Entwurf und Implementierung eines Systems zum Management von Teilnehmerdaten für mobile Dienste

Bachelorarbeit

von

Ngoc Kin Voong

aus

Osnabrück

vorgelegt am

Lehrstuhl für Rechnernetze und Kommunikationssysteme
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

April 2006

Betreuer:
Dipl.-Inf. Michael Stini

Danksagung

Mein größter Dank geht an Michael Stini, der mir während der Bacherlorarbeit kompetent zur Seite stand und mir wertvolle Anregungen gegeben hat.

Als nächster besonderer Person möchte ich ganz herzlich Michael Wirtz danken, der mir viele Verbesserungsvorschläge gegeben hat und mich in manchen Sachen sehr gut beraten konnte.

Mehmet Kolac danke ich für seine praktischen Tipps und die Bereitstellung der Software für die schönen Grafiken.

Last but not least möchte ich auch meiner Familie und Jiabin He danken, die mich in allen Bereichen unterstützt haben und mich so mögen, wie ich bin. Dafür liebe ich euch!

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Zu den Innovationen des 21. Jahrhunderts zählen fraglos die mobilen Endgeräte. Nach der PC-Ära folgt eine mobile Generation. Die Technik wird in immer kleinere und leistungsfähigere Endgeräten wie PDAs, Mobiltelefone etc. eingebaut, deren Leistung und Kapazität sich stark verbessert hat. Mit Techniken wie UMTS, GPRS ermöglichen Endgeräte dem Benutzer jederzeit mobil zu sein. Angebotene Dienste wie zum Beispiel Nachrichten- oder Börsenticker, bringen den Benutzer sofort und jederzeit auf den neusten Stand. Der Verbraucher ist nicht mehr auf seinen Desktop-PC angewiesen, sondern kann mit einem mobilen Gerät die verschiedenen Dienste überall in Anspruch nehmen.

Durch die Vielzahl der Dienstanbieter kann es zu einer lästigen Pflicht werden, sich immer wieder neu registrieren zu müssen. Deshalb liegt es nahe eine zentrale Teilnehmerverwaltung zu entwickeln, so dass die Dienste auf zentral verwaltete Teilnehmerdaten zurückgreifen können. Eine Änderung der Daten kann vom Teilnehmer in der Teilnehmerverwaltung durchgeführt werden. Für die Dienste, die die zentrale Teilnehmerverwaltung nutzen, wird garantiert, dass aktuelle Daten vom Teilnehmer existieren und die Authentizität der Benutzer gewährleistet ist. Durch verschiedene Profilverwaltungen wird dem Dienst, neben den persönlichen Daten, auch Informationen über die Hardware übermittelt, so dass der Dienst anhand dieser Informationen optimalen, gezielten Service für die Hardware bieten kann.

Ein Beispiel für einen mobilen Dienst ist der Tauschdienst, der zur Zeit vom Lehrstuhl für Rechnernetze und Kommunikationssysteme entwickelt wird. Der Dienst ermöglicht das Tauschen von virtuellen Objekten. Registrierte Teilnehmer suchen mit mobilen Geräten nach potentiellen Tauschpartnern um gewünschte Objekte wie Bilder, Sounds oder Short-Videos miteinander zu tauschen. Dieser Tauschdienst kann zum Beispiel Geräteinformationen von der Teilnehmerverwaltung anfordern, die von beiden Gerätebesitzern hinterlegt wurden, um den Tauschvorgang optimal zu gestalten. So wird z.B. bei einem Tauschvorgang ein 300 kB großes Bild vom mobilen Gerät Laptop, vom Tauschdienst in ein kleineres Format konvertiert, um so das Bild auch auf einem Handy anzeigen zu können.

1.2 Problemstellung

Bei einer großen Zahl von Teilnehmern ist eine Verwaltung der Daten unerlässlich. Die Aufgabe dieser Bachelorarbeit ist es, ein System zu entwickeln, dass das Verwalten von Teilnehmerinformationen ermöglicht. Das Teilnehmermanagementsystem speichert in einer zentralen Datenbank persönliche Teilnehmerdaten und optionale Informationen ab, die von den Diensten abgerufen werden können. Um die Sicherheit der Teilnehmerdaten zu gewährleisten, sind spezifische Anwenderrollen für das Pflegen und Analysieren der Daten vorgesehen. Die Interaktion zwischen Teilnehmer und Programm soll durch eine graphische Benutzeroberfläche erleichtert werden.

1.3 Struktur der Arbeit

Im nachfolgendem Kapitel 2 wird die Datenbank für das TMS modelliert und das verwendete Datenbankmanagementsystem PostgreSQL kurz vorgestellt. Kapitel 3 widmet sich der Programmiersprache Java mit einer kurzen Einführung zum Datenbankzugriff mittels JDBC. Die Möglichkeiten, die Java für das User Interface und die Client-Server-Kommunikation bereit stellt, werden in diesem Kapitel vorgestellt. Im 4. Kapitel werden die Funktionalitäten und der Programmablauf des TMS beschrieben. Es folgt eine Schilderung der Idee des Programms, der Implementation, sowie der Architektur des TMS,

folgen im anschließendem 5. Kapitel. Schließlich wird im letzten, dem 6. Kapitel, ein Resümee der Bachelorarbeit gezogen.

Kapitel 2

Die Datenbank vom Teilnehmermanagementsystem (TMS)

Bei einer großen Zahl an Daten, ist es von Nutzen eine Datenbank zu modellieren, die die effiziente Speicherung und Verwaltung ermöglicht. Das TMS benutzt eine relationales Datenbankmanagementsystem, um Teilnehmerdaten zu speichern und so gezielt nach Daten abzufragen. Es wird die Open-Source-Datenbank PostgreSQL verwendet.

2.1 Datenbank-Entwurf

2.1.1 Anforderungen

Die persönlichen Angaben wie Accountname, Passwort, Name, Alter und Geschlecht spielen die Hauptrolle im TMS. Die Teilnehmerverwaltung kann sich mit weiteren optionalen Daten schmücken wie zum Beispiel postalischen Adressdaten oder elektronischen Kontaktdaten. Da es heutzutage vielfältige Wege gibt erreichbar zu sein, sind die verschiedenen Möglichkeiten zu berücksichtigen. Handynummern, Fax, ICQ, Skype etc. können einer Adresse zugeordnet werden, müssen jedoch nicht. Die Authentizität der

Daten wird mit einem Teilnehmerstatus sichergestellt.

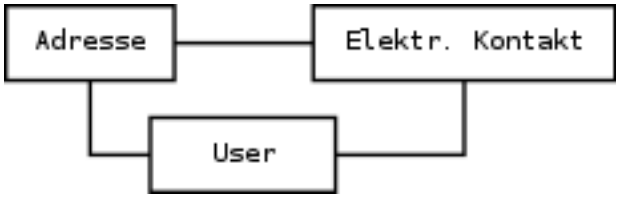

Abbildung 2.1: Die User-Adress-Kontakt-Beziehung

Um mobilen Diensten nicht nur Teilnehmerdaten sondern auch Informationen zu den Teilnehmern anzubieten, ist es sinnvoll Gerätetypen und Geräteleistungen an die Dienste schicken zu können. Die verschiedenen Dienste benötigen individuelle Informationen. So benötigt ein Musikdienst die Datenrate oder Bandbreite des Gerätetyps, während einem Fotodienst die Displaygröße am wichtigsten ist. Es ist eine Abhängigkeit zwischen den Diensten und den Geräten gewünscht, die in der Datenbank festgehalten werden soll. Für die Analyse sind Daten wie Onlinezeiten oder Registrierdatum etc. unentbehr-

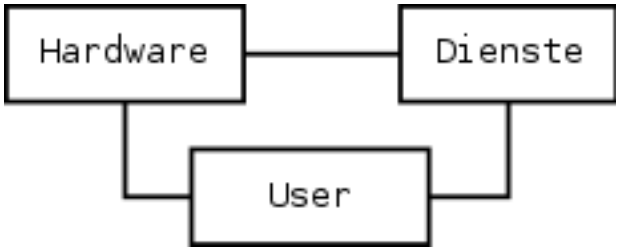

Abbildung 2.2: Die User-Hardware-Dienste-Beziehung

lich. Schließlich sollen Änderungen der Daten protokolliert werden. Die Abbildung 2.3 zeigt das ER-Modell der Datenbank.

2.1.2 Datenbankenmodell

Nachdem das ER-Modell erstellt und die Beziehung klar ist, lässt sich das Datenmodell entwickeln. Um Redundanzen zu vermeiden wird das Datenbankschema in die 3. Normalform überführt. Das bedeutet, dass Nicht-Schlüsselattribute von keinem Schlüsselkandidaten transitiv abhängt und von jedem Schlüsselkandidaten vollständig funktional

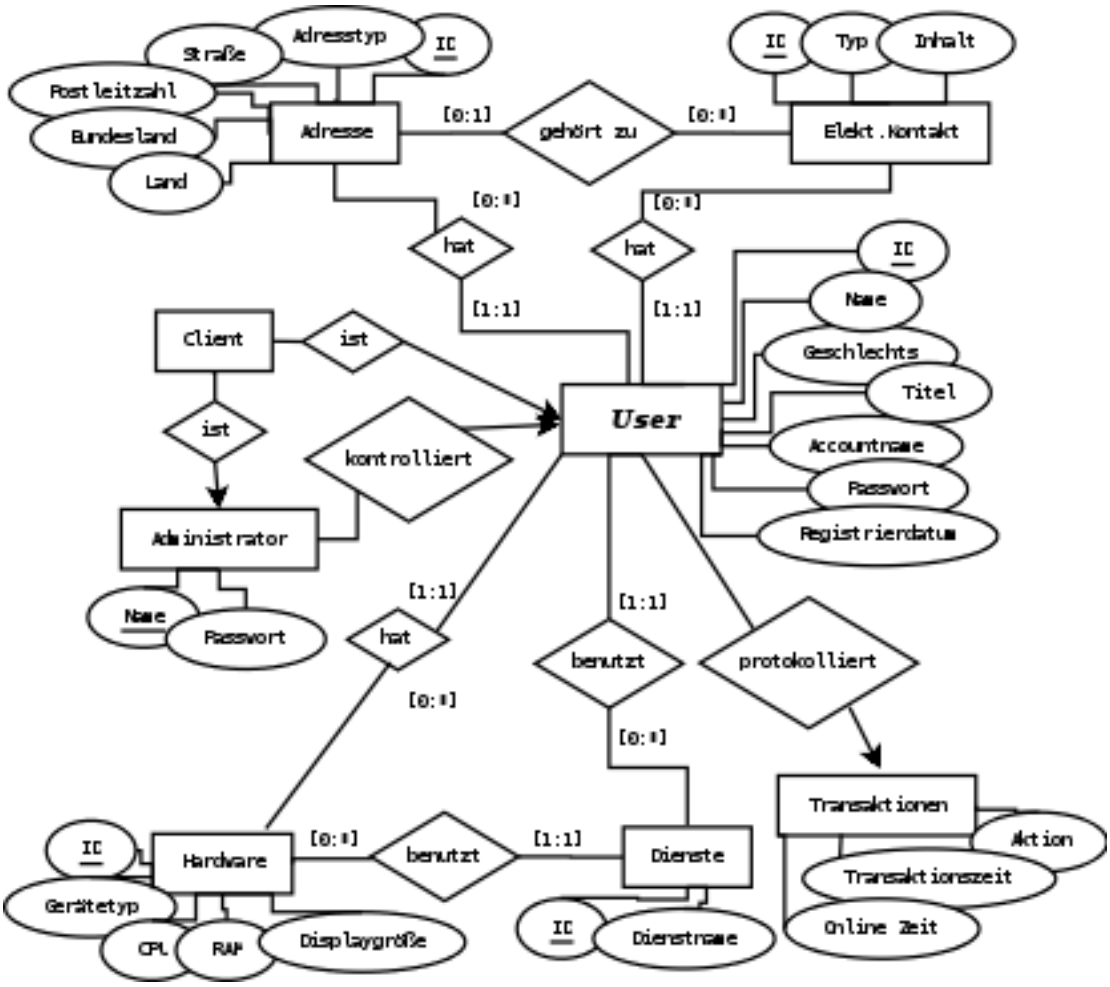

Abbildung 2.3: Das Entity-Relationship-Modell

abhängig ist (vgl. (AH00)). Das hat den Vorteil, dass Mehrfachspeicherungen in einer Tabelle vermieden werden und somit eine Änderung der Daten nur an einer Stelle nötig ist.

Das Wichtigste in dem System sind die persönlichen Teilnehmerdaten. Sie bilden die Grundlage und somit die Pflichtdaten in der Datenbank. Die Pflichtdaten-Tabelle hat die Attribute *Vorname*, *Nachname*, *Geburtsdatum* und *Geschlecht*. Um die eindeutige Identifikation der Person zu gewährleisten, wird vom Datenbankmanagementsystem eine unzweideutige Nummer vergeben. Diese *UserID*-Nummer ist der Primärschlüssel und repräsentiert genau eine Person. Eine weitere wichtige Tabelle ist die *Account*-Tabelle mit den Attributen *Accountname*, *Passwort* und *UserID*. Die *UserID* ist ein Fremdschlüssel, der auf den Primärschlüssel aus der Tabelle *Pflichtdaten* verweist.

Neben der *Pflichtdaten*-Tabelle gibt es eine weitere *Benutzerdetails*-Tabelle, in der optional der akademische Grad eines Teilnehmers angegeben sein kann. Zusätzliche Attribute, wie *Größe*, *Augen*- und *Haarfarbe*, *Konfession*, etc. können wahlweise in der Datenbank ergänzt werden. Hier lassen sich NULL-Werte nicht vermeiden. Es ist daher angebracht weitere Attribute erst dann zu ergänzen, wenn die mobilen Dienste auch diese Informationen tatsächlich benötigen.

Weitere Teilnehmerinformationen sind optional. Jede Person kann beliebig viele Adressen, Kontaktdaten und Geräte haben oder Dienste beanspruchen. Die *Adress*-Tabelle besitzt die Attribute *AdressID* als Primärschlüssel, den *Adresstyp*, die *Straße*, die *Postleitzahl*, die *Stadt* und das *Land*. Bei dem *Adresstyp* kann es sich um eine Hausadresse, Büroadresse oder einen selbstdefinierten Typ handeln. Um die Dreiecksbeziehung zwischen *Pflichtdaten*, *Adresse* und *elektronischem Kontakt* aufzulösen, wird in der *Adress*-Tabelle und der *elektronischer Kontakt*-Tabelle ein weiteres *UserID*-Attribut als Fremdschlüssel eingefügt, mit dessen Hilfe Relationsbeziehungen erhalten bleiben. Die Tabelle *elektronischer Kontakt* besitzt außerdem noch die Attribute *KontaktID* als Primärschlüssel sowie *Typ*, *Inhalt*, und *AddressID*. Da ein Teilnehmer beispielweise die Büroadresse mit einer Telefonnummer hinterlegt hat, wird die *AdressID* benötigt, um den Bezug festzuhalten. Hat der Teilnehmer nur seine E-Mail-Adresse hinterlegt, ist das *AdressID*-Feld NULL. Das Attribut *Typ* gibt an welcher Art der elektronische Kontakt ist. Es kann sich dabei um die Telefon- oder Handynummern, E-Mail, ICQ, Skype, Pager, etc. handeln.

Eine weitere Dreiecksbeziehung bilden die Pflichdaten, die Dienste und die Geräte. Für eine einzelne Person werden mehrere Profile, abhängig vom Dienst, angelegt. Dabei beinhaltet ein Profil einen Dienst und alle Geräte mit denen der Teilnehmer diesen Dienst beansprucht. Die Tabelle *Profil* enthält die Attribute *ProfilID* als Primärschlüssel, die Fremdschlüssel *UserID* und *DienstID*, *Loginname* und *Passwort*. Die letzten beiden Attribute unterscheiden sich von dem *Accountname* und *Passwort* aus der *Account*-Tabelle. Sie werden lediglich bei der Anmeldung über den Dienst benötigt. Bei einer späteren Schnittstelle zwischen dem mobilen Dienst und dem TMS spielt das Paar dann eine wichtigere Rolle. Der Fremdschlüssel *DienstID* gehört zu dem Primärschlüssel aus der Tabelle *Dienste* mit dem weiteren Attribut *Dienstname*. Da der Teilnehmer den Dienst mit verschiedenen Geräten beanspruchen kann, wird die Zuordnung in einer separaten *Profil-Geräte*-Tabelle gespeichert. Diese Tabelle besteht aus den Attributen *ProfilID* und *GeräteID*. Die Tabelle *Geräte* besitzt die Attribute *GeräteID* als Primärschlüssel, *Typ*,

CPU, *RAM*, *Bandbreite*, *Displaygröße* und *UserID*. Das Attribut *Typ* gibt an, um was für ein Gerät es sich handelt. Es kann zum Beispiel ein Handy, PDA, Laptop oder Desktop-PC sein. Weitere Attribute können beliebig ergänzt werden.

Die meisten Daten für die oben genannten Tabellen liefert in der Regel der Benutzer. Andere essentielle Daten, die für die Analyse bedeutend sind, werden vom System selbst in die Datenbank gespeichert. So wird der *Status* und der *Mitgliedsbeginn* in einer *Teilnehmerzusatzinfo*-Tabelle festgehalten. Jede Änderung der Daten, die vom Teilnehmer durchgeführt wird, wird im Protokoll in der *Transaktion*-Tabelle festgehalten. Diese Tabelle besitzt die Attribute *UserID*, *Zeit*, *Aktion*, *Änderungsattribut*, *alter Wert* und *neuer Wert*. Die Werte von *Aktion* können *Änderung*, *Löschvorgang* oder *Hinzufügen* sein. In den Fällen *Änderung* und *Löschvorgang* wird das geänderte Attribut in der Spalte *Änderungsattribut* festgehalten.

Eine letzte Tabelle protokolliert die Online-Zeit. Die *Onlinezeit*-Tabelle bekommt die Attribute *UserID*, *Loginzeit*, *Logoutzeit* und *Loginort*. *Loginort* gibt an, über welchen Dienst der Teilnehmer sich einloggt. Meldet der Teilnehmer sich zum Beispiel beim TMS an, wird eine „0“ als Wert in die Spalte *Loginort* eingetragen. Bei allen anderen Diensten wird die *DienstID* aus der *Dienst*-Tabelle in die Spalte geschrieben. So weiß das TMS, welche Dienste der Teilnehmer benutzt und wie lange der Dienst beansprucht wurde.

Abbildung 2.4 zeigt das endgültige Datenmodell.

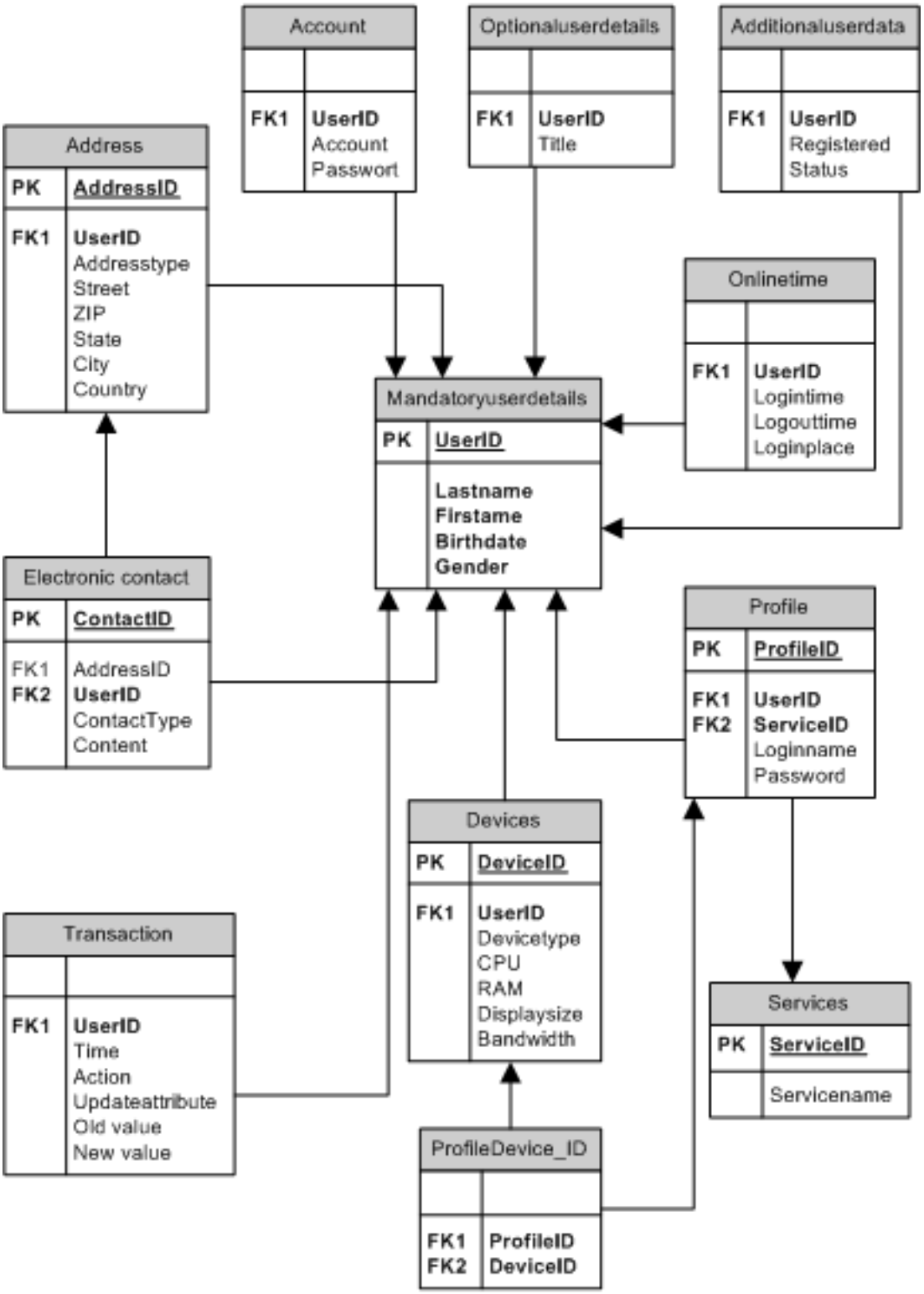

Abbildung 2.4: Das Datenmodell für das TMS

2.2 PostgreSQL

2.2.1 Einleitung

PostgreSQL ist ein objektrelationales Open-Source-Datenbankmanagementsystem, das vom Informatikinstitut der Universität von Kalifornien in Berkeley entwickelt wurde und seit 1996 offiziell den Namen trägt. Wie viele anderen Datenbanksysteme ist auch PostgreSQL konform mit dem SQL92/SQL99-Standard und bietet viele moderne Funktionalitäten, wie Trigger, Views, komplexe Abfragen oder Multiversionskontrolle.

Der Vorläufer von PostgreSQL war POSTGRES. Die Entwicklung des Systems begann 1986 als ein Nachfolger des RDBMS Ingres. Bis 1994 hat man sich hauptsächlich auf die Portierbarkeit und Zuverlässigkeit konzentriert. Im Jahre 1994 wurde ein SQL-Interpreter in POSTGRES eingebaut und viele interne Veränderungen verbesserten die Leistung und Wartungsfreundlichkeit. Getauft auf den Namen Postgres95 wurde es schließlich im Web freigegeben. Mit der rasanten Entwicklung des World Wide Web wurde auch die - nun PostgreSQL genannte - Open-Source Datenbank schnell weiterentwickelt. Auch heute noch erfährt PostgreSQL ständige Verbesserungen.

2.2.2 Datentypen

Für das TMS werden ausschließlich die Datentypen `Character`, `Biginteger`, `Bigserial`, `Integer`, `Date` und `Timestamp` verwendet. Die meisten Datentypen der Tabellenattribute haben *variable Charakterlänge*. Die IDs, die Primärschlüssel sind, sind vom Typ `Bigserial`. Dieser Integer-Typ wird vom Datenbankmanagementsystem automatisch angelegt und inkrementiert. Alle anderen Fremdschlüssel-IDs sind vom Typ `Biginteger`. Das *Geschlecht* aus der *Pflichtdaten*-Tabelle ist vom Typ `Character` der Länge eins. Das *Geburtstdatum* hat den Datentyp `Date` und ist vier Bytes groß. Dieser Typ unterstützt Datumseingaben, die in Europa oder in Amerika gängig sind. Um eine genaue Onlinezeit zu protokollieren wird dem Attribut der Datenyp `Timestamp` gegeben.

Die genaue Syntax der Tabellenanlegung ist im Anhang. Die Größe und der Wertebereich

der Datentypen sind in der folgenden Tabelle aufgelistet.

Datentyp	**Größe**	**Wertebereich**
Bigserial	8 Byte	1 bis 9223372036854775807
Biginteger	8 Byte	-9223372036854775808 bis +9223372036854775807
Integer	4 Byte	-2147483648 bis +2147483647
Date	4 Byte	
Timestamp	8 Bytes	

Tabelle 2.1: Die Datentypen

Kapitel 3

Java

3.1 Einführung

Java ist eine objektorientierte Programmiersprache, die von Sun Microsystems entwickelt wurde. Die rasante Entwicklung seit 1995 verdankt Java seinen vielfältigen Vorteilen. Java wird heute von vielen Softwareentwicklern für die verschiedensten Anwendungen verwendet. Die Programmiersprache ist eng an die Syntax der Sprachen C, C++ angelehnt, wurde jedoch in vielen Bereichen vereinfacht. Zum Beispiel wurde das Verwenden von Zeigern vermieden, das Freigeben nicht mehr benutzer Resourcen geschiet automatisch (garbage collection). Der Java-Compiler erzeugt einen Bytecode, der vom Java-Interpreter ausgeführt wird. Der Bytecode ist plattformunabhängig und kann auf allen Rechnern ausgeführt werden, die eine Java Virtual Machine installiert haben.

Mit der umfangreichen Bibliothek sind bei Java schon viele Standard-APIs vorhanden. So können Java-Anwendungen auf über das Netz verteilte Objekte zugreifen, leicht ein Datenbank-Objekt instanziieren und Klassen von Java-Quellcodes signieren, so dass auch die Sicherheit bei Java gewährleistet wird.

3.2 Datenbankzugriff mit JDBC

Java Database Connectivity (JDBC) ist eine Programmierschnittstelle zwischen dem Java-Programm und der relationalen Datenbank. Mit Hilfe der Schnittstelle können vom Programm aus einfach Datanbankverbindungen auf- und abgebaut, SQL-Anfragen an die Datenbank gestellt und deren Ergebnisse in aufbereiteter Form weiterverarbeitet werden. Ein großer Vorteil von JDBC ist, dass die verwendete Datenbank für das Java-Programm transparent ist. Die Auswahl des richtigen Treibers erledigt JDBC automatisch. Somit kann das Datenbankmanagementsystem (DBMS) ohne Portieraufwand gewechselt werden. Die Plattformunabhängigkeit der Java-Applikation bleibt zusätzlich erhalten, wenn der Treiber außerdem reinen Java-Code enthält.

JDBC besteht im Wesentlichen aus zwei Komponenten, den Datenbanktreibern und dem Treibermanager. Ein JDBC-Treiber ist eine datenbank- und herstellerabhängige Klasse. Er enthält die eigentliche Implementierung für die Kommunikation und ist für den Anschluss eines Datenbanksystems wie PostgreSQL oder DB/2 verantwortlich. Die Hauptaufgabe des Treibermanagers ist das Verwalten der Treiberobjekte. Beim Ausführen des Java-Programms registriert sich der Treiber selbst beim Manager. Alle Klassen und Interfaces sind in den Java-Packages `java.sql` und `javax.sql` gespeichert. `java.sql` enthält alle Standard-Klassen, während `javax.sql` eine Erweiterung der `java.sql`-Klasse ist. Folgende Abläufe sind notwendig, um eine Datenbankverbindung herzustellen. Zuerst muss, wie in diesem Fall exemplarisch gezeigt, der Treiber geladen werden:

Class.forName(„org.postgresql.Driver“);

Das Registrieren am Manager passiert automatisch. Als nächstes verbindet sich der Manager mit der Datenbank. Er stellt die Methode *getConnection* bereit, die mit verschiedenen Parametern aufgerufen werden kann.

Connection dbcon = DriverManager.getConnection(dburl, login, password);

Die URL der Datenbank muss immer angegeben werden, da die Datenbank nicht lokal sein muss, sondern auch auf einem entferntem Rechner sein kann. Die URL hat folgenden Aufbau:

dburl = „jdbc:<Protokoll>:<Datenbank-URL>“

Der Manager wählt entsprechend dem angegebenen Protokoll den passenden Treiber aus seiner Treiberliste aus. Wenn der Verbindungsaufbau erfolgreich war, wird ein Connection-Objekt zurück geliefert. Mit dem Connection-Objekt kann man nun Statements erzeugen und darauf die SQL-Anfragen stellen.

3.3 User Interface (UI)

Das TMS verwendet das Model-View-Controller Pattern (Abschnitt 5.1). Die graphische Benutzeroberfläche bildet eine eigenständige Einheit, und es gibt mehrere Arten sie zu realisieren. Es werden folgende drei graphische Benutzerschnittstellen vorgestellt, die für das TMS möglich sind. Andere UIs, wie beispielsweise die Konsole, sind nicht benutzerfreundlich und deshalb nicht geeignet für das TMS.

3.3.1 Applikation

Eine Applikation ist ein in Java geschriebenes Programm. Der Bytecode, der durch das Kompilieren des Java-Quellcodes entsteht, kann direkt ausgeführt werden.

Die Benutzeroberfläche wird mit Komponenten aus den Packages *java.awt* und *javax.swing* realisiert. Die Klassen in diesen Packages können nach Funktionalität in vier Kategorien eingeteilt werden:

- *Atomare Komponenten* implementieren eine einfache Oberflächenfunktionalität, wie z.B. das Anzeigen eines Labels oder die Auswahlmöglichkeit einer Checkbox.
- *Container* gruppieren mehrere Komponenten zu einer.
- *Layout-Manager* ist für die Anordnung der einzelnen Komponenten innerhalb eines Containers verantwortlich.
- *Listener* wachen über ein auftretendes Ereignis und verarbeiten dieses.

Das TMS verwendet Applikationen für die graphische Benutzeroberfläche. Für Benutzer, die keine Java Entwicklungsumgebung haben und die Applikation nicht ausführen können, kann als Abhilfe Java Web Start dienen. Java Web Start (JWS) ist eine von Java entwickelte Technologie, mit der der Benutzer die Applikation starten kann. Nach der Installation von JWS und Java 2 Runtime Environement (JRE) kann der Benutzer sogar über das Netz die Applikation verwenden. Die Applikation läuft in einer Java-Laufzeitumgebung und unterscheidet sich nicht von lokal installierten Applikationen. Läd der Benutzer die Applikation zum ersten Mal runter, verifiziert JWS die Applikation und speichert sie lokal auf dem Rechner, so dass auch eine Anwendung Offline-Anwendung möglich ist. Eventuelle Updates der Applikation erledigt JWS automatisch. Bei einer Netzwerkverbindung fragt JWS den Server der Applikation, ob eine neue Version vorhanden ist, und läd sie gegebenfalls runter.

JWS bietet außerdem eine Sicherheit durch das Sandbox-Prinzip. Applikationen dürfen wie Applets nicht ohne die explizite Zustimmung des Benutzers auf lokale Resourcen des Rechners zugreifen.

3.3.2 Applet

Ein Applet enthält ebenfalls nur reinen Java-Bytecode. Es ist im Gegensatz zur Applikation kein eigenständiges Programm, sondern wird von einem Java-fähigen Browser oder AppletViewer ausgeführt. Damit das Applet im Browser angezeigt werden kann, wird der Bytecode einfach in eine HTML-Datei eingebettet. Der Browser erkennt beim Parsen an einem speziellem Tag, dass ein Applet folgt und startet den Java-Interpreter.

Applets laufen in einer streng gesicherten *Sandbox*. Dadurch werden gefährliche Operationen, wie das Zugreifen auf die lokal gespeicherten Dateien oder auf das Netzwerk sowie das Ausführen lokaler Programme verhindert. Um diese stark eingeschränkte Umgebung von Applets zu erweitern und dem Programmierer mehr Freiheit zu geben, gibt es seit Java 2.0 zwei verschiedene Möglichkeiten, dem Applet mehr Rechte zuzugestehen.

Die einfachste Methode ist besteht in der Erstellung einer *Policy-Datei*, die lokal auf dem Rechner des Benutzers abgelegt wird und Richtlinien für das Applet enthält. Ein nicht selbst geschriebenes Applet, das z.B. auf das Netz zugreifen möchte, dem der Benut-

zer aber in seiner Policy-Datei nicht das Recht gegeben hat, wird mit einer Fehlermeldung abgebrochen. Die zweite Möglichkeit ist die Verwendung eines *Zertifikats* durch den Applet-Entwickler. Er steht für die Sicherheit des Applets gegenüber dem Applet-Anwender ein. Der Applet-Anwender muss lediglich die Sicherheitsabfrage akzeptieren, damit das Applet die angeforderten Rechte zugeteilt bekommt. Das Zertifikat beruht auf einer Public-Key-Infrastructure (PKI). Kryptographische Methoden von Zertifikaten haben einen asymmetrischen Verschlüsselungsalgorithmus. Das bedeutet, dass der Teilnehmer einen privaten und einen öffentlichen Schlüssel hat, und die Nachricht mit beiden Schlüsseln ver- und entschlüsselt. Der öffentliche Schlüssel wird von der höheren Hierarchie „unterschrieben“. In der Spitze der Hierarchie steht die Root Certification Authority (Root CA).

Um das Applet nun zu signieren, stellt Java das Tool *keytool -genkey* bereit, das das Schlüsselpaar und ein selbstsigniertes Zertifikat automatisch erzeugt. „Echte“ Zertifikate müssen von einer anerkannten Zertifizierungsstelle unterschrieben werden. Der Bytecode eines Applets muss in ein JAR (Java ARchive) File gepackt werden, bevor es endgültig mit dem Java-Tool *jarsigner* unterschrieben wird. Das Problem ist nun, dass dem Applet keine benutzerseitige Sicherheitsberechtigung (durch Policies) gegeben werden kann. Vielmehr bekommt es nur die Freiheiten, die der Applet-Enwickler bestimmt hat.

3.3.3 Web-Frontend

Als Web-Frontend wird der Teil einer Internet-Anwendung bezeichnet, der für den Benutzer über einen Browser sichtbar ist. Die Gestaltungsmöglichkeiten eines Web-Frontends sind vielfältig. So kann die graphische Oberfläche mit Hilfe von Flash, JavaScript, XHTML, DHTML etc. dynamisch gestaltet werden. Hinter dem sichtbaren Teil verbirgt sich die komplexe Programmstruktur des Webservers. Hier bietet sich ebenfalls eine breite Palette an, um den Webserver effizient und dynamisch zu gestalten. Als Beispiel könnte das *Common Gateway Interface* (CGI) als Schnittstelle benutzt werden. Die CGI-Programme können in vielen Sprachen wie z.B. Python oder Perl verfasst werden. Der Browser baut eine Verbindung zum Server auf, welcher das CGI-Programm aufruft und dieses dann dynamisch eine HTML-Seite erzeugt. Trend ist jedoch schon länger die Integrierung der Funktionalitäten im Webserver. Repräsentante Beispiele sind Servlets oder

Java Server Pages die eigenständig im Webserver laufen und dynamisch HTML-Seiten erzeugen.

3.4 Client-Server-Kommunikation

Bei einer Client-Server-Kommunikation über das Netz, kommt man nicht an der Socketprogrammierung vorbei. Obwohl die Socketprogrammierung bei Java im Gegensatz zu anderen Programmiersprachen schon stark vereinfacht ist, muss sich der Programmierer trotzdem Gedanken über das Protokoll oder die Serialisierung machen. Für die Implemetierung des TMS wurden zwei Techniken in Erwägung gezogen, die die Datenübertragung ermöglichen sollten.

3.4.1 Java Message Service

Java Message Service (JMS) ist eine Programmierschnittstelle (API) zum asynchronen und synchronen Austausch von Nachrichten (SuJ). Bei mobilen Endgeräten, die oft die Netzwerkverbindung verlieren, bietet sich JMS hervorragend an, da die Zustellung der Nachrichten gewährleistet wird. JMS ist Bestandteil der Java 2 Plattform, Enterprise Edition (J2EE). Um es zu verwenden, muss auf beiden Seiten der J2EE Applikationsserver laufen.

Die JMS-API stellt zwei verschiedene Modelle zur Verfügung: Beim *Point-to-Point-Queuing*-Modell (P2P) kommuniziert der Dienst (Producer) mit nur einem Client (Consumer). Dieser holt sich die Nachrichten selbst aus dem Übertragungskanal (der Queue) ab und entfernt die Nachricht anschließend aus der Queue. Sowohl der Dienst als auch der Client können *Producer* und *Consumer* sein.

Abbildung 3.1: Das Point-to-Point-Queuing-Modell

Das zweite Modell ist das *Publish-and-Subscribe* (Pub/Sub). Es verwendet eine 1-zu-n-Übermittlung. Dieses Modell wird verwendet, wenn mehrere Clients Nachrichten von einem Dienst bekommen. Der Dienst, auch *Producer* genannt, schickt die Nachricht durch

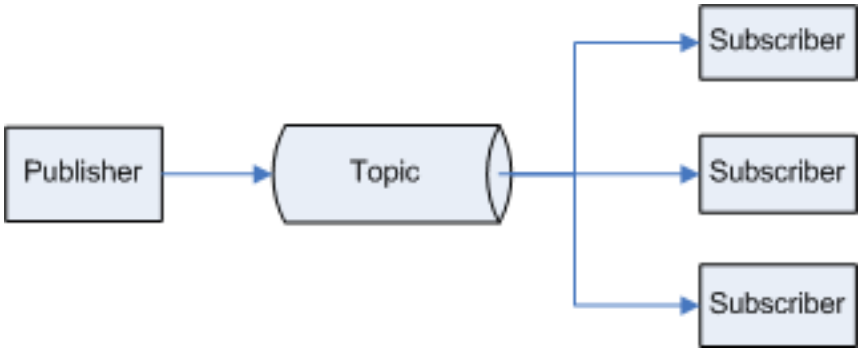

Abbildung 3.2: Das Publish-and-Subsribe-Modell

einen virtuellen Kanal (Topic). Clients, hier *Subscriber* genannt, die diesen Topic abonniert haben, erhalten dann die Nachricht. Für Clients besteht auch die Möglichkeit, auf die Nachrichten zu antworten.

3.4.2 Remote Method Invocation

Remote Method Invocation (RMI) ist eine Technik, bei der das Programm Methoden aufruft, die auf einer anderen Virtual Machine laufen (SuR). RMI fungiert als Protokoll über TCP/IP und erledigt die ganze Netzverkverbindung. Eine sichere Verbindung mit SSL ist möglich. Alle Interfaces und Remote-Exceptions, die für das Verwenden von RMI benötigt werden, befinden sich in dem *java.rmi*-Paket.

Der Server erzeugt ein oder mehrere Remote-Objekte und definiert in einem Remote-Interface die zugehörigen Methoden, die für andere Rechner aufrufbar sind. Die eigentlichen Methoden sind in einer anderen Klasse implementiert, die vom Remote-Interface abgeleitet wurde.

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Object extends Remote {
```

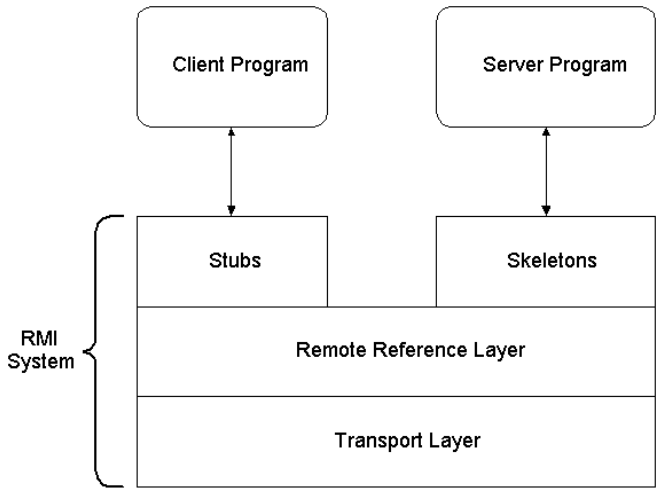

Abbildung 3.3: Architektur von Remote Method Invocation

```
        public Vector login(String login, String pwd) throws RemoteException;
        public Vector getUsersDetails() throws RemoteException;
}
```

Das erzeugte Remote-Objekt wird in einem *Namensdienst* (Registry) registriert, der von potentiellen Clients abgefragt werden kann.

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.RemoteServer;
import java.rmi.server.UnicastRemoteObject;

/* Remote-Objekt Erzeugung */
ObjectImpl admin = new ObjectImpl();
Object stub = (Object) UnicastRemoteObject.exportObject( serverobject, 0 );
RemoteServer.setLog( System.out );

/* Registrieren an dem Namensregister */
try {
    registry.rebind(„serverobject“,stub);
} catch(ConnectException e) {
```

System.out.println(„Object not bound“);
}

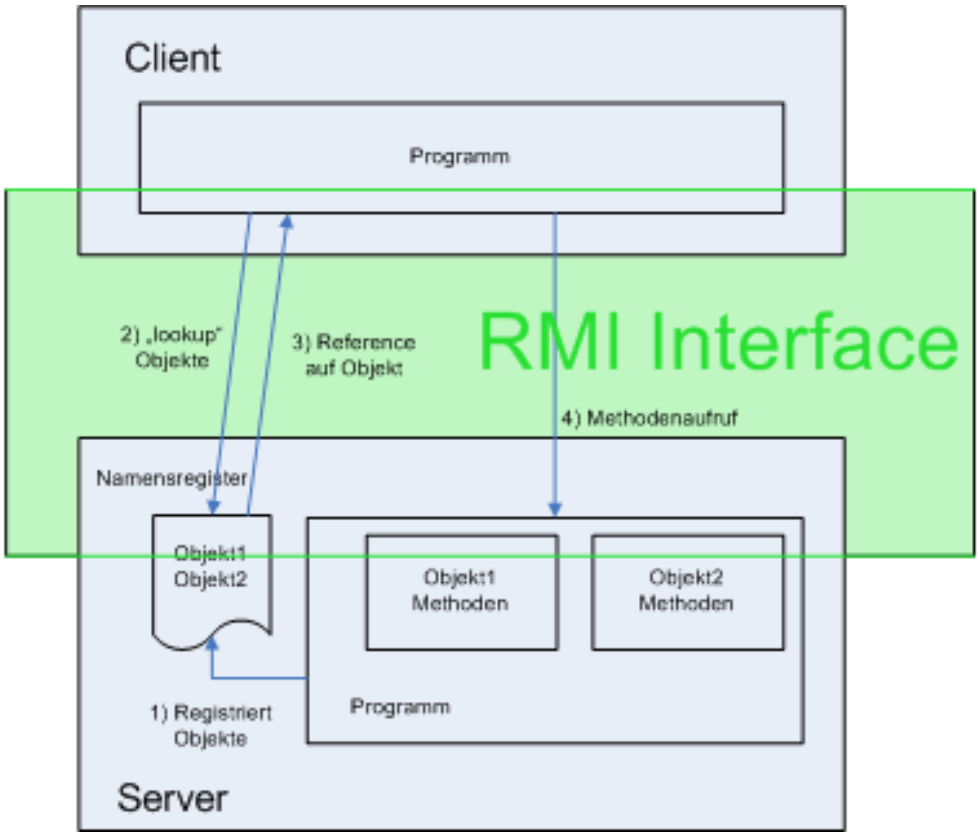

Abbildung 3.4: Der Ablauf von RMI

Die Stellvertreterobjekte *Stub* und *Skeleton* (siehe 3.3) sind für die Netzwerkverbindung zwischen Client und Server verantwortlich. Java bietet das Tool *rmic* an, das die Objekte, welche im Prinzip Methoden sind, selbst erzeugt. So braucht der Programmierer sich nicht weiter um die Kommunikation zu kümmern.

Die Registry muss mit dem Befehl *rmiregistry* gestartet werden, damit der Client die entfernten Objekte finden kann.

Der Client besorgt sich lediglich eine Referenz auf das benötigte Objekt. Bei erfolgreicher Beschaffung, kann der Client die Methoden aufrufen, die sich so verhalten, als ob sie lokal da wären.

import java.rmi.registry.LocateRegistry;

```
import java.rmi.registry.Registry;

Registry registry;
Object clientobject;

try {
    registry = LocateRegistry.getRegistry();
    clientobject = (Object) registry.lookup( „serverobject“ );
 } catch(java.rmi.NotBoundException e) {
    System.out.println(„ Object not found “);
}
```

3.4.3 Remote Method Invocation vs. Java Message Service

Beide Techniken eignen sich hervorragend für das TMS. JMS garantiert, dass Daten beim Ziel ankommen, auch wenn die Netzwerkverbindung für eine kurze Zeit unterbrochen wird. RMI dagegen bietet eine Verteilung der Arbeit auf verschiedene Rechner, somit können zeitaufwendige Prozeduren vom Server erledigt oder gleichmäßig verteilt werden. RMI-Fehler werden mittels Remote-Exceptions abgefangen und können im weiteren Programmablauf behandelt werden.

	RMI	**JMS**
Vorteil	* Verteilung der Aufgaben * entfernter Methodenaufruf * SSL-Verschlüsselung	* Zustellungsgarantie * asynchrone und synchrone Sende- und Empfangmöglichkeit
Nachteil	* keine Garantie auf Datenzustellung bei Netzwerkabbruch	* auf beiden Seiten muss J2EE-Applikationsserver laufen

Tabelle 3.1: Remote Method Invoction vs. Java Message Service

Für das TMS wird RMI verwendet, da die meiste Arbeit vom TMS-Server getan werden soll. Es ist nicht sichergestellt, dass die mobilen Dienste, mit der das TMS später kommuniziert, auch mit der J2EE-Plattform arbeiten. Die Platform von mobilen Endgeräten sind stark begrenzt, so dass sich der J2EE-Applikationsserver nicht auf allen Geräten installieren lässt. Mit RMI können entfernte Methoden aufgerufen werden, die sich so verhalten, als ob sie lokale Methoden wären. Ein vorteilhafter Nebeneffekt ist, dass dem Programmierer die Arbeit der Client-Server-Kommunikation erspart bleibt.

Kapitel 4

Funktionsbeschreibung

4.1 Funktionalitäten

Das TMS ist aufgeteilt in Client und Server. Der Client ist der Verarbeiter der Daten, während der Server nur Daten speichert und aufbereitet anbietet.

4.1.1 Client

Die Hauptfunktion des Clients ist die Dateneingabe, die Kontrolle, der Lösch- und Updatevorgänge. Mit Hilfe der GUI steuert der Benutzer das ganze Geschehen und manipuliert die Daten. Es gibt zwei Benutzerrollen. Nach dem Starten der herunter geladenen Software, kann sich der Benutzer entweder als Teilnehmer oder als Administrator sich anmelden.

Teilnehmer

Der Teilnehmer kann sich am Anfang entweder einloggen oder neu registrieren, falls er noch keinen Account hat.

Hat er sich beim System erfolgreich angemeldet, erscheint das Hauptfenster. Der Teilnehmer hat hier die Möglichkeit seine Daten und Transaktionsprotokolle, aufgeteilt in

Abbildung 4.1: Der Loginscreen

fünf Registerkarteien, anzuschauen. In der ersten Kartei werden seine persönlichen Daten angezeigt. Er hat die Möglichkeit die Einstellung für Namen, Titel und Geschlecht zu ändern. Der Accountname und das Passwort sind ebenfalls beliebig änderbar, sofern der Accountname nicht vergeben ist. Der Vertrauensgrad des Teilnehmers ist in einem Status festgehalten. Neue Teilnehmer bekommen automatisch den Status *User* zugewiesen. Die Teilnehmer haben die Möglichkeit ihren Status zu ändern, indem sie ein Echtheitszertifkat oder ein Pfand hinterlegen. Eine andere Möglichkeit wäre die Einführung von Bewertungspunkten, die Benutzer bei den Diensten sammeln müssen. Die genaue Handhabung des Status bleibt noch offen, solange keine wirklichen Dienste das TMS benutzen. Kontaktdaten, die keiner Adresse zugeordnet sind, befinden sich ebenfalls in der ersten Registerkartei. Der Teilnehmer kann Kontaktdaten, wie E-Mail-Adressen oder Telefonnummern, beliebig hinzufügen oder löschen.

Die nächsten zwei Registerkarteien sind Adresse und Geräte. Der Teilnehmer kann beliebig viele Adressen und mobile Geräte angeben. Zum Einfügen der Adresse wird ein neues Fenster mit einem Formular erzeugt, in das der Teilnehmer die Adressstandarddaten, das sind Straße, Postleitzahl, Stadt, Land und Adresstyp, eintragen kann ().

Außerdem hat er die Möglichkeit seine Kontaktdaten, die zu der Adresse gehört, einzugeben. Wird aber nur diese eingegeben, und ist das Abschicken der Daten erfolgreich,

Abbildung 4.2: Adresseneingabe

so erscheint diese in der ersten Registerkartei. Die Standdarddaten für die Geräte sind Gerätetyp, Displaygröße, Bandbreite, RAM und CPU und werden in der Geräte-Kartei aufgelistet.

In der nächsten Registerkartei sind alle Dienste mit den verwendeten Geräten aufgelistet. Das Auflisten der Dienste geschieht erst, wenn der Dienst das TMS benutzt, und wenn vom Dienst selbst eine Bestätigung der Mitgliedschaft vom Teilnehmer vorliegt. Dann erst wird die Abhängigkeit zwischen Dienst und Teilnehmer in der Datenbank festgehalten und die Software kann die Beziehung zwischen Teilnehmer und Dienst erkennen. Der Teilnehmer hat die Möglichkeit seine mobilen Geräte, die er in der zentralen Datenbank hinterlegt hat, dem Dienst zuzuordnen oder die Zuordnung zu entziehen. Damit kann er dem Dienst mitteilen, mit welchen Geräten er den Dienst beanspruchen möchte (siehe).

Die letzte RegisterKartei ist das Protokoll. Hier werden alle Lösch-, Einfüge- und Updatetransaktionen aufgelistet, die der Teilnehmer durchgeführt hat, absteigend sortiert nach dem Datum.

Abbildung 4.3: Geräteauswahl für den Dienst

Administrator

Das Handwerkszeug des Administrators ist ebenfalls die graphische Benutzeroberfläche. In der ersten Registerkartei werden, statt der Daten des einzelnen Benutzers, alle Teilnehmer aufgelistet. Gezielt kann der Administrator mehrere Teilnehmer auswählen und vergleichen. Außerdem ist er bemächtigt persönliche Teilnehmerdaten, Adresse und Kontaktdaten der Teilnehmer zu ändern.

In der zweiten Registerkartei sind alle Dienste und die Teilnehmer, die den jeweiligen Dienst beanspruchen, verzeichnet. Dadurch kann sich der Administrator einen schnellen Überblick darüber verschaffen, welcher Teilnehmer, mit welchem Loginnamen, Zugang zu welchem Dienst hat.

Ähnlich verhält es sich auch mit der Geräte- Kartei. Dort werden alle Geräte vom jeweiligen Teilnehmer aufgelistet (siehe 4.5).

In der Protokoll-Kartei sind alle Protokolle von Teilnehmern und Administratoren aufgelistet. Das Admin-Tool hat im Gegensatz zum Teilnehmer-Tool zwei weitere Registerkarteien, die Statistik-Kartei und die Such-Kartei. In der Statistik-Kartei werden alle

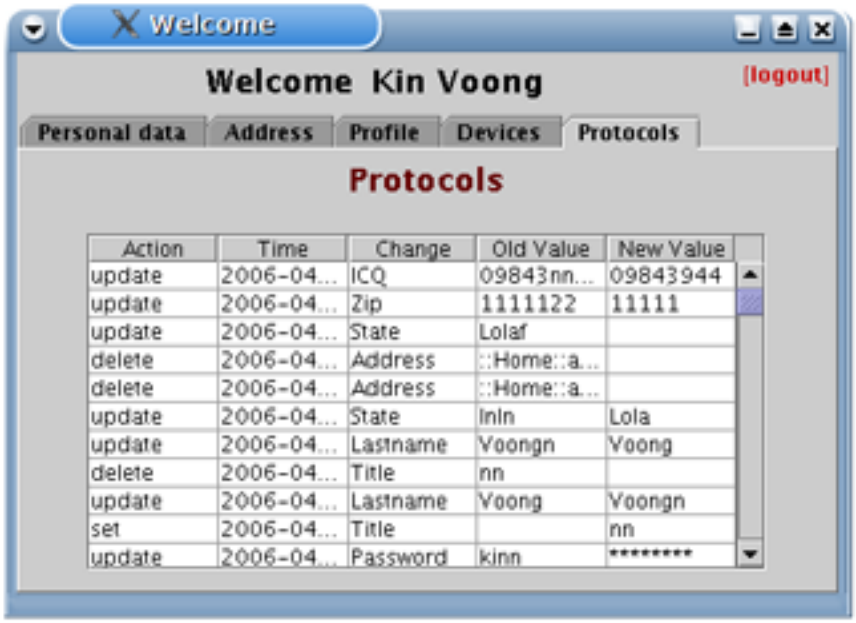

Abbildung 4.4: Das Protokoll

wichtigen Daten von Teilnehmern angezeigt.

Eine wichtige Tätigkeit des Administrators ist das Herausfiltern der Teilnehmer mit bestimmten Merkmalen. Das TMS bietet daher die Such-Kartei mit nützlichen Funktionen. Es kann nach (Teil-) Wörtern in der Rubrik Vor-, Nachname oder Accountname gesucht werden oder nach Alter, Geschlecht und Land selektiert werden. Die Verknüpfung der Kriterien mit AND- oder OR-Operatoren ist möglich (siehe 4.6).

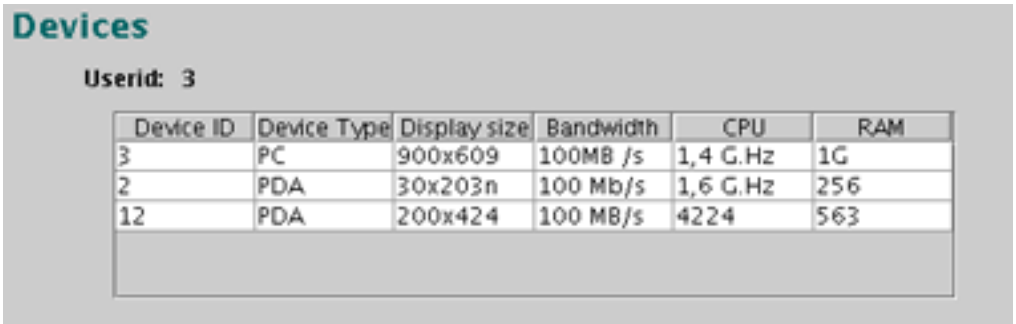

Abbildung 4.5: Geräte von einem Teilnehmer

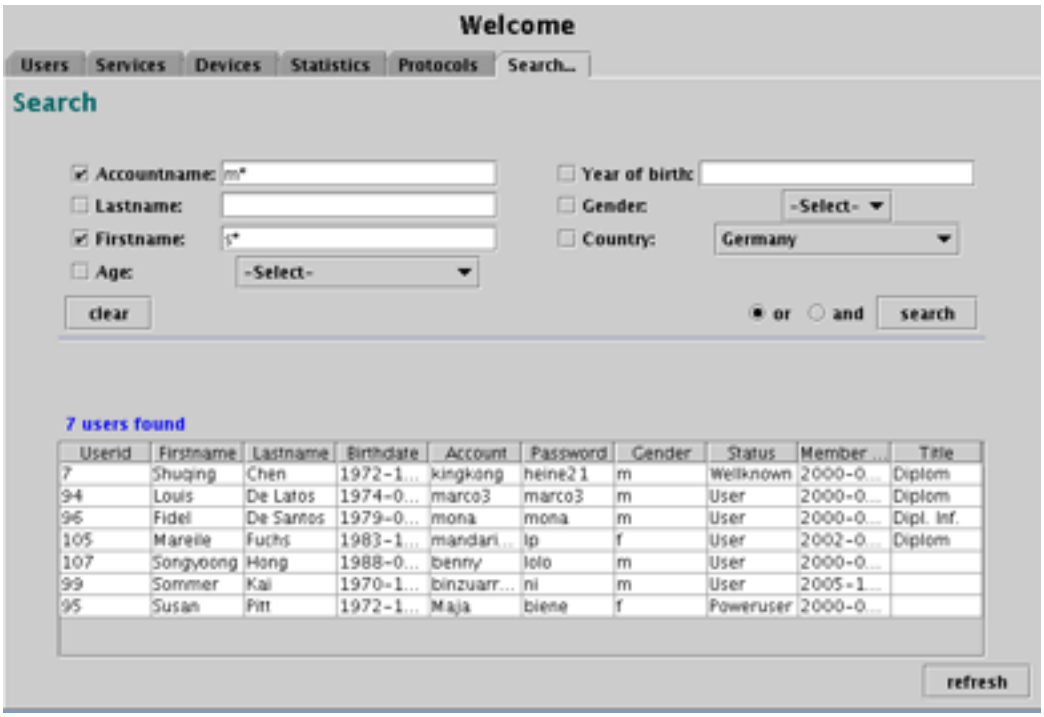

Abbildung 4.6: Suchmöglichkeiten

4.1.2 Server

Die Hauptaufgabe des Servers ist den Client mit Daten aus der Datenbank zu versorgen. Er liest nach Anforderung Teilnehmerdaten aus der Datenbank und schickt sie dem anfragenden Client. Er ist der Liefererant aller Teilnehmerdaten und stellt dem Client Funktionen zur Verfügung, mit denen dieser seine Informationen updaten, löschen einfügen oder anschauen kann.

4.2 Programmablauf

Teilnehmer

Nach dem Start der vom Internet heruntergeladenen Software hat der Teilnehmer die Möglichkeit, sich einzuloggen oder sich zu registrieren. Nach einem erfolgreichen Ein-

loggen, kommt der Benutzer zum Hauptfenster. Von dort aus hat er die Möglichkeit, seine persönlichen Daten, Adressen, E-Kontakte und Geräte in einem separaten Fenster zu aktualisieren. War die Aktualisierung erfolgreich, schließt sich das Fenster und das Hauptfenster wird auf den neusten Stand gebracht. Bei einer erfolglosen Aktion wird eine Fehlermeldung ausgegeben und der Benutzer kann eine Korrektur vornehmen oder bei einem Netzwerkausfall das Fenster schließen und später die Änderung nochmals durchführen. Beim Hinzufügen einer neuen Adresse, eines Gerätes oder E-Kontakts erzeugt das System ein Formular in einem neuen Fenster. Auch hier werden eventuell Fehlermeldungen angezeigt und bei erfolgreichem Update das Fenster geschlossen. Beim Löschen eines Geräte- oder Adress-Eintrags muss der Benutzer eine Sicherheitsabfrage vor dem endgültigen Entfernen aus der Datenbank bestätigen. Vom Hauptfenster aus gelangt man zudem zu einem Fenster, in dem man Geräte einem Dienst zuordnen kann. Nach einem erfolgreichem Update befindet sich der Benutzer wieder im Hauptfenster.

Der Registrierdialog besteht aus einem Hauptformular, das dem Benutzer die wichtigsten Daten abfragt. Da es sich hierbei um Pflichtdaten handelt, werden diese zunächst von der Middleware überprüft. Wurden keine oder fehlerhafte Eingaben gemacht, erscheint im Fenster eine Nachricht, die mitteilt welche Änderung der Teilnhemer vornehmen muss. Ist die Eingabe dann in Ordnung, werden die Pflichtdaten zum Server weitergeleitet. Ein erfolreiches Anlegen des Accounts öffnet das nächste Fenster, in dem der Teilnehmer zusätzlich seine Adresse eingeben kann. Möchte er dies nicht tun, kann er direkt zum Hauptfenster mit dem Abbrechknopf gelangen. Bei nicht erfolgreicher Registrierung oder Adressanlegung wird eine Fehlermeldung ausgegeben. Abbildung 4.7 zeigt den Programmablauf einer Teilnehmer-Sitzung.

Administrator

Nach Starten der Software, kann der Administrator sich mit seinem Namen und Passwort in das TMS einloggen. Er hat die Möglichkeit, gezielt Teilnehmerdaten anzuschauen und zu ändern. Wie bei dem Teilnehmer wird ein Aktionsfenster erzeugt, in dem der Administrator die Änderungen durchführen kann. Die Eingabe wandert durch die Middleware zum Server und eine Bestätigung nimmt den gleichen Weg wieder zurück. Beispielsweise beim Suchen von Teilnehmern mit bestimmten Merkmalen in der Such-Kartei: Der

Administrator schickt seine Anfrage ab – gefundene Daten aus der Datenbank werden zurückgeschickt und aufgelistet. Netzwerkausfall oder Eingabefehler werden in einem Fenster angezeigt und verhindern das Weiterleiten der Eingabe zum Server. Abbildung 4.8 zeigt den Programmablauf einer Administrator-Sitzung.

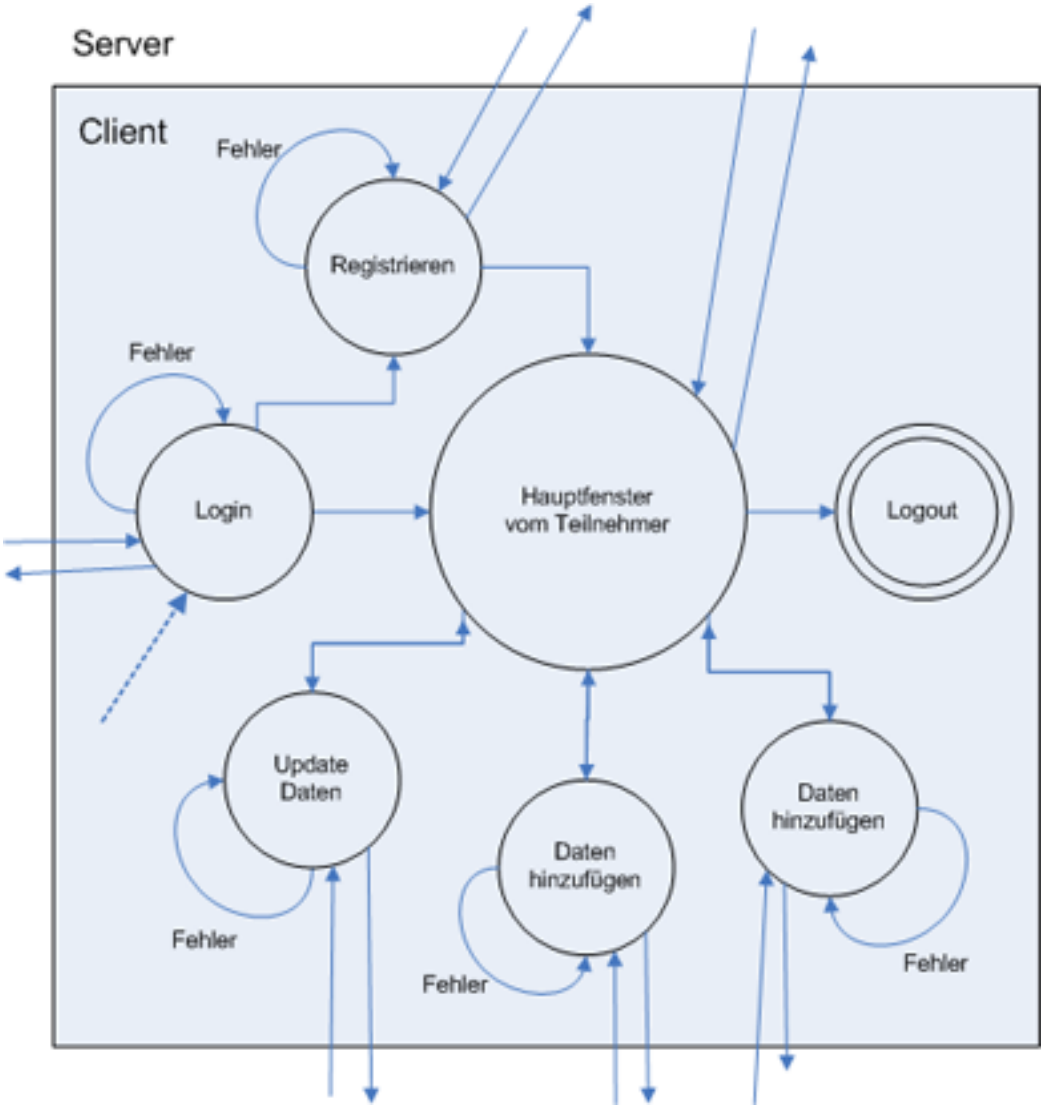

Abbildung 4.7: Der Programmablauf eines Teilnehmers

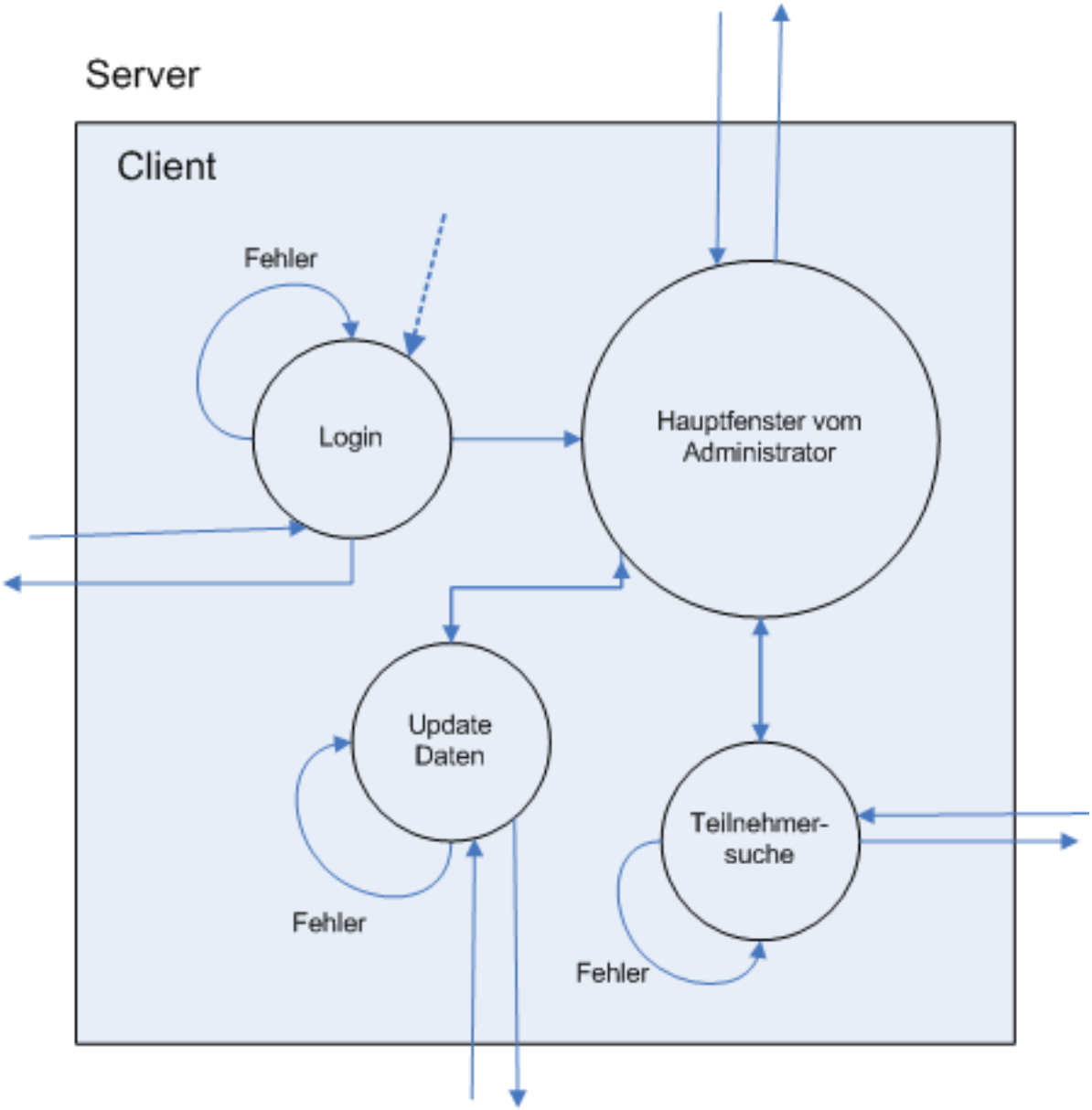

Abbildung 4.8: Der Programmablauf eines Administrators

Kapitel 5

Implementation

Das ganze TMS ist in Java geschrieben, wodurch Plattformunabhängikeit und Portierbarkeit gewährleistet werden. (Abschnitt 3). Als Entwicklungsumgebung wurde die Java-IDE *Netbeans* verwendet. Die Software ist modular aufgebaut und umfasst die Schnittstelle zwischen dem Client und dem Server, die Datenbanklogik und -aufbau und die graphische Benutzeroberfläche. Für die Client-Server-Kommunikation verwendet das TMS die RMI-Technik. Da Java objektorientiert ist, bietet sich das Anlegen weiterer Datenobjekte hervorragend an. Diese Objekte sind *Person*, *Address*, *EContact*, *Device* und *Profile*. Durch die Objekte wird das Portieren der Teilnehmerdaten vereinfacht und das Hinzufügen weiterer Attribute erleichtert. Dies macht das Programm fehlerresistenter und trägt allgemein zur Stabilität des Systems bei.

Person	**Address**	**EContact**	**Device**	**Profile**
persId	addrId	eId	dId	profileId
persfname	addrType	userId	dType	serviceName
perslname	addrStreet	addrId	dDisplaysize	userId
persbirthd	addrZip	eType	dCPU	serviceLoginName
persgender	addrState	eContent	dRAM	serviceLoginPassword
perstitle	addrCity		dBandwidth	
persaccount	addrCountry		dUserId	
perspwd				
persstatus				
persregistered				

Tabelle 5.1: Die Datentypen

5.1 Model-View-Controller

Was eine gute Software ausmacht, ist eine klare Strukturierung und Aufteilung der einzelnen Komponenten. Das Konzept des Model-View-Controller (MVC) ist geradezu prädestiniert für das TMS. Die Idee des MVC besteht in der logischen Aufteilung der Software in *Datenmodell*, *Präsentation* und *Programmsteuerung* (SuM).

Die Einheit des *Datenmodells* beinhaltet lediglich die reinen Daten und die API zur Datenbank. Beim TMS wird diese Einheit durch die PostgreSQL Datenbank und der API repräsentiert. Die Aufgabe der Datenspeicherung und Verwaltung fallen dieser Einheit zu.

Die *Präsentations*-Einheit ist dafür verantwortlich, dass die Daten dem Benutzer in geeigneter Form dargestellt werden. Sie ist gänzlich abgekoppelt von der internen Datenverarbeitung. Dies bedeutet, dass eine Änderung in der Benutzungsoberfläche keine direkte Auswirkung auf die interne Verarbeitung der Daten und der Datenstruktur hat. Daten können in mehreren unterschiedlichen Präsentationen bzw. Sichten dargestellt werden, zum Beispiel könnte das Teilnehmergeschlecht als Tabelle aufgelistet, als Balken- oder Tortendiagranm angezeigt werden. Die graphische Benutzeroberfläche des TMS hat die Aufgabe verschiedene Sichten zu produzieren und diese dem Benutzer diese zu präsentieren. Sie ist für die aktuelle Darstellung der Eingangs- und Ausgangsdaten in den jeweiligen Anzeigeobjekten (Editfelder, Menüs, Dialogboxen etc.) verantwortlich.

Die Logik und somit der Kontrollfluss obliegt der *Controller*-Komponente. Diese mittlere Ebene ist die Brücke von der graphischen Benutzeroberfläche zur Datenbank. Ausgabedaten werden von der Datenmodell-Komponente an die Controller-Komponente weitergegeben. Die Rückrichtung mit den Eingaben passiert ebenfalls die Controller-Einheit. Änderungen der Modelldaten werden also vom Controller gesteuert. Im TMS repräsentiert die Middleware die Controller-Einheit.

Das flexible Modelldesign schafft eine klare Ordnung im System und bewahrt die Transparenz. Jede Einheit kann so von Spezialisten weitergeführt werden ohne Beeinflussung von anderen Einheiten. Zum Beispiel kann der Datenbankentwickler die Datenbank neu modellieren und der Designer die Benutzeroberfläche umgestalten. Der Softwareent-

wickler kann an der Effizient der Controller-Einheit weiterarbeiten und sogar die Client-Server-Kommunikation durch eine andere Technik ersetzen.

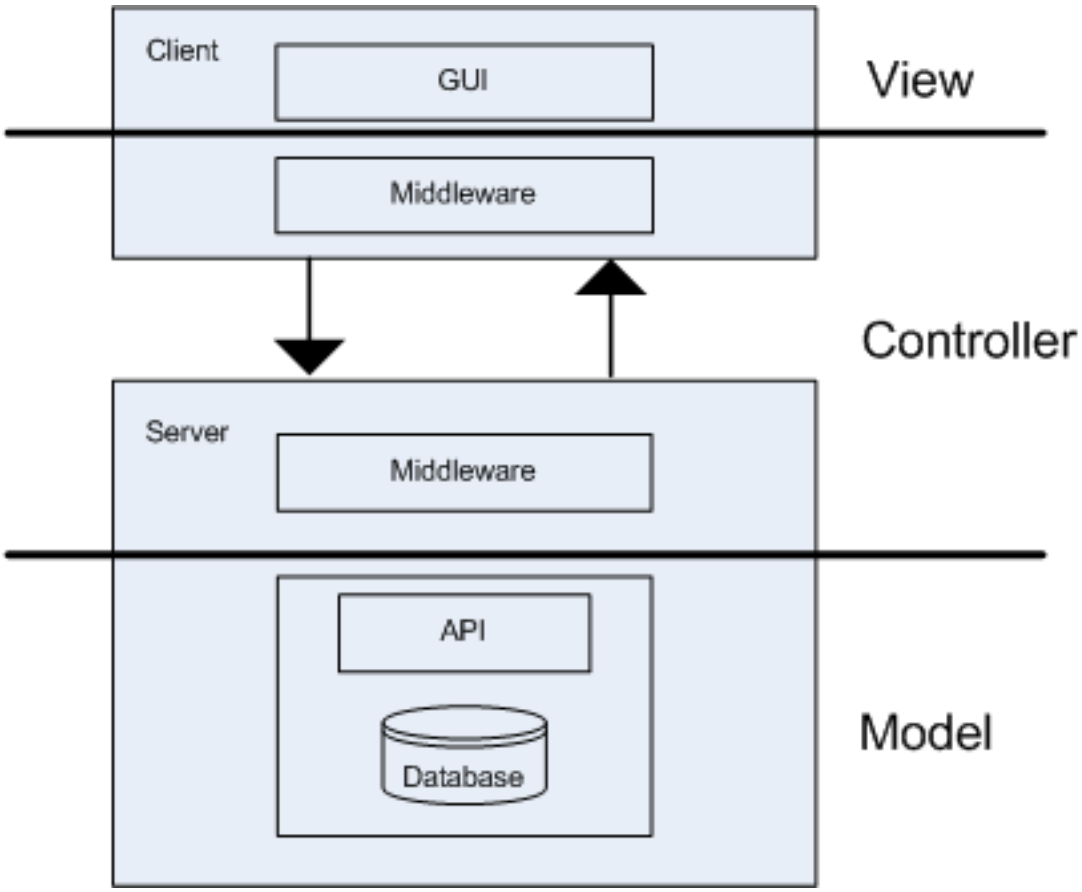

Abbildung 5.1: Model-View-Controller-Konzept für das TMS

5.2 Server

Der Server ist aufgeteilt in einen Middleware-Teil, welcher zur *Controller*-Einheit gehört, und einen Datenbankbereich. Die Middleware nimmt Anfragedaten vom Client oder geforderte Daten von der Datenbank entgegen, und leitet sie an die entsprechend andere Partei weiter.

Ein Teil der Middleware ist in den Datenbankbereich integriert. Dieser Teil der Datenmodell-Komponente kennt die Datenbankstruktur und ist auch für den Lauf der Daten verant-

wortlich. Welche Daten zu welcher Datenbanktabelle hinzugefügt oder gelöscht werden, wird hier gesteuert.

5.2.1 Middleware

Die Middleware ist die Schicht zwischen dem Client und der Datenbank. Hier ist die RMI-Schnittstelle zur Client-Middleware implementiert. Im Server muss für jedes Objekt eine Schnittstelle deklariert werden. Diese Objekte sind *Admin* und *User* und befinden sich in den Klassen *Admin.java* und *User.java*, die von der Klasse *Remote* abgeleitet werden müssen. Die zugehörigen Methoden sind in den Klassen *AdminImpl.java* und *UserImpl.java* zu finden. Für die weiteren Netzwerkverbindungen hilft uns das Tool *rmic*, das weitere Klassen automatisch anlegt. Nachdem die Objekte angelegt sind, müssen sie bei einem Namensdienst veröffentlicht werden, damit der Client auf die Objekte zugreifen und so die Methoden der Objekte aufrufen kann (Abschnitt 3.4.2).

Einige Methoden werden von Teilnehmern und Administrator gemeinsam verwendet. Methoden, die nur für das Admin-Objekt bzw. User-Objekt geschrieben sind, die aber dasgleiche bezwecken, werden gebündelt, indem eine weitere globale Methode aufgerufen wird, die unabhängig von den Objekten benutzt werden kann. Beispielsweise möchte der Teilnehmer seine persönlichen Daten von der GUI anzeigen lassen. Das User-Objekt in der Middleware ruft seine entfernte Methode auf, um die persönlichen Daten anzufordern. Das Admin-Objekt könnte nun ebenfalls die persönlichen Daten von dem Teilnehmer anfordern, indem er seine entfernte Methode aufruft. Beide Methoden rufen dann im Middleware auf der Serverseite eine gemeinsam benutzte Methode auf, die unabhängig von der Anwenderrolle ist.

Die dazugehörige Logik steckt in der Klasse *MiddleLayer.java* (siehe Abbildung). Sind die Anfragen nicht vertretbar, ist z.B. der Vektor des Parameters leer oder ein NULL-Wert, so wird schon hier der „Fehler“ abgefangen und eine entsprechende Nachricht wird über RMI zurück zum Client verschickt. Das frühe Abfangen von Fehlern hat den Vorteil, dass die Datenbank möglichst unberührt bleibt. Hier werden auch die Zeiten, wann ein Teilnehmer sich registriert oder eingeloggt hat, festgehalten und die Daten werden direkt zur Datenbank-API geleitet.

5.2.2 Datenbanken

Das Modul Datenbanken besteht aus der PostgreSQL-Datenbank selbst und der API. (Bild) Die Methoden, viele davon werden von der Middleware aufgerufen, sind in der *DBLayer*-Klasse implementiert. Diese Klasse arbeitet eng mit der Datenbank zusammen. Sie kennt die Datenbankstruktur, die Tabellen und bekommt durch Methodenaufrufe mit, welche Datenänderungen stattfinden. Das Protokollieren der Datenänderung in der Datenbank gehört deshalb zu ihren Aufgaben.

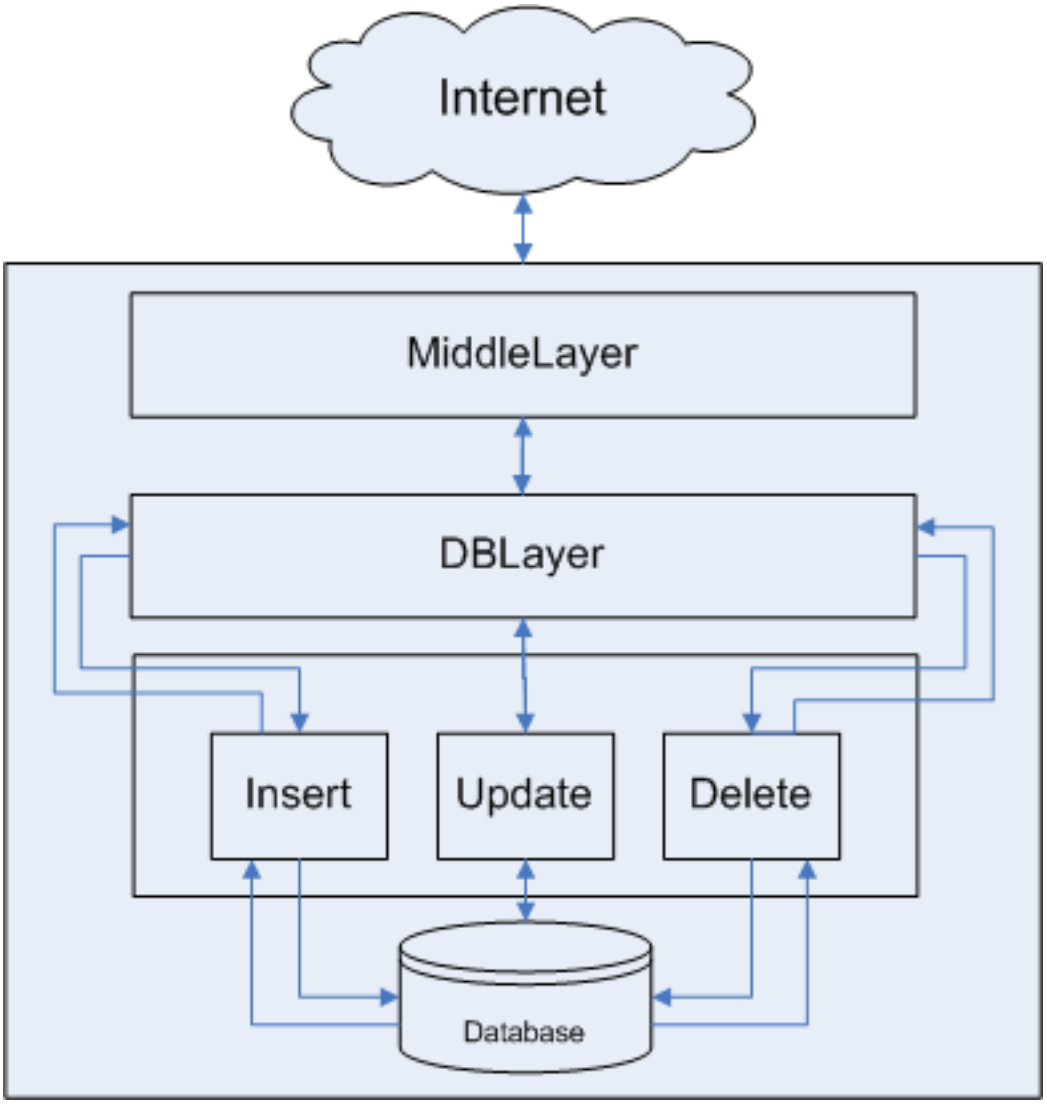

Abbildung 5.2: Die Architektur vom Server

Eine weitere Aufgabe ist, die Daten in der Datenbank zu aktualisieren, zu löschen oder geforderte Daten aus der Datenbank zu lesen. Je nachdem welche Aktion verlangt wird, wird von *DBLayer* die entsprechende Methode in der *Insert-*, *Get-*, *Delete-* oder *Update-*Klasse aufgerufen. Die DBLayer-Klasse ordnet also die Anweisungen nach Funktionalität und ruft die entsprechenden Methoden auf.

Für jede Insert-, Get-, Update- und Deleteoperation gibt es eine separate Klasse. In den Klassen werden hauptsächlich SQL-Anfragen, die „Query“, gestellt, ausgeführt und das Ergebnis zurück geschickt. Die meisten Rückgabewerte sind vom Typ *Vektor* oder *boolean*. Alle vier Klassen benutzen die Standardschnittstelle *DBStandard*, über die Konstanten, wie Datenbank-URL oder Datenbanktreiber, abrufbar sind. Eine ähnliche Gemeinsamkeit aller vier Klassen ist das Verwenden der Methode *sendsql(String query)*. Diese Methode arbeitet eng mit der DBConnection-Klasse zusammen, in der die Datenbankverbindung aufgebaut wird und der Zugriff zur Datenbank stattfindet. Die *sendsql*-Methoden, in der Insert-, Update- und Delete-Klasse, schicken ein boolean-Wert als Ergebnis zurück. Die Get-Klasse arbeitet, statt mit dem *ResultSet*, mit dem *RowSet*. Das RowSet ist flexibler und hat den Vorteil, dass das Ergebnis auch bei einer geschlossener Datenbankverbindung noch zur Vearbeitung zur Verfügung steht. Die Methode, die die *sendsql(String query)* aufgerufen hat, bekommt als Ergebnis das RowSet zurück, und kann dort das Ergebnis in die gewünschte Datenstruktur umwandeln.

Alle vier Klassen benutzen die DBConnection-Klasse, um die Query auszuführen. Die DBConnection-Klasse lädt den Treiber und stellt die Verbindung zur Datenbank her (Abschnitt 3.2). Zwar sollte man die Datenbankverbindung und sämtliche Statements möglichst wieder freigeben, doch sind die Statement-Objekte kostspiele Resourcen (Krü05). Deshalb speichert die Klasse *DBConnection* Statement-Instanzen nach dem Benutzen in einem Zwischenspeicher um sie wiederzuverwenden. Hier wurden angepasste Code-Teile aus dem Buch *Handbuch der Javaprogrammierung* (Krü05) verwendet.

Es ist möglich, dass mehrere Teilnehmer gleichzeitig auf die Datenbank zugreifen. Dabei wird nicht für jeden Benutzer eine eigene Datenbankverbindung oder ein eigenes Statement-Objekt erzeugt. Die Benutzer benutzen die gleiche Datenbankverbindung. Bei jedem Gebrauch eines Statements wird im Cache geschaut, ob eine Instanz existiert. Ist kein Statement-Objekt im Cache vorhanden, wird ein neues Statement-Objekt erzeugt, ansonsten wird es einfach aus dem Cache „rausgeholt“. Nach dem Benutzen wird die Instanz wieder im Cache abgelegt.

Die bestehende Datenbankverbindung könnte abbrechen. Dies hat aber keine große Auswirkung auf das System, denn auch dies wird abgefangen und automatisch eine neue Verbindung mit einem zugehörigen Cache erzeugt. Beim Einloggen benutzt der Teilnehmer das Standard-Datenbankkonto “knvng„ und das zugehörige Passwort. Die Rechte des Kontos sind darauf beschränkt Daten zu lesen sowie Datenänderungen und -updates durchzuführen.

Der Administrator benutzt in den meisten Fällen das gleiche Konto, obwohl er selbst eins hat. Beim Einloggen prüft die Klasse *DBConnection.java* den Administratornamen und Passwort und versucht eine Verbindung herzustellen. Ist sie erfolglos, wird ein Fehler gemeldet. Bei einer erfolgreichen Verbindung wird die Verbindung sofort wieder geschlossen und eine positive Nachricht wird zurück geschickt. Bei weiteren Datenbankzugriffe verwendet der Administrator dieselbe Verbindung zur Datenbank wie die Teilnehmer. Die Benutzer teilen sich Datenbankverbindung und Statements. Dadurch wird verhindert, dass zuviele Statements produziert werden. Wenn 100 Teilnehmern, die gleichzeitig online sind, jeweils 20 Anfragen pro Minute verschicken, würden bei einzelnem Verbindunsaufbau 2000 Statements kreiert. Dies kann den Server schon sehr belasten, da es ihm viel Speicher und Rechenzeit kostet.

5.3 Client

Der Client ist aufgeteilt in Middleware, die ebenfalls zur *Controller*-Komponente gehört, und der *View*-Komponente. Die Middleware hat die Aufgabe, Daten von der Sicht weiter in Richtung Datenbank zu leiten und umgekehrt. Sie führt die Anweisungen aus, die von der View ausgelöst werden. Die View-Einheit sorgt für die Anzeige der Teilnehmerdaten und Interaktion zwischen Benutzer und Programm.

5.3.1 Middleware

Die Middleware ist die Controller-Komponente des Clients und Gegenstück der Middleware auf der Serverseite. Sie nimmt die Eingaben von der View entgegen und schickt sie mittels RMI zur Datenbank. Um die RMI-Technik zu verwenden, wurden für den

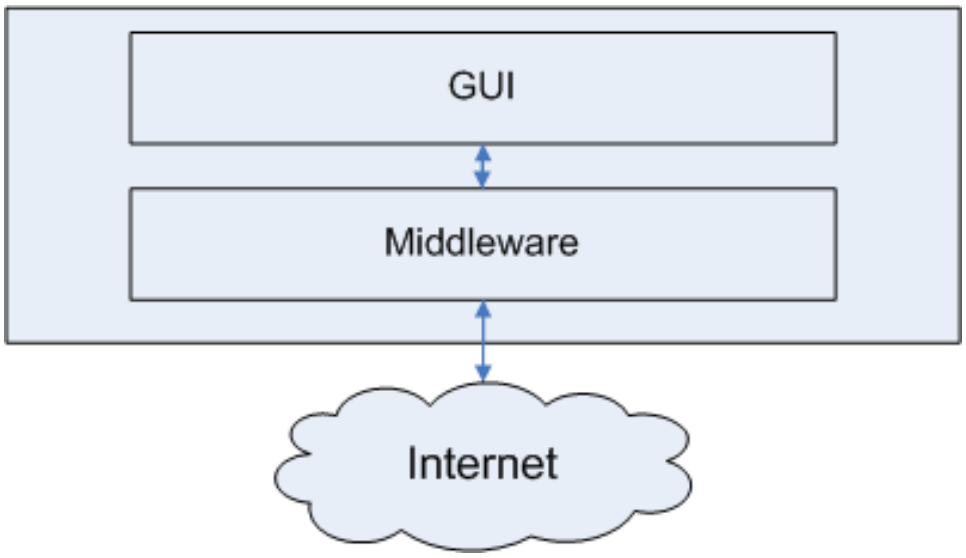

Abbildung 5.3: Die Architektur vom Client

Administrator und Teilnehmer zwei separate Klassen geschrieben: Das sind die Klassen *AdminClient.java* und *UserClient.java*. Jede Klasse sucht in dem vom Server veröffentlichten Register nach dem entfernten Objekt *Admin* bzw. *User*, um entfernte Methoden aufrufen zu können. Ist die Suche erfolgreich, kann die Klasse das entfernte Objekt mit den verfügbaren Methoden verwenden (Abschnitt 3.4.2).

Bevor die Eingabedaten per RMI verschickt werden, findet in der Middleware eine Eingabe-Kontrolle statt. Falls keine Eingaben gemacht wurden oder Fehleingaben, wie Datums- oder Zeichenfehler, existieren, wird an dieser Stelle abgebrochen und eine Nachricht mit Fehlermeldung wird an die View zurück gegeben. Die Standard-Fehler sind in der Klasse *Announcement.java* aufgelistet. Der Administrator und die Teilnehmer benutzen beide die Klasse *MiddleLayer.java*, wobei die public-Methoden für beide Benutzer unabhängig sind. Lediglich die privaten Methoden, wie das Sortieren der Tabelle oder die Datumsüberprüfung, werden von beiden gemeinsam benutzt.

5.3.2 View

Die View-Einheit ist mit der graphischen Benutzerschnittstelle (Graphical User Interface, GUI) realisiert, welche hauptsächlich aus *Swing*-Komponenten besteht. Für den

Administrator und die Teilnehmer existieren zwei unterschiedliche GUIs. Jedes Fenster ist in einer eigenen Klasse beschrieben. Im Hauptfenster werden die wichtigsten Daten angezeigt. Der Teilnehmer kann in seinem Hauptfenster Änderungen vornehmen oder Kontaktdaten hinzufügen. Das Hinzufügen und die Änderung passieren in einem neuem Fenster, wo der Teilnehmer in den vorgegebenen Textfeldern seine Daten eingeben bzw. korrigieren kann. Für den Dienst kann er mit dem Checkbox auswählen, welche Geräte zu welchen Dienst gehören. Das Speichern geschieht mit einem Knopfdruck, wodurch das Action-Event ausgelöst wird und die Daten durch die Middleware weiter zur Datenbank gehen. Fehlermeldungen werden im Fenster angezeigt.

Der Administrator hat ebenfalls ein Hauptfenster, in dem alle Teilnehmerdaten, Dienste und Geräte in einer Tabelle aufgelistet sind. Die Tabelle wird intern sortiert, das heißt die Tabelle wird von der Middleware auf der Clientseite alphabetisch sortiert. Auch der Administrator ist berechtigt Teilnehmerdaten zu ändern. Es ist aber wenig sinnvoll Adressen und Kontaktdaten für den Teilnehmer neu hinzuzufügen, da der Administrator in der Regel diese Daten nicht bekommt und somit auch nicht kennt. Deshalb steht ihm diese Funktion nicht zur Verfügung. Das Suchen der Teilnehmerdaten kann in der Such-Registerkarte durchgeführt werden. Mit dem Abschicken einer Suchtext-Eingabe in den Namensfeldern oder Auswahl vom Popup-Menü und durch anschließendem Knopfdruck, werden die Kriterien an die Datenbank weitergeleitet und diese sucht für den Administrator die Teilnehmer mit den gewünschten Eigenschaften heraus, die in der Tabelle aufgelistet werden. Auch hier werden allgemeine Fehlermeldungen in einem entsprechenden Fenster angezeigt.

Kapitel 6

Resümee

Das Teilnehmermanagementsystem (TMS) verwaltet in der zentralen Datenbank Teilnehmerdaten, so dass mobile Dienste oder Teilnehmer selbst auf diese Daten zurückgreifen können. Das TMS wurde nach dem Client-Server-Prinzip realisiert. Der Client produziert die Daten, während der Server diese Daten in der modellierten PostgreSQL-Datenbank abspeichert. Als Daten werden hauptsächlich persönliche Daten, Adressen, Kontaktdaten und Geräteeigenschaften erhoben. Der Benutzer des TMS kann mit der graphischen Benutzeroberfläche, die mit *Swing*-Komponenten realisiert wurde, Teilnehmerdaten manipulieren oder anzeigen lassen. Das Relationsschema wurde normalisiert, dadurch werden Redundanzen vermieden und die Wartung vereinfacht. Für die Client-Server-Kommunikation wird RMI verwendet. Damit bleibt die Software auf der Clientseite schlank und die Arbeit mit der Netzwerkverbindung unkompliziert.

Die Architektur des Systems folgt dem Model-View-Controller-Paradigma. Die Aufgaben sind in drei große Einheiten aufgeteilt: die *View*, also die graphische Benutzeroberfläche, in der Daten visualisiert werden und der Programmverlauf mittels Kontrollmechanismen wie Buttons gesteuert wird, der *Controller*, der die Logik des TMS darstellt, und das *Datenmodell*, das bestimmt, wie Daten in der Datenbank gespeichert sind.

Das System ist vollständig in Java geschrieben, um Einheitlichkeit und Plattformunabhängigkeit zu gewährleisten. Mit der Idee des MVC sind die Einheiten flexibel gehalten und einzelne Einheiten können erweitert bearbeitet werden. Somit ist die Integration zusätzlicher Schnittstellen für die Dienste leicht möglich.

Anhang A

Datenbanktabellen

```
CREATE TABLE mandatoryuserdetails (
        userid bigserial NOT NULL PRIMARY KEY,
        firstname character varying,
        lastname character varying,
        birthdate date,
        gender character(1)
);


CREATE TABLE account (
        id bigint REFERENCES mandatoryuserdetails (userid),
        accountname character varying,
        password character varying
);


CREATE TABLE optionaluserdetails (
        userid bigint REFERENCES mandatoryuserdetails (userid),
        title character varying
);
```

```
CREATE TABLE address (
        id bigserial NOT NULL PRIMARY KEY,
        adrtype character varying,
        street character varying,
        zip character varying,
        city character varying,
        country character varying,
        userid bigint REFERENCES mandatoryuserdetails (userid),
        state character varying
);



CREATE TABLE electcontact (
        userid bigint REFERENCES mandatoryuserdetails (userid),
        adrid bigint REFERENCES address (id),
        type integer,
        content character varying,
        eid serial NOT NULL PRIMARY KEY
);



CREATE TABLE device (
        deviceid bigserial NOT NULL PRIMARY KEY,
        type character varying,
        displaysize character varying,
        cpu character varying,
        ram character varying,
        bandwidth character varying,
        userid bigint REFERENCES mandatoryuserdetails (userid)
);



CREATE TABLE profiles (
        profileid bigserial NOT NULL PRIMARY KEY,
```

```
        userid bigint REFERENCES mandatoryuserdetails (userid),
        serviceid bigint REFERENCES service (serviceid),
        loginname character varying,
        password character varying
);



CREATE TABLE profiledevice_id (
        deviceid bigint REFERENCES device (deviceid),
        profileid bigint REFERENCES profiles (profileid)
);



CREATE TABLE service (
        serviceid bigserial NOT NULL PRIMARY KEY,
        servicename character varying
);


CREATE TABLE additionaluserdata (
        userid bigint NOT NULL REFERENCES mandatoryuserdetails (userid),
        registered timestamp with time zone,
        status character varying
);


CREATE TABLE onlinetime (
        userid bigint NOT NULL REFERENCES mandatoryuserdetails (userid),
        logindate timestamp with time zone,
        logoutdate timestamp with time zone,
        loginplace character varying
);
```

```
CREATE TABLE transaction (
        userid bigint NOT NULL REFERENCES mandatoryuserdetails (userid),
        time timestamp with time zone,
        action character varying,
        oldvalue character varying,
        tablecolumn character varying,
        newvalue character varying
);
```

Literaturverzeichnis

AH00 ANDREAS HEUER, Gunter S.: *Datenbanken: Konzepte und Sprachen, 2. Auflage*. Bonn : mitp-Verlag, 2000

Cha CHAN, Patrick: *The Java Developers Almanac 1.4, Volume 1*. – Examples and Quick Reference

CRoa Get disconnected with CachedRowSet. `http://java.sun.com/developer/technicalArticles/javaserverpages/cachedrowset/index.html`

CRob *Verbindunsabbruch, RowSets als Erweiterung von JDBC ResultSets*. Javamagazin `http://www.javamagazin.de/itr/online_artikel/psecom,id,561,nodeid,11.html`

Deh DEHNHARDT, Wolfgang: *Java und Datenbanken*. Hanser-Verlag. – Anwendungsprogrammierung mit JDBC, Servlets und JSP

Die DIETRICH, Ernst-Wolfgang: *Java 2*. Oldenbourg-Verlag. – Von den Grundlagen bis zu Threads und Netzen

EA ERIC ARMSTRONG, Stephanie Bodoff Debbie Bode Carson Ian Evans Dale Green Kin Haase Eric J.: *The J2EE 1.4 Tutorial*. – For Sun Java System Applicaion Server Platform Edition 8.2

Eid EIDEMILLER, Sean

GE98 GERALD EHMAYER, Siegfried R.: *Java in der Anwendungsentwicklung*. Heidelberg : dpunkt-Verlag, 1998

Hen HENDRICH, Norman: *Java*. Springer-Verlag. – für Fortgeschrittene

Krü05 KRÜGER, Guido: *Handbuch der Javaprogrammierung, 3. Auflage*. Addison-Wesley, 2005

MS MARTIN SCHADER, Lars Schmidt-Thieme: *Java*. Springer-Verlag. – Eine Einführung

SuJ *Java Message Service (JMS)*. Sun Developer Network (SDN) `http://www.sun.com/software/products/message_queue/index.xml`

SuM *Java BluePrints, Model-View-Controller*. Sun Developer Network (SDN) `http://java.sun.com/blueprints/patterns/MVC-detailed.html`

SuR *Java Remote Method Invocation (Java RMI)*. Sun Developer Network (SDN)

Ull06 ULLENBOOM, Christian: *Java ist auch eine Insel, 5. Auflage*. Galileo Computing. – Das umfassende Handbuch. `http://www.galileocomputing.de/openbook/javainsel5/`

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 31. März 2006 Kin Voong