <profile>
        <name>String</name>
        <description>String</description>
      <product>String</product>
      <model>String</model>
        <class>String</class>
        <protocol>String</protocol>
        <protocolSide>String</protocolSide>
        <devicePoolName>String</devicePoolName>
        <userHoldMOHAudioSourceId>int</userHoldMOHAudioSourceId>
        <callingSearchSpaceName>String</callingSearchSpaceName>
        <loginUserId>int</loginUserId>
        <phoneTemplate uuid="">String</phoneTemplate>
        <lines>
          <line index="">
            <dirn uuid="">int</dirn>
            <pattern>int</pattern>
          </line>
          <line index="">
            …
          </line>
        </lines>
        <services>
          <service>
            <telecasterService uuid="">String</telecasterService>
            <name>String</name>
            <URL>String</URL>
          <service>
          <service>
            …
          </service>
        </services>
      </profile>
      <profile>
        …
      </profile>
    </profiles>
  </user>
</users>
```

addUser

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<user>
  Gleich wie User-Tag in getUser Antwort
  <action>add|update|delete</action>
</user>
```

Zusätzlich zur `getUser` Antwort wird ein `action`-Tag mitgesendet. Mittels diesem Tag wird definiert, was für eine Aktion die Middleware mit diesem Objekt durchführen muss. Ohne diesen Tag wäre es für die Middleware nicht möglich, alle Änderungen auf dem Zielsystem durchzuführen. Wird z.B. ein Telefon aus einem Benutzer entfernt, muss die Middleware das entfernte Telefon-Objekt kennen, da ansonsten kein Update, in welchem das Telefon aus dem Benutzer entfernt wird, ausgeführt werden kann.

Die Rückgabe der Middleware sieht wie folgt aus:

```
<user>
  <uuid>String</uuid>
  <phones>
    <phone mac="">
      <uuid>String</uuid>
      <lines>
        <line number="">
          <uuid>String</uuid>
        </line>
        <line number="">
          …
        </line>
      </lines>
    </phone>
    <phone mac="">
      …
    </phone>
  </phones>
  <profiles>
    <profile name="">
      <uuid>String</uuid>
      <lines>
        <line number="">
          <uuid>String</uuid>
        </line>
        <line number="">
          …
        </line>
      </lines>
    </profile>
    <profile name="">
      …
    </profile>
  </profiles>
</user>
```

deleteUser

Als Parameter an die Middleware wird ein Benutzername, welcher gelöscht werden soll, mitgegeben.

```
<user>
   <userId>String</userId>
</user>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<user>
   <userId>String</userId>
</user>
```

updateUser

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<user>
  Gleich wie addUser
</user>
```

In der Rückgabe der Middleware ist ersichtlich, welcher Task auf welchem Objekt ausgeführt wurde:

```
<user>
   <uuid>String</uuid>
   <phones>
      <phone mac="">
         <action>String</action>
         <uuid>String</uuid>
         <lines>
            <line number="">
            <action>String</action>
               <uuid>String</uuid>
            </line>
            <line number="">
               …
            </line>
         </lines>
      </phone>
      <phone mac="">
         …
      </phone>
   </phones>
   <profiles>
      <profile name="">
         <action>String</action>
         <uuid>String</uuid>
         <lines>
            <line number="">
            <action>String</action>
               <uuid>String</uuid>
            </line>
            <line number="">
               …
            </line>
         </lines>
      </profile>
      <profile name="">
         …
      </profile>
   </profiles>
</user>
```

portUser

Der XML-String für die Parameterübergabe ist genau gleich aufgebaut, wie der des `addUser`. Dabei enthält das User-Objekt alle Telefone und Profile mitsamt den Nummern, welche der User nach der Portierung behält.

Auch die Rückgabe ist mit der des `addUser` identisch. Die `UUID` des Communications Managers bestätigt die erfolgreiche Portierung der einzelnen Objekte.

getGroup

Anhand des Root-Tags wird unterschieden, um welche Gruppenfunktion es sich handelt.

Die Parameterübergabe an die Middleware für Pickup Gruppen ist wie folgt spezifiziert:

```
<callPickupGroup>
  <uuid>String</uuid>
  <name>String</name>
</callPickupGroup>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<callPickupGroup>
   <uuid>String</uuid>
   <pattern>String</pattern>
   <name>String</name>
   <description>String</description>
   <usage>String</usage>
   <pickupNotification>String</pickupNotification>
   <routePartitionName>String</routePartitionName>
   <lines>
      <line number="">
         <uuid>String</uuid>
      </line>
      <line number="">
         …
      </line>
   </lines>
</callPickupGroup>
```

Die Parameterübergabe an Middleware für Hunt Gruppen ist wie folgt spezifiziert:

```
<huntGroup>
  <groupName>String</groupName >
</huntGroup>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<huntGroup>
   <uuid></uuid>
   <distribution>String</distribution>
   <distributionTimeOut>int</distributionTimeOut>
   <description>String</description>
   <lines>
      <line number="">
         <uuid>String</uuid>
      </line>
      <line number="">
         …
      </line>
   </lines>
   <pilots>
      <pilot>
         <pattern>String</pattern>
         <description>String</description>
         <usage>String</usage>
         <routePartitionName>String</routePartitionName>
         <huntListName>String</huntListName>
      </pilot>
   </pilots>
</huntGroup>
```

addGroup

Anhand des Root-Tags wird unterschieden, um welche Gruppenfunktion es sich handelt.

Die Parameterübergabe an die Middleware für Pickup Gruppen ist wie folgt spezifiziert:

```
<callPickupGroup>
   <pattern>String</pattern>
   <name>String</name>
   <description>String</description>
   <usage>String</usage>
   <pickupNotification>String</pickupNotification>
   <routePartitionName>String</routePartitionName>
</callPickupGroup>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<callPickupGroup>
   <uuid>String</uuid>
</callPickupGroup>
```

Die Parameterübergabe an die Middleware für Hunt Gruppen ist wie folgt spezifiziert:

```
<huntGroup>
   <name>String</name>
   <description>String</description>
   <callManagerGroupName>String</callManagerGroupName>
   <routeListEnabled>String</routeListEnabled>
   <members>
      <member>
         <lineGroupName>String</lineGroupName>
         <selectionOrder>int</selectionOrder>
      </member>
   </members>
</huntGroup>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<huntGroup>
   <uuid>String</uuid>
</huntGroup>
```

deleteGroup

Anhand des Root-Tags wird unterschieden, um welche Gruppenfunktion es sich handelt.

Die Parameterübergabe an die Middleware für Pickup Gruppen ist wie folgt spezifiziert:

```
<callPickupGroup>
   <uuid>String</uuid>
</callPickupGroup>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<callPickupGroup>String</callPickupGroup>
```

Die Parameterübergabe an die Middleware für Hunt Gruppen ist wie folgt spezifiziert:

```
<huntGroup>
   <name>String</name>
</huntGroup>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<huntGroup>String</huntGroup>
```

updateGroup

Anhand des Root-Tags wird unterschieden, um welche Gruppenfunktion es sich handelt.

Die Parameterübergabe an die Middleware für Pickup Gruppen ist wie folgt spezifiziert:

```
<callPickupGroup>
   <uuid>String</uuid>
   <newPattern>String</newPattern>
   <newName>String</NewName>
   <description>String</description>
   <usage>String</usage>
   <pickupNotification>String</pickupNotification>
   <newRoutePartitionName>String</newRoutePartitionName>
</callPickupGroup>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<callPickupGroup>
   <uuid>String</uuid>
</callPickupGroup>
```

Die Parameterübergabe an die Middleware für Hunt Gruppen ist wie folgt spezifiziert:

```
<huntGroup>
   <uuid>String</uuid>
   <NewName>String</NewName>
   <description>String</description>
   <callManagerGroupName>String</callManagerGroupName>
   <routeListEnabled>String</routeListEnabled>
   <members>
      <member>
         <lineGroupName>String</lineGroupName>
         <selectionOrder>int</selectionOrder>
      </member>
   </members>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<huntGroup>String</huntGroup>
```

getEnvironment

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<environment>
   <css />
   <partition />
   <musicOnHold />
   <devicePool />
   <phoneButtonTemplates />
</environment>
```

Die Tags im `<environment>` sind fakultativ. Falls der Tag vorhanden ist, wird eine Liste mit Objekten zurückgegeben.

Die Rückgabe der Middleware enthält alle gewünschten Objekte:

```
<environment>
   <csss>
      <css>
         <uuid>String</uuid>
         <name>String</name>
      </css>
      <css>
         …
      </css>
   </csss>
   <partitions>
        <partition>
         <uuid>String</uuid>
         <name>String</name>
      </partition>
      <partition>
         …
      </partition>
   </partitions>
   <musicOnHolds>
      <musicOnHold>
         <uuid>String</uuid>
         <name>String</name>
         <sourceId>int</sourceId>
      </musicOnHold>
      <musicOnHold>
         …
      </musicOnHold>
   </musicOnHolds>
   <devicePools>
      <devicePool>
         <uuid>String</uuid>
         <name>String</name>
         <softkeyTemplate>String</ softkeyTemplate >
      </devicePool>
      <devicePool>
         …
      </devicePool>
   </devicePools>
   <phoneButtonTemplates>
      <phoneButtonTemplate>
         <uuid>String</uuid>
         <name>String</name>
         <maxDirNumbers>int</maxDirNumbers>
      </phoneButtonTemplate>
      <phoneButtonTemplate>
         …
      </phoneButtonTemplate>
   </phoneButtonTemplates
</environment>
```

getImportData

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<importData>
   <user />
   <phone />
   <profile />
   <number />
   <huntGroup />
   <callPickupGroup />
</importData >
```

Die Tags im <importData> sind fakultativ. Falls der Tag vorhanden ist, wird eine Liste mit Objekten zurückgegeben.

Die Rückgabe der Middleware liefert die gewünschten Objekte:

```
<importData>
   <users>
      <user>
         Gleich wie bei getUser
      </user>
      <user>
         …
      </user>
   </users>
   <phones>
      <phone>
         Gleich wie Phone-Tag im getUser
      </phone>
      <phone>
         …
      </phone>
   </phones>
   <profiles>
      <profile>
         Gleich wie Profile-Tag im getUser
      </profile>
      <profile>
         …
      </profile>
   <numbers>
      <number>
         Gleich wie Number-Tag im getUser
      </number>
      <number>
         …
      </number>
   </numbers>
```

```
   <huntGroups>
      <huntGroup>
        Gleich wie getGroup (HuntGroup)
      </huntGroup>
      <huntGroup>
        …
      </huntGroup>
   </huntGroups>
   <callPickupGroups>
      <callPickupGroup>
        Gleich wie getGroup (CallPickupGroup)
      </callPickupGroup>
      <callPickupGroup>
        …
      </callPickupGroup>
   </callPickupGroups>
</importData>
```

Falls die Benutzer ausgewählt wurden, werden die Telefone und Profile, welche schon in einem Benutzer enthalten sind, nicht noch einmal übertragen. Das heisst, wenn Telefon1 bereits in einem User-Tag vorhanden ist, wird dieses Telefon1 nicht mehr im Phones-Tag aufgelistet. Durch dieses Vorgehen wird der Informationsgehalt der Rückmeldung erhöht.

getDirNumber

Ein ganzes Nummern-Objekt wird anhand der Rufnummer aus dem Communications Manager herausgeholt.

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<dirNumber>
  <pattern>String</pattern>
</dirNumber>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<dirNumber>
  <action>String</action>
  <pattern>String</pattern>
  <uuid>String</uuid>
  <description>String</description>
  <routePartition>
  Gleich wie Partition-Tag im getEnvironment
  </routePartition>
  <css>
  Gleich wie Css-Tag im getEnvironment
  </css>
  <mohUserHold>
  Gleich wie musicOnHold-Tag im getEnvironment
  </mohUserHold>
  <mohNetworkHold>
  Gleich wie musicOnHold-Tag im getEnvironment
  </mohNetworkHold>
  <pickupGroup>
  Gleich wie callPickupGroup-Tag im getEnvironment
  </pickupGroup>
  <numberType>String</numberType>
  <displayName>String</displayName>
  <callForwardAll>String</callForwardAll>
  <callForwardBusy>String</callForwardBusy>
  <callForwardBusyInt>String</callForwardBusyInt>    <callForwardNoAns-
wer>String</callForwardNoAnswer>
  <callForwardNoAnswerInt>String</callForwardNoAnswerInt>  <callForwar-
dAllNoCoverage>String</callForwardNoCoverage>
  <callForwardAllNoCoverageInt>String</callForwardNoCoverageInt>
  <callForwardAllOnFailure>String</callForwardOnFailure>
</dirNumber>
```

addDirNumber

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<dirNumber>
   Gleich wie dirNumber-Tag in getDirNumber Antwort
</dirNumber>
```

Die Rückgabe der Middleware enthält die `uuid` sowie die Telefonnummer:

```
<dirNumber>
   <pattern>String</pattern>
   <uuid>String</uuid>
</dirNumber>
```

deleteDirNumber

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<dirNumber>
   <pattern>String</pattern>
</dirNumber>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<dirNumber>
   <uuid>String</uuid>
</dirNumber>
```

Die `uuid` ist entweder auf 1 (erfolgreich) oder auf -1 (nicht erfolgreich) gesetzt.

updateDirNumber

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<dirNumber>
   Gleich wie dirNumber-Tag in getDirNumber Antwort
</dirNumber>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<dirNumber>
   <uuid>String</uuid>
</dirNumber>
```

Die `uuid` ist entweder auf 1 (erfolgreich) oder auf -1 (nicht erfolgreich) gesetzt.

getPhone

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<phone>
  <MAC>String</MAC>
</phone>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<phone>
  Gleich wie phone-Tag in getUser Antwort
</phone>
```

getProfile

Die Parameterübergabe an die Middleware ist wie folgt spezifiziert:

```
<profile>
  <profileName>String</profileName>
</profile>
```

Die Rückgabe der Middleware sieht wie folgt aus:

```
<profile>
  Gleich wie profile-Tag in getUser Antwort
</profile>
```

Fehlerbehandlung

Für jedes bearbeitete Objekt gibt die Middleware in der Antwort an, ob die Aktion erfolgreich durchgeführt werden konnte oder nicht. Dadurch kann in der Applikation eine eigene Fehlerbehandlung implementiert werden. Diese wird über die `uuid` des jeweiligen Objektes bewerkstelligt. Ist die `uuid` „-1“, konnte die Aktion auf diesem Objekt nicht ausgeführt werden.

5.5.7. Middleware Logik

Durch die Mabata Business Case Vereinfachung ergeben sich Änderungen in den Middleware Anforderungen. Neu muss diese komplette Business Cases verarbeiten können und ist nicht mehr nur für die Weiterleitung der Requests verantwortlich.

Diese neue Anforderung führt dazu, dass das gesamte Know How über das Zielsystem in der Middleware ist. Ein Applikationsentwickler muss nur noch die Schnittstellen der Middleware kennen, und nicht mehr das Zielsystem.

Da in Mabata kein Wissen über die Funktion des Communications Managers vorhanden ist, muss dies in der Middleware abgebildet werden. Aus dem empfangenen XML-String können die Objekte wiederhergestellt werden, welche für die Kommunikation mit dem Communications Manager gebraucht werden. Dabei muss die Reihenfolge der Erstellung eingehalten werden. Aus diesem Grund werden zuerst die Objekte erstellt, welche keine anderen Objekte referenzieren.

Falls zum Beispiel ein Telefon mit einer Nummer erstellt werden soll, wird zuerst die Nummer auf dem Communications Manager erstellt und erst danach das Telefon, welchem bereits die soeben erstellte Nummer zugewiesen werden kann.

5.6. Middleware

In diesem Kapitel wird beschrieben, wie die Middleware technisch umgesetzt wurde. Dabei wurde sie, abgesehen von einer kleinen Änderung, wie geplant umgesetzt. Die einzige Abweichung besteht darin, dass die Middleware gegen Aussen nicht vier einzelne WebServices, sondern ein einzelner WebService mit vier WebMethoden publiziert. Dies erleichtert vor allem die Erstellung des Adapters, da pro WebService im Adapter zwei Proxy-Klassen generiert werden müssen.

Nachfolgend wird der Aufbau des Codes mitsamt den einzelnen Klassen sowie dem Ablauf eines Methodenaufrufes beschrieben.

5.6.1. Package Diagramm

Nachstehendes Diagramm zeigt das Zusammenspiel zwischen den verschiedenen Packages der Middleware auf.

Eine wichtige Eigenschaft ist, dass die XML-Messages welche über den WebService empfangen werden, in der Façade in Java-Objekte umgewandelt werden. Das heisst, dass ab der Façade nur noch mit Java-Objekten gearbeitet wird. Dies hat den Vorteil, dass Änderungen des Übertragungsformates nicht die gesamte Middleware betreffen würde, sondern bloss den Teil der Façade, welcher für die Umwandlung der XML-Strings in Java-Objekte zuständig ist.

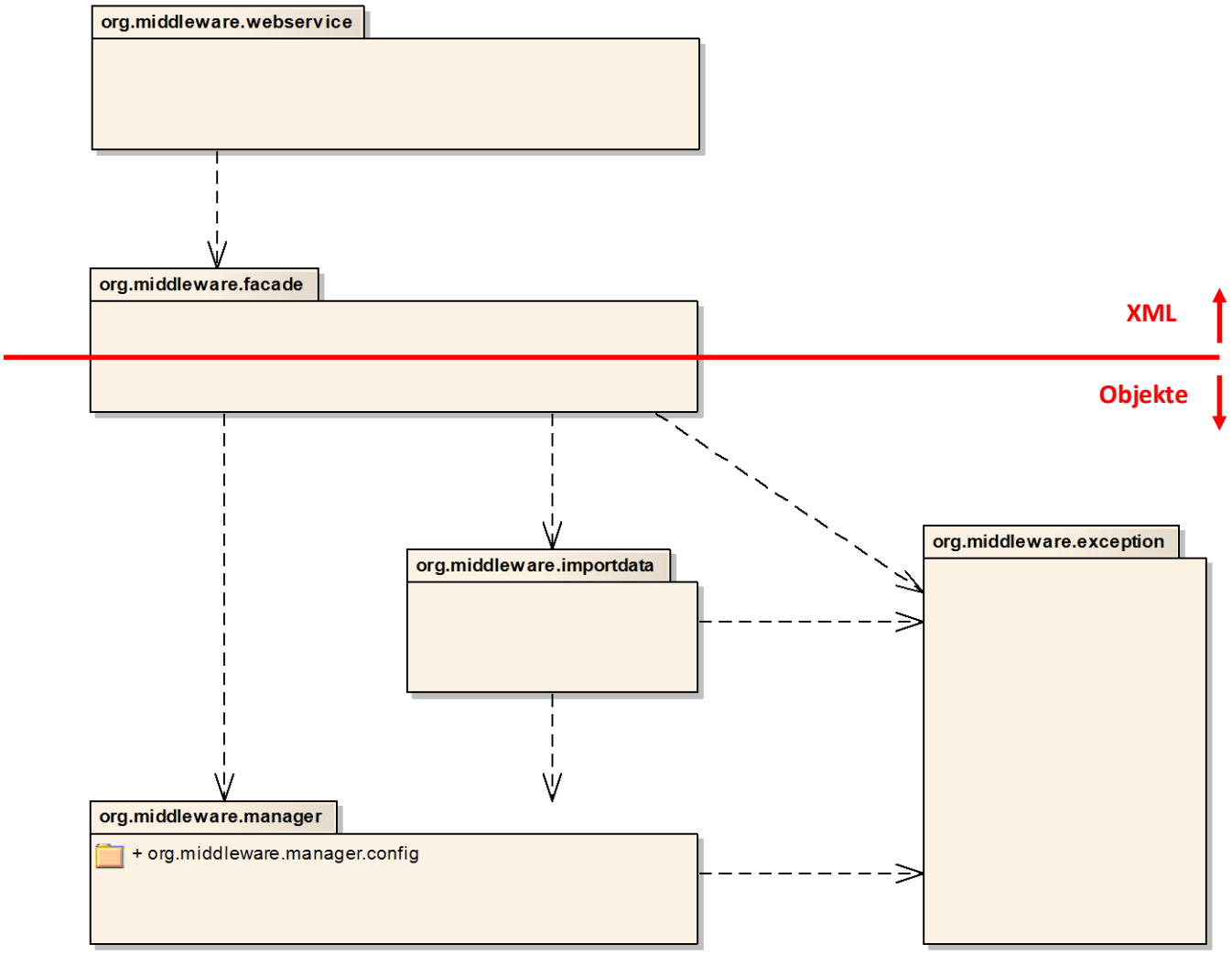


Abbildung 5-27 Package Diagramm Middleware

5.6.1.1. org.middleware.webservice

Enthält den WebService mit den WebService Methoden. Ist für die Publizierung des WebServices und dessen Methoden verantwortlich. Enthält ausserdem die Überprüfung um sicherzustellen, dass der Aufrufer berechtigt ist, den WebService zu benutzen. Leitet die Aufrufe dem Façade weiter.

5.6.1.2. org.middleware.façade

Stellt den Einstiegspunkt vom WebService in die Middleware dar. Aufgabe der Façade ist, aus dem XML-String anhand des Root-Tags das richtige Objekt zu erstellen und den Aufruf an den entsprechenden Manager weiterzuleiten. Bei der Rückmeldung muss das Façade aus einem Objekt einen XML-String erstellen, welcher an den WebService weitergeleitet wird.

5.6.1.3. org.middleware.importdata

Ist für den Import verantwortlich. Greift nicht direkt auf das myFramework zu, sondern über die entsprechenden Objekt-Manager. Dieses Package ist transparent. Es wird nur für den Import verwendet. Ansonsten wird es von der Façade übergangen.

5.6.1.4. org.middleware.manager

Enthält die eigentliche Funktionalität der Middleware. Im Manager befinden sich zum einen jene Klassen, welche für die Abstraktionen der Daten verantwortlich sind. Das heisst, dass diese Klassen

auf dem Communications Manager nicht existieren. Zum andern befinden sich die Klassen im Manager, welche den Zugriff auf das myFramework herstellen. Es wurde Wert darauf gelegt, dass es für jedes auf dem Communications Manager existierende Objekt eine Manager Klasse gibt, welche für alle Requests verantwortlich ist.

5.6.1.5. org.middleware.manager.conifg

Ist für die Verbindungsdaten des myFrameworks verantwortlich. Liest die Daten aus einer Konfigurationsdatei und füllt diese in eine Klasse, mit welcher jeweils die Daten bei einem Request konfiguriert werden.

5.6.1.6. org.middleware.exception

Enthält die in der Middleware geworfenen Exceptions. Diese Exceptions werden im Manager, Importdata und Façade Package verwendet.

5.6.2. Klassendiagramm

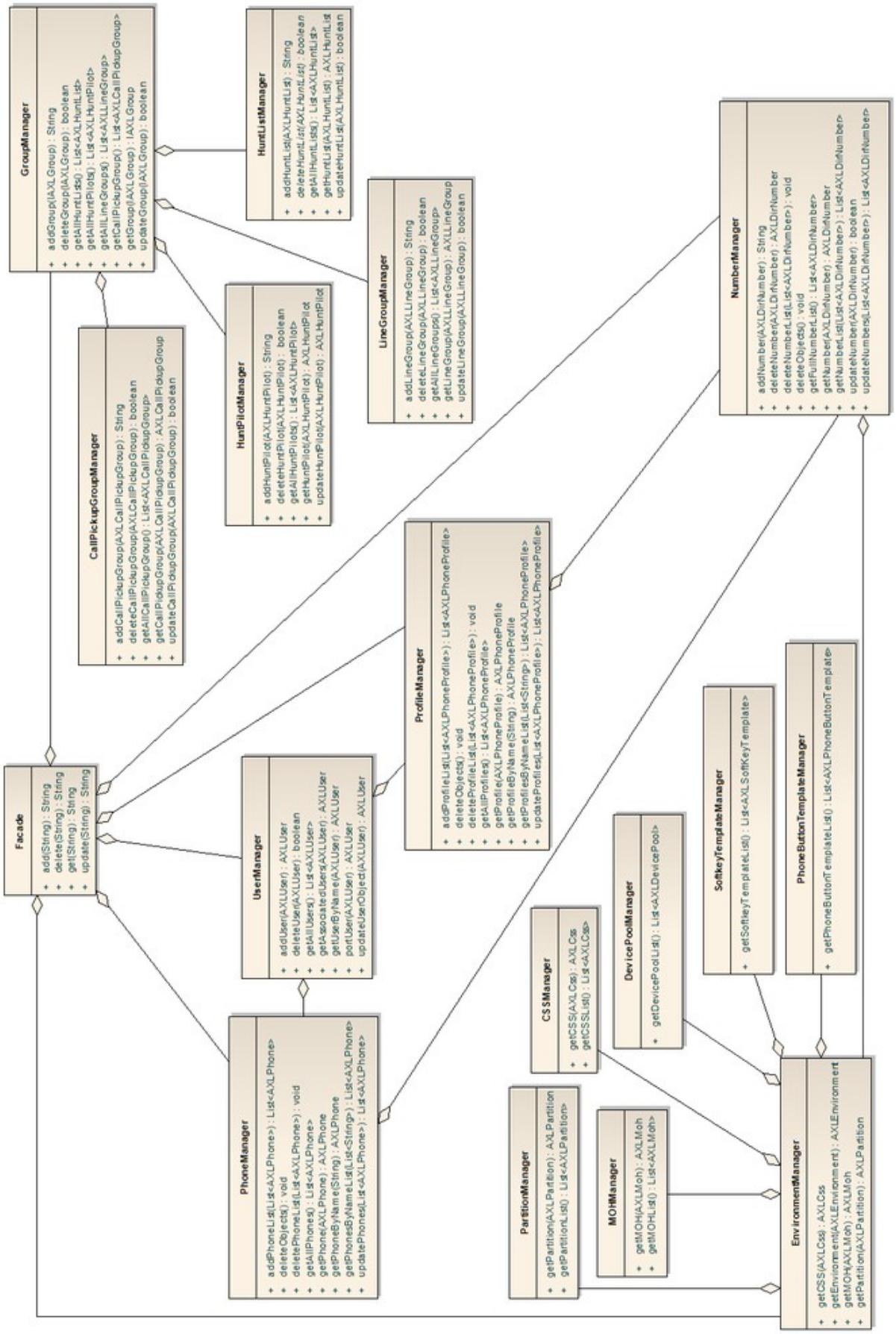


Abbildung 5-28 Klassendiagramm Middleware

Um die Übersicht zu verbessern, wurden im Klassendiagramm nur die Klassen des Package «org.middleware.manager» abgebildet. Da die Hauptaufgabe der Middleware in diesem Package geschieht, macht es durchaus Sinn, sich auf diese Klassen zu beschränken. Ansonsten würde das Diagramm nur unnötig grösser und komplizierter, ohne daraus einen grösseren Nutzen zu schöpfen. Als einzige Klasse ausserhalb dieses Packages wurde das `Façade` abgebildet, um zu zeigen, auf welche Objekte direkt und auf welche indirekt zugegriffen wird.

Nachfolgend werden die wichtigsten Klassen mit ihren Funktionen kurz beschrieben.

5.6.2.1. Façade

Stellt den Einstiegspunkt in die Middleware Logik dar. Aufgabe der Façade ist, den empfangenen XML-String nach dem Root-Tag zu untersuchen und entsprechend weiterzuleiten. In der weitergeleiteten Methode, welche sich ebenfalls noch in der Façade befindet, wird aus dem XML-String ein Java-Objekt erstellt. Ab diesem Zeitpunkt wird nur noch mit Objekten gearbeitet.

5.6.2.2. UserManager

Der `UserManager` trägt die Verantwortung dafür, um User Objekte zu bearbeiten. Dieser Manager stellt über das myFramework die Verbindung zum Communications Manager her. Der Manager ruft nicht nur die einzelnen Requests des myFrameworks auf, sondern stellt auch die Verbindungen eines Benutzers her. Das heisst, bei einem `getUser` wird überprüft, welche Telefone diesem Benutzer zugewiesen sind. Diese Telefone werden anschliessend einzeln im `PhoneManager` abgeholt, einer Liste hinzugefügt und im Benutzer gespeichert. Analog dazu werden die Profile gehandhabt.

Eine weitere wichtige Aufgabe besteht darin, Fehler zu erkennen und diese soweit möglich zu beheben. Diese können bei einem `addUser` oder `updateUser` auftreten, falls mehrere AXLRequests nacheinander an den Communications Manager gesendet werden. Diese Fehlerbehandlung ist im Kapitel 5.6.4 Transaction Handling genauer beschrieben.

5.6.2.3. PhoneManager

Der `PhoneManager` hat die gleichen Aufgaben wie der UserManager. Er ist für die Verarbeitung des Phone-Objektes zuständig. Dabei ist ein Phone-Objekt ebenfalls wie ein User-Objekt wieder verschachtelt. Es enthält eine Liste von Nummern. Diese Liste von Nummern wird jeweils wieder in einer Schleife durchlaufen und im `NumberManager` abgearbeitet.

5.6.2.4. ProfileManager

Der `ProfileManager` ist das Pendant zum `PhoneManager`. Er verhält sich gleich und deckt die Verantwortlichkeit für die PhoneProfiles ab.

5.6.2.5. GroupManager

Von Seiten des `GroupManagers` gibt es keinen direkten Zugriff auf das myFramework. Die Aufgabe besteht darin, die verschiedenen Gruppen, welche auf dem Communications Manager bestehen, zu abstrahieren. Dieser Klasse wird jeweils ein Objekt des Interfaces `IAXLGroup` übergeben. Dieses Interface wird von jeder Klasse implementiert, welche eine Group abbildet. Dabei überprüft der `GroupManager` um welchen Typ es sich bei dem übergebenen Objekt handelt und leitet den Methodenaufruf an den entsprechenden Manager weiter.

5.6.2.6. EnvironmentManager

Vom `EnvironmentManager` erfolgen keine Zugriffe auf das myFramework. Der `EnvironmentManager` dient dazu, Elemente, welche häufig zusammen verwendet werden, zusammenzufassen.

5.6.3. Ablaufdiagramm

Nachfolgend wird zuerst ein allgemeiner Ablauf beschrieben, wie ein einfacher Aufruf von Mabata über den Mabata-Adapter auf die Middleware erfolgt. Dabei wurde das Augenmerk nur auf diesen Ablauf gelegt, und jegliche andere Aktivitäten wurden zu Gunsten der besseren Übersicht weggelassen.

In einem zweiten Diagramm wird die Aktion `addUser` in der Middleware abgebildet, da dieser Prozess verschachtelte Objekte beinhaltet und aus diesem Grund komplexer ist.

5.6.3.1. Allgemeiner Ablauf

Das folgende Ablaufdiagramm verbildlicht das Zusammenspiel von Mabata, Mabata-Adapter, der Middleware und dem myFramework. Als Beispiel wird eine Nummer aus Mabata erstellt. Zur besseren Übersicht wurde das Diagramm in zwei Teil unterteil. Im ersten Teil ist sichtbar, wie Mabata auf den Mabata-Adapter zugreift und dieser die XML-Meldung versendet. Im zweiten Teil ist die Verarbeitung in der Middleware abgebildet.

Der Zugriff von Mabata auf den Mabata-Adapter erfolgt ausschliesslich über die `EntityManager`. Im untenstehenden Beispiel erstellt der `DirNumberEntityManager` ein `DirNumberMethods` Objekt, welches die Methoden des Adapters zur Verfügung stellt. Das `DirNumberMethods` Objekt generiert in der `addDirNumber`-Methode den XML-String, welcher in der Middleware benötigt wird, um die aufzurufende Methode zu bestimmen. Über den `Connector`, welcher den Hash-Wert aus dem übergebenen Parameter generiert, wird die Verbindung zum WebService hergestellt. Ausserdem besitzt der Connector den Benutzernamen sowie das Passwort, um den WebService aufzurufen.

Auf das nachfolgende Beispiel bezogen, greift der Mabata Core auf den `DirNumberEntityManager` zu und ruft die `addDirNumber` Methode auf. In dieser Methode wird auf den Mabata-Adapter zugegriffen. Dazu wird ein `DirNumberMethods`-Objekt erstellt und dessen `addDirNumber` Methode, welche ein `AdapterDirNumber` Objekt empfängt, aufgerufen. Ein `AdapterDirNumber Objekt` kann aus einem Mabata `DirNumber`-Objekt generiert werden. Im Mabata-Adapter greift die `DirNumberMethods`-Klasse über den `Connector` auf das WebService-Objekt zu, um die Verbindung zur Middleware herzustellen.

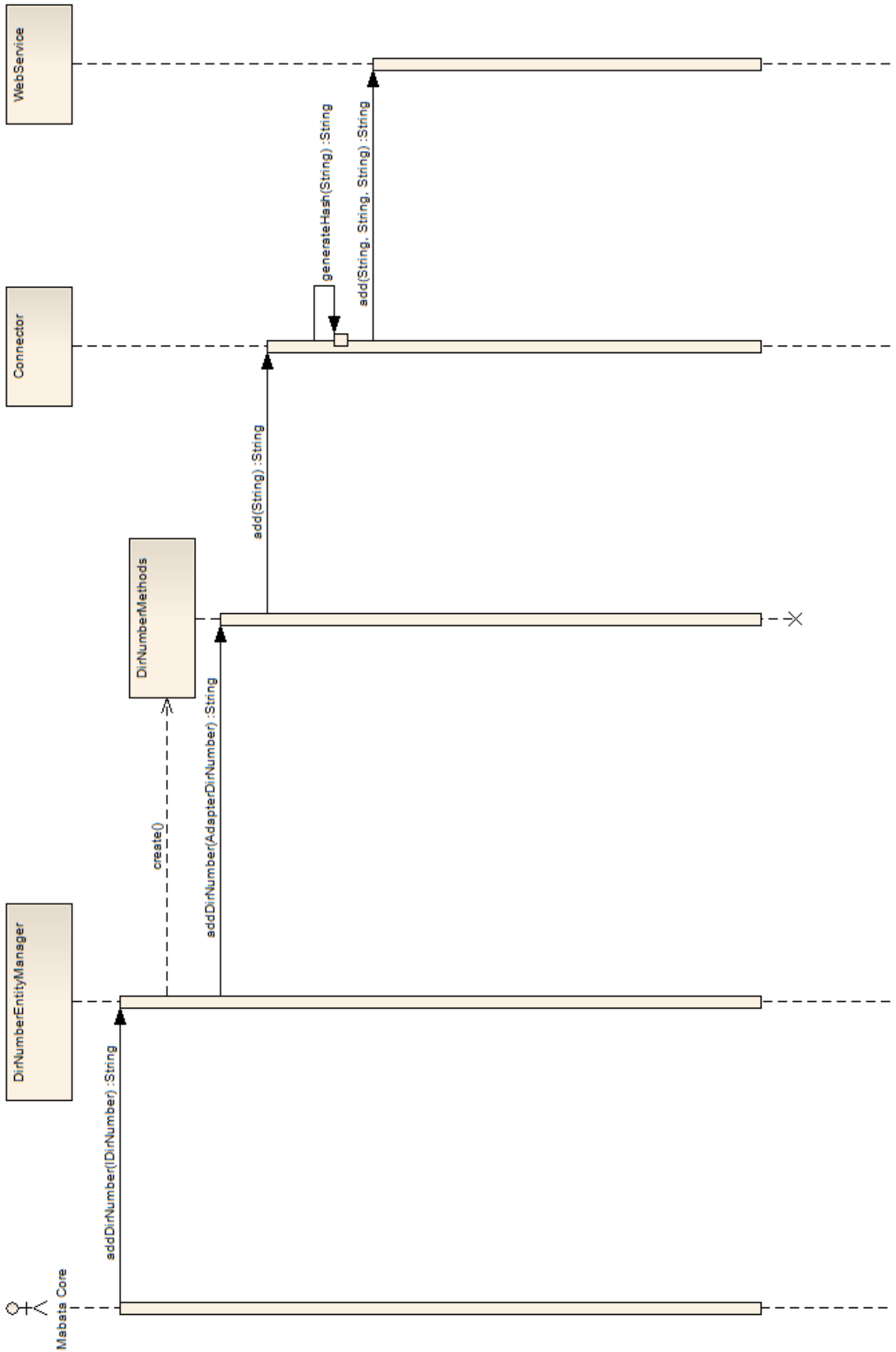


Abbildung 5-29 Ablauf addDirNumber

Der WebService des Mabata-Adapters überträgt den XML-String, Hash-Wert und den Benutzernamen an den add-WebService der Middleware. Als Rückmeldung sendet die Middleware einen XML-String.

```
add(String,String,String):String
```

Der WebService überprüft als Erstes, ob mit dem empfangenen Benutzernamen ein gleicher Hash-Wert generiert werden kann. Dies kann er, indem er mit dem empfangenen Benutzernamen das serverseitig abgespeicherte Passwort ermittelt und aus diesem wieder den Hash-Wert generiert. Falls dieser identisch zum mitgelieferten Hash-Wert ist, wird der Root-Tag des XML-Strings untersucht. Anhand des Root-Tags (in diesem Beispiel `number`) kann bestimmt werden, um welches Objekt es sich handelt. Danach kann aus dem XML-String mittels der Methode `importToMiddleware` ein `AXLDirNumber`-Objekt erzeugt werden. Ab diesem Zeitpunkt wird in der Middleware nur noch mit Objekten gearbeitet. Dieses Objekt wird dem `NumberManager` übergeben, welcher für den Aufruf des myFrameworks verantwortlich ist.

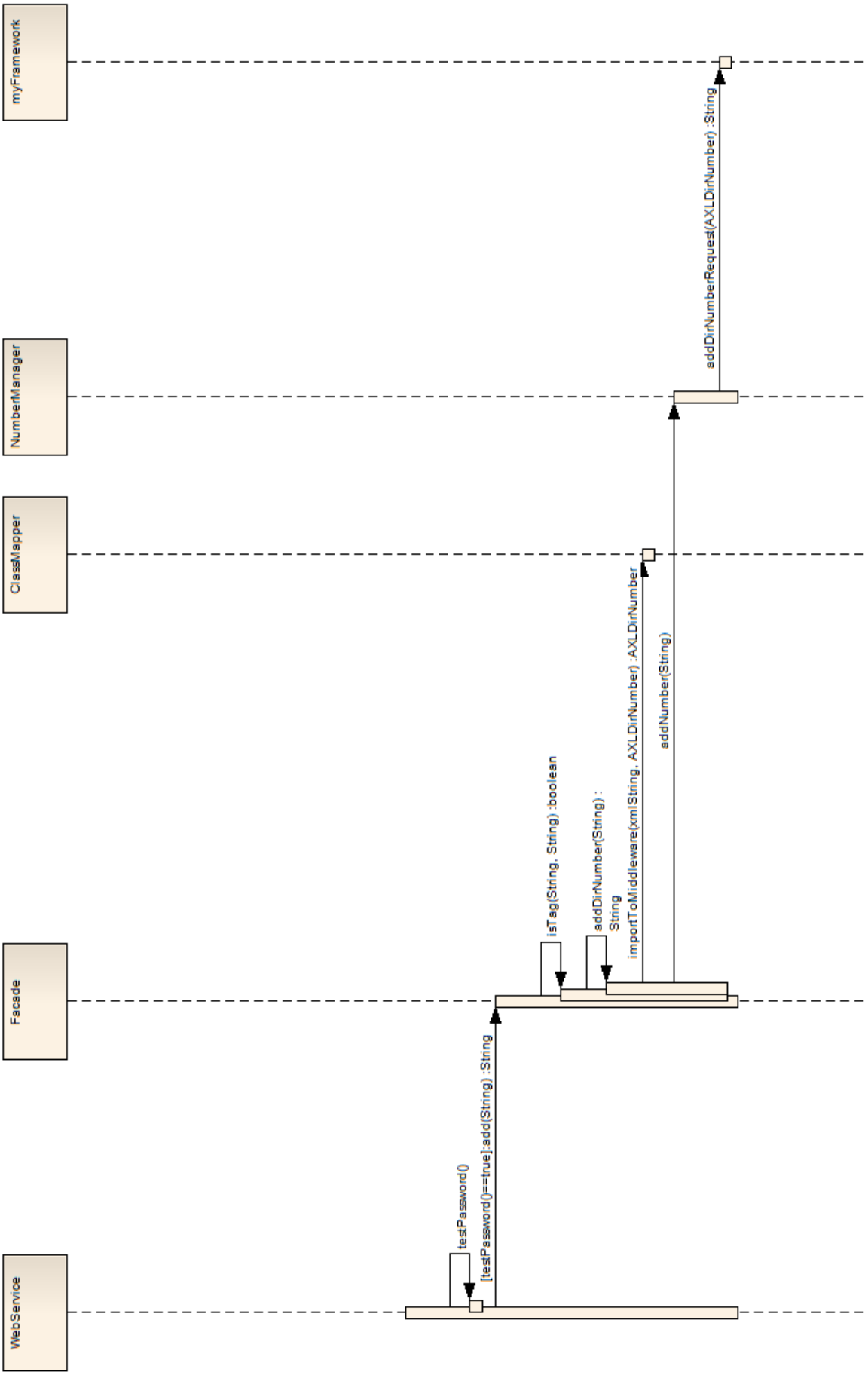


Abbildung 5-30 Hash-Wert Überprüfung

5.6.3.2. addUser[1]

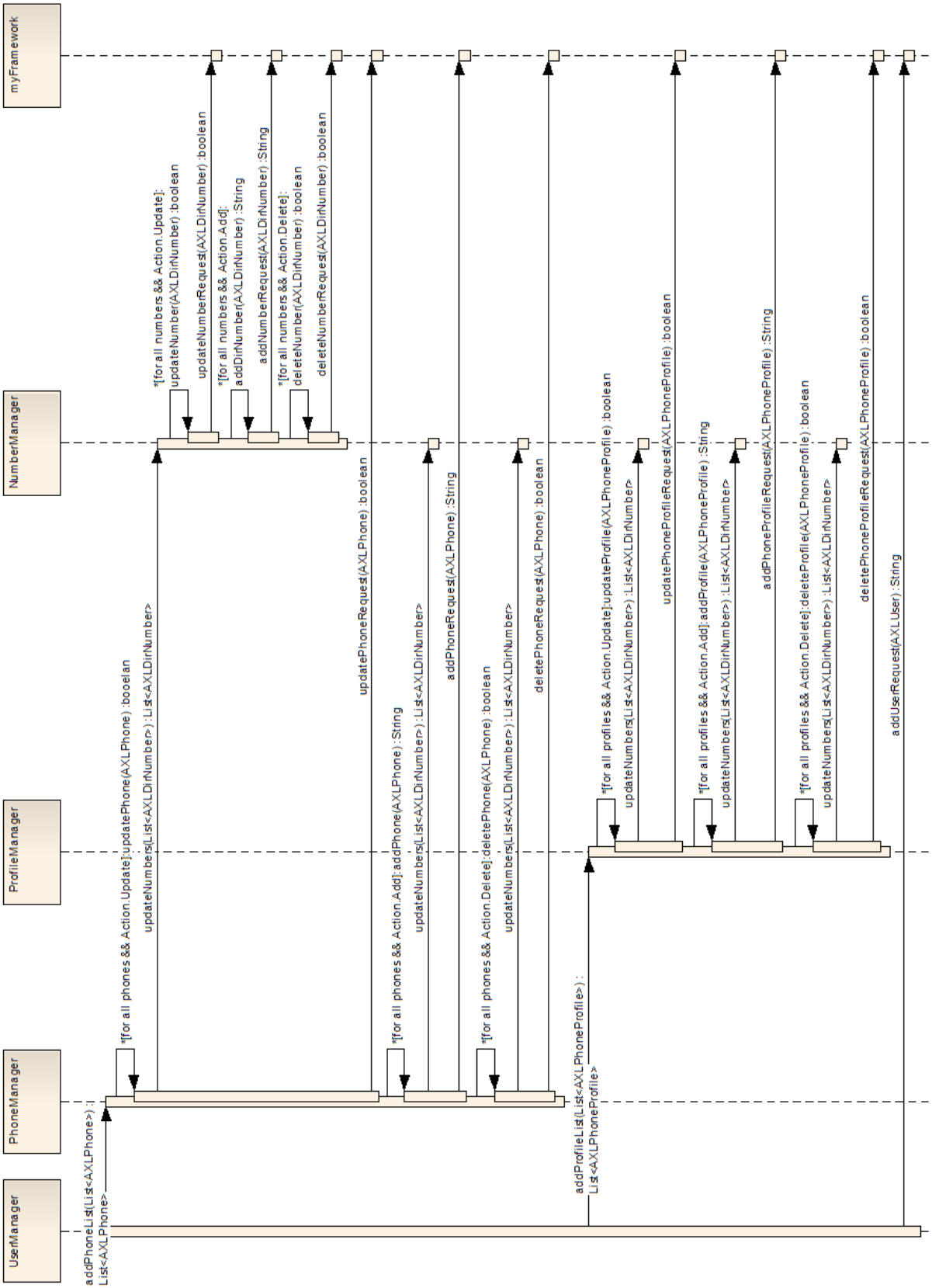


Abbildung 5-31 Ablauf addUser

[1] Die `updateNumbers`-Methode wurde nur das erste Mal vollständig aufgezeichnet. Bei jedem weiteren Aufruf wird der gleiche Prozess wie beim ersten Aufruf abgearbeitet

Um einen Benutzer hinzuzufügen, muss die Middleware mehrere Methoden durchlaufen. Als Erstes wird aus dem `AXLUser` die Liste mit `AXLPhone`-Objekten genommen und dem `PhoneManager` übergeben. Dieser geht in einer Liste jedes `AXLPhone` durch und übergibt zuerst die im `AXLPhone` enthaltene Liste mit `AXLDirNumber`-Objekten dem `NumberManager`. Im `NumberManager` wird anhand der `Action` entschieden, welche Aktion auf der Nummer durchgeführt wird. Sind alle `AXLDirNumber`-Objekte abgearbeitet, kehrt das Programm zum `AXLPhone` zurück. Nun wird das `Action` Feld ausgewertet und die entsprechende Aktion ausgeführt. Diese Schritte werden für alle `AXLPhone`-Objekte durchgeführt.

Sind alle `AXLPhone` abgearbeitet, folgt der gleiche Prozess für die `AXLPhoneProfile`. Zuerst werden bei jedem Profil die Nummern, anschliessend das Profil aktualisiert.

Zum Abschluss des gesamten Prozesses wird der Benutzer selbst über das myFramework auf dem Communications Manager hinzugefügt.

5.6.4. Transaction Handling

Für die Benutzer-Funktionen wurde ein Transaction Handling eingeführt. Der Ablauf ist im nachfolgenden Diagramm abgebildet:

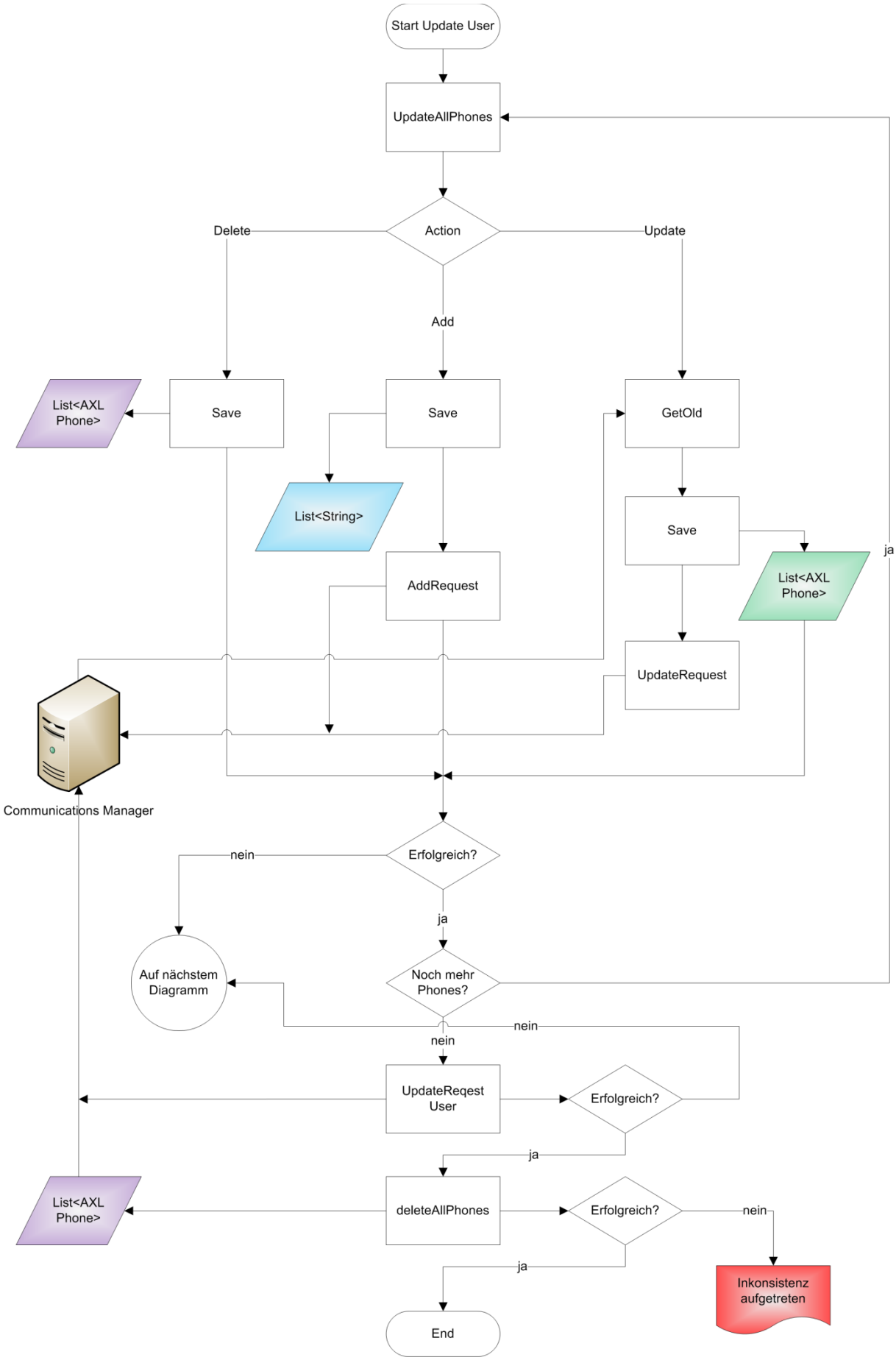


Abbildung 5-32 Ablauf Transaction Handling

Als erstes wird die Liste mit `AXLPhone` im Benutzer durchlaufen. Je nach gesetzter `Action` in diesem Objekt werden folgende Aktionen durchgeführt:

- `Delete`:
  Die Objekte werden in einer Liste zwischengespeichert. Auf dem Communications Manager wird noch nichts gelöscht.
- `Add`:
  Der `PhoneName` wird in einer Liste abgespeichert. Danach wird auf dem Communications Manager das neue Phone erstellt.
- `Update`:
  Zuerst wird das alte Phone-Objekt vom Communications Manager abgeholt und in einer Liste zwischengespeichert. Erst danach wird ein Update auf dem Communications Manager durchgeführt.

Konnte dieser Vorgang für alle `AXLPhone` erfolgreich durchgeführt werden, wird der Benutzer auf dem Communications Manager aktualisiert. Andernfalls wird der Restore, welcher im Kapitel 5.6.4.2 beschrieben ist, aufgerufen. Der Restore wird Ebenfalls aufgerufen, falls der `updateUser` fehlschlägt. Ist bis zu diesem Zeitpunkt alles fehlerfrei abgelaufen, werden zum Abschluss die in einer Liste zwischengespeicherten Telefone gelöscht. Ist diese letzte Aktion erfolgreich, ist die ganze Transaktion abgeschlossen. Konnte allerdings ein Telefon nicht gelöscht werden, ist eine Inkonsistenz entstanden und der Benutzer muss darüber informiert werden.

Der gesamte Transaktionskontext erstreckt sich natürlich auch über die `AXLPhoneProfile` und `AXLDirNumber`-Objekte. Für alle diese Objekte ist das Transaction Handling identisch Implementiert.

5.6.4.1. Listen

Nachfolgend werden die verwendeten Listen beschrieben, in welchen die Objekte zwischengespeichert werden:

- Delete: `List<AXLPhone>`
  In dieser Liste werden die `AXLPhone`-Objekte zwischengespeichert. Diese Liste wird am Schluss verwendet um die Objekte zu löschen. Dies ist nötig, da die Objekte auf dem Communications Manager erst zum Schluss einer Transaktion gelöscht werden dürfen. Denn diese können nach dem Löschen nicht wiederhergestellt werden, da die Abhängigkeiten zu anderen Objekten nur mit einem enormen Aufwand zwischengespeichert werden können.
- Add: `List<String>`
  Um ein hinzugefügtes Objekte wieder vom Communications Manager zu löschen, wird nur sein Name gebraucht. Dieser Name wird in einer Liste abgespeichert.
- Update: `List<AXLPhone>`
  Damit alle Felder eines Objektes wiederhergestellt werden können, muss vor einem Update das alte Objekt vom Communications Manager abgeholt und zwischengespeichert werden.

5.6.4.2. Restore

Falls im Prozess ein Fehler aufgetreten ist versucht die Middleware, die gesamte Transaktion rückgängig zu machen. Für diesen Restore werden die vorhin beschriebenen Listen benötigt. Als Erstes wird die Liste durchlaufen, in welcher die `PhoneIDs` gespeichert wurden. Diese enthält alle IDs von den Telefonen, welche neu auf dem Communications Manager erstellt wurden. Alle diese Telefone werden nun wieder mit Hilfe der `PhoneID` gelöscht.

Als nächstes werden die Updates rückgängig gemacht. Hierzu wird die Liste mit `AXLPhone`-Objekten verwenden. Dabei wird für jedes Objekt in der Liste jeweils ein Update auf dem Communications Manager ausgeführt. Konnten alle diese Aktionen erfolgreich durchgeführt werden, wurde die Transaktion zurückgesetzt. Falls nicht, wurde die Transaktion teilweise ausgeführt, was ein Eingreifen des Administrators erfordert. Dazu kann die Konsistenzprüfung verwendet werden.

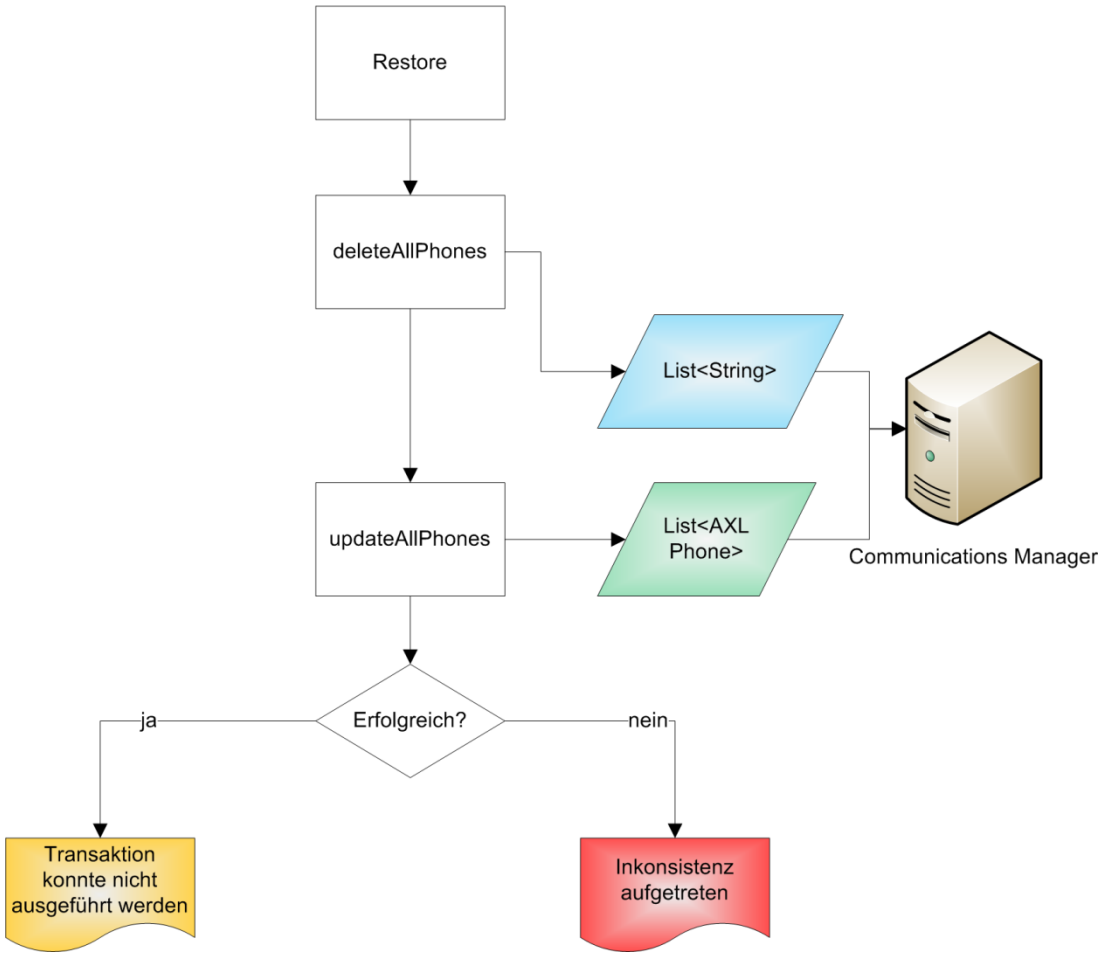


Abbildung 5-33 Ablauf Restore

5.6.5. XML-Parsing

Die Datenübertragung zum WebService geschieht über XML. Dieser XML-String muss in der Middleware wieder in ein Objekt umgewandelt werden. Dies wird mit dem JAXB Parser bewerkstelligt.

5.6.5.1. JAXB Parser

Zur Erstellung des XML-Strings aus Objekten, und wieder zurück, kann die «Java Architecture for XML Binding (JAXB)» verwendet werden. Diese erstellt automatisch aus Objekten, welche annotiert wurden, XML-Strings und umgekehrt auch wieder Objekte aus einem XML-String.

Dies ist nachfolgend anhand eines Beispiels erläutert.

Das Testprogramm:

```
public class Test {

 public static void main(String[] args) {
  try{
   JAXBContext jc =
    JAXBContext.newInstance(new Class[] {com.foo.FooObject.class});

   Unmarshaller u = jc.createUnmarshaller();
   FooObject fooObj = new FooObject("foo");

   Marshaller m = jc.createMarshaller();

   StringWriter sWriter = new StringWriter();
   m.marshal( fooObj, sWriter );
   Charset charset = Charset.forName("UTF-8");
   FooObject fooObj2 =
    (FooObject)u.unmarshal(new ByteArrayInputStream(xmlString
                        .getBytes(charset)));
  }
  catch(Exception e){
   System.out.println(e.getMessage());
  }
 }
}
```

Der `Marshaller` erstellt mit der Methode `marshal` den XML-String:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<fooObject>
   <name>foo</name>
   <testobject>
      <name>bar</name>
   </testobject>
   <barobjects>
   <barobject>
      <name>barlist1</name>
   </barobject>
   <barobject>
      <name>barlist2</name>
   </barobject>
</fooObject>
```

Das Objekt, welches annotiert wurde, sieht folgendermassen aus:

```
@XmlRootElement
public class FooObject {
      @XmlElement
      public String name;

      @XmlTransient
      public String notinteresting;

      @XmlElement(name="testobject")
      public BarObject barobject = new BarObject("bar");

      @XmlElementWrapper(name="barobjects")
      @XmlElement(name="barobject")
      public List<BarObject> list = new ArrayList<BarObject>();

      public FooObject(){}

      public FooObject(String name){
            this.name = name;
            list.add(new BarObject("barlist1"));
            list.add(new BarObject("barlist2"));
      }
}
```

Kurzerklärung der Annotations

@XmlRootElement	Bezeichnet das Root Element des XML-Strings. Der Name des Tags kann mit «name=“newname“» überschrieben werden.
@XmlElement	Ein Element das in den XML-String aufgenommen wird. Alle `public` Variablen werden automatisch aufgenommen. Der Name des Tags kann mit «name=“newname“» überschrieben werden.
@XmlTransient	Verhindert das Aufnehmen in den XML-String.
@XmlElementWrapper	Fügt ein zusätzlicher Tag um das eigentliche Element hinzu.

Weiterführende Informationen sind unter [JAXB] vorhanden.

5.6.5.2. Verwendung in Middleware

Da die `importToMiddleware`-Methode für jedes Objekt gebraucht wird und die untenstehende rot markierte Zeile für jedes Objekt ein anderes Resultat gibt, wurde eine generische Methode eingeführt. Dieser Methode kann der empfangene XML-String und das Objekt, welches aus dem XML-String generiert werden soll, übergeben werden.

```
public static <T> T importToMiddleware(String xmlString, T object)
      throws JAXBException {
      JAXBContext jc = JAXBContext.newInstance(new Class[] {
            object.getClass() });
      Unmarshaller u = jc.createUnmarshaller();
      Charset charset = Charset.forName("UTF-8");
      object = (T) u.unmarshal(new
            ByteArrayInputStream(xmlString.getBytes(charset)));
      return object;
}
```

Für den umgekehrten Weg, um aus einem Objekt ein XML-String zu erstellen, wurde ebenso eine generische Methode erstellt. Dieser Methode muss das Objekt übergeben werden, aus welchem der XML-String generiert werden soll. Aus diesem Objekt erhält die Methode alle benötigten Informationen, welche der JAXB Parser für die Erstellung des XML-Strings benötigt.

```
public static <T> String extractToXML(T object) {
      JAXBContext jContext = null;
      String result = "";
      jContext = JAXBContext.newInstance(new Class[] { object.getClass()
            });
      Marshaller marshaller = jContext.createMarshaller();
      StringWriter sWriter = new StringWriter();
      marshaller.marshal(object, sWriter);
      return sWriter.toString();
}
```

5.6.6. Weiterentwicklung

Um die Endanforderungen (mehrere Ziel- und Quellsysteme unterstützen) der Middleware zu erreichen, muss sie weiterentwickelt werden. In diesem Kapitel wird versucht, denn zukünftigen Entwicklern eine Grundidee für die Weiterentwicklung zu vermitteln.

5.6.6.1. Anbindung neuer Quellsysteme

Ziel der Anbindung eines neuen Quellsystems ist es, wie in der nachfolgenden Grafik zu sehen ist, von einer zweiten Applikation aus Daten mit dem Communications Manager auszutauschen.

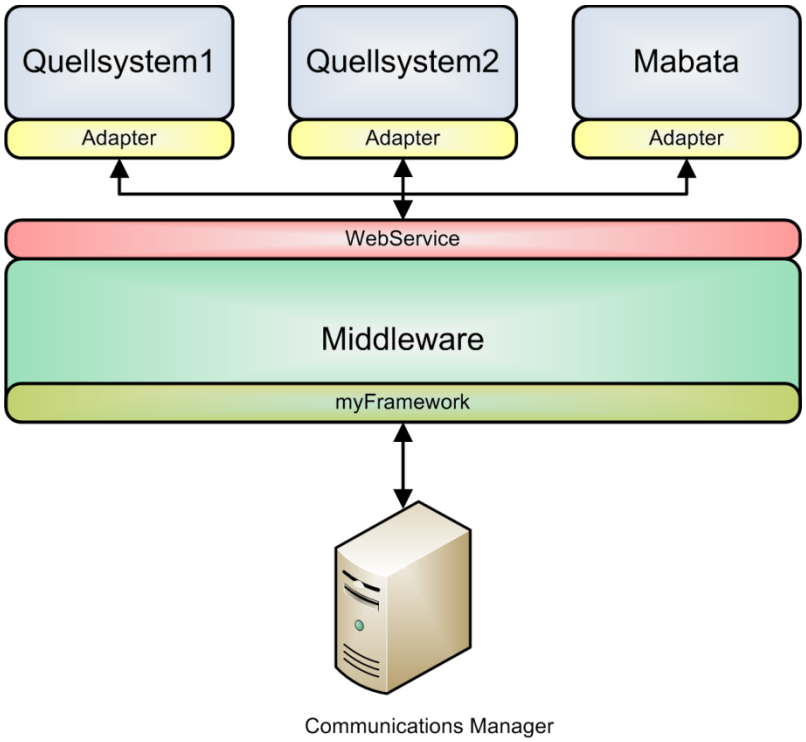


Abbildung 5-34 Anbindung mehrerer Quellsysteme

Um diese Anbindung zu bewerkstelligen müssen die folgenden Punkte beachtet werden. Als Erstes muss für die neue Applikation ein Adapter erstellt werden, welcher auf die Middleware zugreift. Für diesen Adapter gibt es kein Standardverfahren, da die Architekturen der anzubindenden Applikationen unbekannt sind.

Nebenläufigkeit

Ein wichtiger Punkt, mit welchem man sich unweigerlich auseinandersetzen muss, ist die Nebenläufigkeit. Da die Middleware in keiner Container-Umgebung läuft, ist der Programmierer selber dafür verantwortlich, Nebenläufigkeiten zu verhindern. Dabei gilt es sich zu überlegen, ob es eventuell reicht, bestimmte Methoden `synchronized` zu machen oder ob es sinnvoller ist, andere Locking-Mechanismen zu verwenden. Entscheiden dabei ist, dass die Performance-Einbussen nicht zu hoch sein dürfen. Wobei hier ganz klar zu beachten ist, dass momentan das Nadelöhr der Middleware die Kommunikation mit dem Communications Manager darstellt, auf welche der Programmierer keinen Einfluss hat.

Applikationen Callback

Da die meisten Applikationen über eine eigene Datenbank verfügen, müssen diese informiert werden, falls sich Daten ändern welche in der lokalen Datenbank abgespeichert sind.
Beispiel: Zwei Applikation sind über die Middleware an den Communications Manager angeschlossen. Falls die eine Applikation einen Benutzer erstellt, muss die Middleware die andere Applikation über diese Erstellung informieren.
Eine denkbare Lösung für dieses Problem wäre, dass auch die Adapter einen WebService anbieten, welcher die Middleware aufrufen kann. Damit könnte eine Art Publish/Subscribe-Pattern implementiert werden. Dabei kann sich der Adapter bei der Middleware für jedes Zielsystem anmelden. Dadurch wird sichergestellt, dass die Applikation über alle gewünschten Änderungen informiert wird.

5.6.6.2. Anbindung neuer Zielsysteme

Das Endziel der Middleware sieht vor, mehrere Quell- und Zielsysteme zu unterstützen.

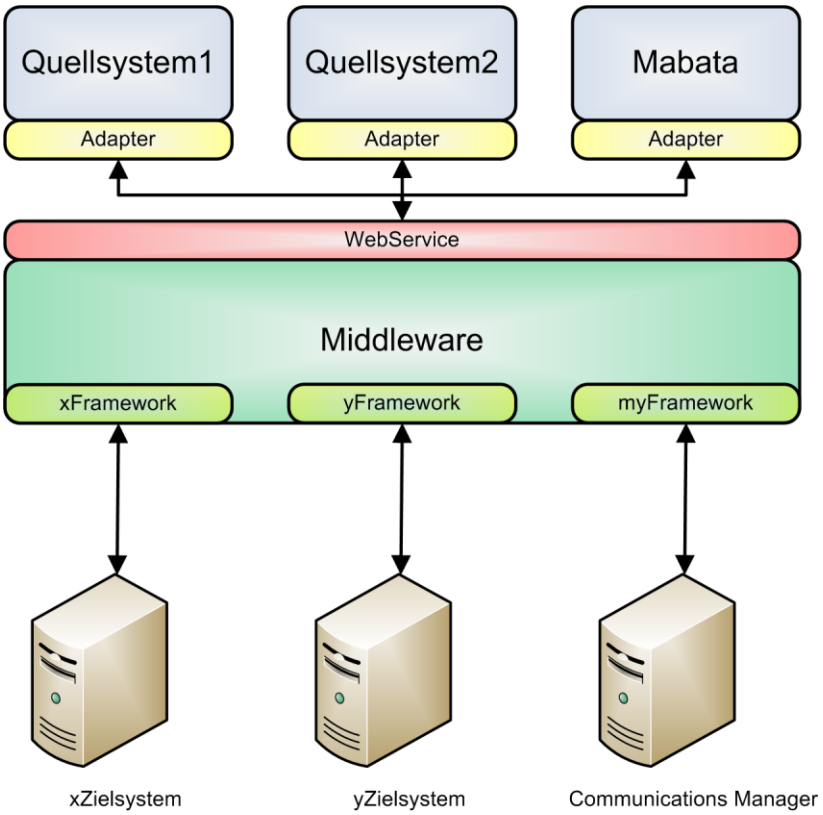


Abbildung 5-35 Anbindung mehrerer Zielsysteme

Um dieses Endziel zu erreichen, muss die Middleware weiterentwickelt werden.

Eigene Models

Da im Moment der Communications Manager als einziges Zielsystem an die Middleware angebunden ist, wurden in der Middleware keine eigenen Models eingeführt, sondern die Models des myFrameworks verwendet. Sobald neue Zielsysteme angebunden werden, macht es durchaus Sinn, auch in der Middleware eigene Models einzuführen, da die Wahrscheinlichkeit sehr gross ist, dass das neue Zielsystem andere Datenfelder braucht wie der Communications Manager. Da das Parsen der XML-Strings nicht von Hand, sondern mit dem JAXB-Parser realisiert wurde, funktioniert das Parsen nach dem Annotieren der neuen Models genau gleich wie bis anhin.

Konfiguration der gewählten Zielsysteme

Um zu konfigurieren, auf welchen Zielsystemen eine Applikation ihre Funktionen ausführen soll, bietet sich die Konfigurationsdatei des Adapters der jeweiligen Applikation an. In dieser Datei kann definiert werden, welche Zielsysteme abgedeckt werden sollen. Beim Versenden einer Message muss der Adapter diese Information dem XML-String anhängen. In der Middleware kann dieser angehängte Tag aus dem XML-String gelesen und anhand dessen bestimmt werden, welche Zielsysteme angesprochen werden müssen.

Neue Schicht

Nachdem die Middleware die Information vom Quellsystem bekommen hat, in welchen Zielsystemen die Funktion ausgeführt werden soll, muss dies in der Middleware noch abgebildet werden.

Im untenstehenden Package Diagramm ist ein Vorschlag für die Realisierung abgebildet. Allgemeine Abläufe, welche nicht spezifisch auf ein System gerichtet sind, laufen im «org.middleware.core» Package ab. In diesem Package findet auch die Entscheidung statt, in welchem Zielsystem die Funktion ausgeführt werden soll. Falls eine Funktion in einem Zielsystem ausgeführt werden soll, wird die entsprechende Methode im richtigen Package eine Stufe weiter unten aufgerufen.

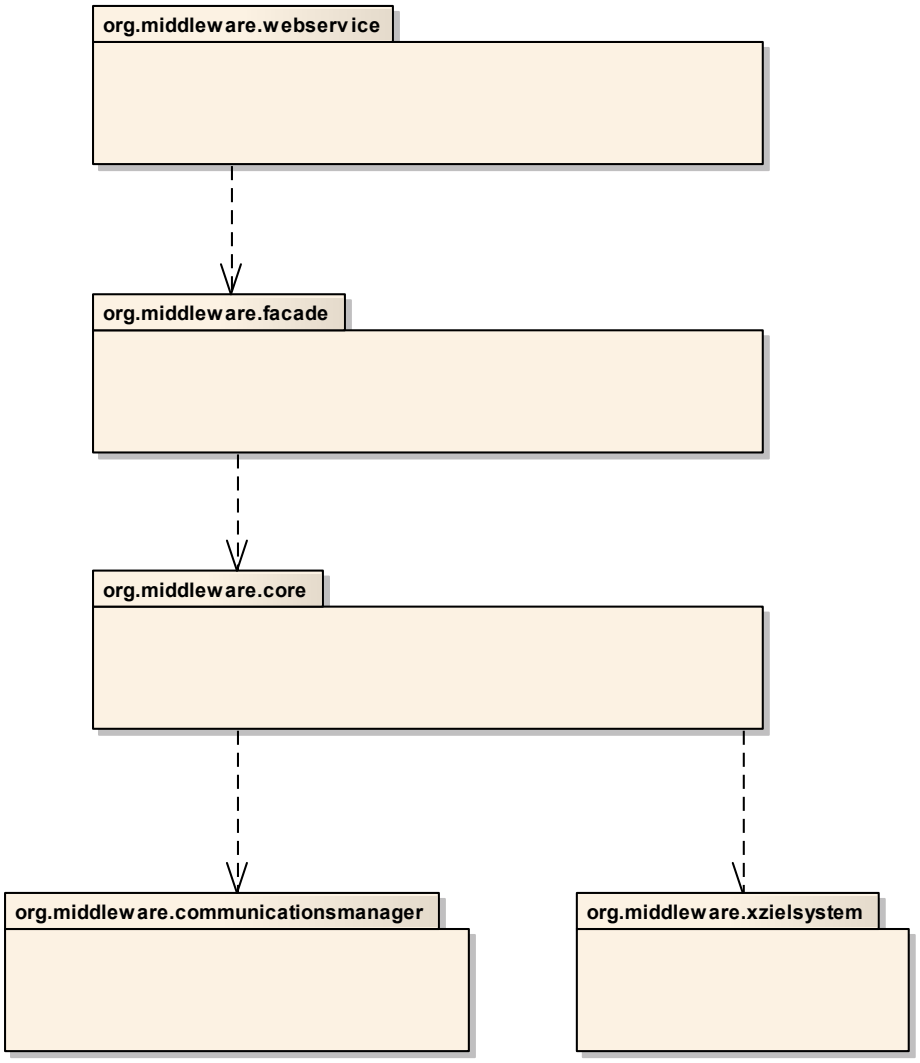


Abbildung 5-36 Package Diagramm Middleware Weiterentwicklung

Beispiel: Mabata will eine Nummer auf dem Communications Manager und auf dem xZielsystem erstellen. Im «org.middleware.façade» wird der empfangene XML-String in ein Objekt umgewandelt und an die richtige Methode in einer Klasse im «org.middleware.core» weitergeleitet. Im Core wird überprüft, welche Zielsysteme von diesem Aufruf betroffen sind. Nun werden die `addNumber`-Methoden in den `NumberManager` im «org.middleware.communicationsmanager» und «org.middleware.xzielsystem» aufgerufen. Diese beiden Packages greifen jeweils auf ein eigenes spezifisches Framework zu, mit welchem der Zugriff auf den Zielserver hergestellt wird.

5.7. Mabata-Adapter

Dieses Kapitel beschreibt die technische Umsetzung des Mabata Adapters. Die Umsetzung konnte wie geplant durchgeführt werden.

5.7.1. Package Diagramm

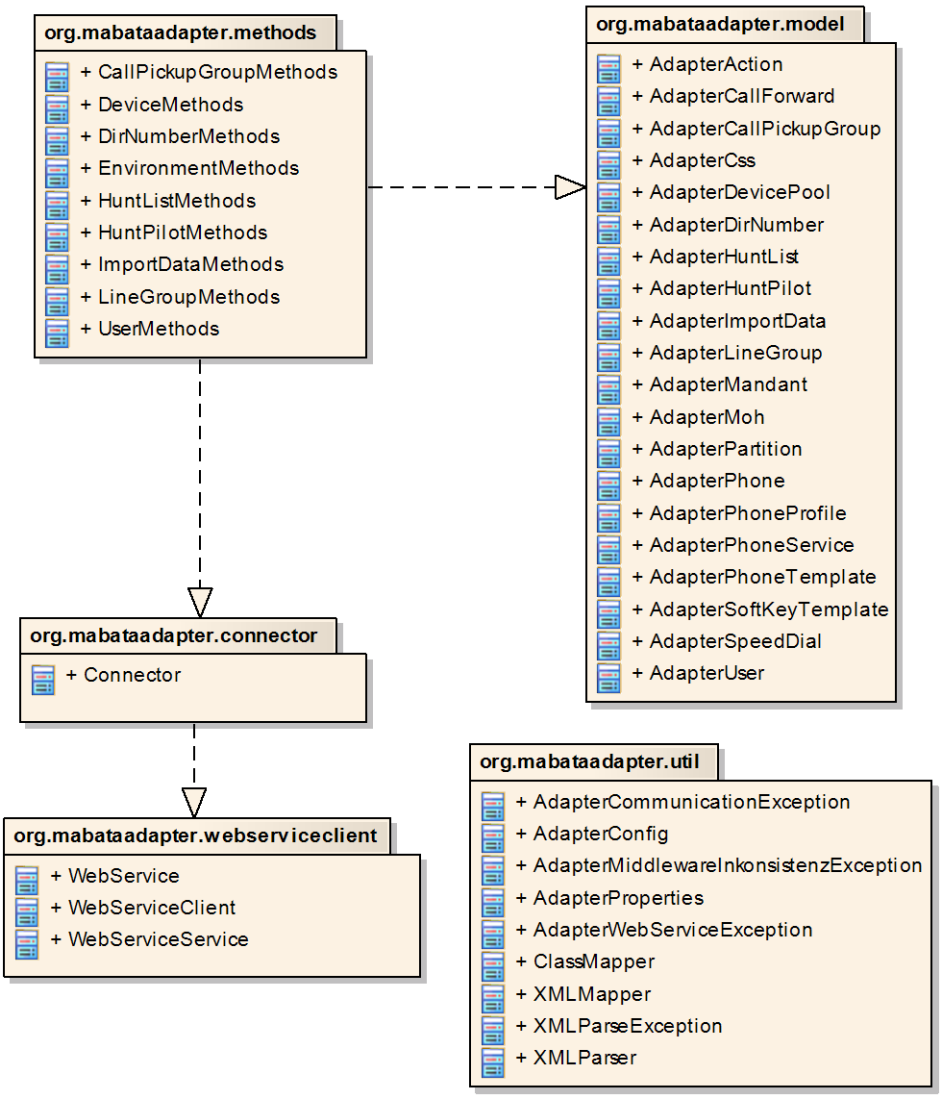


Abbildung 5-37 Package Diagramm Adapter

5.7.1.1. org.mabataadapter.methods

Dieses Package beinhaltet die Methoden welche für Mabata zur Verfügung gestellt werden. Jeder Zugriff auf die Middleware geschieht über ein Methods-Objekt, welches den Aufruf an die Middleware weiterleitet.

5.7.1.2. org.mabataadapter.connector

Die einzige in diesem Package enthaltene Klasse `Connector` sorgt dafür, dass immer nur eine Verbindung zum WebService besteht. Zudem fügt er dem XML-String den Benutzernamen sowie ein für die Authentifizierung benötigter Hash-Wert dazu. Dieser Hash-Wert wird aus dem Benutzernamen, dem Passwort sowie dem Objekt erstellt.

5.7.1.3. org.mabataadapter.webservice

Dies ist der Client des WebServices. Die Server Version läuft auf der Middleware. Der WebService leitet die Methodenaufrufe aus dem Package Methods weiter an die Middleware.

5.7.1.4. org.mabataadapter.model

Enthält eigene Models welche zusätzlich noch mit XML-Anweisungen annotiert wurden.

5.7.1.5. org.mabataadapter.util

Enthält Klassen, welche von verschiedenen Packages verwendet werden. Die Hilfsklasse für das XML-Marshalling und Unmarshalling, eine Klasse um die Properties-Datei auszulesen sowie eigene Exceptions.

5.7.2. Klassendiagramm

Das Klassendiagramm zeigt nicht alle Klassen, da diese sehr ähnlich aufgebaut sind. Alle Methods Klassen enthalten die vier Methoden, um die Aufrufe über den `Connector` an den WebService weiterzuleiten. Weiter enthalten sie noch eine Methode um den Rückgabewert der Middleware auf Fehler zu überprüfen.

Die beiden Klassen `ClassMapper` und `XMLMapper` enthalten je eine statische Methode um generisch aus dem XML-String ein Objekt oder umgekehrt zu erstellen.

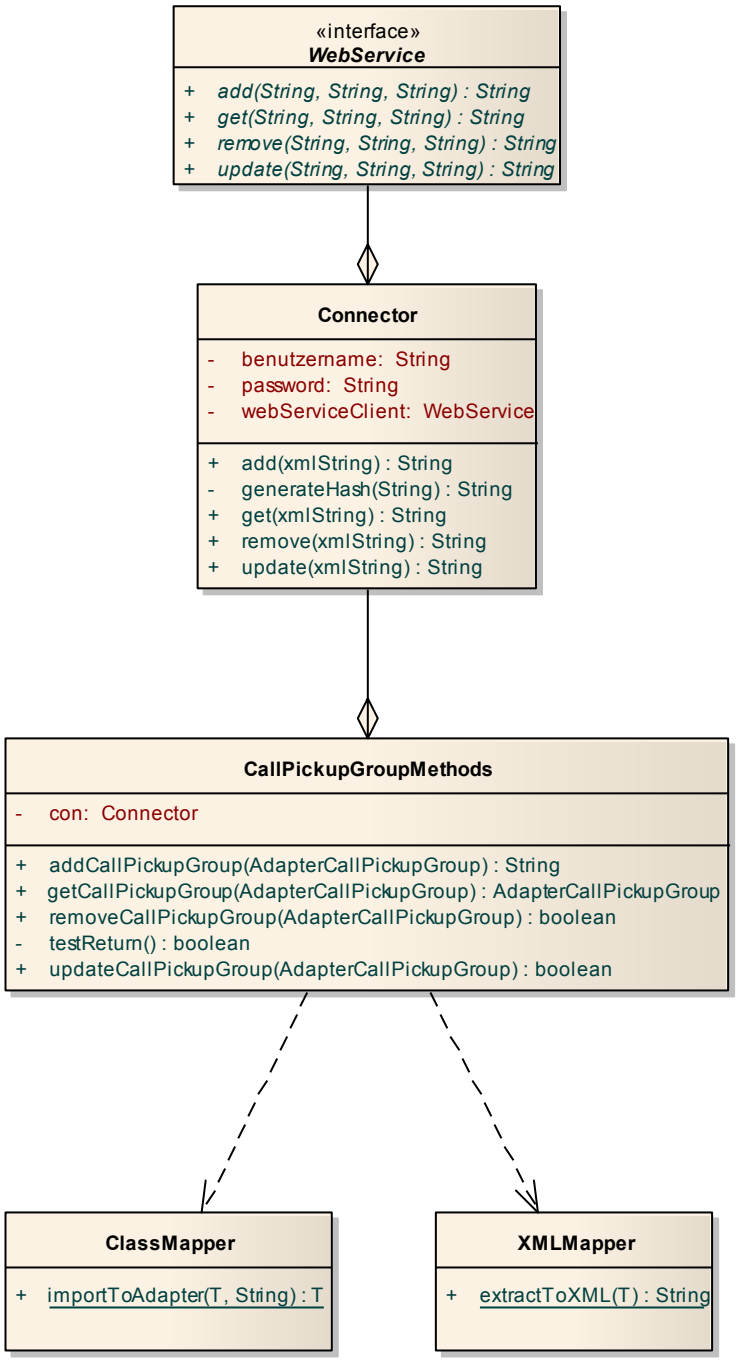


Abbildung 5-38 Klassendiagramm Adapter

5.7.3. Ablaufdiagramm

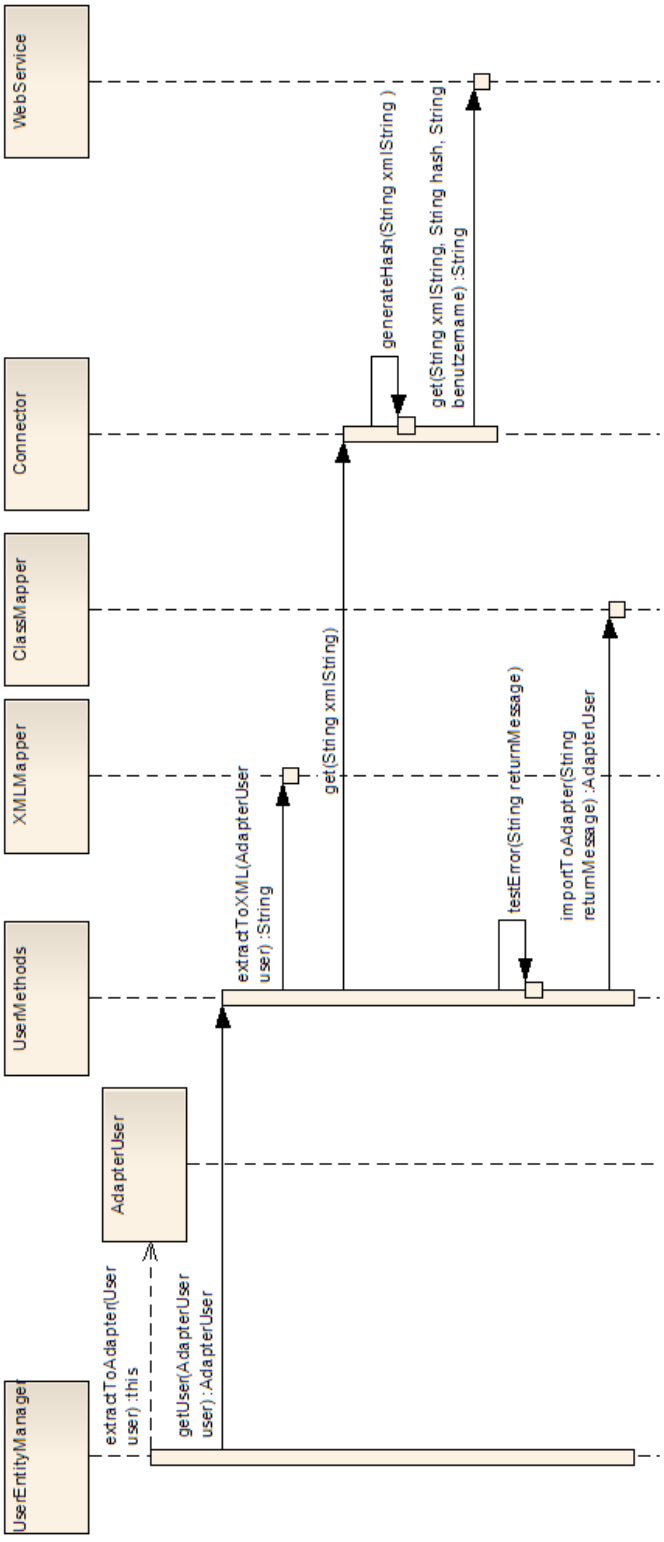


Abbildung 5-39 Ablaufdiagramm Adapter

Dieses Ablaufdiagramm zeigt den Ablauf der `getUsers`-Methode. Zuerst erstellt Mabata ein `AdapterUser`-Objekt und übergibt dieses Objekt an den Adapter. Dieser wiederum erstellt daraus einen XML-String für die Übertragung an den WebService. Die Antwort wird auf Fehler überprüft und wieder zurück in ein Objekt umgewandelt. Dieses wird schlussendlich an Mabata zurückgegeben.

5.8. Nummerntyp

Dieses Kapitel beinhaltet die Erweiterung des Nummerntyps um die Hunt- und Pickup Gruppen Funktion. Mittels dieser Erweiterung werden die von einer Hunt- oder Pickup Gruppe verwendeten Telefonnummern gekennzeichnet.

5.8.1. Benutzeroberfläche

5.8.1.1. Prototyp

Der Typ der Telefonnummern wird um Pickup und Hunt erweitert. Angezeigt wird dies durch zwei neue Symbole. Bei einem Klick öffnete sich die Detailansicht der Telefonnummer. Diese bleibt für die bisherigen Typen bestehen. Bei Pickup- und Hunt Nummern erscheint eine Information, dass diese Nummern durch Pickup- oder Hunt Gruppen verwendet werden und in dieser Ansicht nicht bearbeitet werden können.

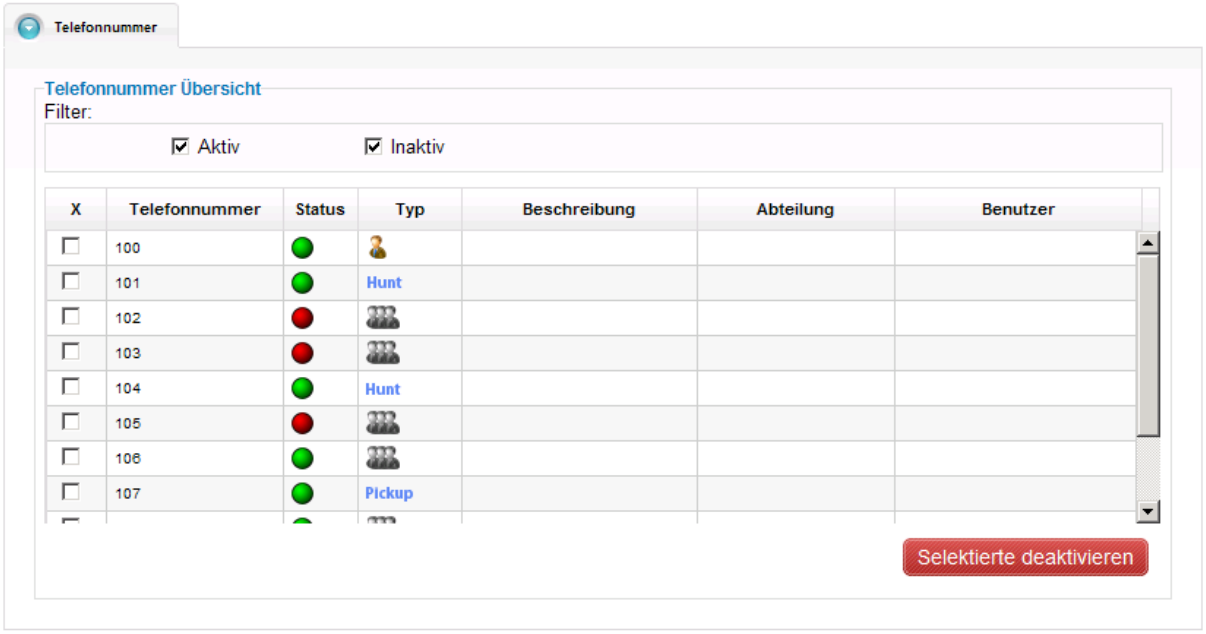


Abbildung 5-40 Prototype Nummerntyp

5.8.1.2. Umsetzung

Die Benutzeroberfläche wurde wie im Prototyp beschrieben Umgesetzt. Bei einem Klick auf eine Pickup- oder Hunt Nummer erscheint eine Meldung, dass diese Nummern nicht bearbeitet werden können.

5.8.2. Implementation

Im `DirNumber` Model wurde der Numbertype um die Einträge «PICKUP» und «HUNT» erweitert. Dadurch können die von einer Gruppe verwendeten Nummern markiert werden.

5.9. Reporting

5.9.1. Benutzeroberfläche

Die vom Administrator und Mandanten durchgeführten Änderungen müssen protokolliert und in Mabata ersichtlich sein. Mithilfe dieser Daten können Änderungen nachvollzogen werden. Zudem können diese Daten als Grundlage für das manuelle Rückgängig machen verwendet werden.

5.9.1.1. Prototype – Administrator-Ansicht

Die Übersicht aller Reporting Daten (nur Kategorie 5 – Notice) werden in der bestehenden Ansicht angezeigt. Durch einen Klick auf einen Eintrag wird die Detailansicht des Eintrages angezeigt. Unter «Einstellungen» kann der Debug-Modus aktiviert werden.

Fenster 1: Reporting Übersicht

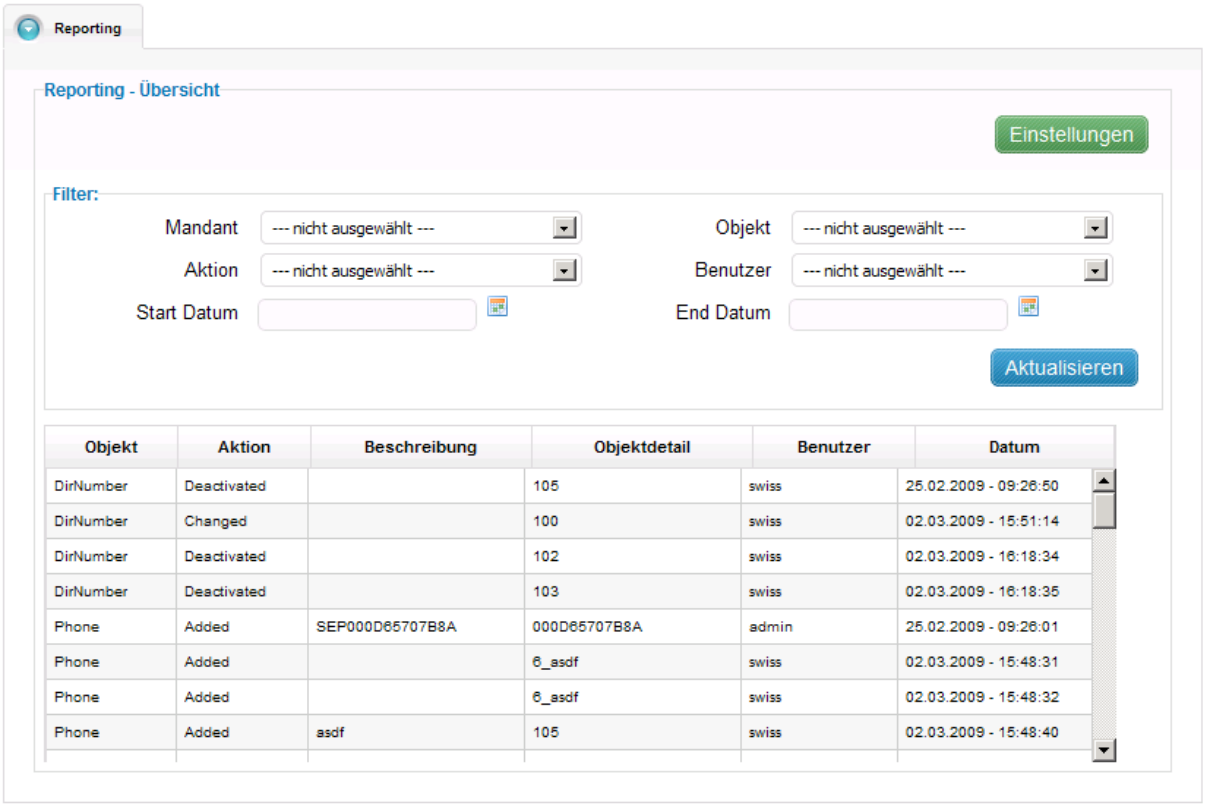


Abbildung 5-41 Prototyp Administratoroberfläche Reporting

Fenster 2: Detailansicht

In dieser Ansicht werden dem Administrator die Details zu dieser Transaktion angezeigt. Unter «Objektdetails» wird ein String mit den gesamten Attributen angezeigt. Falls es sich bei dieser Transaktion um das Editieren eines Objektes handelt, werden die alten sowie die aktualisierten Attribute angezeigt.

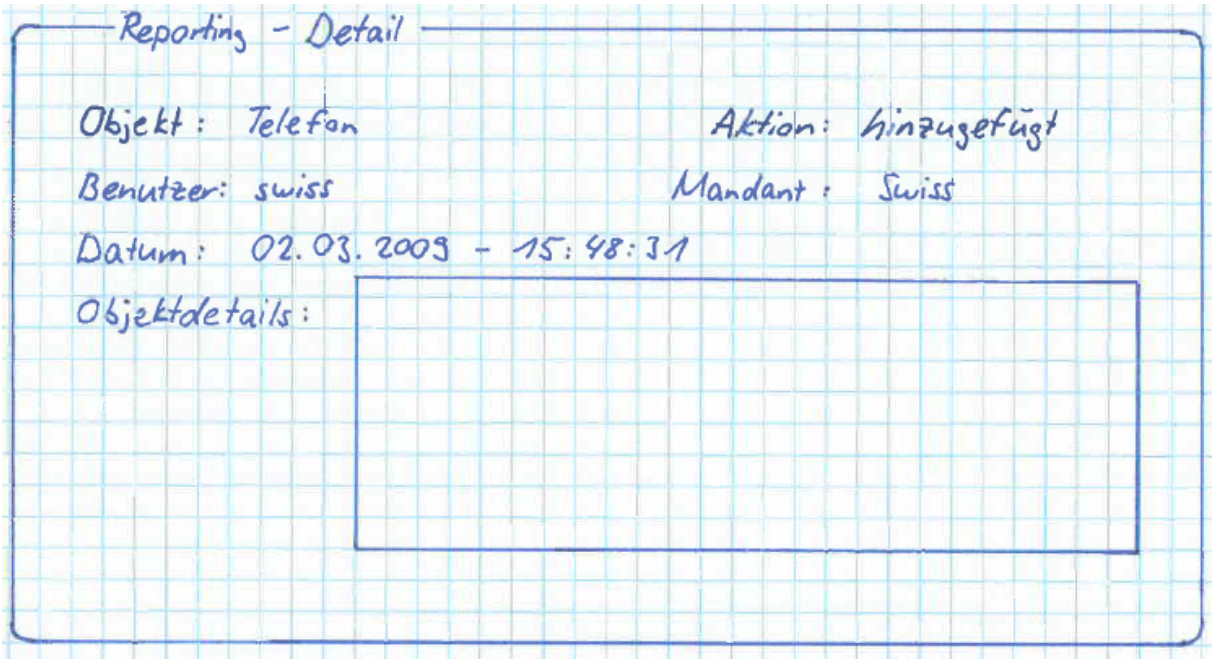


Abbildung 5-42 Prototyp Detailansicht Reporting

Fenster 3: Einstellungen

In den Einstellungen kann der Administrator den Syslog-Server konfigurieren, an welchen die Log-Daten gesendet werden sollen. Über Checkboxen können die Loglevels ausgewählt werden, welche an den Server übertragen werden sollen. Der Debug-Level wird demnach durch das Aktivieren der entsprechenden Checkbox eingeschaltet.

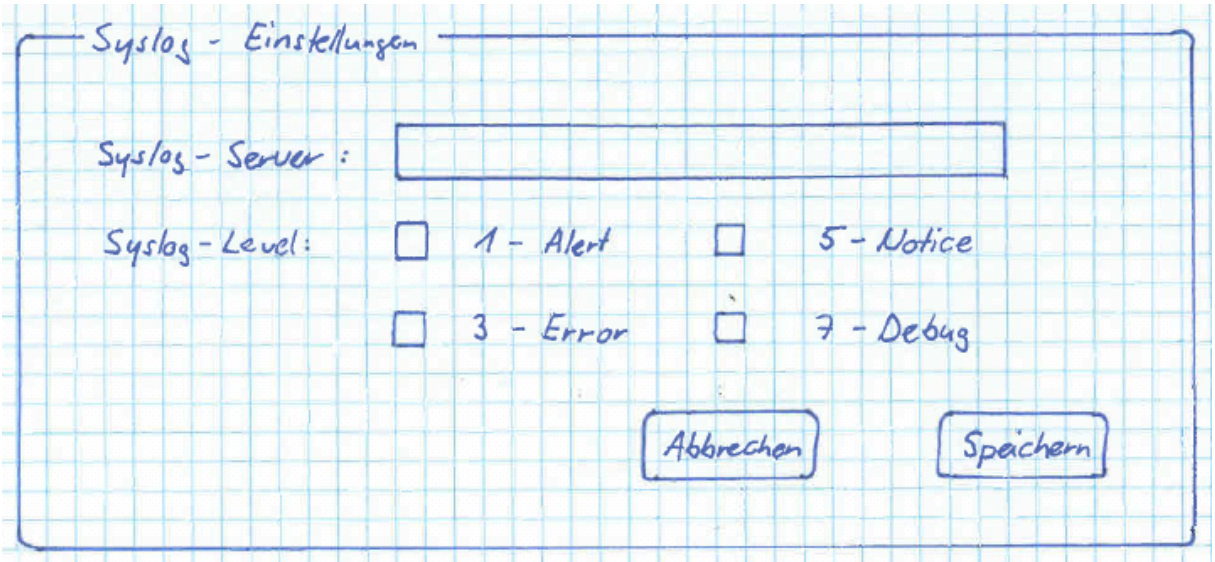


Abbildung 5-43 Prototyp Einstellungen Reporting

5.9.1.2. Prototype – Mandator-Ansicht

Die Benutzeroberfläche des Superusers entspricht grösstenteils der des Administrators. Einzig der Mandant kann auf der Benutzerebene des Superusers nicht ausgewählt werden und die Einstellungen für den Syslog-Server können nicht vorgenommen werden.

5.9.1.3. Umsetzung – Administrator-Ansicht

Fenster 1: Reporting Übersicht

Die Übersicht wurde grösstenteils wie im Prototyp geplant umgesetzt. Bei den Filter-Optionen wurde der Benutzer weggelassen, denn Objekte können nur vom Mandant oder vom Administrator erstellt, geändert oder gelöscht werden. Die Objektdetails wurden in die Spalte Attribute integriert, dadurch entfällt die Detailansicht aus dem Prototyp. Durch diese Spalte können die Objekte einfacher miteinander verglichen werden.

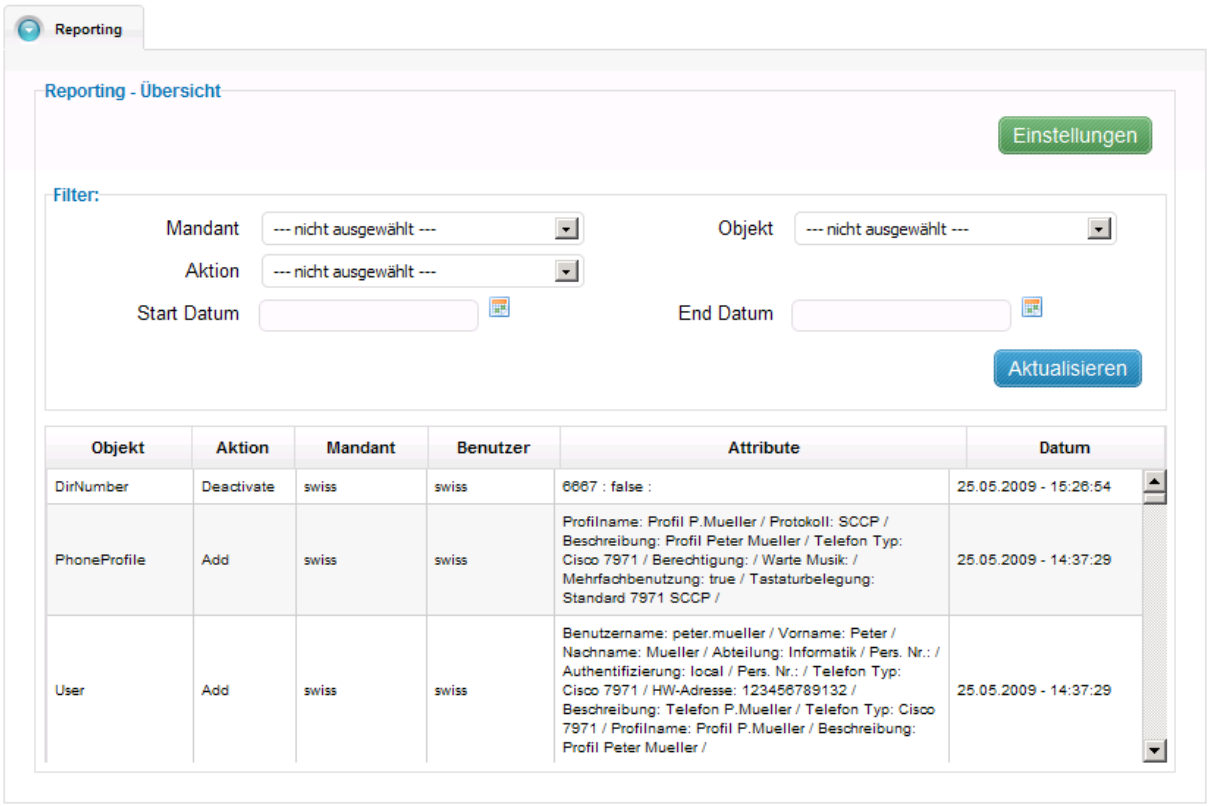


Abbildung 5-44 Umsetzung Administratoroberfläche Reporting

Fenster 2: Einstellungen

Diese Ansicht wurde gemäss Prototype umgesetzt.

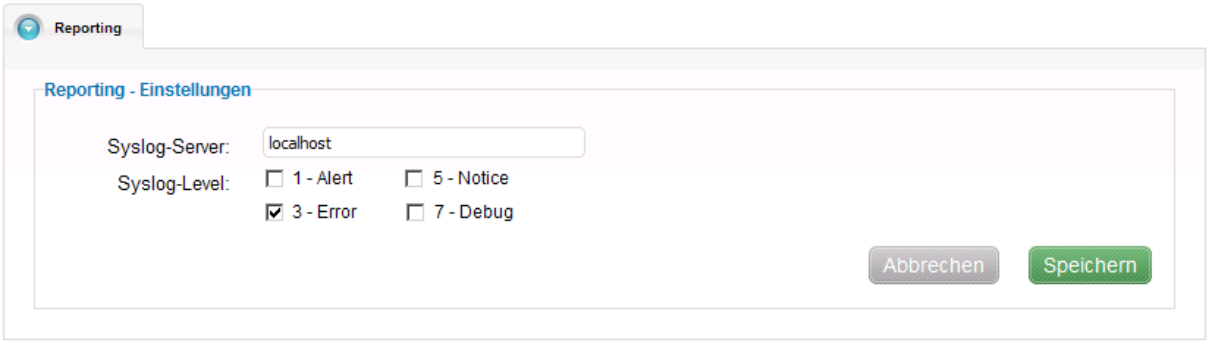


Abbildung 5-45 Umsetzung Einstellungen Reporting

5.9.1.4. Umsetzung – Mandator-Ansicht

Fenster 1: Reporting Übersicht

Der Mandant erhält dieselbe Ansicht wie der Administrator, jedoch werden nur die Einträge des eigenen Mandanten angezeigt.

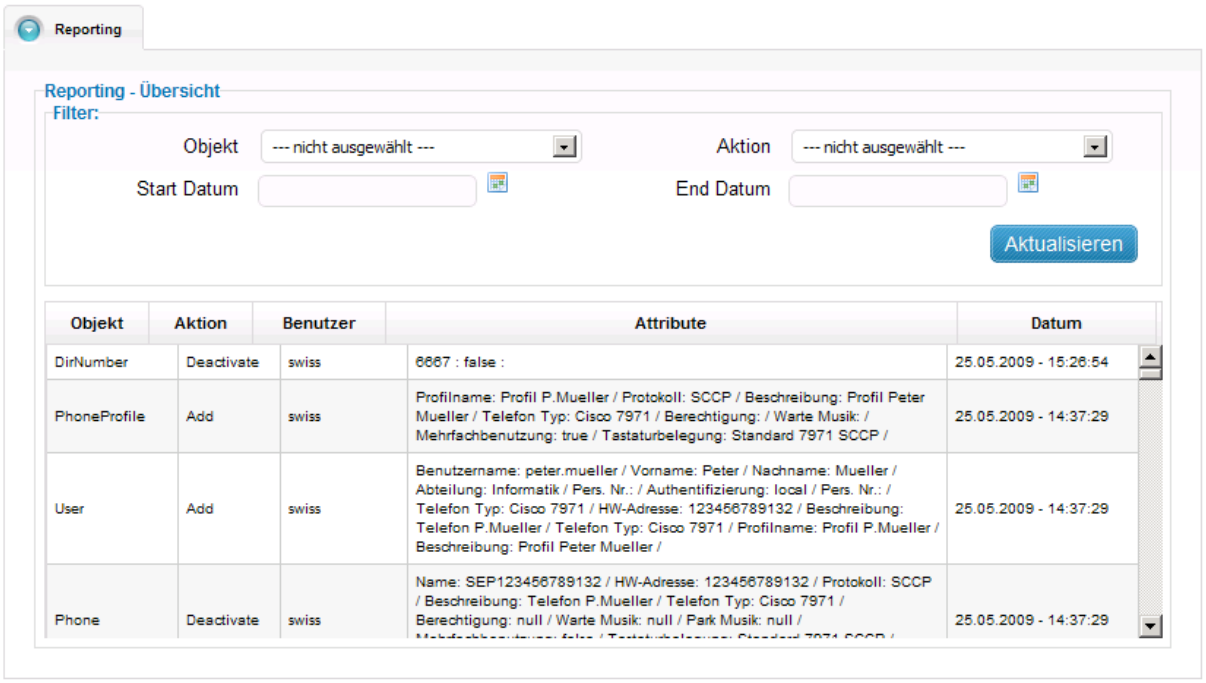


Abbildung 5-46 Umsetzung Mandatoroberfläche Reporting

5.9.2. Design

Das Design für die Reporting-Funktionalität musste komplett neu erstellt werden. Hierzu wurden die Models `DebugLogEntry`, `NoticeLogEntry` und `ErrorLogEntry` sowie ein neuer EntityManager (`LoggingEntityManager`) gemäss Abbildung 5-8 erstellt. Für das Model `NoticeLogEntry` musste zusätzlich eine neue Tabelle «Reporting» in der Mabata Datenbank erstellt werden (siehe Datenbankmodel im Kapitel 5.1.4).

5.9.2.1. Reporting Tabelle

In der «Reporting» Tabelle werden die vom Administrator und Mandanten durchgeführten Aktionen gespeichert. Die Tabelle ist wie folgt definiert:

Spaltenname	Definition	Beschreibung
logEntryID	bigint NOT NULL	ID des Datensatzes
modulo	integer	Wird für das Rotieren der Tabelle verwendet
username	character varying(255)	Angemeldeter Mabata-Benutzer
objectType	character varying(255)	Objekttyp (z.B. Telefon, Profil, Nummer, Benutzer, …)
action	character varying(255)	Durchgeführte Aktion (z.B. Add, Delete, Update, …)
attributs	text	Attribute des entsprechenden Objektes
date	timestamp	Datum des Logeintrages
mandatorName	character varying(255)	Name des Mandanten

Tabelle 5-3 Reportingtabelle

Spalte Attributs

Alle Attribute der Objekte (z.B. Benutzername, Vorname, Nachname, Abteilung, …) werden zu einem String zusammengefügt und in dieser Spalte abgespeichert. Mithilfe dieser Informationen können Änderungen nachvollzogen und gegebenenfalls auch rückgängig gemacht werden.

Spalte mandatorName

Es wird bewusst nur der Name des Mandanten und nicht die ID gespeichert. Wird ein Mandant gelöscht, können die Datensätze von diesem Mandanten anhand des Namens immer noch identifiziert werden.

5.9.3. Implementation

5.9.3.1. Log-Einträge

Log-Einträge werden über die statische Klasse `LoggingEntityManager` erstellt. Der Methode `logEvent` wird ein Objekt welches das Interface `ILogEntry` implementiert, übergeben. Dieses Interface besitzt die Methode `getLogString`, welche den String für den Logeintrag zurückgibt. Zusätzlich zu diesem Objekt wird der `logEvent` Methode auch der Log-Level (Alert, Error, Notice, Debug) übergeben. Anhand des Log-Levels wird anschliessend entschieden, ob der Eintrag in die Datenbank, Log-Datei oder an den Syslog-Server gesendet wird.

5.9.3.2. Notice Log-Einträge

Notice Log-Einträge sind die vom Administrator und Mandanten durchgeführten Aktionen. Diese werden durch die EntityManager der Objekte in den entsprechenden Methoden (z.B. addPhone, deletePhone, updatePhone) erstellt. Um bei einem Update die geänderten Attribute ersichtlich zu machen, wird jeder update-Methode immer das Aktualisierte sowie das bisherige Objekt übergeben. Anschliessend werden zwei Log-Einträge erstellt, einmal für das bisherige und einmal für das aktualisierte Objekt.

5.9.3.3. Error Log-Einträge

Error Log-Einträge enthalten die Syslog-Meldungen der Kategorie 1 (Alert) und 3 (Error). Diese werden durch die Exceptions generiert, welche im Fehlerfall geworfen werden. Mittels log4j werden diese Einträge in eine Log-Datei geschrieben.

5.9.3.4. Debug Log-Einträge

Mit den Debug Log-Einträgen wird der gesamte Mabata-Core protokolliert. Diese Meldungen werden durch die EntityManager erstellt und können nur an einen Syslog-Server übertragen werden. Hierzu wird wiederum log4j verwendet.

5.9.3.5. Syslog-Einstellungen

Die vom Administrator konfigurierbaren Syslog-Einstellungen werden in einer Property-Datei «log4j.properties» abgespeichert. In dieser Datei kann zudem auch der Pfad zur Log-Datei sowie die Dateigrösse definiert werden.

5.9.3.6. Logrotate Datenbank

Um ein kontinuierliches Wachsen der Datenbank zu verhindern wurde die Spalte «modulo» eingefügt. Vor jedem Einfügen wird die höchste ID + 1 aus der Datenbank ausgelesen. Die Spalte «modulo» ist nun der modulo Wert dieser neuen ID. Ist beim Einfügen des neuen Eintrages dieser Wert bereits in der Spalte «modulo» vorhanden, wird der alte Eintrag überschrieben. Damit wird gewährleistet, dass die «Reporting» Tabelle nicht unaufhörlich wächst.

5.10. Kostenabrechnung

Um die Kostenabrechnung zu erstellen wird eine neue Tabelle in der Datenbank erstellt. In dieser Tabelle werden die Daten, welche zur Berechnung der Kosten benötigt werden, abgelegt. Wird ein neues Objekt erstellt, wird automatisch ein Eintrag in dieser Tabelle erstellt. Das aktuelle Datum wird in der Spalte «Erstellt» eingetragen. Bei der Erstellung der Abrechnung wird die Differenz der beiden Daten («Erstellt» und «Gelöscht») zur Berechnung der Anzahlt der Tagessätze verwendet. Ist noch kein «Gelöscht» Datum eingetragen, wird das Abrechnungsdatum verwendet.

Objekt	ID	Beschreibung	Mandant	Erstellt	Gelöscht
Telefon	000D65707B8A	Telefon Empfang	Swiss	12.12.2008	
Benutzer	Pmueller	Peter Müller	Swiss	01.01.2009	28.02.2009
Telefonnummer	456845	Peter Müller	Swiss	01.01.2009	28.02.2009
Telefon	000D65707A41	Telefon Support	Swiss	15.02.2009	20.02.2009

Tabelle 5-4 Kostenabrechnung

5.10.1. Benutzeroberfläche Prototype

Die Ansicht gibt dem Benutzer eine Einsicht in die aktuelle Kostenabrechnung. Zugleich erhält er die Möglichkeit, die Kosten von vergangenen Zeitperioden anzuzeigen und diese als CSV-Datei zu exportieren.

5.10.1.1. Administrator

Reporting Übersicht

In dieser Ansicht kann der Administrator die Kosten der einzelnen Mandanten über festgelegte Zeiträume abrufen. Die angezeigten Daten können zusätzlich als CSV-Datei extrahiert werden und dadurch in andere Systeme importiert werden.

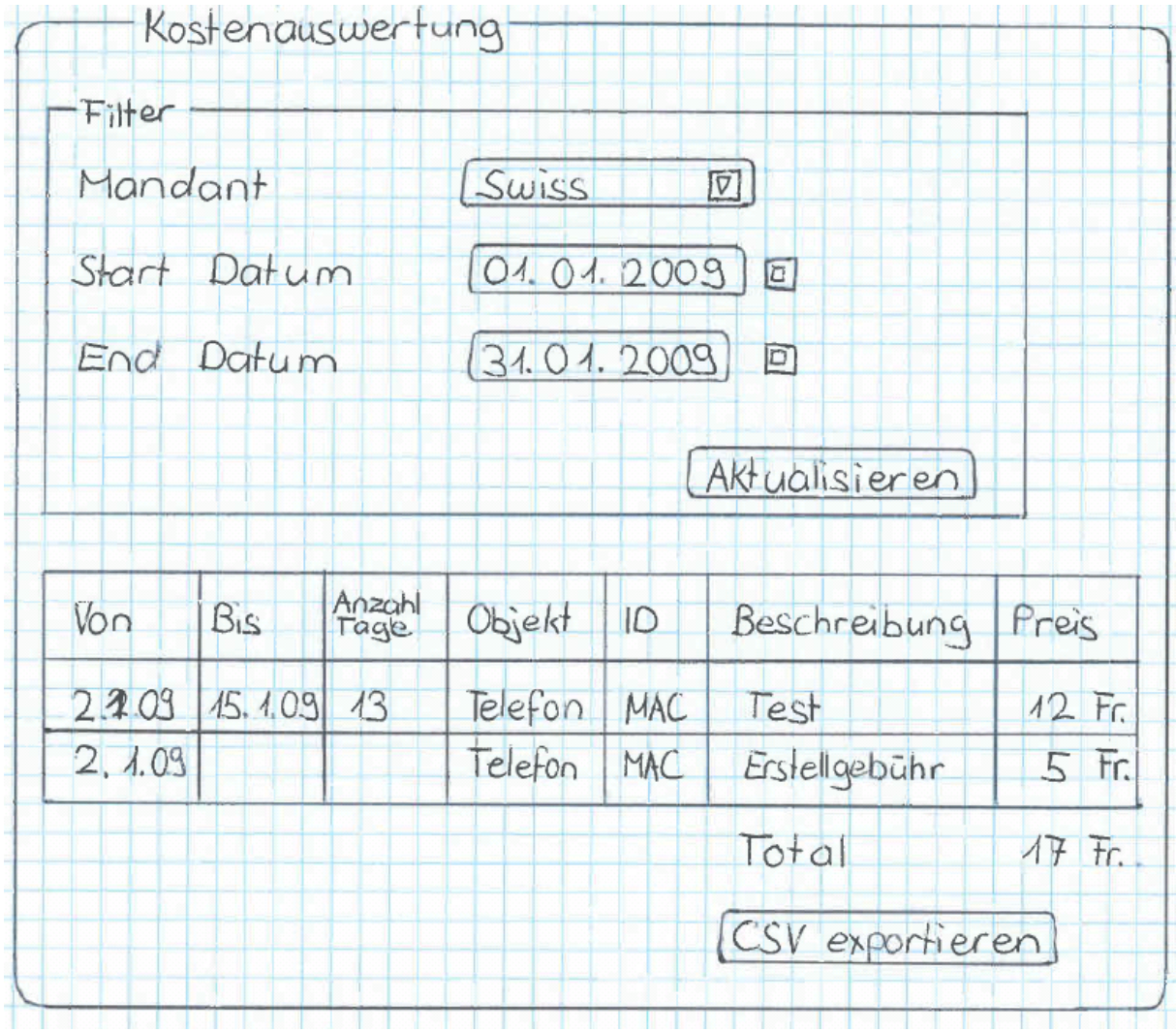


Abbildung 5-47 Paper Prototype Kostenauswertung

Kosten einrichten

Damit der Administrator die Einrichtungspauschalen sowie die Tagessätze für jeden Mandanten festlegen kann, wird die folgende Benutzeroberfläche erstellt.

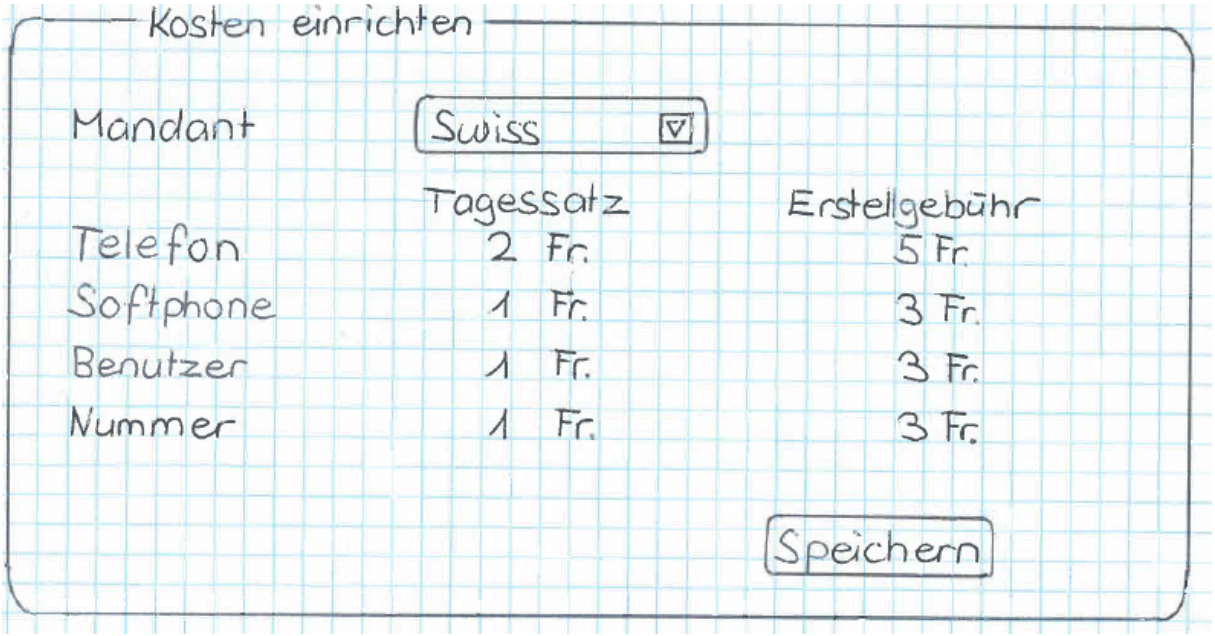


Abbildung 5-48 Paper Prototype Kosten einrichten

5.10.1.2. Superuser

Der Superuser erhält dieselbe Ansicht (Reporting Übersicht) wie der Administrator. Einzig die Auswahl des Mandanten bleibt ihm verborgen.

5.10.2. Externe Schnittstelle

Da die Buchhaltung verantwortlich für die Rechnungen ist, wird eine externe Schnittstelle angeboten, mit welcher eine Applikation eine Rechnung eines Mandanten abfragen kann. Dabei muss der Mandant, Start- und Enddatum angegeben werden. Die Rückgabe enthält einen CSV-String, welcher anschliessend weiter verarbeitet werden kann.

```
public String getBill(String mandant, Date startDate, Date endDate);
```

5.11. SoftKey Template

In diesem Kapitel wird beschrieben wie die SoftKey Templates realisiert werden.

5.11.1. Voraussetzung

Damit die Tasten «Pickup»/«GPickup» zur Verfügung stehen, müssen vorgängig entsprechende SoftKey Templates auf dem Communications Manager erstellt werden. Dies wurde im Kapitel 1.2.2 Communications Manager Konfiguration beschrieben.

5.11.2. Benutzeroberfläche - Mandator-Ansicht

Die Telefondetailansicht wird um einen zusätzlichen Parameter «SoftKey Template» erweitert. Mit diesem kann dem Telefon ein zugehöriges SoftKey Template zugewiesen werden. Es werden alle SoftKey Templates angezeigt die auf dem Communications Manager verfügbar sind.

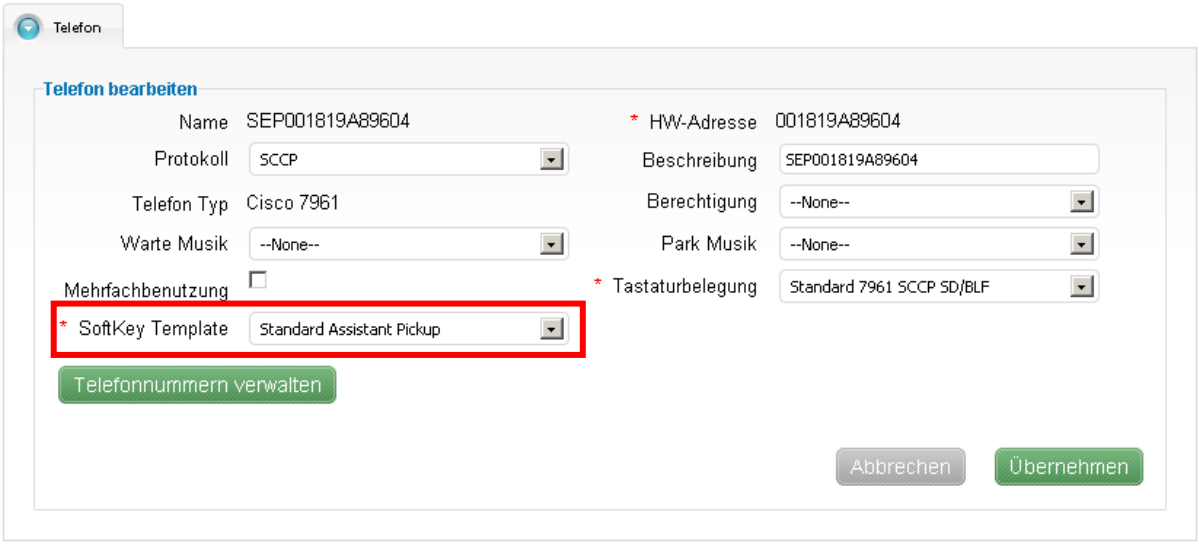


Abbildung 5-49 Benutzeroberfläche SoftKey Template

5.12. PhoneButton Template mit Busy Lamp

In diesem Kapitel wird beschrieben wie die Kurzwahltasten mit der Busy Lamp Funktion erweitert werden.

5.12.1. Voraussetzung

Damit die Funktion Busy Lamp verwendet werden kann, müssen vorgängig Busy Lamp PhoneButton Templates auf dem Communications Manager erstellt werden. Dies wurde im Kapitel 1.2.2 Communications Manager Konfiguration beschrieben.

5.12.2. Benutzeroberfläche – Mandator-Ansicht

Für den Superuser ändert sich an der Oberfläche von Mabata nichts. Er hat jetzt zusätzliche PhoneButton Templates zur Auswahl, die er den Telefonen zuweisen kann.

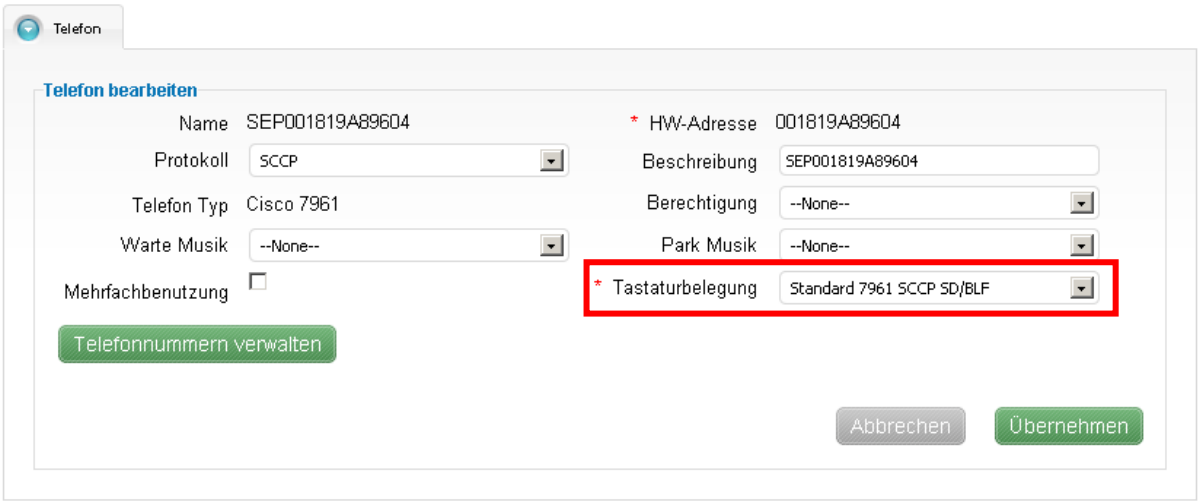


Abbildung 5-50 Benutzeroberfläche PhoneButton Template

5.12.3. Erweiterung

Bis anhin wurde in der Auswahlliste «Tastaturbelegung» alle vorhandenen Templates angezeigt. Neu sollen nur noch diejenigen angezeigt werden, welche auch vom Telefon unterstützt werden. Wird ein unbekannter Telefontyp eingegeben, werden alle Templates angezeigt.

5.13. Services Framework

In diesem Kapitel wird beschrieben wie das bestehende Services Framework durch eine eigenständige Version, welche über Mabata konfigurierbar ist, abgelöst werden kann.

5.13.1. Reduzierung bestehendes Projekt

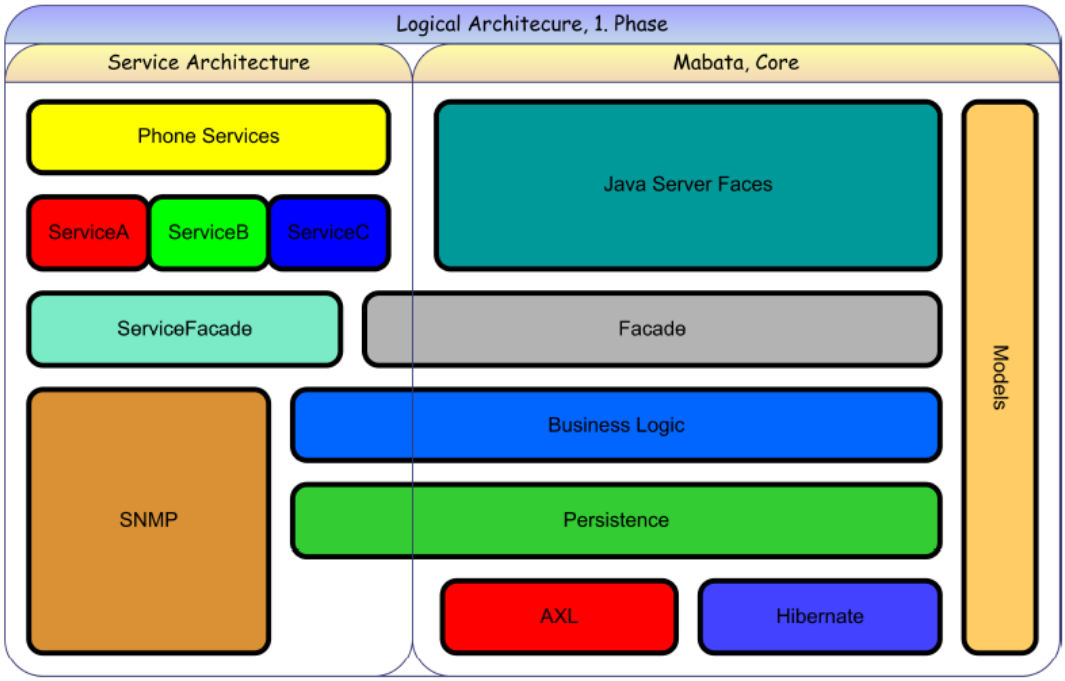


Abbildung 5-51 Bestehende Architektur Services Framework

Momentan besteht das Services Framework gemäss Abbildung 5-51. Im Grunde dessen, dass die im Services Framework verwendete Mabata-Version nicht mehr aktuell ist, wird diese Architektur komplett entfernt. Das Services Framework wird dadurch unabhängig und eigenständig von Mabata lauffähig. In Abbildung 5-52 ist gezeigt, wie die Architektur neu aussieht:

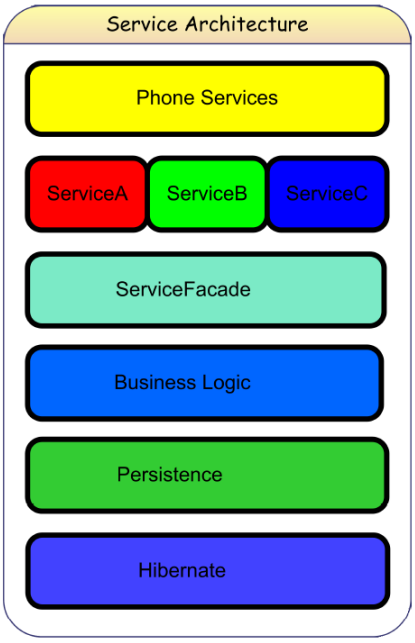


Abbildung 5-52 Neue Architektur Services Framework

5.13.2. Interaktion Telefon – Services Framework – Service

Folgendes Sequenzdiagramm soll das Zusammenspiel zwischen Telefon, Services Framework und Service aufzeigen:

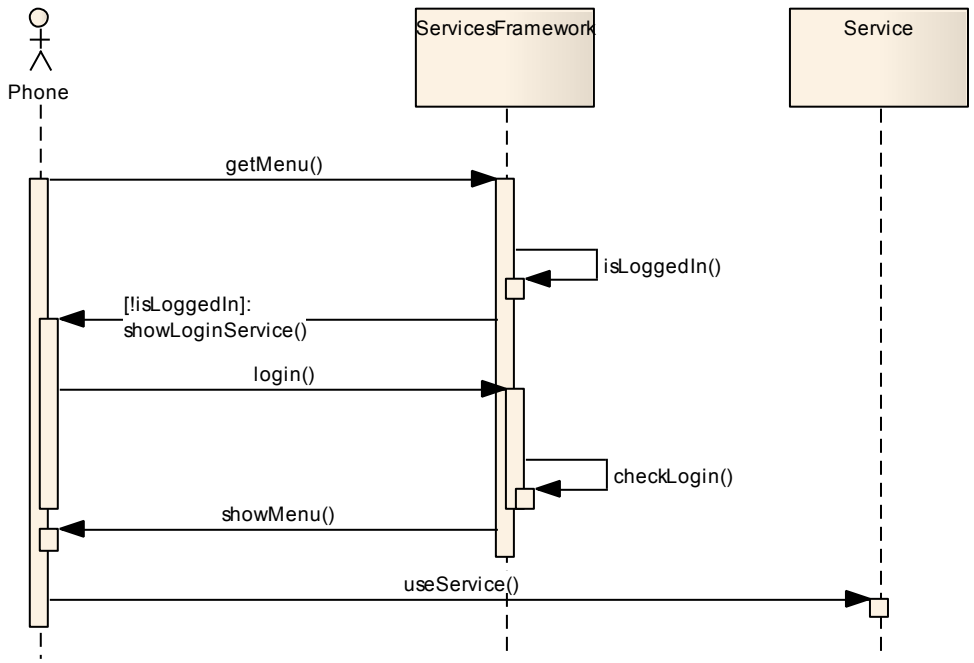


Abbildung 5-53 Interaktion Services Framework - Services

Falls der Benutzer auf seinem Telefon den Services Button drückt, schickt das Telefon dem Services Framework eine `getMenu()` Message. Das Services Framework prüft, ob der Benutzer angemeldet ist oder nicht. Falls nicht, schickt er ein «LoginService» zum Telefon, mit welchem sich der Benutzer anmelden kann. Nachdem der Benutzer erfolgreich angemeldet wurde, werden alle Services, welche der Benutzer konfiguriert hat, an das Telefon geschickt. Wählt ein Benutzer einen Service aus, kommuniziert das Telefon direkt mit dem ausgewählten Service.

5.13.3. Anmeldeverfahren

Da im Services Framework das Anmeldeverfahren für den Benutzer zu lange dauert, muss dieses geändert werden.

5.13.3.1. Altes Verfahren

Aktuell bietet das Services Framework zwei verschiedene Verfahren an:

Weak-Auth — Dieses Verfahren ermittelt anhand den Telefon- bzw. Netzwerk-Eigenschaften den Mandanten des aktuell angemeldeten Benutzers. Somit geschieht die Anmeldung automatisch. Der Benutzer merkt von diesem Prozess nichts.

Secure-Auth — Der Benutzer meldet sich mit Benutzernamen und Passwort an.

Das Weak-Auth Verfahren ist für die aktuellen Anforderungen an das Services Framework unbrauchbar, da nur auf Stufe Mandant unterschieden werden kann. Die Anforderungen sind aber, dass auf Stufe Benutzer unterschieden werden muss. Denn dem Benutzer soll die Möglichkeit geboten werden, selbst zu bestimmen welche Services er auf seinem Telefon hat. Zudem müssen die Mandanten einzelne Services nur ausgewählten Benutzern zur Verfügung stellen können.

5.13.3.2. Neues Verfahren

Der Benutzer wird automatisch angemeldet, somit entfällt die manuelle Anmeldung. Wird Extension Mobility eingesetzt, kann der auf dem Telefon angemeldete Benutzer über den Communications Manager ermittelt werden. Falls kein Extension Mobility verwendet wird, muss in Mabata eine Zuweisung zwischen Benutzer und Telefon erfolgen. Um dies zu bewerkstelligen, muss das GUI von Mabata erweitert werden. Durch diese Zuweisung kann für ein Telefon der entsprechende Benutzer ermittelt und die gewünschten Services angezeigt werden.

5.13.3.3. Optionale Anmeldung

Durch das neue Verfahren bleiben die Services auf den Telefonen, welche nicht mit Extension Mobility verwendet werden, immer zugänglich. Um den Zugriff auf private Daten (wie z.B. privates Telefonbuch) zu schützen, muss dem Benutzer eine An- und Abmeldung zur Verfügung gestellt werden. Dadurch erhält er die Möglichkeit, sich vom Telefon abzumelden, falls er z.B. seinen Arbeitsplatz verlässt.

Die Anmeldung wird durch einen globalen Service zur Verfügung gestellt. Der Benutzer meldet sich daraufhin mit seinem Benutzernamen und PIN an. In einer Mabata Umgebung können die Credentials von Mabata überprüft werden. Andernfalls wird vom Services Framework eine Authentifizierung auf dem Communications Manager durchgeführt. Ist diese erfolgreich, sind die Credentials korrekt und die Anmeldung kann erfolgreich abgeschlossen werden.

5.13.4. Architektur

Wie in den Anforderungen beschrieben muss das Services Framework eigenständig (ohne Mabata) lauffähig sein. Um dies zu bewerkstelligen wird eine Datenbank für das Framework benötigt. In ihr werden die Services sowie die Service-Benutzer Beziehungen gespeichert.

Zugleich muss die Möglichkeit bestehen, das Services Framework in Mabata zu integrieren. In einer solchen Umgebung werden die Services aus der Mabata-Benutzeroberfläche konfiguriert.

Demzufolge kann das Services Framework zusammen mit Mabata oder auch alleine betrieben werden.

5.13.4.1. Umgebung mit Mabata

In dieser Umgebung wird Mabata zur Administration des Services Frameworks verwendet. Das Services Framework erhält über einen WebService Zugriff auf die Mabata Datenbank. Durch diesen WebService wird ein direkter Zugriff des Frameworks auf die Mabata Datenbank umgangen. Zugleich ist das Framework dadurch auch funktionsfähig, wenn Mabata nicht ausgeführt wird.

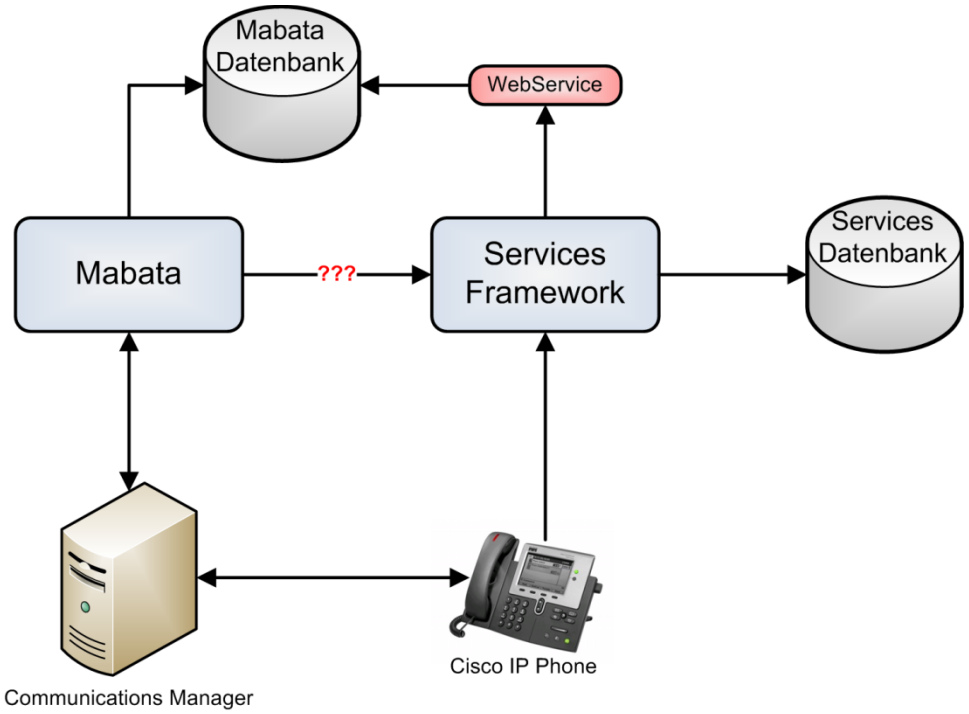


Abbildung 5-54 Services Framework mit Mabata

Für die Administration der Services durch Mabata wird eine Kommunikation zwischen Mabata und dem Services Framework benötigt. Für diese Kommunikation werden anschliessend zwei Lösungsvarianten ausgearbeitet.

Lösungsvariante 1

Bei dieser Lösungsvariante wird eine grafische Benutzeroberfläche im Services Framework erstellt. Um nun das Services Framework von Mabata aus konfigurieren zu können, wird diese Benutzeroberfläche in das GUI von Mabata eingebunden. Die URL des Services Framework wird als zusätzlicher Parameter in dem Konfigurationstool angegeben.

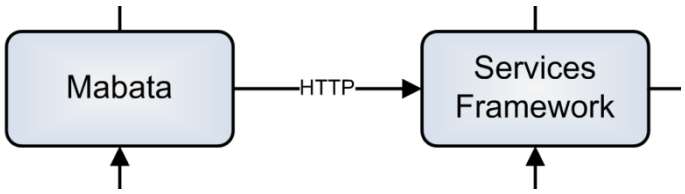


Abbildung 5-55 Verbindung GUI Services Framework Variante 1

Durch diese Variante wird verhindert, dass zwei GUIs erstellt werden müssen welche praktisch die gleiche Funktionalität bereitstellen.

Lösungsvariante 2

In dieser Lösungsvariante wird das Services Framework mit einem WebService ausgestattet. Über diesen WebService können Applikationen, wie z.B. Mabata auf die Funktionen des Services Framework zugreifen.

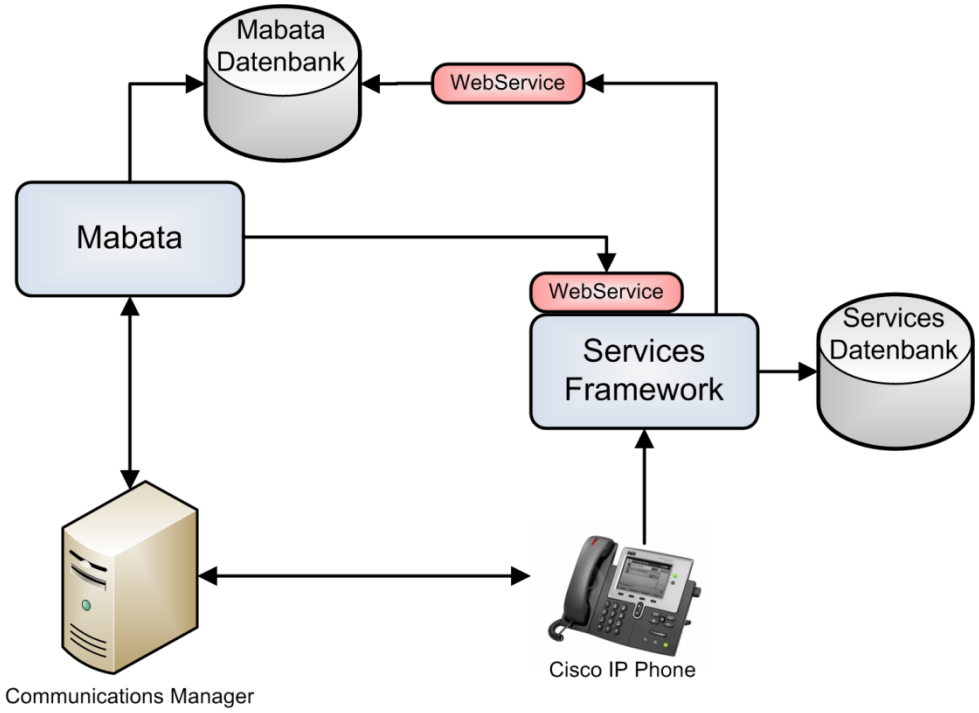


Abbildung 5-56 Verbindung GUI Services Framework Variante 2

Der Nachteil dieser Variante besteht darin, dass jede Applikation, welche das Services Framework benutzt, eine eigene grafische Benutzeroberfläche für diese Funktionen zur Verfügung stellen muss. Durch die Verwendung eines WebServices können die Funktionen des Frameworks jedoch von verschiedenen Applikationen verwendet werden.

Fazit

Die Architekturidee hat sich grundsätzlich als gut herausgestellt. In den Diskussionen stellte sich heraus, dass das Services Framework die Benutzer vom Communications Manager beziehen muss, da dieser für die Benutzer den Datenmaster darstellt.

Das GUI des Services Frameworks soll in Mabata eingebunden werden. Dies ergibt den Vorteil, dass Änderungen welche in der Benutzeroberfläche zu machen sind, nur einmal durchgeführt werden müssen.

Die finale Architektur, welche im nächsten Kapitel beschrieben wird baut auf all den vorangegangen Überlegung auf.

Finale Architektur

Aus den Designüberlegungen, welche in den vorangegangenen Kapiteln beschrieben wurden resultiert die nachstehende Architektur.

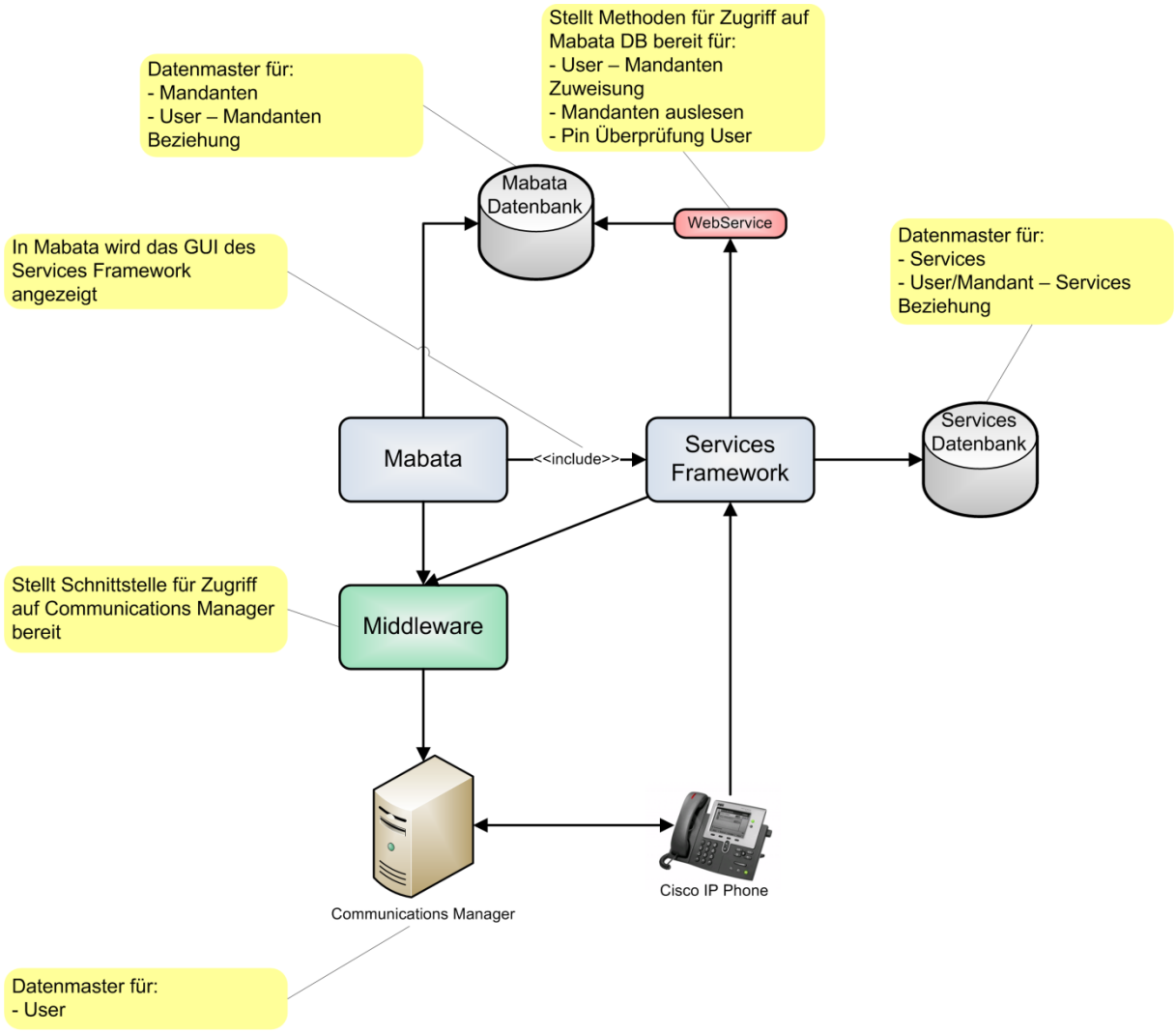


Abbildung 5-57 Finale Architektur Services Framework mit Mabata

Mabata weiss selber nichts von den Services. Das heisst, dass in der Mabata Datenbank keine Informationen über Services abgespeichert werden. Einzig wird in Mabata das GUI des Services Frameworks eingebunden, damit aus Mabata auch Services erstellt und konfiguriert werden können.

Das Services Framework hat eine Verbindung über die Middleware zum Communications Manager. Da die Verbindung über die Middleware hergestellt wird, können bestehende Funktionen dafür verwendet werden. Das Services Framework muss vom Communications Manager die Benutzer bekommen, damit diesen Services zugewiesen werden können. Weiter muss das Services Framework vom Communications Manager die Information bekommen, welcher Benutzer aktuell an einem Telefon angemeldet ist. Dies wird jedoch nur bei Extension Mobility unterstützt. Falls kein Extension Mobility eingesetzt wird, muss in Mabata eine Zuweisung zwischen Telefon und Benutzer erfolgen. Durch diese Zuweisung kann für jedes Telefon (anhand der MAC-Adresse) der dazugehörende Benutzer ermittelt werden.

In der Services Datenbank werden alle Information über die Services abgespeichert. Um Services Benutzern und Mandanten zuweisen zu können, müssen auch diese beiden Objekte in der Datenbank erfasst werden. Jedoch haben Benutzer und Mandanten in dieser Datenbank keine Beziehung. Diese Beziehung wird nur in der Mabata Datenbank abgebildet.

Um im Services Framework herauszufinden, zu welchem Mandant ein Benutzer gehört, wird ein WebService der Mabata Datenbank vorangestellt, welcher diese Abfrage tätigen kann. Der WebService stellt zudem eine Funktion zur Verfügung, mit welcher das Services Framework die Mandanten abfragen kann, um sie in der eigenen Datenbank zu speichern.

Datenbankschema

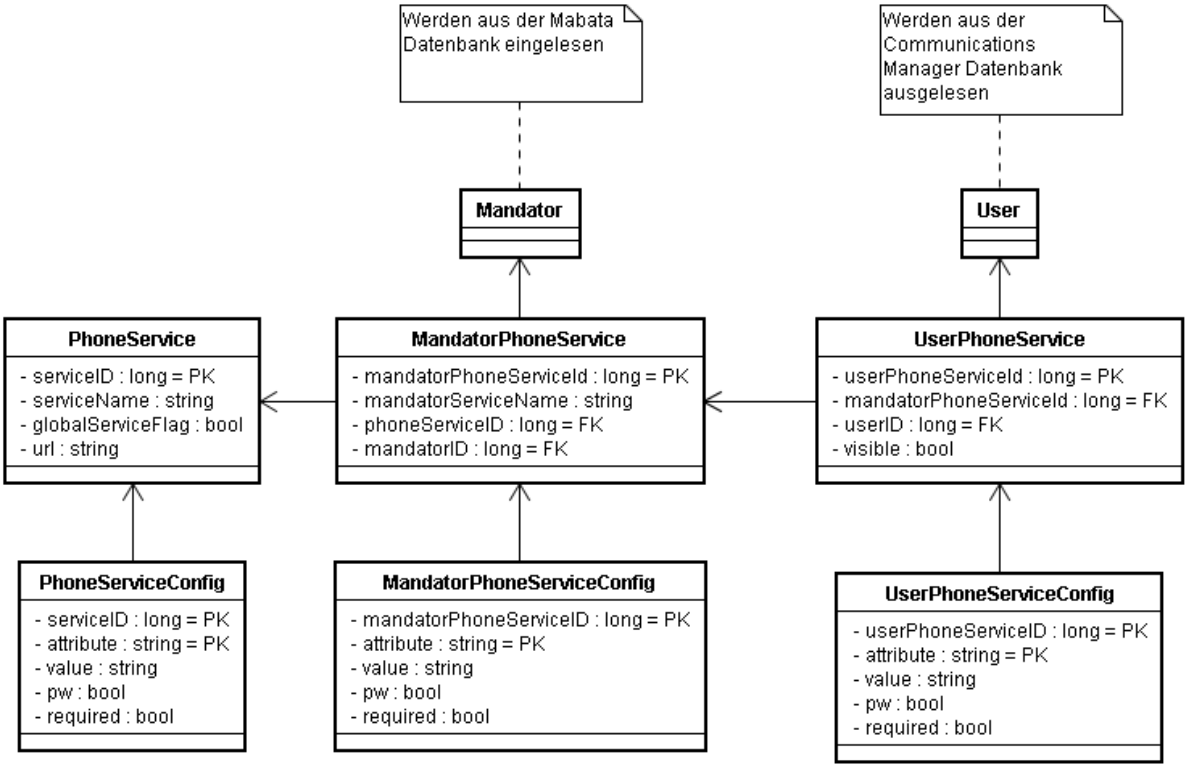


Abbildung 5-58 Datenbankschema Services Framework mit Mabata

Das Datenbankschema umfasst alle Daten die benötigt werden um die Services und deren Zugehörigkeit abzuspeichern. Die restlichen Daten werden entweder aus der Mabata Datenbank oder aus dem Communications Manager ausgelesen.

5.13.4.2. Umgebung ohne Mabata

In diesem Modus wird das Services Framework einzeln betrieben. Alle für den Betrieb benötigten Daten werden in einer lokalen Datenbank abgespeichert. Für die Administration wird eine grafische Benutzeroberfläche verwendet.

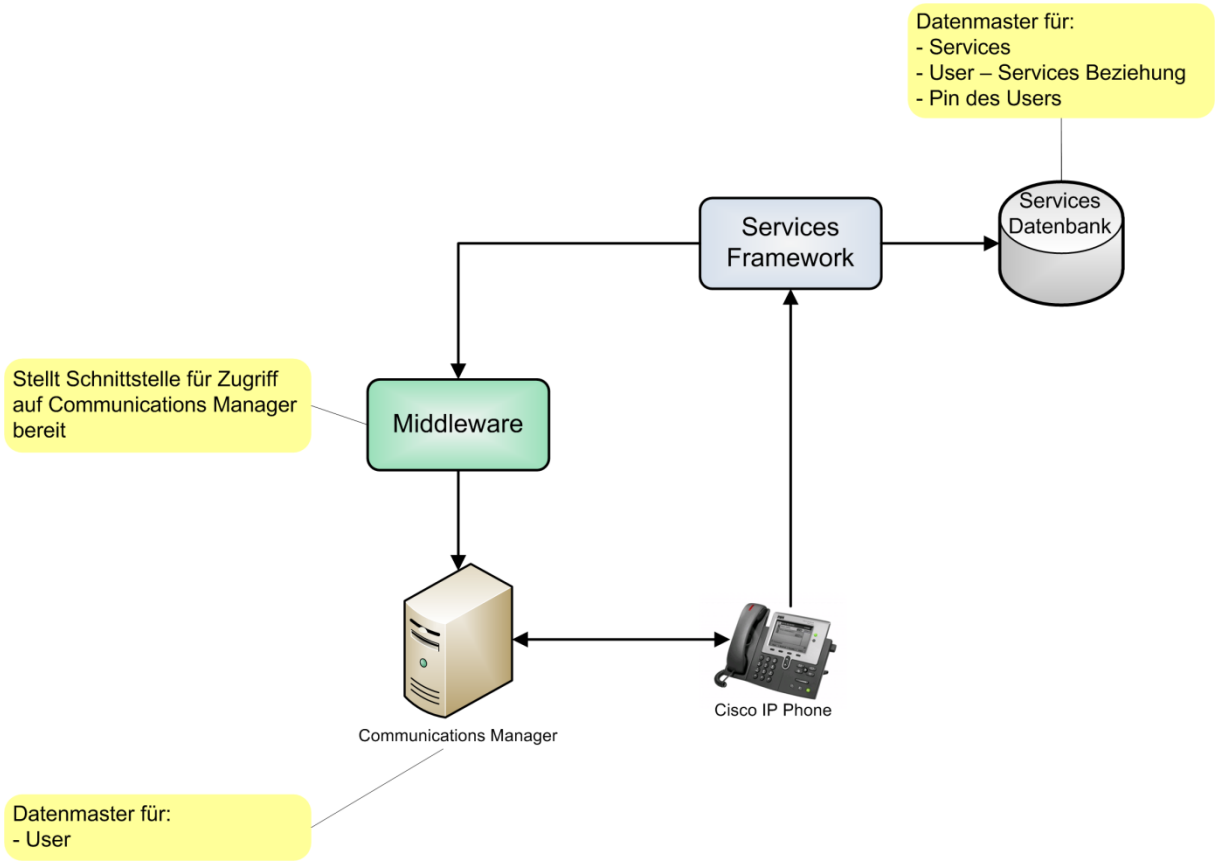


Abbildung 5-59 Finale Architektur Services Framework ohne Mabata

Grundsätzlich ist die Architektur dieselbe wie mit Mabata. Der wesentlichste Unterschied besteht darin, dass ohne Mabata das Services Framework nicht mehr mandantenfähig ist. Ein Service ist somit entweder global – für alle User benutzbar – oder er wird einem Benutzer direkt zugewiesen.

Der Datenmaster für die Benutzer bleibt der Communications Manager.

Datenbankschema

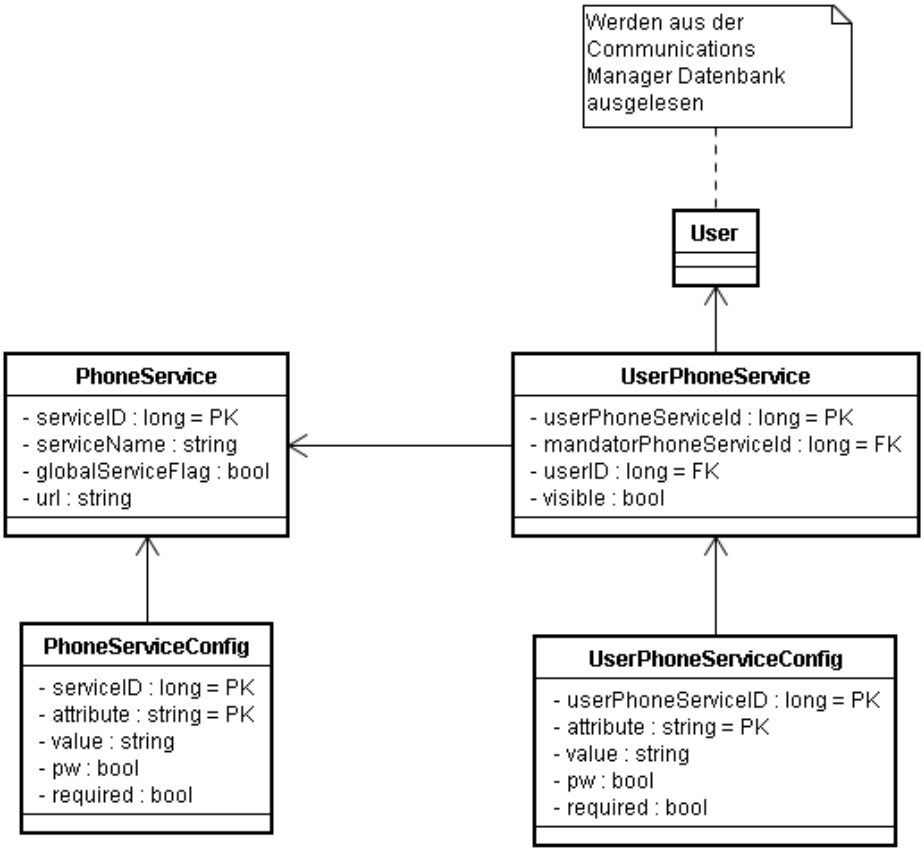


Abbildung 5-60 Datenbankschema Services Framework ohne Mabata

Das Datenbankschema braucht die Tabellen, welche mandantenspezifisch waren, nicht mehr. Ansonsten ist das Schema identisch mit dem Schema, welches im Zusammenhang mit Mabata gebraucht wird.

5.13.5. Paper Prototype

5.13.5.1. Fenster 1: Phone Service Übersicht Admin

Im ersten Fenster werden alle Phone Services aufgelistet. Die wichtigsten Informationen wie Name, Service URL und ob der Service global ist, werden direkt in der Übersichtstabelle angezeigt. Auch kann hier ein Service gelöscht werden.

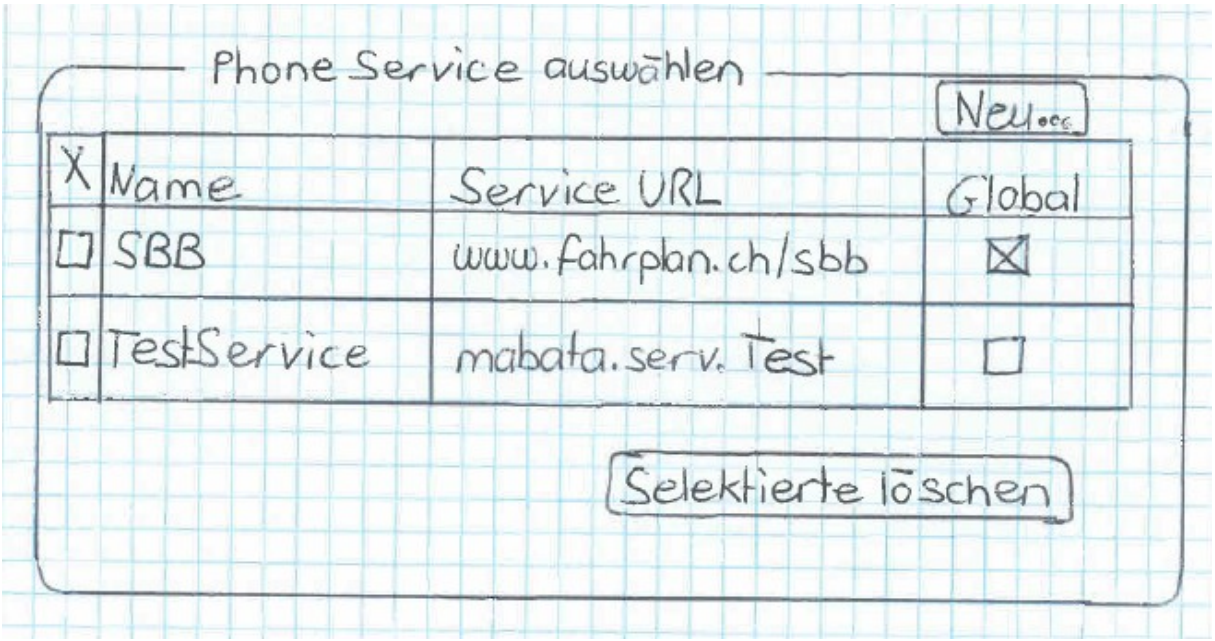


Abbildung 5-61 Paper Prototype PhoneService Übersicht Admin

Nach dem Klick auf einen Tabelleneintrag öffnet sich der ausgewählte Service im Fenster 2 in der Detailansicht.

5.13.5.2. Fenster 2: Phone Service Detailansicht

In diesem Fenster wird ein Service bearbeitet. Der Name sowie die URL werden festgelegt. Ist der Service nicht global definiert, können Mandanten ausgewählt werden, welche für diesen Service berechtigt sind.

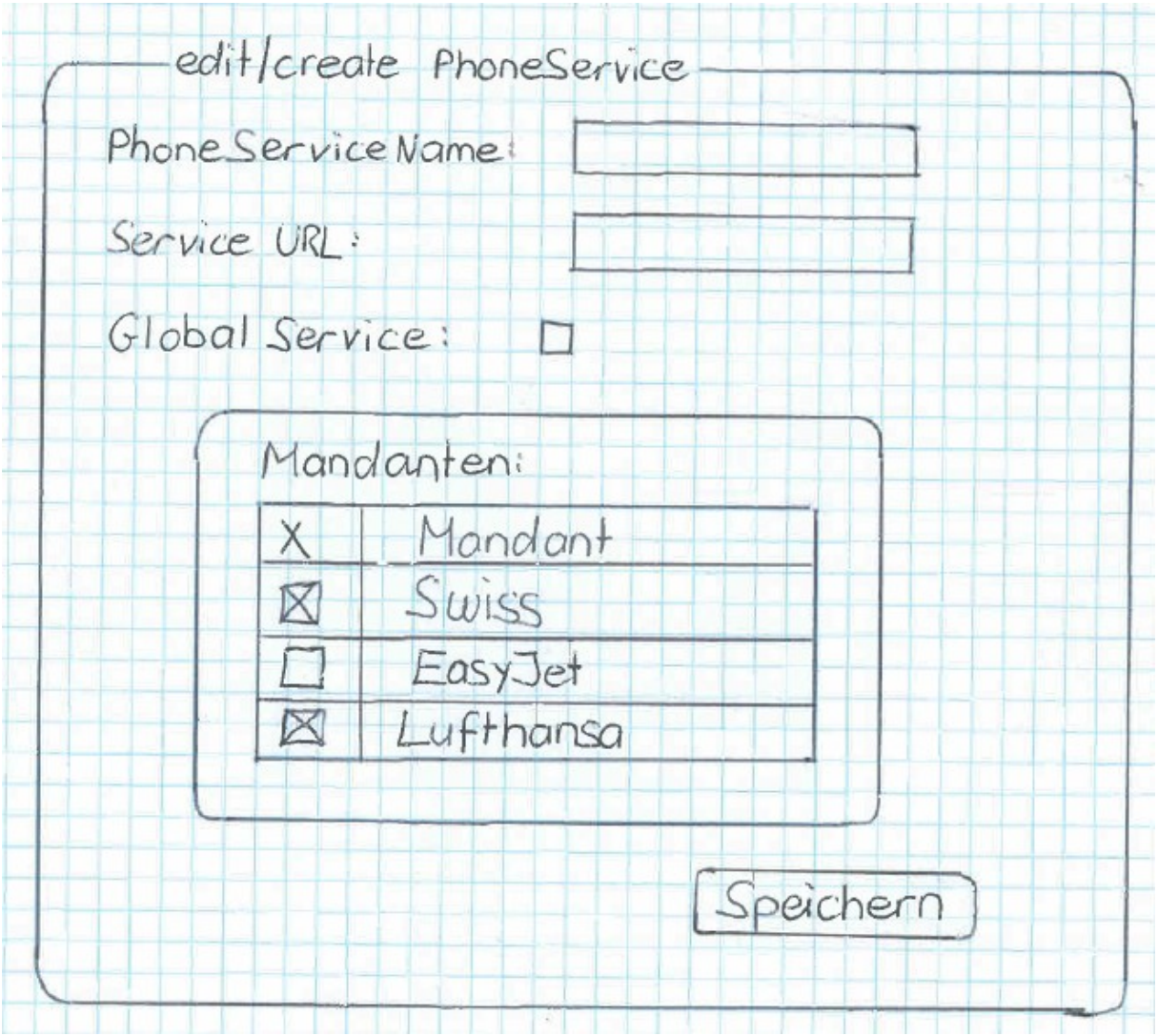


Abbildung 5-62 Paper Prototype Phone Service Detailansicht

Mittels der «Speichern» Schaltfläche werden die Änderungen übernommen und es wird zum Fenster 1 Phone Services Übersicht gewechselt.

5.13.5.3. Fenster 3: Phone Service Übersicht Admin/Superuser

Im ersten Fenster werden alle Phone Services aufgelistet. Die wichtigsten Informationen wie Name, Service URL und ob der Service global ist werden direkt in der Übersichtstabelle angezeigt.

Hier kann ein Service nicht gelöscht werden, da diese Ansicht auch dem Superuser zur Verfügung steht.

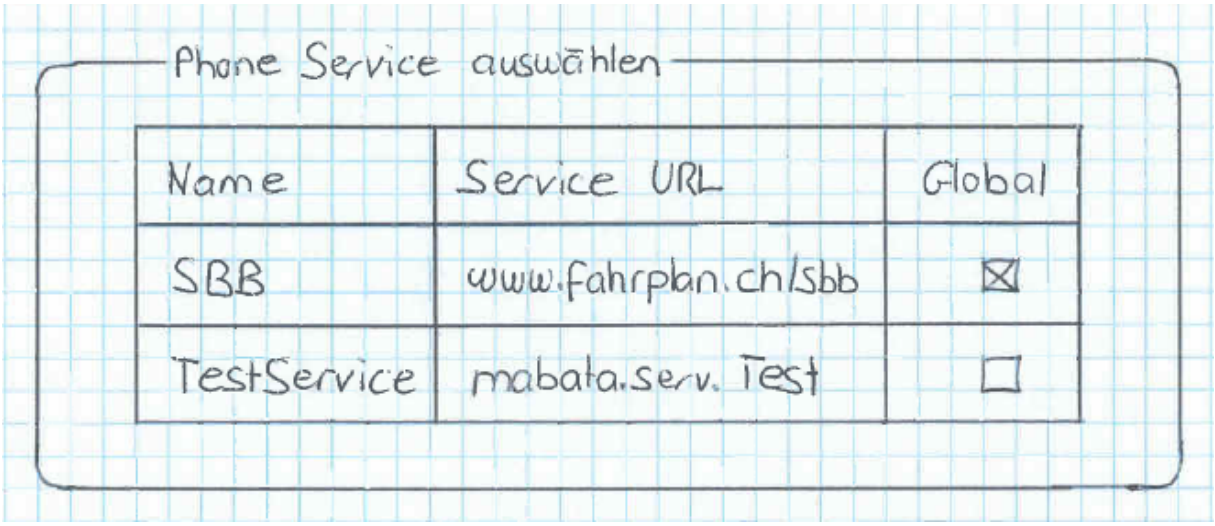


Abbildung 5-63 Paper Prototype Phone Service Admin-/Superuser Übersicht

Durch anwählen eines Services aus der Tabelle wird die Ansicht zu Fenster 4 weitergeleitet.

5.13.5.4. Fenster 4: Service editieren

In diesem Fenster können die Parameter des Services konfiguriert werden. Einen globalen Service kann nur der Administrator editieren. Der Superuser kann die eigenen Services seines Mandanten selber konfigurieren. Der Benutzer kann die Parameter eines Services nur setzen, nicht aber ändern oder löschen.

Ist ein Service global, fällt das Drop-Down Menü Mandant weg.

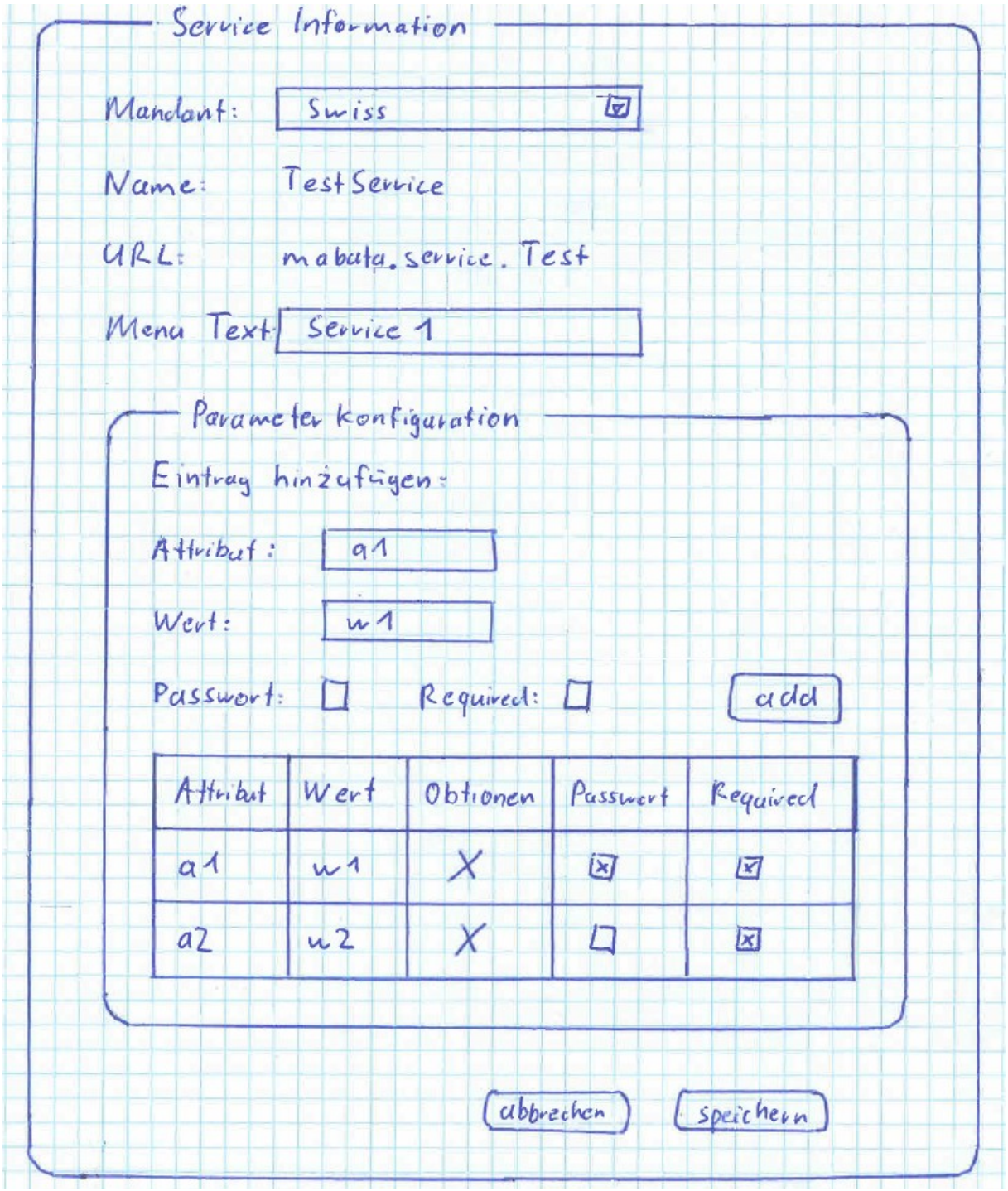


Abbildung 5-64 Paper Prototype Service editieren

5.13.5.5. Fenster 5: Services Berechtigungen Superuser

In diesem Fenster kann der Mandant die Services seinen Benutzern freigeben. Der Superuser kann einzelne Benutzer anwählen oder mittels der Schaltfläche «Alle» den Service allen Benutzer zur Verfügung stellen.

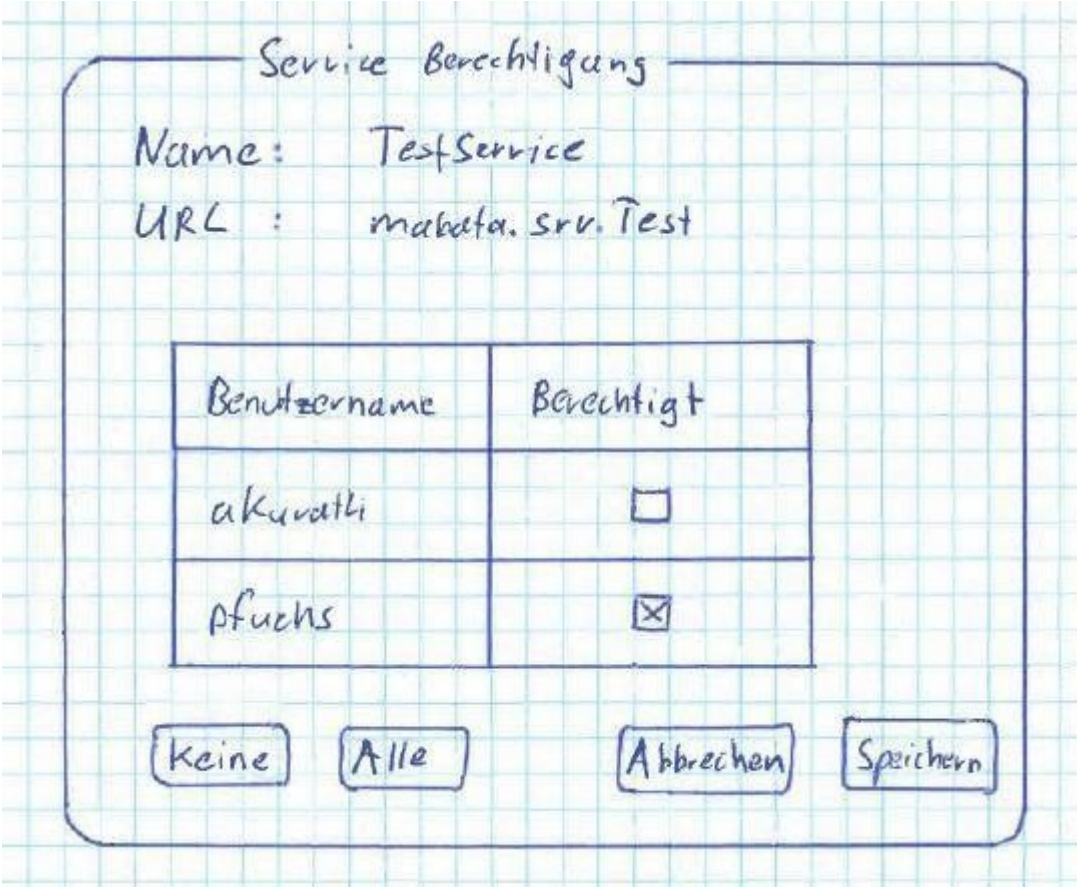


Abbildung 5-65 Paper Prototype Services Berechtigungen

5.13.5.6. Fenster 6: Service Berechtigung Benutzer

Hier kann der Benutzer auswählen, welche ihm zugewiesene Services er auf dem Telefon angezeigt haben will. Klickt er auf einen Service kommt er auf das Fenster 4 und kann die benutzerspezifischen Parameter selbst definieren.

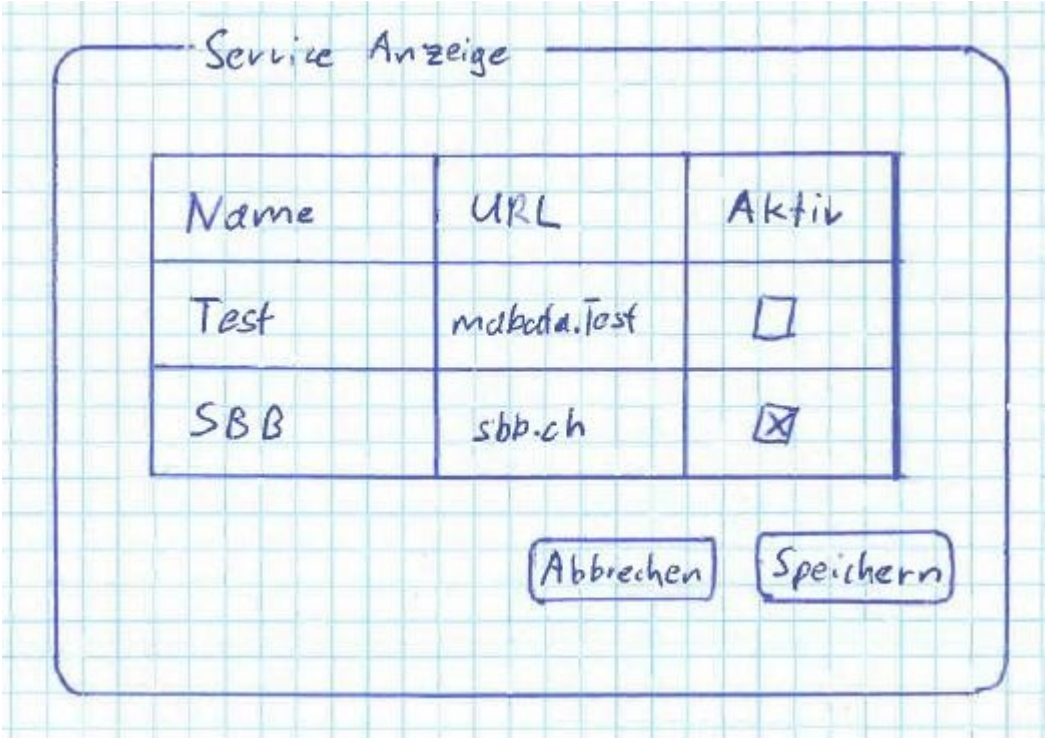


Abbildung 5-66 Paper Prototype Service Berechtigung Benutzer

5.14. Mabata Core Verbesserungen

In diesem Kapitel werden Verbesserungen des Mabata Cores dokumentiert, welche einem Benutzer entscheidende Vorteile bringen.

5.14.1. GetException

Falls in der alten Mabata Version ein Objekt abgeholt werden wollte, welches auf dem Communications Manager nicht mehr vorhanden war, wurde dieses Objekt jeweils mit `null` instanziiert. Dies führte früher oder später unweigerlich zu einer `NullPointerException`, da auf diesem Objekt Funktionen aufgerufen werden.

Neu wird bei jeder get-Operation überprüft, ob das abgeholte Objekt `null` ist. Für diesen Fall wurde eine neue Exception (`GetException`) eingeführt. Sie wird immer an das GUI geworfen, falls ein abgeholtes Objekt `null` ist. Mittels dieser Exception kann der Benutzer auf eine Inkonsistenz hingewiesen werden, ohne dass ein Fehlverhalten in der Applikation auftritt.

5.14.2. Unterstützung von Umlauten

Bei den Tests der neuen Middleware sind Probleme mit Umlauten aufgetreten. In diesem Zusammenhang konnte herausgefunden werden, dass das myFramework keine Umlaute unterstützt.

Das Problem war sowohl in der Middleware wie auch im myFramework die `getBytes()` Funktion auf einem String. Nach diesem Funktionsaufruf waren im String-Array die Umlaute jeweils nicht mehr richtig dargestellt.

Als Lösung für dieses Problem musste der `getBytes()` Funktion der UTF-8 `Charset` mitgegeben werden. Nachfolgendes Beispiel zeigt die entsprechende Codezeile aus dem myFramework vor der Änderung (wobei die Variable `request` vom Typ `String` ist):

```
outSocket.write(request.getBytes());
```

Neu ist dies wie folgt gelöst:

```
Charset charset = Charset.forName("UTF-8");
outSocket.write(request.getBytes(charset));
```

5.14.3. Passwort als Hash-Wert in Datenbank ablegen

In der alten Mabata Version wurden die Passwörter der Benutzer jeweils im Klartext in der Datenbank abgelegt. Dies ist aus zwei Gründen schlecht. Einerseits verwenden viele Benutzer für alle Accounts das gleiche Passwort und andererseits ist dies nicht zeitgemäss.

Aus diesen Gründen wird das Passwort neu mit einem MD5 Hash-Wert in der Datenbank abgespeichert. Für die Benutzung von Mabata ändert sich selbstverständlich nichts.

5.14.4. Importassistent Erweiterung

Im Verlauf dieser Bachelorarbeit wurde vermehrt festgestellt, dass der Importassistent erweitert und verbessert werden muss. Im folgenden Kapitel wird erläutert, um welche Funktionen der Importassistent erweitert wurde und wie seine Performance und Abläufe verbessert werden konnten.

5.14.4.1. Native SQL Querys

Um den Importassistenten zu verbessern/erweitern, wurden mehrere AXL-Requests durch native SQL Querys, welche ebenso per AXL an den Communications Manager gesendet werden, ersetzt. Nur durch diese Umstellung können alle benötigten Daten abgerufen werden. Bis zum jetzigen Zeitpunkt gibt es nicht für alle von Mabata benötigten Informationen einen eigenen AXL-Aufruf. Aber mit den nativen SQL-Abfragen können die AXL-Aufrufe selbst zusammengestellt und dadurch alle gewünschten Daten vom Communications Manager abgefragt werden.

Ein «executeSQLQuery» Aufruf per AXL muss folgendermassen aussehen:

```
public AXLSQLRequest(String AXLsrv,String AXLuser,String AXLpw){
      super(AXLsrv,AXLuser,AXLpw,"executeSQLQuery",”<sql></sql>”,true);
}
```

Innerhalb des Tags `<sql></sql>` kann nun ein beliebiges SQL Query stehen. Die Syntax wird überprüft und falls Fehler vorhanden sind werden diese in einer AXL-Response angegeben.

```
<faultstring>
The specified table (wrongtable) is not in the database.
</faultstring>
```

Als Resultat kommt eine Response zurück die jede Zeile in einen `<row>`-Tag umklammert.

```
<return>
<row><uuid>50a465d9-feb8</uuid><nr>7200</nr><usage>Device</usage></row>
<row><uuid>e1b93a1c-ff27</uuid><nr>7105</nr><usage>Device</usage></row>
<row><uuid>4f6aaf92-5e8c</uuid><nr>7102</nr><usage>Device</usage></row>
</return>
```

Als Quelle für das Datenbankschema des Communications Manager wurde das offizielle PDF von Cisco verwendet, welches auf der CD beigelegt ist «Einarbeitung\CUCM7.0.pdf».

5.14.4.2. Verbesserungen

In den folgenden Kapiteln werden die Verbesserungen des Importassistenten beschrieben.

Import der DirNumbers

Bis anhin wurden die Nummern anhand der vom Administrator eingegebenen Nummernblöcke importiert. Dies führte dazu, dass versucht wurde jede einzelne Nummer innerhalb der Nummernblöcke per AXL vom Communications Manager abzuholen. Falls dies nicht erfolgreich war, bedeutete dies, dass diese Nummer nicht aktiv ist. Diese Lösung bietet eine sehr schlechte Performance. Für jede in Mabata erstellte Nummer wird ein einziger AXL-Request abgesetzt obwohl diese nicht einmal aktiv sein muss. Neu werden alle aktiven Nummern mit einem einzigen `AXLSQLRequest` abgeholt.

Gleichzeitig kann nun überprüft werden, ob die Nummer ein Hunt Pilot oder eine Pickup Gruppen Nummer ist. Zusätzlich wird auch geprüft, ob die Nummer einer `CallPickupGroup` zugewiesen ist. Ist dies der Fall, wird die ID der `CallPickupGroup` in der Nummer gesetzt.

```
SELECT n.pkid as dirnumberpkid, n.description, n.dnorpattern,
       t.name as usage ,plm.fkpickupgroup as cppkid
FROM typepatternusage t, numplan n
LEFT JOIN pickupgrouplinemap plm ON plm.fknumplan_line = n.pkid
WHERE n.tkpatternusage = t.enum
AND (t.enum = 2 OR t.enum = 4 or t.enum = 7)
```

In der Tabelle «numplan» auf dem Communications Manager sind nicht nur alle Nummern, sondern auch die Route Patterns (z.B. 41XXXXXXXXX) gespeichert. Aus diesem Grund wird am Schluss dieser Query gefiltert, dass nur jene Nummern ausgelesen werden, welche entweder eine Nummer «`t.enum=2`», ein Hunt Pilot «`t.enum=4`» oder eine Pickup Gruppe «`t.enum=7`» sind.

Import der Pickup Gruppen

Da ein AXL-Request, welcher alle Pickup Gruppen liefert, fehlt, wurden die Pickup Gruppen bis anhin über die Nummern abgeholt. Es wurden die IDs der Pickup Gruppen gespeichert, welche an Nummern zugewiesen waren und danach separat über AXL abgeholt. Dadurch wurden aber Pickup Gruppen, welche keiner Nummer zugewiesen waren, gar nie importiert. Um dieses Problem zu beseitigen und weil die Nummern neu per SQL importiert werden, wurden auch die Pickup Gruppen auf eine Importierung per SQL umgestellt.

```
SELECT p.pkid as cppkid, p.name, n.dnorpattern
FROM pickupgroup p, numplan n
WHERE p.fknumplan_pickup = n.pkid
```

Der Join von der «pickupgroup» Tabelle auf die «numplan» Tabelle ist nötig, um die Pickup Gruppen Nummer zu erhalten. Nur mit Hilfe dieser Nummer kann die Pickup Gruppe in Mabata einem Mandanten zugewiesen werden.

5.14.4.3. Neue Funktionen

Zusätzlich können nun auch Hunt Piloten mit den dazugehörigen Hunt Lists und Line Groups importiert werden. Diese werden korrekt anhand der Hunt Piloten Nummer dem richtigen Mandanten zugeordnet. Wird ein Hunt Pilot importiert, dessen Nummer in keinem gültigen Nummernblock vorhanden ist, kann der Hunt Pilot nicht importiert werden.

Für die Hunt Gruppen Funktionalität müssen drei verschiedene Objekte importiert werden: Hunt Pilots, Line Groups und Hunt Lists. Der Import der Line Groups kann einfach über eine einzelne Tabelle gelöst werden. Die Abfragen der beiden anderen Objekten sind ein wenig komplizierter. Als Beispiel werden hier die Hunt Lists dargestellt:

```
SELECT d.name as hltname, d.pkid as hltpkid , d.description as hltde-
scription, d.routelistenabled as hlroutelistenabled,
c.name as callmanagergroupname,
l.name as linegroupname, l.pkid as linegrouppkid
FROM
device d, typeProduct t, routelist r, linegroup l, callmanagergroup c
WHERE t.name = 'Hunt List'
AND t.tkmodel = d.tkmodel
AND r.fkDevice = d.pkid
AND r.fklinegroup = l.pkid
AND d.fkcallmanagergroup = c.pkid
```

Dies ist dadurch zu erklären, dass die Hunt Lists keine eigene Tabelle besitzen sondern in der «Device» Tabelle abgespeichert sind.

5.14.5. Fortschrittsanzeige

Durch die Umsetzung der Middleware ist die direkte Information, wie weit der Import fortgeschritten ist, verloren gegangen. Demzufolge kann Mabata nicht mehr periodisch informiert werden. Daher wurde der Fortschrittsbalken durch einen bewegenden Balken ausgetauscht und der Benutzer wird informiert, dass die Bearbeitung noch nicht abgeschlossen ist.

5.14.6. Erweiterung des Nummerndatentyps

Der Communications Manager kann auch Telefonnummern gemäss dem internationalen Standard [E.164] verwalten. Damit diese auch in Mabata verwendet werden können, wurde eine Erweiterung eingeführt. Die Nummern sind in der ganzen Architektur von Mabata als `long` verankert. Dies wird benötigt, um Vergleiche mit den Nummernblöcken anzustellen. Somit konnte der Datentyp nicht von `long` auf `String` geändert werden. Die Lösung des Problems lag darin, ein weiteres Feld in der Datenbank-Tabelle «dirNumber» zu erstellen, welches den Präfix der Nummer beinhaltet. Dieser kann entweder aus einem «+» oder «00» bestehen.

Zum Beispiel sind folgende Nummern möglich:

```
0041712241234
 +41712241234
```

Bei der Erstellung eines Nummernblocks kann nun der Präfix angegeben werden:

Abbildung 5-67 Erweiterung Nummerndatentyp

5.14.7. Benutzerpasswort beim Import

Bei den importierten Benutzern wurde bis anhin kein Passwort gesetzt. Neu wird dem Administrator die Möglichkeit geboten, ein Standardpasswort festzulegen. Beim Importvorgang wird aus diesem Password einen Hash-Wert generiert und in der Datenbank abgespeichert.
Das Standardpasswort kann über das Konfigurationstool gesetzt werden.

Folgende Codezeile musste im `ImportEntityManager` eingefügt werden:

```
user.setPassword(User.generateHash(MabataProperties.get()
.getProperty("default_password")));
```

5.14.8. Erfassen von Telefontypen

Bis anhin konnten Telefontypen ohne Namen erfasst werden. Dies führte dazu, dass in dem Drop-Down Menü leere Einträge vorhanden waren. Um dies zu verhindern wird überprüft, ob ein Name eingegeben wurde. Ist dies nicht der Fall, wird eine Fehlermeldung ausgegeben.

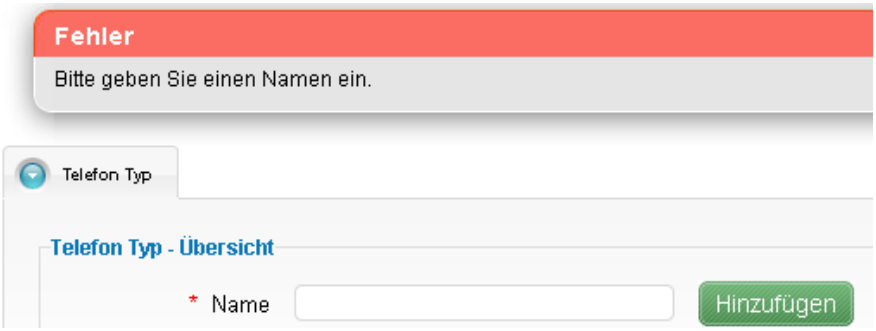


Abbildung 5-68 Telefontyp Erfassung

5.14.9. Mabata-Setup

Das Mabata-Setup beinhaltete folgende drei Probleme:

Falls das Datenbank-Setup nicht erfolgreich war, wurden die Passwortfelder zurückgesetzt, jedoch wurde keine Fehlermeldung angezeigt. Dieses Problem wurde behoben in dem alle Exceptions abgefangen werden und dem Benutzer eine Fehlermeldung ausgegeben wird.

Nach erfolgreichem Setup musste die Adresse von Mabata manuell in den Browser eingeben werden. Hierzu dient neuerdings ein «Login» Button, welcher mit der Einstiegsseite von Mabata verknüpft ist:

Abbildung 5-69 Mabata Setup inkl. Button

Dieser Button stellt einen `CommandLink` dar:

```
<ice:commandLink value="#{msg['common.button.login']}" id="login"
action="index" styleClass="btn green right rightspace"/>
```

Die Action dieses `CommandLinks` ist mit einer NavigationRule in der Datei «faces-config.xml» verknüpft, welche auf die «index.iface» Seite verweist:

```
<navigation-rule>
      <navigation-case>
            <from-outcome>index</from-outcome>
            <to-view-id>/index.iface</to-view-id>
            <redirect/>
      </navigation-case>
</navigation-rule>"/>
```

Die Passwörter wurden nicht darauf überprüft, ob sie identisch sind. In die Datenbank wurde nur immer das Passwort gespeichert, welches im ersten Passwortfeld eingegeben wurde. Neu wurde eine Passwortüberprüfung eingebaut.

5.14.10. Mabata-Login

Falls beim Mabata-Login das Passwort falsch eingegeben wurde, sah die Seite folgendermassen aus:

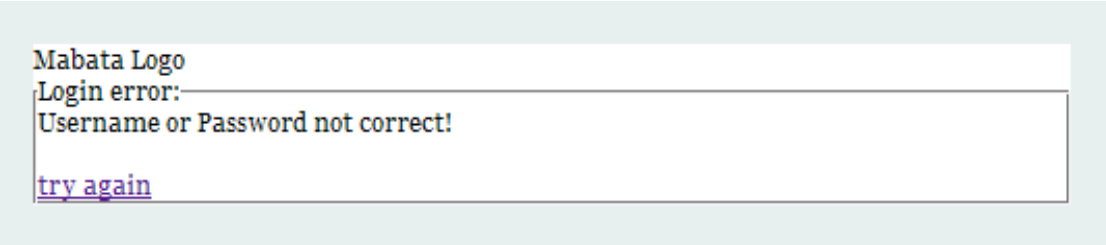


Abbildung 5-70 Alte Login Seite

Diese Seite ist überhaupt nicht Mabata konform. Aus diesem Grund musste sie überarbeitet werden. Dabei wurde als Problem erkannt, dass die Fehlerseite eine JSP-Datei sein muss. Das heisst, ICEfaces kann nicht verwendet werden. Um dieses Problem zu umgehen, wurde in der Fehlerseite eine Umleitung eingebaut, welche auf eine JSPX-Seite verweist. Mit diesem Umweg ist es nun auch möglich eine ICEfaces Seite anzuzeigen:

Abbildung 5-71 Neue Login Seite

5.14.11. Unterstützung für Softphones

Als Softphone wird die Cisco Software bezeichnet welche es erlaubt über den PC zu telefonieren. Die korrekte Bezeichnung dafür lautet «Cisco IP Communicator». Weiter gibt es von Cisco eine Software die auch auf einem PC installiert wird und zum Telefonieren genutzt werden kann. Zusätzlich hat diese Software noch weitere Funktionen wie Chatten oder Videotelefonie. Diese Software ist mit «Cisco Unified Personal Communicator» bezeichnet.

Damit Mabata mit diesen beiden Telefontypen umgehen kann, musste eine Unterscheidung bei der Erstellung und Handhabung der Telefone eingefügt werden. Diese zeichnet sich dadurch aus, dass nun unterschieden wird, ob es sich um ein Hardphone, welches eine MAC-Adresse besitzt, oder ein Softphone handelt. Da ein Softphone keine MAC-Adresse besitzt, muss bei diesem der Name eindeutig sein.

Dazu wurde im GUI eine Checkbox hinzugefügt mit der ein Telefon als Hard- oder Softphone gekennzeichnet werden kann.

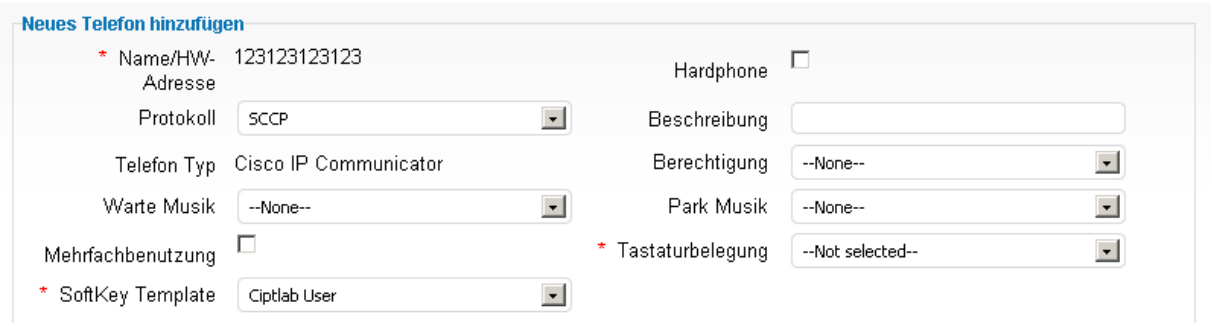


Abbildung 5-72 Telefon Ansicht inkl. SoftPhone Erweiterung

5.14.12. Verbesserung der Navigation

Bis anhin verhielten sich die Schaltflächen der Navigation nicht wie allgemein üblich. Die Schaltflächen wurden während dem Darüberfahren mit der Maus hervorgehoben. Dies zeigte sich darin, dass sie farbig dargestellt wurden. Nach dem Betätigen einer Schaltfläche wurde diese aber wieder grau. Aus diesem Grund konnte nicht mehr festgestellt werden, welcher Menüpunkt gerade aktiv ist.

Durch das Einfügen von Bedingungen im `NavigationController` und zusätzlichen Styles im «layout.css» behalten die ausgewählten Schaltflächen ihre Farbe während sie aktiv sind.

Im Beispiel ist die Topnavigation «System» ausgewählt, als Sekundärnavigationspunkt die «Konsistenzprüfung».

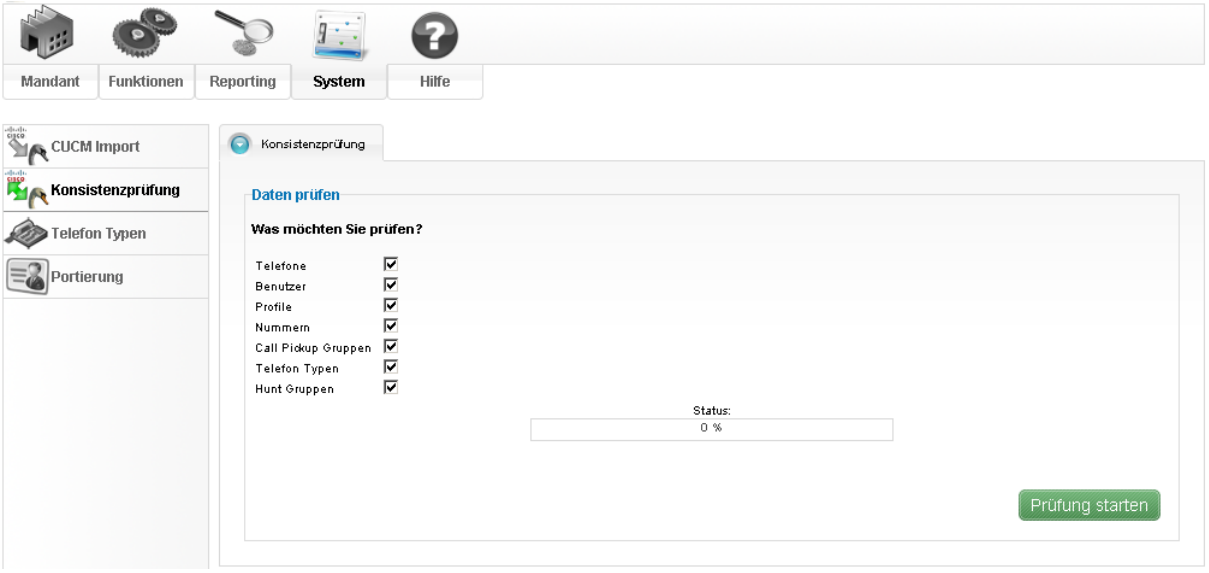


Abbildung 5-73 Navigationsverbesserung

Anhang A: Begriffe Mabata <-> Communications Manager

Mabata	Communications Manager
Berechtigung	Calling Search Space (CSS)
Rufgebiet	Partition
Warte Musik	Music on Hold (MOH)
HW-Adresse	MAC Address
Profil	Device Profile
Gruppennummer	Hunt Pilot

Tabelle A-0-1 Begriffsdefinition

Anhang B: Literaturverzeichnis

Cisco

- Cisco CallManager Fundamentals
- Konfiguration Communications Manager 4.x, 5.x und 6.x
  http://www.andtek.de/datasheet-172-.pdf
- Cisco Unified Communications Manager Administration Guide
  http://www.ciscosystems.com/en/US/products/sw/voicesw/ps556/prod_maintenance_guides_list.html
- Weitere Referenzen sind unter: „CD:\Einarbeitung\CommunicationsManager“

AXL

- SOAP Message
  http://en.wikipedia.org/wiki/Soap
- AXL Schnittstelle vom Communications Manager
  http://www-user.tu-chemnitz.de/~ronsc/misc/CCM-cisco-2006/
- AXL API
  http://www.selso.com/doc/axl/api.html
- XML Parser
  http://www.uzi-web.de/parser/parser_toc.htm

ICEfaces

- ICEfaces Beschreibung
  http://de.wikipedia.org/wiki/ICEfaces
- Allgemeine Information zu ICEfaces
  http://www.icefaces.org
- ICEfaces Komponenten
  http://component-showcase.icefaces.org/component-showcase/showcase.iface

Tomcat

- Allgemeine Information zum Apache Tomcat Server
  http://tomcat.apache.org/

Datenbank

- Information zu Postgresql
  http://www.postgresql.org/

Mabata

- Alte Diplomarbeit
  Technische Referenz – Mandantenfähiges IP-PBX Administrationstool, 25. April 2008

Services Framework

- Alte Diplomarbeit
  Adding Security Functions to Mandate based VoIP Solution, 22. November 2007

Corba

- http://de.wikipedia.org/wiki/Common_Object_Request_Broker_Architecture
- http://www.corba.org

WebServices

- http://www.stefan-rinke.de/articles/webservices/

JavaService

- http://forge.ow2.org/projects/javaservice/

Anhang C: Quellenverzeichniss

Kapitel	Quelle	Besucht am
3.2.3.4	Kapitel BizTalk Server Übersicht http://www.microsoft.com/germany/biztalk/einsatz/eai.mspx	24.02.09
3.2.3.4	Kapitel BizTalk Server Adapter http://www.microsoft.com/germany/biztalk/interop/adapter.mspx	24.02.09
5.13.1	DA Adding Security Functions to Mandate based VoIP Solution 22. November 2007	
3.7.3 / 3.8.3	[AXL-API] CD-ROM: Einarbeitung\AXL-API.zip	
4.2.4.1 / 4.2.5.1	http://www.voipintegration.com/images/phoneremote.png	25.02.09
4.2.10.1 / 5.14.6	[E.164] http://de.wikipedia.org/wiki/E.164	30.02.09
5.2.3.1	Kapitel Authentifizierung mit Benutzername/Passwort http://www.stefan-rinke.de/articles/webservices/ch11.html	11.03.09
5.6.5.1	[JAXB] https://jaxb.dev.java.net/guide/	28.05.09
5.1.2.1	Kapitel ICEfaces http://de.wikipedia.org/wiki/ICEfaces	

Anhang D: Bilderverzeichniss

Anhang E: Tabellenverzeichniss