Fachhochschule Köln
Abteilung Gummersbach
Fachbereich Informatik
Studiengang Wirtschaftsinformatik

Diplomarbeit

Konzepte einer auf verteilter Software basierenden Koordination von Pflegedienstleistungen und deren Implementierung in Java

Von

Martin Braches
Matrikelnummer 11014673

Prüfer:

Frau Prof. Dr. Faeskorn-Woyke

Herr Dipl. Math. Peters

Wermelskirchen, November 2000

1 EINFÜHRUNG IN DIE THEMATIK 6

1.1 Pflegeversicherung 7

1.1.1 Pflegestufen 8

1.2 Servicezentren 9

1.3 Konkurrenz und Kooperation der Dienste 10

1.4 Ziel der Arbeit 11

1.5 SWIFT 11

2 BESCHREIBUNG VON BEISPIELDIENSTEN 14

2.1 ambulante Pflegedienste 14

2.1.2 Leistungen der ambulanten Pflegedienste 15

2.1.3 Relevante Daten für den Servicekatalog 15

2.2 Hausnotrufdienste 16

2.2.4 Leistungen von Hausnotrufdiensten 17

2.2.5 Relevante Daten für den Servicekatalog 18

2.3 Mahlzeitendienste 18

2.3.1 Leistungen der Mahlzeitendienste 19

2.3.2 Relevante Daten für den Servicekatalog 19

3 METHODEN 21

3.1 UML 21

3.1.3 Entstehung der UML 21

3.1.4 Diagramme der UML 22

3.1.5 Anwendungsfalldiagramme 22

3.1.6 Klassendiagramme 23

3.1.7 Aktivitätsdiagramme 26

3.2 Java 27

3.2.1 RMI (Remote Method Invocation) 28

3.2.2 Grundaufbau von RMI 28

3.2.3 Architektur von RMI 29

3.2.4 Komponenten eines RMI Systems 30

3.3 Datensicherheit 31

3.3.1 Verschlüsselung mit Java 32

3.4 JDeveloper 34

3.4.1 Gründe für die Auswahl 34

3.4.2 Benutzeroberfläche des JDevelopers 35

4 IST-SOLL ANALYSE DER ABLÄUFE EINES SERVICEZENTRUMS 37

4.1 Beschreibung des Ist- Zustandes 37

4.1.3 Koordinierung mit ambulanter Pflege 38

4.1.4 Zusammenfassung 40

4.2 Beschreibung des Soll-Zustandes ... 41
4.2.5 Koordinierung mit ambulanter Pflege ... 41
4.3 Kundenanfrage ... 44
4.3.1 Daten in der Kundenanfrage ... 44
4.4 Persönliches Leistungsprofil ... 44
4.4.1 Daten im persönlichen Leistungsprofil ... 45
4.5 Leistungsanfrage ... 45
4.5.1 Daten in der Leistungsanfrage ... 45
4.6 Leistungsangebot ... 46
4.6.1 Daten im Leistungsangebot ... 46
4.7 Anforderungen an den Softwareprototyp ... 46

5 DER SERVICEKATALOG ... 49

5.1 Daten im Servicekatalog ... 50
5.2 Darstellung des Servicekatalogs ... 52
5.2.2 Beschreibung der Baumstruktur ... 52
5.3 Umsetzung des Servicekatalogs in Java ... 53

6 BESCHREIBUNG DES PROGRAMMS ... 55

6.1 Beschreibung des Koordinationsservers ... 55
6.1.3 Servicekatalog ... 56
6.1.4 Kundenanfrage ... 58
6.1.5 Hausnotrufdienste ... 64
6.1.6 Mahlzeitendienste ... 65
6.1.7 ambulante Pflegedienste ... 65
6.2 Beschreibung des Koordinationsclients ... 67
6.2.1 Kundendaten ... 69
6.2.2 Leistungsanfrage ... 70
6.3 Datenmodell ... 72
6.4 Kommunikation zwischen Client und Server ... 74
6.5 Sicherheit im Prototyp ... 76
6.5.1 Verschlüsseln von Java-Objekten ... 76
6.5.2 Signieren von Java-Objekten ... 77
6.6 Fazit der Programmierung ... 78

7 MÖGLICHKEITEN DES SERVICEKATALOGS ... 79

7.1 Darstellung im Internet ... 79
7.2 Datenaustausch der Dienste untereinander ... 79
7.3 Reporting ... 80
7.4 Dokumentenfluß im Pflegeprozeß ... 80
7.5 XML ... 80

8 VALIDIERUNG DES PROTOTYPS 82

8.1 GUI 82

8.2 Funktionen 82

8.3 Abläufe 83

8.4 Fazit 83

9 ZUSAMMENFASSUNG UND AUSBLICK 84

ANHANG A 85

Quellcode des Prototyps: 85

Die Klasse CoordinationModule 85

Die Klasse MainFrame 85

Die Klasse CoordinationPanel 87

Die Klasse Message 93

Die Klasse Server 94

Die Klasse ServerInterface 96

Die Klasse ServerContainerThread 97

Die Klasse ServiceTree 97

Die Klasse ServiceTreeRenderer 99

Die Klasse ServiceItemNode 99

Die Klasse ServiceNode 100

Die Klasse ClientRequestPanel 101

Die Klasse SearchPersonDialog 104

Die Klassse SearchDBConnect 106

Die Klasse SearchTableModel 107

Die Klasse CareClientRequestedServicesDialog 108

Die Klasse CareClientPhysicianDialog 112

Die Klasse SearchPhysicianDialog 113

Die Klasse CareClientPaymentInstitutionDialog 115

Die Klasse SearchPaymentInstitutionDialog 117

Die Klasse CoordinationPanel3 119

Die Klasse AnswerListDialog 122

Die Klasse SecureCareClient 123

Die Klasse ServiceProfile 124

Die Klasse ServiceProfileTableModel 125

Die Klasse ServiceProfileCellRenderer 126

Die Klasse ServiceProfileCellEditor 127

Die Klasse CoodinationClient 127

Die Klasse ClientFrame 128

Die Klasse Client 130

Die Klasse ClientInterface 132

Die Klasse ClientContainerThread 132

Die Klasse ClientServicePanel 132
Die Klasse ServiceDataPanel..... 134
Die Klasse ServiceDataResoure..... 135
Die Klasse CareClientPanel..... 136
Die Klasse ClientRequestedServicesDialog..... 137
Die Klasse ClientPhysicianDialog 140
Die Klasse ClientPaymentInstitutionDialog 141
Die Klasse CareClientServiceProfilePanel 142

ANHANG B 144

Übersicht über die Tabellen der Datenbank: 144

ANHANG C 145

Literaturverzeichnis:..... 145

1 Einführung in die Thematik

Heute gibt es in der Bundesrepublik Deutschland über 1,8 Millionen Menschen, die ständig auf Pflege angewiesen sind. Davon leben ca. 535.000 Pflegebedürftige in Heimen. Die übrigen ca. 1,27 Millionen werden zu Hause versorgt [Bmg00]. Diese Versorgung wird zum Teil von Familienangehörigen, Nachbarn oder ehrenamtlichen Helfern geleistet. Dort wo diese Hilfen nicht oder nur teilweise zur Verfügung stehen, kommen Pflegedienste zum Einsatz. Die Pflegeleistungen, welche von Diensten angeboten werden, können in drei Hauptgruppen unterteilt werden:

- ambulante Pflege:

  Die ambulante Pflege, die oft auch als häusliche Pflege bezeichnet wird, hat die Aufgabe den zu pflegenden Personen bei Tätigkeiten des täglichen Lebens zu helfen, um ein Verbleiben in der gewohnten Umgebung zu ermöglichen. Zu diesen Tätigkeiten gehören z.B. Waschen, Duschen, Hilfen bei der Ernährung oder Reinigung der Wohnung des Pflegebedürftigen. Diese Tätigkeiten werden normalerweise in die vier Gruppen Körperpflege, Mobilität, Ernährung und hauswirtschaftliche Versorgung eingeteilt.

- stationäre Pflege:

  Die stationäre Pflege kann in vollstationäre Pflege, teilstationäre Pflege und Kurzzeitpflege unterteilt werden. Unter vollstationärer Pflege versteht man die Unterbringung in einem Pflegeheim. Dies hat den Vorteil, daß qualifizierte Fachkräfte ständig und sofort zur Verfügung stehen, und daß dadurch ein Höchstmaß an pflegerischer Betreuung gewährleistet wird. Die teilstationäre Pflege kann in Tagespflege und Nachtpflege eingeteilt werden. Diese Formen der Pflege finden immer dann Anwendung, wenn die häusliche Pflege nicht mehr ausreicht, oder wenn die Pflegeperson nicht mehr in der Lage ist, die Pflege allein zu gewährleisten. Sie sind der vollstationären Pflege vorzuziehen, um ein teilweises Verbleiben in der eigenen Wohnung zu ermöglichen. Die Kurzzeitpflege ist vollstationäre Pflege mit einer zeitlichen Begrenzung. Sie wird in der Übergangszeit nach einem Krankenhausaufenthalt oder nach einer Krisensituation angewendet, wenn die Leistungen der teilstationären oder der ambulanten Pflege aufgrund eines erhöhten Leistungsbedarfs nicht ausreichen.

- sonstige Leistungen:

  Sonstige Leistungen sind ergänzende Leistungen für pflegebedürftige Menschen, welche über die Angebote der beiden anderen Bereiche hinausgehen oder diese ergänzen. Dazu zählen z.B. die Leistungen der Hausnotrufdienste, Mahlzeitendienste oder mobilen sozialen Dienste. Diese Leistungen müssen in der Regel vom Pflegebedürftigen selbst bezahlt werden.

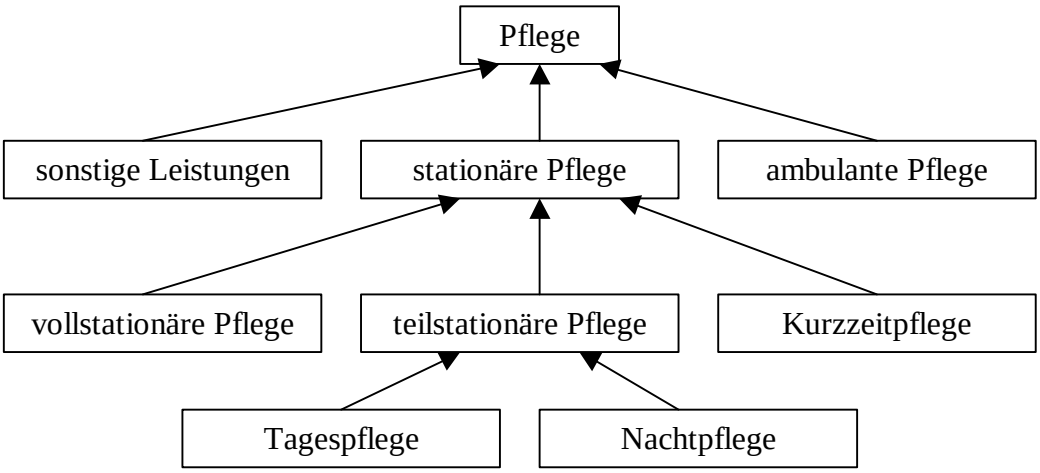


Abbildung 1-1 Übersicht über die Leistungsarten

Abbildung 1-1 gibt eine Übersicht über die Einteilung der Leistungsarten von professionellen Pflegediensten. Der Einsatz dieser Dienste kann zu erheblichen Kosten für die zu pflegende Person sowie deren Angehörige führen. Um dieses Kostenrisiko zu mildern wurde die Pflegeversicherung eingeführt.

1.1 Pflegeversicherung

Das Gesetz zur Einführung der Pflegeversicherung wurde im Mai 1994 verabschiedet und trat im Januar 1995 in Kraft [Klie99, 12]. Das Gesetz beinhaltet die Regelung der Leistungen für die häusliche Pflege, die teilstationäre Pflege, die Kurzzeitpflege und die vollstationäre Pflege. Die Pflegeversicherung ist durch das 11. Sozialgesetzbuch (SGB XI) geregelt. Sie ist als begrenzte Sozialleistung bei Pflegebedürftigkeit konzipiert und geht von Eigenbeiträgen der Pflegebedürftigen sowie von ergänzenden Sozialhilfeleistungen aus. Dabei soll eine Mischung von Hilfen bei Pflegebedürftigkeit entstehen, die einerseits aus familiären Hilfen, andererseits aus staatlich garantierten Basisleistungen und schließlich aus dem Zukauf weiterer Dienstleistungen besteht. Das

Ziel der Pflegeversicherung ist nach §2 SGB XI die Förderung der Selbstbestimmung und Selbständigkeit der Pflegebedürftigen mit der Zielsetzung, die körperlichen und geistigen Kräfte wiederzugewinnen oder zu erhalten. Ein zentraler Bestandteil der Selbstbestimmung ist das Wahlrecht der Leistungen. Dies bezieht sich einerseits auf die Art der Leistungen, was bedeutet, daß zwischen ambulanter Pflege, Geld- oder Sachleistungen, teilstationärer oder vollstationärer Pflege gewählt werden kann. Andererseits besteht die Möglichkeit, den in Anspruch genommenen Pflegedienst oder die gewählte Pflegeeinrichtung frei zu wählen. Außerdem ist nach §3 SGB XI vorrangig die häusliche Pflege und die Pflegebereitschaft der Angehörigen oder Nachbarn zu fördern sowie die teilstationäre Pflege und Kurzzeitpflege der vollstationären Pflege vorzuziehen.
Da die Pflegebedürftigkeit nicht immer gleich hoch ist, werden Pflegebedürftige in drei Pflegestufen eingeteilt.

1.1.1 Pflegestufen

Die Einteilung der Pflegebedürftigen in die Pflegestufen wird durch den Medizinischen Dienst der Krankenversicherung (MDK) vorgenommen. Dieser beurteilt die Pflegebedürftigkeit der zu pflegenden Person nach einer eigenen Untersuchung, sowie nach Rücksprachen mit dem Hausarzt oder anderen behandelnden Ärzten. Außerdem werden noch Auskünfte von evtl. bereits pflegenden Personen eingeholt.

- Pflegestufe I : Erheblich-Pflegebedürftig
  Hier wird ein Pflegebedarf von mindestens einmal täglich 90 Minuten vorausgesetzt. Davon müssen mindestens 45 Minuten pflegerische Hilfe sein. Darüber hinaus ist festgelegt, daß die Hilfen aus wenigstens zwei der vier Bereiche Körperpflege, Mobilität, Ernährung und hauswirtschaftliche Versorgung bestehen müssen.
- Pflegestufe II: Schwer-Pflegebedürftig
  Bei Pflegestufe II wird ein Hilfebedarf von drei Stunden täglich vorausgesetzt. Dieser sollte sich auf drei unterschiedliche Tageszeiten verteilen und mindestens eine Stunde pflegerische Versorgung beinhalten.
- Pflegestufe III: Schwerst-Pflegebedürftig
  Bei der höchsten Pflegestufe die es in der Pflegeversicherung gibt, ist ein Bedarf an

Hilfen von insgesamt fünf Stunden erforderlich. Diese müssen bei Tag und Nacht benötigt werden und es müssen vier der fünf Stunden pflegerische Versorgung sein.

Die Höhe der Sachleistung der Pflegeversicherung richtet sich in erster Linie nach der Pflegestufe. Sie liegt bei:

- 750 DM pro Monat bei Pflegestufe I
- 1800 DM pro Monat bei Pflegestufe II
- 2800 DM pro Monat bei Pflegestufe III

Darüber hinaus gibt es noch eine Ausnahmeregelung:
In besonderen Härtefällen wird eine Leistung von bis zu 3750 DM gewährt. Dies wird auch als Pflegestufe 3a bezeichnet.

1.2 Servicezentren

Die Verantwortlichkeit für das Bereithalten einer leistungsfähigen und wirtschaftlichen Versorgungsstruktur, welche die im SGB XI beschriebenen Anforderungen erfüllt wird durch §9 SGB XI vom Bund an die Länder delegiert.
Um die Forderungen aus §9 SGB XI zu erfüllen, haben die Bundesländer eigene Pflegegesetze erlassen. So entstand auch das Landespflegegesetz des Landes Nordrhein-Westfalen, welches im März 1996 verabschiedet wurde. Darin ist unter anderem die Verpflichtung zur Kooperation von Pflegediensten und Einrichtungen festgeschrieben. Außerdem wird in §4 PfG NW die trägerunabhängige Beratung der Pflegebedürftigen und deren Angehörigen gefordert. Um diesen Forderungen zu genügen wurden sogenannte Servicezentren geschaffen.
Diese Servicezentren stehen normalerweise unter der Leitung von Wohlfahrtsverbänden wie z.B. der Johanniter oder der Caritas, können aber auch von privaten Trägern oder von Städten geleitet werden. Sie haben die Aufgabe die trägerübergreifende Beratung und Koordination von Dienstleistungen für einzelne Stadtteile oder Bezirke zu gewährleisten. Da die Träger dieser Servicezentren meistens auch ambulante oder andere Pflegedienste unterhalten, werden diese in der Praxis bei der Vergabe von Aufträgen bevorzugt. Um hier eine unabhängige Beratung und gleichberechtigte Verteilung zu erreichen, muß eine verstärkte Kooperation zwischen den Servicezentren und den verschiedenen Diensten geschaffen werden. Ein Ansatz zur Erreichung dieses

Ziels ist die Schaffung einer vollständigen und einheitlichen Datenbasis, des Servicekatalogs. Die Wichtigkeit einer solchen Datenbasis ergibt sich aus der Überlegung heraus, daß man nur über alle Dienste in einem Stadtbezirk beraten kann, die man kennt. Daher ist es notwendig, daß alle Dienste, über die ein Servicezentrum beraten soll auch ein vollständiges Angebot mit ihren Leistungen zur Verfügung stellen. Wünschenswert wären sicherlich auch Informationen über die Verfügbarkeit dieser Leistungen, um diese gegebenenfalls direkt buchen zu können. Dies ist jedoch nur bei einzelnen Diensttypen möglich, wie z.B. bei Heimplätzen der vollstationären Pflege, wo man einfach die Anzahl der noch freien Betten angeben kann. Bei anderen Diensttypen, wie z.B. bei der ambulanten Pflege, ist eine Angabe der Verfügbarkeit schwieriger, da diese oft von einzelnen Pflegekräften abhängt und sich sehr kurzfristig ändert.
Ein weiteres Problem, das bei der Koordination verschiedener, unabhängiger Dienste in einem Servicezentrum auftritt, ist das Spannungsfeld zwischen Konkurrenz und Kooperation.

1.3 Konkurrenz und Kooperation der Dienste

Einerseits sind die Anbieter von Pflegedienstleistungen vom Gesetz her verpflichtet miteinander zu kooperieren, um für die Pflegebedürftigen die möglichst beste Versorgung zu gewährleisten. Andererseits sind dieselben Anbieter Konkurrenten auf dem freien Markt der Pflegedienstleistungen und müssen um ihr Überleben kämpfen. Dadurch entsteht ein Spannungsfeld, welches durch Abbildung 1-2 verdeutlicht werden soll.

Abbildung 1-2 Spannungsfeld zwischen Kooperation und Konkurrenz

Mitten in diesem Spannungsfeld steht das Servicezentrum und erwartet weitreichende Angaben über das Angebot und die Leistungen der Dienste. Die Erstellung eines

gemeinsamen Servicekatalogs läuft somit dem Bestreben der einzelnen Dienste möglichst wenig Informationen an Konkurrenten weiterzugeben entgegen.
Das so entstandene Spannungsfeld kann nur durch langwierige vertrauensbildende Maßnahmen abgebaut werden. Diese sollten eine offene Informationspolitik und die absolute Gleichbehandlung aller Kooperationspartner beinhalten. Nur wenn dieses Spannungsfeld abgebaut wird, also alle Dienstanbieter in einem Bereich den Willen haben zusammen zu arbeiten können die Ziele der Servicezentren erreicht werden.
Einen Ansatz zur Lösung dieses Problems soll die in dieser Arbeit entstehende Software bieten.

1.4 Ziel der Arbeit

Das Ziel dieser Arbeit ist die Konzeption einer verteilten Software für die Beratung und Koordination von Pflegedienstleistungen in einem Servicezentrum.
Dies beinhaltet die Erstellung eines Prototyps einer verteilten Software in Java. Ein Bestandteil dieses Prototyps soll der Servicekatalog sein, der die Grundinformationen über die zum Koordinierungsbereich gehörenden Einrichtungen und Dienste enthält. Desweiteren soll das Programm den Arbeitsablauf einer Koordination in einem Servicezentrum unterstützen. Hierbei wird sich die Arbeit auf die Untersuchung der Koordinierungsabläufe für die Diensttypen ambulante Pflege-, Hausnotruf- und Mahlzeitendienste beschränken.
Da diese Arbeit im Rahmen des SWIFT Projekts entsteht, ergeben sich daraus einige grundsätzliche Anforderungen und Vorgaben.

a) Es soll nur ein Prototyp einer Software entstehen.
b) Die Programmierung soll in Java geschehen, da die bisherigen Prototypen ebenfalls in Java geschrieben wurden und um eine Plattformunabhängigkeit zu gewährleisten.
c) Es sollen bereits entwickelte Komponenten, Klassen und Modelle wiederverwendet werden.
d) Es sollen offene Schnittstellen zu anderen Modulen geschaffen werden.

1.5 SWIFT

SWIFT ist ein von der Europäischen Union gefördertes Projekt, welches unter dem Titel „Coordinated Service Delivery for the Elderly“ steht. Das Ziel dieses Projektes ist die

Entwicklung und Bereitstellung einer Telematik[1] –Architektur zur Vereinheitlichung von Pflegedienstleistungen für ältere Menschen. Diese Architektur soll auf Workflow[2] Konzepten und Technologien basieren.
Es hat nicht das Ziel am Ende der Projektlaufzeit eine voll funktionsfähige und marktreife Software zu haben, sondern die Erstellung und Erprobung eines Architekturkonzeptes für eine Software der Zukunft. Dabei wurde schon frühzeitig eine objektorientierte Vorgehensweise festgelegt.
Das Projekt wird von einem multinationalen Konsortium durchgeführt. Dies besteht aus englischen, italienischen und deutschen Kooperationspartnern. In den verschiedenen Ländern gehören Forschungseinrichtungen, Softwarehäuser sowie Wohlfahrtsverbände und öffentliche Einrichtungen zu den Projektgruppen. In Deutschland werden die Aktivitäten des SWIFT Projekts von der DLR[3] in Köln-Porz geleitet und koordiniert.
Die Arbeiten an SWIFT wurden im Dezember 1998 begonnen und sollen voraussichtlich Ende 2001 abgeschlossen sein. Gegenwärtig befindet sich das Projekt in der Phase der Definition der Anforderungen. Hierbei werden in Gesprächen mit möglichen späteren Nutzern die Anforderungen und Anwendungsfälle einer zukünftigen Software ermittelt und deren Realisierbarkeit überprüft. Partner in diesen Gesprächen sind normalerweise Vertreter von Wohlfahrtsverbänden, welche auch Mitglieder im SWIFT Konsortium sind. Die so gewonnenen Erkenntnisse werden durch explorative Prototypen überprüft.
Explorative Prototypen sind lauffähige und benutzbare Dialoge oder Dialogsequenzen, welche die zuvor definierten Anwendungsfälle illustrieren [Oest97, 104].
Die so entstehenden Softwareprototypen sollen sich zu einem modularen Gesamtprototyp zusammenfassen lassen. Das Hauptinteresse der deutschen Projektgruppe liegt dabei auf dem Bereich der ambulanten Pflegedienste.
Durch das Aufteilen des Gesamtprojektes in kleinere Teilaspekte sind in der bisherigen Projektlaufzeit schon folgende Prototypen entstanden:

[1] Telematik ist ein Kunstwort, welches die Verknüpfung von Hard- und Software in der Telekommunikation und Informatik umfaßt.
[2] Workflow ist eine computergestützte Geschäftprozeßabwicklung, bei der die Reihenfolge vorgegeben ist.
[3] Deutsches Zentrum für Luft- und Raumfahrt e.V.

- Ein Personen- Modul, welches für die Verwaltung von Kunden- sowie Mitarbeiterstammdaten zuständig ist.
- Eine graphische Benutzeroberfläche, welche die Möglichkeiten einer Darstellung der Einzelkomponenten in einer Gesamtoberfläche verdeutlicht.
- Ein Tourenplanungs-Modul, welches die einfache Planung von Kundenwünschen und Mitarbeiterverfügbarkeit zu Touren für ambulante Pflegedienste mit Hilfe von Drag & Drop Techniken ermöglicht.
- Ein Leistungs- und Tätigkeitskatalog- Modul, mit dem es möglich ist, die Leistungs- und Tätigkeitskataloge, welche die Abrechnungs- und Arbeitsgrundlage für ambulante Pflegedienste sind, frei zu erstellen und zu verändern.
- Ein Ausführungs- Modul, welches die Funktionen der anderen Module mit dem Zweck eine Darstellung auf einem Hand-held [4] zu ermöglichen, verknüpft. Dieser Hand-held soll von ambulanten Pflegediensten bei der Durchführung der Pflege mitgeführt werden, um Informationen über den Kunden, die Touren und die auszuführenden Tätigkeiten zu geben, und um Zeiten sowie Bemerkungen direkt zu erfassen.

Die im Laufe dieser Arbeit entstehende Software soll sich in den Rahmen der bereits existierenden Prototypen einfügen und soweit möglich auf bestehende Datenstrukturen und Klassenmodelle zurückgreifen.

[4] Ein Hand-held, ist ein Computer im Taschenformat

2 Beschreibung von Beispieldiensten

Um die Aufgabe der Pflegedienstkoordination zu bewältigen oder mit einer Software zu unterstützen muß ein detailliertes Wissen über diese Dienste vorhanden sein. Dieses Hintergrundwissen soll das nachfolgende Kapitel schaffen. Dazu werden im Folgenden drei Typen von Pflegediensten beispielhaft beschrieben. Dies sind:

1. ambulante Pflegedienste
2. Hausnotrufdienste
3. Mahlzeitendienste

Die in dieser Beschreibung enthaltenen Informationen und Beispiele stammen größtenteils aus persönlichen Gesprächen mit Mitarbeitern dieser Dienste. Sie bilden die Grundlage für die Erstellung des Servicekatalogs.

2.1 ambulante Pflegedienste

Ambulante Pflegedienste sind selbständig wirkende Einrichtungen, die Pflegebedürftige in ihrer Wohnung pflegen und hauswirtschaftlich versorgen. Sei bilden eine eigene Gruppe in der Einteilung der Leistungen. Die pflegerische Verantwortung für einen solchen Dienst liegt nach § 71 SGB XI bei einer ausgebildeten Pflegekraft. Dies muß entweder eine ausgebildete/r Krankenschwester/-pfleger, eine/r Kinderkrankenschwester/-pfleger oder eine/r Altenpflegerin/-pfleger sein, und darüber hinaus muß diese Person über zwei Jahre Berufserfahrung verfügen [Klie99, 104].
Die Tätigkeiten in der ambulanten Pflege lassen sich in die vier Bereiche Körperpflege, Ernährung, Mobilität und hauswirtschaftliche Versorgung einteilen.
Zum Bereich der Körperpflege gehören z.B. das Waschen, Duschen, Baden, die Zahnpflege, das Kämmen und Rasieren. In den zweiten Bereich, die Ernährung, fallen z.B. das mundgerechte Zubereiten oder die Aufnahme der Nahrung. Der Bereich der Mobilität umfaßt das selbständige Aufstehen und Zu-Bett-Gehen, An- und Auskleiden, Gehen, Stehen, Treppensteigen oder das Verlassen und Wiederaufsuchen der Wohnung. Der vierte Bereich, die hauswirtschaftliche Versorgung, schließt Tätigkeiten wie Einkaufen, Kochen, Reinigen der Wohnung, Spülen, Wechseln und Waschen der Wäsche und Kleidung oder das Beheizen der Wohnung ein.

2.1.2 Leistungen der ambulanten Pflegedienste

Die Leistungen der ambulanten Pflegedienste werden in zwei Gruppen eingeteilt. Nach diesen Gruppen findet auch die Abrechnung statt. Dies sind zum einen die Leistungen nach SGB XI, welche über die Pflegeversicherung abgerechnet werden. Dazu zählen die oben aufgeführten Tätigkeiten. Das SGB XI definiert hier detaillierte Leistungskomplexe, welche zum Teil aus andern Leistungskomplexen sowie aus einzelnen Tätigkeiten bestehen. Aus diesen Leistungskomplexen setzt sich das persönliche Leistungsprofil eines Kunden zusammen. Zum anderen gibt es die Leistungen nach SGB V, wie z.B. die Krankenhausersatzpflege, die mit der Krankenversicherung abgerechnet wird.
Die gewünschten Leistungen werden im Erstgespräch mit dem Kunden ausgewählt und zu einem persönlichen Leistungsprofil zusammengestellt, nach dem dann die Pflege erfolgt. Ambulante Pflegedienste bieten ihre Leistungen nur für einen festgelegten Bereich an, wie z.B. für einen Stadtteil. Der Grund dafür ist, daß aus Wirtschaftlichkeitsgründen zu lange Fahrstrecken vermieden werden müssen.

2.1.3 Relevante Daten für den Servicekatalog

Für die Beratung von Pflegebedürftigen und deren Angehörigen über die Leistungen von ambulanten Pflegediensten, sowie deren Koordination in Servicezentren wird eine Übersicht über die angebotenen Pflegeleistungen und das Einsatzgebiet des jeweiligen Dienstes benötigt.
Angaben über die Verfügbarkeit der einzelnen Leistungen sind im Servicekatalog nicht sinnvoll, da die Einzelleistungen erst beim Erstbesuch des ambulanten Pflegedienstes mit dem Kunden ausgehandelt werden. Daraus ergibt sich zum einen der Preis, den der Kunde zu bezahlen hat. Zum anderen kann erst jetzt bestimmt werden, wie oft und wie lange ein Mitarbeiter des Pflegedienstes mit der Pflege dieses Kunden beschäftigt ist, was Einfluß auf die Dienst- und Tourenplanung des ambulanten Pflegedienstes hat. Daher ist eine Bestimmung der Verfügbarkeit von ambulanten Pflegedienstleistungen im voraus nicht möglich.
Aus den vorangestellten Überlegungen ergeben sich folgende Daten über ambulante Pflegedienste, die für den Servicekatalog benötigt werden:

- Name

- Straße
- Ort
- PLZ
- Telefonnummer
- Ansprechpartner
- E-Mail Adressen
- Bankverbindungen
- Leistungskataloge mit Preisen und dem Einzugsgebiet des Dienstes
- Sonstige Informationen

2.2 Hausnotrufdienste

Die Hausnotrufdienste gehören zur Kategorie Sonstige Leistungen und werden oft auch als komplementäre Dienste bezeichnet. Sie haben das Ziel älteren oder behinderten Menschen eine sehr einfache Möglichkeit zu geben, in Notfällen Hilfe zu rufen. Zu diesem Zweck wird bei den Kunden eines Hausnotrufdienstes ein Hausnotrufgerät eingerichtet.
Dieses Gerät besteht aus einer Basisstation und einem oder mehreren sogenannten Funkfingern. Darüber hinaus besteht oftmals auch die Möglichkeit Rauchmelder an das Hausnotrufsystem anzuschließen.
Der Hauptbestandteil des Hausnotrufgerätes ist die Basisstation. Dies ist ein ca. 20 x 24 cm^5 großes Gerät, welches über einen Telefonanschluß im Falle eines Notrufes die fest einprogrammierte Nummer der Hausnotrufzentrale anwählt. Darüber hinaus ist dieses Gerät noch mit einem leistungsstarken Lautsprecher und Mikrofon ausgestattet. Dies ermöglicht den Mitarbeitern einer Hausnotrufzentrale die sofortige Kontaktaufnahme mit den Kunden im Falle eines Notrufes, um die Art des Notfalles herauszufinden.
Außerdem ist ein Basisgerät in der Regel noch mit drei Tasten ausgestattet, die folgende Funktionen haben:

- Eine Notruftaste zum Auslösen eines Notrufes an der Basisstation.
- Eine Tagesmeldungstaste, welche spätestens alle 25 Stunden betätigt werden muß, wenn eine Tagesmeldung mit dem Kunden vereinbart worden ist, da sonst ein Notruf ausgelöst wird.

- Eine An- Abmeldetaste, mit der sich der Kunde an der Hausnotrufzentrale abmelden kann. Diese wird benutzt, wenn der Kunde in Urlaub fährt oder in ein Krankenhaus geht, um die Funktion der Tagesmeldungstaste außer Kraft zu setzen.

Der zweite Bestandteil des Hausnotrufgerätes ist der Funkfinger. Dies ist ein schlüsselanhängergroßes Gerät mit einer Taste zum Auslösen des Notrufes. Ein Notrufsignal wird über Funk an die Basisstation weitergeleitet, welche dann die Notrufzentrale anwählt. Das Gerät ist Batterie betrieben und stoß- sowie wasserfest. Es sollte vom Kunden jederzeit mitgeführt werden, um einen Notruf auslösen zu können. Die Reichweite dieses Gerätes beträgt ca. 50 m in einem Haus und ca. 300 m außerhalb geschlossener Räume. Als dritte Geräteart sind noch Rauchmelder verfügbar. Diese werden wie normale Rauchmelder an der Decke eines Raumes montiert und lösen im Falles einer Rauchentwicklung ebenfalls einen Notruf aus.

2.2.4 Leistungen von Hausnotrufdiensten

Die meisten Hausnotrufdienste bieten zwei unterschiedliche Leistungspakete an. Zum einen wird ein Grundpaket mit Basisleistungen angeboten. Dieses umfaßt die Bereitstellung eines Hausnotrufgerätes sowie dessen Installation. Darüber hinaus bieten manche Dienste die Angabe einer Bezugsperson an, die im Falle eines Notrufes verständigt werden kann. Dies kann z.B. ein Nachbar sein, der einen Schlüssel für die Wohnung des Kunden hat. Falls ein Notruf ausgelöst wird und der Kunde nicht über die Lautsprecher- Mikrofonkombination der Basisstation ansprechbar ist, kann die Bezugsperson gebeten werden nachzusehen warum der Notruf ausgelöst wurde oder wenn möglich Hilfe zu leisten. Dies ist erheblich einfacher und billiger als die Alarmierung eines Rettungswagens. Die Kosten für einen Grunddienst liegen bei ca. 35 DM im Monat.

Zum anderen gibt es Hausnotrufdienste, die zusätzlich ein erweitertes Leistungsangebot bieten. Dieses Leistungsangebot besteht aus den Leistungen des Grundpaketes, welche um die Möglichkeit der Hinterlegung eines Wohnungsschlüssels bei der Hausnotrufzentrale und die Angabe von bis zu drei Bezugspersonen erweitert ist. Bei diesen Angeboten wird im Fall eines Notrufes und für den Fall das keine der angegebenen Bezugspersonen erreicht werden kann, ein Fahrer des Hausnotrufdienstes

[5] Die Größe variiert je nach Ausführung und Hersteller.

mit dem Wohnungsschlüssel losgeschickt, um den Grund für den Notruf herauszufinden. Die Kosten für diese Dienste liegen bei ca. 75 DM im Monat.

2.2.5 Relevante Daten für den Servicekatalog

Für die Beratung über Hausnotrufdienste, sowie deren Koordination in Servicezentren werden außer einer kompletten Leistungsbeschreibung auch noch der Aktionsradius jedes einzelnen Dienstes benötigt. Ideal hierfür wäre eine Liste mit allen Straßen und Stadtteilen, die in den Bereich des Dienstes fallen. Angaben über die Verfügbarkeit des Angebotes sind nicht notwendig, da diese so gut wie immer gewährleistet ist.
Deshalb werden im Servicekatalog folgende Daten über Hausnotrufdienste benötigt:

- Name
- Straße
- Ort
- PLZ
- Telefonnummer
- Ansprechpartner
- E-Mail Adressen
- Bankverbindungen
- Leistungskataloge mit Preisen und dem Aktionsradius des Dienstes
- Sonstige Informationen

2.3 Mahlzeitendienste

Mahlzeitendienste werden oft auch als „Essen auf Rädern“ bezeichnet und sind ebenfalls in die Gruppe der Sonstigen Leistungen einzuordnen. Sie haben das Ziel, ihre Kunden täglich mit einer warmen Mahlzeit zu versorgen. Der Ausdruck „warme Mahlzeit“ wird in diesem Bereich als Mittagessen interpretiert. Die Leistungen dieser Dienste sind genau wie die der anderen Dienste auf einen festen Bereich beschränkt. In diesem Gebiet fahren die Fahrer der Mahlzeitendienste täglich im Zeitraum zwischen ca. 10:00 Uhr und 14:00 Uhr die Mahlzeiten aus. Die Auslieferungszeit variiert je nach Dienst und hängt zudem noch von der Fahrtroute des jeweiligen Fahrers ab. Die Bestellung der Mahlzeiten erfolgt normalerweise über Telefon, kann aber auch von einem Fahrer aufgenommen oder per Fax geschickt werden. Mahlzeitendienste

schließen mit ihren Kunden keine Verträge über feste Leistungen wie bei den andern Diensten ab. Das heißt, es gibt keine Mindestabnahme oder ähnliche Vertragsbindungen.

2.3.1 Leistungen der Mahlzeitendienste

Die meisten Mahlzeitendienste haben täglich warme Mahlzeiten und tiefgekühlte Mahlzeiten in ihrem Angebot. Einige bieten darüber hinaus noch Frühstück oder spezielle Abendessen an. Die tiefgekühlten Mahlzeiten sind komplett fertig zubereitet und müssen nur noch vom Kunden aufgewärmt werden. Tiefkühlkost kann in der Regel aus einem Katalog ausgewählt werden und muß zwei Wochen im Voraus bestellt werden. Eine Lieferung besteht hier normalerweise aus einem Karton mit 7 Mahlzeiten. Warme Mahlzeiten hingegen werden täglich frisch zubereitet und sofort ausgeliefert. Daher ist es in diesem Bereich auch leicht möglich kurzfristige Bestellungen zu bearbeiten. Der Kunde eines Mahlzeitendienstes hat bei warmen Mahlzeiten normalerweise die Möglichkeit sein Essen aus einer Wochenkarte auszuwählen.
Die Preise für ein Essen liegen zwischen ca. 7,50 DM und 10 DM. Hierbei gibt es Schwankungen je nach gewähltem Essen und Anbieter.

2.3.2 Relevante Daten für den Servicekatalog

Für die Koordination von Mahlzeitendiensten und eine umfassende Beratung über deren Leistungen wird eine vollständige Beschreibung der Leistungen jedes Dienstes benötigt. Zudem sind Informationen über die Lieferzeiten und das Gebiet, welches beliefert wird wünschenswert. Angaben über die Verfügbarkeit der Leistungen sind normalerweise nicht notwendig, da diese Dienste immer Kapazitäten frei haben. Der Servicekatalog sollte folgende Angaben über Mahlzeitendienste enthalten:

- Name
- Straße
- Ort
- PLZ
- Telefonnummer
- Ansprechpartner
- E-Mail Adressen
- Bankverbindungen

- Leistungskataloge mit Preisen, Lieferzeiten und Einzugsgebiet
- Sonstige Informationen

Die in diesem Kapitel aufgezählten Datenfelder bilden die Grundlage für das Datenmodell des Servicekataloges.

3 Methoden

In diesem Kapitel werden die für die Konzeption des Prototyps verwendeten Methoden, Techniken sowie die für die Programmierung verwendete Entwicklungsumgebung vorgestellt und beschrieben.

3.1 UML

Die Unified Modeling Language (UML) ist eine Sprache und Notation zur Spezifikation, Konstruktion, Visualisierung und Dokumentation von Modellen für Softwaresysteme [Oest97, 143]. UML in der Version 1.1 wurde im November 1997 von OMG [6] verabschiedet und wurde dadurch zu einer Art Standard für die objektorientierte Modellierung.

3.1.3 Entstehung der UML

Die Anfänge der objektorientierten Methoden gehen auf die frühen, objektorientierten Programmiersprachen wie z.B. Smalltalk oder C++ zurück. Für eine strukturierte Programmierung in diesen Sprachen entstanden schnell eine Vielzahl von objektorientierten Methoden. Die wichtigsten sind:

- Shalaer/Mellor
- Coad/Yourdon
- OMT (von J. Rumbaugh et al.)
- Booch
- OOSE (von I. Jacobson)

Da die meisten Methoden ihre eigenen Vor- und Nachteile hatten, wurde versucht durch eine Kombination von verschiedenen Methoden diese Nachteile auszugleichen.
1997 wurde die Version 1.0 von UML von Jim Rumbaugh, Grady Booch und Ivar Jacobson, welche die Erfinder der drei bis dahin populärsten Methoden waren, bei der OMG als Standardisierungsvorschlag eingereicht. Diese hatten ihre eigenen Methoden vereinigt und daraus UML entwickelt. UML ist heute die am weitesten verbreitete objektorientierte Notation. Der Hauptvorteil von UML ist neben der sehr großen

[6] Die Object Management Group (OMG) ist ein Zusammenschluß von über 800 Hard- und Softwareherstellern, der sich für die Standardisierung von Hard- und Softwarearchitekturen, Komponenten und Methoden einsetzt [OMG00].

Verbreitung die Prozeßneutralität, da nur eine Notation und Sprache definiert wurde, aber keine Prozesse. Dies erlaubt den Einsatz von UML für die objektorientierte Modellierung in einer Vielzahl von Anwendungsbereichen. Die Hauptbestandteile von UML sind verschiedene Diagramme, welche aus unterschiedlichen graphischen Elementen bestehen.

3.1.4 Diagramme der UML

Die UML umfaßt insgesamt 8 verschiedene Diagrammtypen:

- Anwendungsfalldiagramme
- Klassendiagramme
- Aktivitätsdiagramme
- Kollaborationsdiagramme
- Sequenzdiagramme
- Zustandsdiagramme
- Komponentendiagramme
- Einsatzdiagramme

Die wichtigsten Diagramme sind das Anwendungsfalldiagramm (Use-Case-Model), das Klassendiagramm und das Aktivitätsdiagramm, welche auch in dieser Arbeit verwendet werden.

3.1.5 Anwendungsfalldiagramme

Ein Anwendungsfalldiagramm besteht aus mehreren einzelnen Anwendungsfällen oder Use-Cases und einer Anzahl von Objekten, die daran beteiligt sind. Die Anwendungsfälle werden durch Ellipsen und die Objekte, welche auch als Akteure bezeichnet werden, durch Strichmännchen oder ähnliche Symbole dargestellt. Anwendungsfalldiagramme beschreiben das Zusammenwirken von Personen mit einem System und sind hauptsächlich aus der Arbeit von Ivar Jacobson entstanden. Die Beschreibung in diesen Diagrammen beschränkt sich auf die Aktivitäten, die durch die zu entwickelnde Software unterstützt werden sollen.
Anwendungsfälle werden als Hilfsmittel zur Anforderungsermittlung eingesetzt und sind dazu da, die Kommunikation zwischen den zukünftigen Anwendern und den Programmierern oder Systemdesignern zu verbessern. Sie stellen das externe Systemverhalten dar, wohingegen die Anwendungsfalldiagramme die Zusammenhänge

zwischen den einzelnen Anwendungsfällen darstellen. Abbildung 3-1 zeigt, daß die einzelnen Anwendungsfälle durch Linien mit den beteiligten Objekten verbunden werden. Innerhalb eines Diagramms können die Anwendungsfälle durch Beziehungen miteinander verbunden werden. Dafür werden die Verbindungslinien mit << *uses* >> oder << *extends* >> beschriftet.

- << *uses* >> wird verwendet, wenn mehrere Anwendungsfälle einen gleichen oder ähnlichen Aufbau haben. Dann kann der identische Teil in einen separaten Anwendungsfall ausgelagert und durch die << *uses* >> Verbindung wieder in verschieden Anwendungsfällen eingebunden oder mitbenutzt werden. Der Pfeil zeigt dabei in die Richtung des mitbenutzten Anwendungsfalles.
- << *extends* >> wird verwendet, um das Verhalten von Anwendungsfällen zu erweitern. Der Pfeil zeigt dabei von der Erweiterung zum Normalfall.

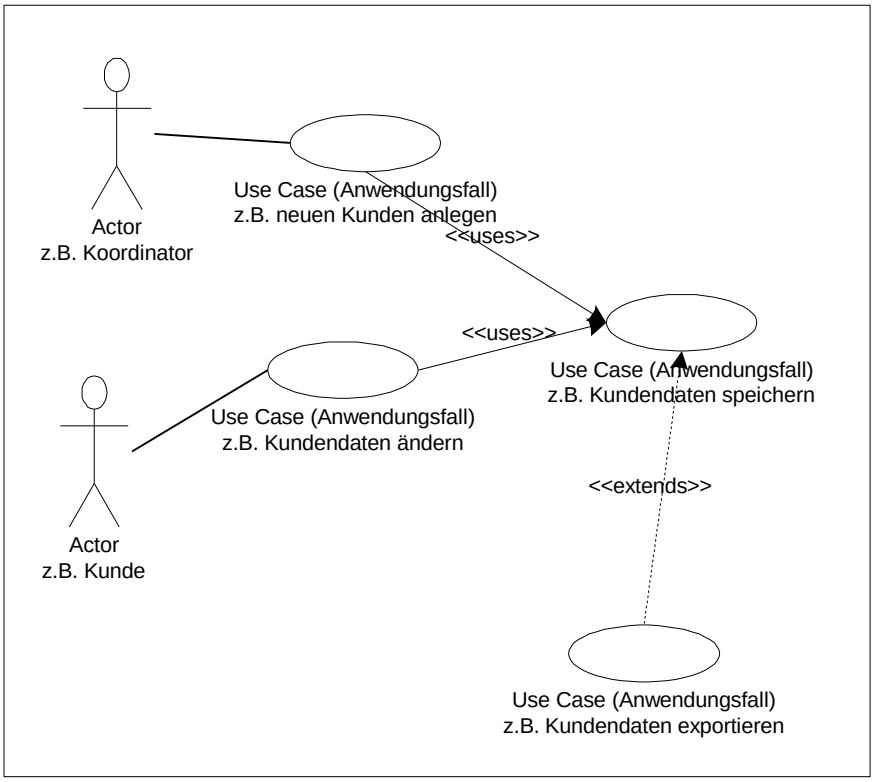
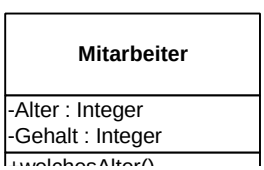


Abbildung 3-1 Beispiel Klassendiagramm

3.1.6 Klassendiagramme

Klassendiagramme sind der zentrale Bestandteil für objektorientierte Systeme und somit



auch von UML, sowie zahlreicher anderer objektorientierter Methoden. Sie können zu fast jedem Zeitpunkt im Analyse- und Designprozeß einer Softwareentwicklung eingesetzt werden. Die Klassendiagramme weisen eine sehr starke Ähnlichkeit zu Enitity-Relationship Diagrammen auf, die aus der Benutzung von ähnlichen Konzepten herrührt. Klassen bestehen aus Attributen und Methoden. Die Attribute stellen die Eigenschaften der Klasse dar, wie z.B. bei der Klasse Mitarbeiter das Alter oder das Gehalt. Die Methoden einer Klasse sind festgelegte Operationen, die mit den Attributen der Klasse durchgeführt werden dürfen, wie z.B. die Methode *setzeAlter()*, die das Alter des Mitarbeiters verändern könnte oder die Methode *welchesGehalt()*, die das Gehalt des Mitarbeiters ausgeben könnte. Klassen werden in UML durch ein Rechteck dargestellt, welches entweder nur den Namen der Klasse enthält oder noch zusätzlich, durch horizontale Linien getrennt, die Attribute und Methoden. Hinter dem Namen eines Attributes folgt normalerweise ein Doppelpunkt sowie der Datentyp des Attributes. Methoden werden durch runde Klammern hinter dem Namen gekennzeichnet. Abbildung 3-2 zeigt ein Beispiel einer Klasse.
Die Klassendiagramme beschreiben die Struktur und die Beziehungen der Objekte in einem System. Aus diesen Diagrammen kann mit Hilfe von Softwareentwicklungswerkzeugen, wie z.B. dem Programm Rose der Firma Rational, direkt Programmcode generiert werden. Klassendiagramme können neben Klassen auch

Abbildung 3-2 Beispiel einer Klasse

Schnittstellendefinitionen (Interfaces) enthalten. Diese werden durch einen Kreis dargestellt worunter der Name der Schnittstellendefinitionen steht. Die einzelnen Elemente in einem Klassendiagramm sind mit Linien verbunden. Diese Linien können folgende Elemente darstellen:

- Assoziation:

  Die Assoziation stellt eine allgemeine Beziehung zwischen zwei Klassen dar und wird als einfache Linie gezeichnet. An den Enden der Assoziation können Kardinalitäten[7] angegeben werden.

[7] Kardinalitäten sind Zahlen, die angeben wieviele Objekte miteinander in Beziehung stehen.

- Aggregation:

  Eine Aggregation ist eine besondere Form der Assoziation, welche durch eine Raute gekennzeichnet wird. Sie kennzeichnet, daß die eine Klasse ein Teil der anderen Klasse ist, z.B. ist die Klasse *Physician* ein Teil der Klasse *CareClient*. Dabei wird die Raute auf die Seite der Klasse gezeichnet, welche die andere Klasse enthält.

- Komposition:

  Die Komposition ist eine stärkere Form der Aggregation, bei der eine Existenzabhängigkeit besteht. Die Komposition wird durch eine ausgefüllte Raute gekennzeichnet. Auch hier wird die Raute auf die Seite der Klasse gezeichnet, welche die andere Klasse enthält.

- Vererbung:

  Die Vererbung ist eine Verallgemeinerung von Eigenschaften (Generalisierung). Sie ist ein Grundprinzip von objektorientierten Programmiersprachen. Dabei werden alle Attribute (Eigenschaften) und Methoden von der Ober- an die Unterklasse weitergegeben. Sie wird durch einen Pfeil dargestellt, der von der Unter- zur Oberklasse zeigt. Siehe Abbildung 3-3.

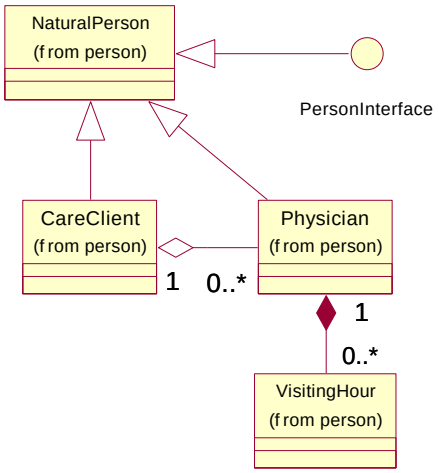


Abbildung 3-3 Beispiel Klassendiagramm

3.1.7 Aktivitätsdiagramme

Aktivitätsdiagramme beschreiben einzelne Schritte in einem Verarbeitungsablauf. Sie sind auf der Grundlage der Zustands- und Flußdiagramme sowie der Petrinetze entstanden. Das Kernelement dieser Diagramme sind die Aktivitäten, die als abgerundetes Rechteck gezeichnet werden. Daneben gibt es noch Synchronisationslinien, die als waagerechter Balken gezeichnet werden. Der Startpunkt eines Aktivitätsdiagramms wird als ausgefüllter Kreis und der Endpunkt als Kreis mit einem Punkt im Inneren gezeichnet.
Mit Aktivitätsdiagrammen lassen sich sehr gut Prozeßabläufe mit Parallelitäten beschreiben. So sind in Abbildung 3-4 die Prozesse *Rechnung erstellen* und *Lieferung zusammenstellen* als parallele Prozesse zu verstehen.

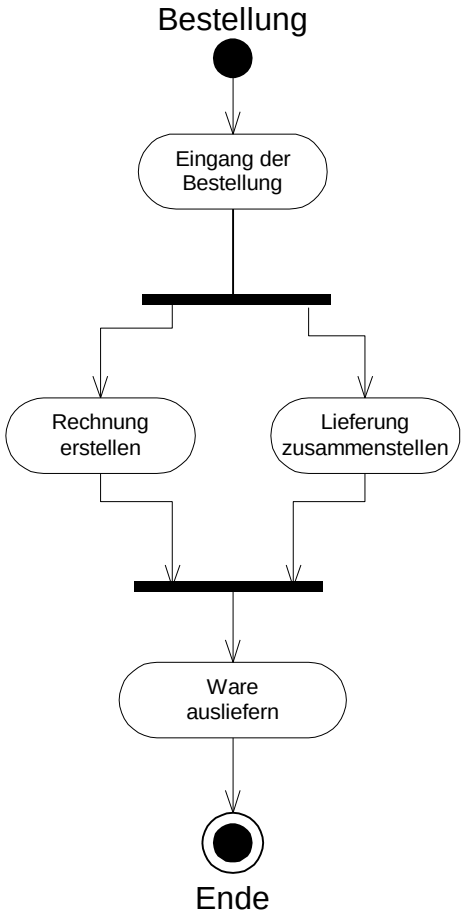


Abbildung 3-4 Beispiel Aktivitätsdiagramm

3.2 Java

Die Programmiersprache Java wurde Ende 1995 von der Firma Sun Microsystems eingeführt und hat sich seitdem mit großer Geschwindigkeit weiterentwickelt. Ab der Version 1.2 wurde Java in Java 2 umbenannt. Für die Programmierung in dieser Arbeit wird Java 2 in der Version 1.3 benutzt welchen im Mai 2000 veröffentlicht wurde. Der Hauptunterschied zwischen Java und anderen objektorientierten Programmiersprachen ist, daß Java eine interpretierte Sprache ist. Das bedeutet, daß in Java geschriebene Programme nach der Programmierung nur in einen universellen Bytecode kompiliert werden. Dieser Bytecode wird zur Laufzeit von einer Java Virtual Machine (JVM) interpretiert und ausgeführt. Daraus ergibt sich die gute Portabilität von Java Programmen, da der universelle Bytecode auf jedem Computer ausgeführt werden kann, für den eine JVM existiert. Die Hauptmerkmale von Java sind:

- Objektorientierung
- Architekturneutralität und Portabilität
- Sicherheit
- Dynamik und Verteilung

Näheres hierzu ist in [Flan98] zu finden.
Java Programme werden heute hauptsächlich für die Programmierung in heterogenen Netzwerken wie z.B. dem Internet eingesetzt. Java ist aber auch bei der reinen Applikationsprogrammierung auf dem Vormarsch. Die Hauptgründe für den bisher zögerlichen Einsatz in diesem Bereich sind wohl die Geschwindigkeitsnachteile gegenüber herkömmlichen Programmiersprachen wie C++, die sich aus der Interpretation zur Laufzeit ergeben. Diese werden aber durch die ständige Verbesserungen von Java sowie die andauernde Leistungssteigerung der Hardware immer unbedeutender.
Einer der Gründe für den Einsatz von Java bei der Prototypenentwicklung in SWIFT, ist die Unterstützung der verteilten Programmierung. Die Programmiersprache Java unterstützt die Verteilte- oder Netzwerkprogrammierung auf einer sehr hohen Ebene durch RMI.

3.2.1 RMI (Remote Method Invocation)

Herkömmliche Ansätze für die Kommunikation von Programmen auf verschiedenen in einem Netzwerk verteilten Computern sind sehr kompliziert und daher für Fehler anfällig. Das beste Beispiel dafür ist CORBA[8], was durch seine hohe Komplexität viele potentielle Anwender abschreckt. Der einfachste Ansatz zur Lösung dieses Problems ist die Vorstellung, daß ein entferntes Objekt genau wie ein lokales Objekt angesprochen werden kann. Das bedeutet, man kann Nachrichten an dieses Objekt senden und die Resultate entgegennehmen, ohne zu bemerken, daß sich dieses Objekt nicht auf dem eigenen Computer befindet. Dabei sollte die gesamte Netzwerkkommunikation vor dem Programmierer verborgen werden.
Genau diese Funktion hat Java RMI.
Java RMI ist eine Erweiterung des Client-Server Modells mit dem allgemeineren Modell entfernter Objekte. In diesem Modell definiert der Server Objekte, welche die Clients über ein Netzwerk benutzen können [Flan98-2, 339]. RMI hat den Nachteil gegenüber CORBA, daß sowohl der Client wie auch der Server ein Java Programm sein müssen. RMI ist in Java seit der Version 1.1 enthalten und wurde ab der Version 1.2 entscheidend verbessert.

3.2.2 Grundaufbau von RMI

RMI basiert auf dem Prinzip der Trennung von Definition und Implementation. Siehe Abbildung 3-5. Damit ist es möglich, daß der Code, der das Verhalten eines Objektes bestimmt (Definition), und der Code, welcher das Objekt darstellt (Implementation) auf verschiedenen JVM's und damit auf verschiedenen Rechnern ausgeführt werden können.
Die Definition wird dabei in einem Java Interface codiert, während die Implementation in einer Java Klasse codiert wird.

- Definition -> Verhalten des Objektes -> Interface
- Implementation -> Das Objekt selbst -> Klasse

[8] Die Common Object Request Broker Architecture (CORBA) ist eine von der OMG herausgegebene Architektur, welche die Kommunikation von in verschiedenen Programmiersprachen geschriebenen Programmen über Netzwerke ermöglichen soll.

Diese Aufteilung paßt genau auf die allgemeinen Anforderungen von verteilten Systemen, bei denen der Client nur an der Definition eines Dienstes interessiert ist und der Server für die Bereitstellung diese Dienstes zuständig ist.

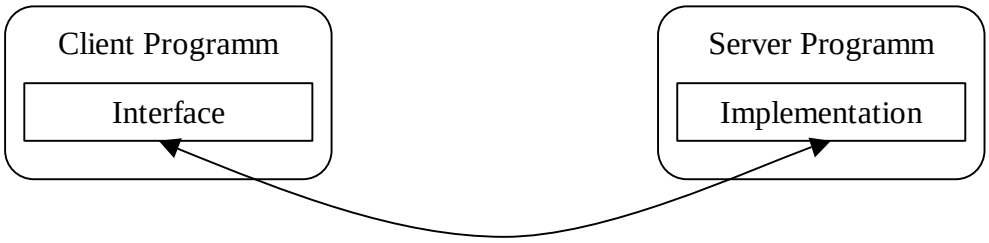


Abbildung 3-5 Grundmodell der Kommunikation über RMI

3.2.3 Architektur von RMI

RMI besteht aus einer Drei-Schicht Architektur, die aus folgenden Schichten aufgebaut ist:

- Stub & Skeleton Layer:
  Das Stub & Skeleton Layer liegt direkt unterhalb der Sicht des Programmierers und beruht auf dem Proxy Entwurfsmuster[9] wie es in [Gamma96, 254] beschrieben wird. Diese Schicht fängt die Methodenaufrufe des Client ab und leitet diese an den Remote Reference Layer um. RMI benutzt dabei den Standardmechanismus der Stubs und Skeletons, um mit entfernten Objekten zu kommunizieren.
  Ein Stub ist eine clientseitige, lokale Repräsentation oder Proxy für das entfernte Objekt. Der Client ruft Methoden auf dem lokalen Stub auf, welcher für deren Ausführung auf dem entfernten Objekt verantwortlich ist.
  Ein Skeleton ist das serverseitige Gegenstück zum Stub. Er nimmt die Anfragen des Stub entgegen, und leitet diese an das Objekt weiter. Außerdem nimmt er die Ergebnisse der Methodenaufrufe entgegen und gibt diese an den Stub zurück.
  Ab der Version 1.2 von Java ist es möglich, die Skeletons weggefallen zu lassen und durch ein neues Netzwerkprotokoll zu ersetzen.
- Remote Reference Layer:
  Das Remote Reference Layer verbindet die Clients mit den entfernten Objekten.

[9] Entwurfsmuster sind aus Erfahrungen anderer Programmierer entstandene, vorgefertigte Problemlösungen.

Dies ist in Java 1.1 immer eine eins-zu-eins Verbindung (unicast link). Ab der Java Version 2 wurde diese Schicht um die Möglichkeit der Objektaktivierung sowie um die Fähigkeit eins-zu-n Verbindungen (multicast links) aufzubauen erweitert.

- Transport Layer :

  Das Transport Layer stellt die grundlegenden Verbindungen zwischen verschieden JVM's sowie einige Mechanismen, welche die Kommunikation über Firewalls[10] hinweg ermöglichen sollen, zur Verfügung. Es basiert bis zur Version 1.2 ausschließlich auf TCP/IP [11] Verbindungen und wurde in der Version 1.3 um die Unterstützung des IIOP[12] Protokolls erweitert. Dies soll eine Verknüpfung zwischen RMI und CORBA ermöglichen.

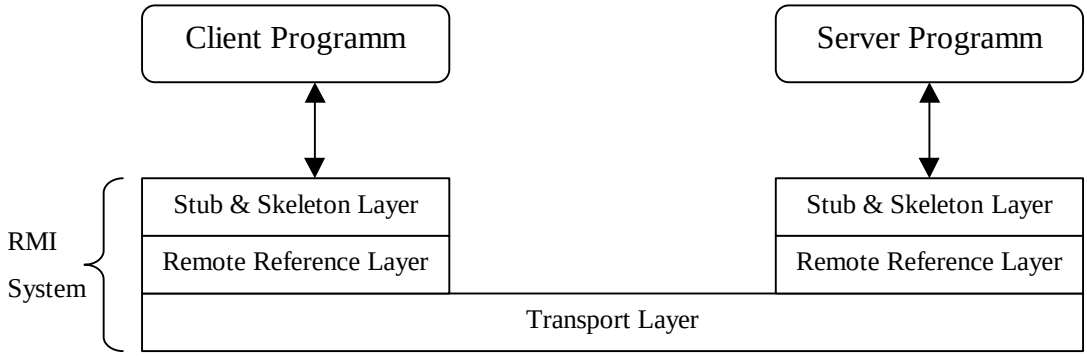


Abbildung 3-6 Modell der Kommunikationsschichten von RMI

Abschließend ist zu beachten, daß eine Kommunikation über Java RMI, die zwischen zwei Programmen auf einem Computer abläuft, genau wie eine Kommunikation zwischen zwei in einem Netzwerk verteilten Computern über das in Abbildung 3-6 dargestellte Modell abgewickelt wird.

3.2.4 Komponenten eines RMI Systems

Für eine Kommunikation über RMI müssen die folgenden Programm-Komponenten erstellt werden:

[10] Firewalls sind Programme, die Netzwerkverbindungen einschränken und überwachen, um die Sicherheit vor unbefugter Benutzung und Mißbrauch zu erhöhen.
[11] Transmission Control Protocol / Internet Protocol (TCP/IP) ist ein Netzwerkprotokoll auf dem z.B. das Internet basiert.
[12] IIOP ist das Internet Inter Orb Protokoll, welches für die Kommunikation zwischen CORBA Anwendungen verwendet wird.

- Eine Interfacedefinition, die das Verhalten des Objekts (Remote-Objekt) bestimmt, welches später über RMI zur Verfügung steht. Dieses Interface definiert die Methoden mit ihren Parametern und Rückgabewerten, die später vom Clientprogramm aufgerufen werden.
- Die Implementation der Klasse, die das Remote-Objekt repräsentiert. In dieser Klasse sind die Methoden, die durch die Interfacedefinition vorgegeben wurden ausprogrammiert.
- Stub- und Skeletonklassen, sie werden aus der zuvor erstellten Klasse generiert. Dies geschieht mit Hilfe des RMIC Compilers, der standardmäßig in Java enthalten ist.
- Ein Serverprogramm, welches das Remote-Objekt zur Verfügung stellt.
- Ein Clientprogramm, welches das Remote-Objekt benutzt.

3.3 Datensicherheit

Für den in dieser Arbeit entwickelten Softwareprototyp ist die Sicherheit der Daten ein entscheidendes Thema, da die Versendung von sensiblen Kundendaten über ein Netzwerk ein Hauptbestandteil des Programmes ist. Um aus der großen Vielfalt der verschieden Methoden Daten zu sichern die richtigen auswählen zu können, werden folgende Schutzziele definiert:

- Integrität:
  Dies ist die Sicherstellung, daß die Informationen auf dem Transportweg nicht unbemerkt verändert werden können.
- Vertraulichkeit:
  Dies bedeutet, daß sichergestellt werden kann, daß eine Information nur von demjenigen gelesen werden kann, für den sie bestimmt ist.
- Authentizität:
  Dies ist die Sicherstellung, daß eine Information auch wirklich von dem Absender stammt, der angibt, der Absender zu sein.

Am Grad der Erreichung dieser Schutzziele kann die Sicherheit der Daten überprüft werden. Die hier definierten Schutzziele leiten sich aus den in [Müller97] beschriebenen Schutzzielen ab.

3.3.1 Verschlüsselung mit Java

Für die Gewährleistung der zuvor definierten Schutzziele stellt die Programmiersprache Java verschiedene Möglichkeiten zur Verfügung, die auf den Konzepten der Kryptographie mit öffentlichen und privaten Schlüsseln basieren, wie sie z.B. in [Horstmann00, 852] beschrieben werden.
Hierfür werden mit Hilfe eines bestimmten mathematischen Algorithmus Schlüsselpaare erzeugt, mit denen dann die eigentliche Verschlüsselung durchgeführt wird. Jedes Schlüsselpaar besteht aus einem öffentlichen Schlüssel, der frei zugänglich gemacht wird und einem privaten Schlüssel, der gesichert aufbewahrt werden muß. Mit Hilfe dieser Schlüssel können nun Maßnahmen zur Erreichung der einzelnen Schutzziele durchgeführt werden.

- Integrität

  Zur Erreichung dieses Schutzzieles, also der Sicherstellung, daß das zu versendende Objekt auf dem Transportweg nicht verändert wurde, stellt Java den Mechanismus der "Signed Objects" zur Verfügung. Abbildung 3-7 zeigt, daß hierbei das zu versendende Objekt mit einer digitalen Signatur versehen wird. Diese Signatur wird mit dem privaten Schlüssel des Absenders erzeugt.
  Der Empfänger eines so signierten Objektes kann nun mit dem öffentlichen Schlüssel des Absenders prüfen, ob die Signatur mit dem richtigen privaten Schlüssel erzeugt und seither nicht verändert wurde.

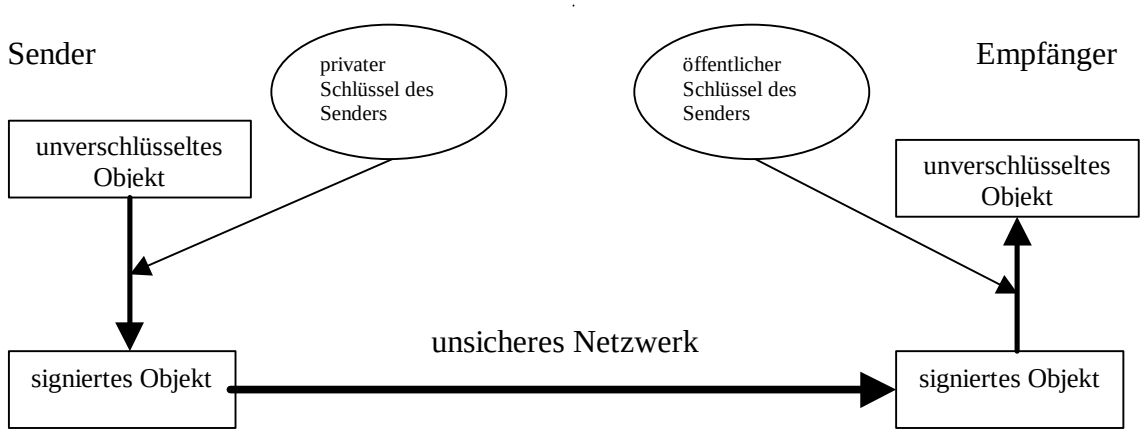


Abbildung 3-7 Signatur von Objekten

- Vertraulichkeit

  Um dieses Schutzziel zu erreichen, sollten die über das Netzwerk zu versendenden Objekte verschlüsselt werden. Java stellt hierfür den Mechanismus der "Sealed Objects" zur Verfügung. Dabei werden die zu versendenden Objekte mit dem öffentlichen Schlüssel des jeweiligen Empfängers verschlüsselt. Die so verschlüsselten Objekte kann nur der Empfänger mit dem richtigen privaten Schlüssel wieder entschlüsseln. Siehe Abbildung 3-8.

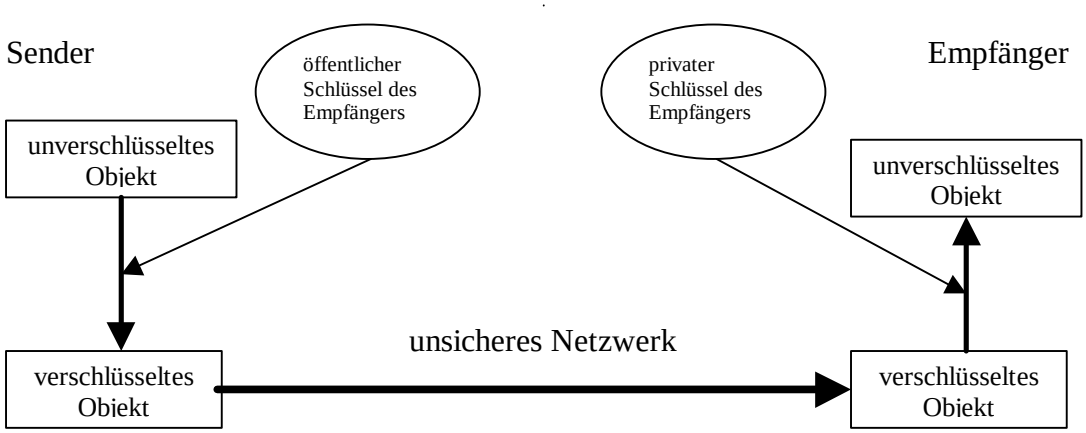


Abbildung 3-8 Verschlüsselung von Objekten

- Authentizität, dieses Schutzziel kann nicht mit Java erfüllt werden, da es hier darum geht die Identität des Senders eines Objektes sicherzustellen. Der einfachste Lösungsansatz hierfür ist die Inanspruchnahme einer Zertifizierungsstelle. Diese Zertifizierungsstellen sind Firmen oder Einrichtungen, die Zertifikate nach dem deutschen Signaturgesetz ausgeben. Ein Zertifikat muß nach Artikel 3 §7 des IuKDG[13] die folgenden 8 Punkte beinhalten:
  1. Name des Zertifikatinhabers
  2. Den zugeordneten öffentlichen Schlüssel
  3. Die Bezeichnung des dahinterstehenden Algorithmus
  4. Die laufende Nummer des Zertifikates
  5. Beginn und Ende der Gültigkeit des Zertifikates
  6. Den Namen der Zertifizierungsstelle

[13] IuKDG ist das Informations- und Kommunikationsdienste-Gesetz

7. Angaben, ob die Nutzung des Signaturschlüssels auf bestimmte Anwendungen nach Art und Umfang beschränkt ist.
8. Hinweise auf zugehörige Zertifikate.

Zertifizierungsstellen überprüfen bei der Erstellung eines Zertifikates die Identität des Anwenders und bürgen dafür, daß der unter dem Namen des Anwenders registrierte öffentliche Schlüssel auch zu diesem gehört. Darüber hinaus bieten diese Einrichtungen auch die Möglichkeit, die Generierung der Schlüssel zu übernehmen sowie für die Verbreitung der öffentlichen Schlüssel zu sorgen, in dem sie diese z.B. auf ihrer Internet Seite zum Abruf bereit stellen.

3.4 JDeveloper

Für die Programmierung des Prototyps in Java wurde das Programm JDeveloper von der Firma Oracle verwendet. Dies ist eine integrierte Java-Entwicklungsumgebung, welche die Möglichkeit bietet vom Java Applet bis hin zur CORBA-Awendung jede Art von Java-Programmen zu entwickeln. Darüber hinaus verfügt dieses Programm über einige Funktionen, welche die Entwicklung von Datenbankanwendungen sowie von mehrschichtigen Architekturen erleichtern sollen.
In dieser Arbeit wird der JDeveloper in der Version 3.1 verwendet. Dieser wird zusammen mit den Java Versionen 1.1.8 sowie 1.2 ausgeliefert und ist nur in englischer Sprache verfügbar. Er bietet aber die Möglichkeit auf Java 1.3 umgestellt zu werden.

3.4.1 Gründe für die Auswahl

Einer der Hauptgründe für die Auswahl des JDevelopers als Entwicklungsumgebung ist die Verfügbarkeit. Das Programm ist für den nicht kommerziellen Gebrauch frei und ohne jede funktionelle Einschränkung verfügbar [ORA00]. Darüber hinaus bietet der JDeveloper einige weitere entscheidende Vorteile gegenüber anderen frei verfügbaren Java-Entwicklungsumgebungen. Dies sind z.B. die Add-In-API Funktion, welche bei der Programmierung im Quellcode automatisch die entsprechenden Teile der API-Dokumentation einblendet und somit ein zeitraubendes Suchen nach Funktionen, Methoden oder Parametern in der API-Dokumentation vermeidet. Außerdem bietet der JDeveloper die Möglichkeit, zwei oder mehrere Programme gleichzeitig auszuführen, was besonders bei der Erstellung von verteilten Systemen von Vorteil ist, da hierbei z.B. oft an Server- und Client-Programmen gleichzeitig gearbeitet wird.

Ein weiterer Grund für die Auswahl des JDevelopers ist, daß dieser neben einem normalen Editor noch über eine graphische Designoberfläche, den Visual Designer, verfügt. Damit ist es sehr leicht und schnell möglich die GUI-Komponenten eines Programmes zu gestalten. Dies geschieht ähnlich wie in einem Zeichenprogramm durch eine direkte Manipulation der Komponenten oder durch das Ändern ihrer Eigenschaften in einem Dialog, dem Property Inspector.

3.4.2 Benutzeroberfläche des JDevelopers

Die Benutzeroberfläche des JDevelopers kann in den Kommandobereich und den Entwicklungsbereich unterteilt werden. Siehe Abbildung 3-9.
Der Kommandobereich unterteilt sich wiederum in drei Bereiche:

- Die Menüleiste, hier sind alle Standard Menüs wie z.B. File, Edit oder View sowie einige zusätzliche Menüs wie z.B. Project, Run oder Wizards zu finden.
- Die Toolbar, hier sind die wichtigsten Funktionen der Menüleiste zusätzlich auf Schaltflächen hinterlegt wie z.B. Run, Debug, Search, Copy, Paste.
- Die Komponentenpalette, hier finden sich die Komponenten, welche mit der graphischen Designoberfläche verbaut werden können. Dies sind einerseits alle Standard Java GUI Komponenten wie z.B. JButton, JLabel oder AWT Button und andererseits Java Komponenten anderer Hersteller, welche sehr einfach in die Palette eingefügt werden können.

Der Entwicklungsbereich unterteilt sich ebenfalls in drei Bereiche. Dies sind:

- Das Nachrichtenfenster oder Message View, dieses Fenster übernimmt die Aufgabe der Systemkonsole und stellt Compiler-, Fehler- oder Systemmeldungen dar.
- Der Navigator ist ein Mehrzweckfenster für den Zugriff auf die Java-Quelldateien. Er ist unterteilt in einen oberen und einen unteren Teil. Im oberen Teil werden die Quelldateien dargestellt und im unteren die Struktur der Java Datei z.B. Variablen, Methoden oder importierte Pakete. Außerdem wird der Navigator noch in die drei Karteikarten Workspace, Opend und Directory unterteilt. Der Workspace dient zu Verwaltung der Projekt- und Javadateien. Die Opend Karteikarte ist eine zweite einfachere Sicht auf Javadateien, die sich nicht innerhalb des Projektes befinden. Hier werden zusätzlich geöffnete Dateien angezeigt, aus denen man z.B. Teile des

Codes herauskopieren möchte. Die Directory Karteikarte ist eine Sicht auf das Dateisystem, ähnlich dem Windows Explorer. Sie kann z.B. zum Auffinden von Dateien benutzt werden und bietet den Vorteil, daß man dafür nicht extra das Programm wechseln muß.

- Das Editorfenster, welches auch Viewer genannt wird. Seine Hauptaufgabe ist die Darstellung des Quellcode-Editors. Darüber hinaus hat es aber noch weitere Funktionen, die sich aus der Unterteilung in Karteikarten ergeben. Diese verändert sich je nach Typ der im Navigator gewählten Datei. Das bedeutet, daß dieser Editor außer Javadateien genauso auch XML-, HTML- oder Bilddateien anzeigen kann. Für Java-Quellcode stehen die Karteikarten Source, Design, Bean und Doc zur Verfügung. Die Source-Karteikarte stellt den Editor für den Quellcode zur Verfügung. Die Design-Karteikarte startet die graphische Designoberfläche. Die Bean-Karteikarte bietet verschiedene Möglichkeiten für die Konvertierung von Javadateien in Java Beans und die Doc-Karteikarte stellt die entsprechende Seite der Java API Dokumentation im Viewer dar.

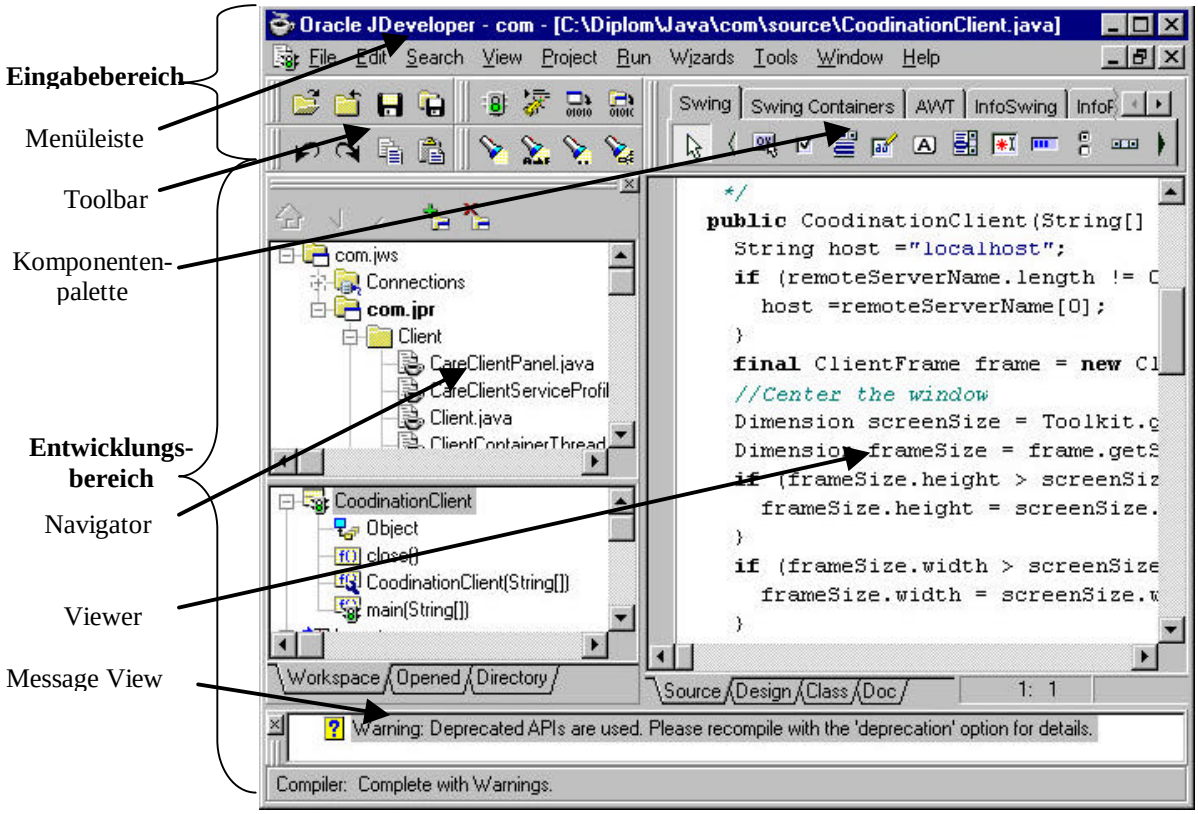


Abbildung 3-9 Benutzeroberfläche des JDevelopers

4 Ist-Soll Analyse der Abläufe eines Servicezentrums

Das nachfolgende Kapitel beschäftigt sich mit der Analyse der Koordinierungsabläufe in einem Servicezentrum und der Erarbeitung der Anforderungen an einen zukünftigen Koordinierungsablauf. Dazu wird zunächst der Ist-Zustand in einem Servicezentrum und die damit verbundenen Arbeitsabläufe beschrieben. Danach wird ein Soll-Zustand mit einem softwareunterstützten Koordinierungsablauf beschrieben. Die Betrachtung der Koordinierungsabläufe beschränkt sich dabei auf die drei in dieser Arbeit untersuchten Diensttypen. Anschließend werden die einzelnen für den Soll-Prozeß benötigten Dokumente beschrieben und die Anforderungen an einen Softwareprototyp definiert.

4.1 Beschreibung des Ist- Zustandes

Die Tätigkeiten in dem untersuchten Servicezentrum laufen bisher ohne EDV-Unterstützung ab. Das bedeutet, daß alle Vorgänge auf Formularen basieren, die von Hand ausgefüllt werden. Die Arbeitsabläufe in einem Servicezentrum beginnen normalerweise mit dem Anruf oder dem persönlichen Erscheinen eines Pflegebedürftigen oder eines Angehörigen. Daraufhin wird zunächst die Art der Anfrage, unterschieden. Hier geht es darum abzuschätzen, ob es sinnvoll ist, die persönlichen Daten des Kunden aufzunehmen oder ob es sich nur um eine einfache Auskunft handelt, nach der keine weiteren Arbeitsschritte notwendig sind.
Handelt es sich um eine Anfrage die weitere Schritte nach sich zieht, wird ein Formular mit der Bezeichnung *Kundenanfrage* ausgefüllt. In dieses Formblatt werden die persönlichen Daten des Kunden eingetragen. Dies sind Name, Geburtsdatum, Adresse des Kunden, der Name des Hausarztes, Angaben über Bezugspersonen (Name, Anschrift, Telefon), die Bezeichnung der Krankenkasse sowie die Versicherungsnummer bei der Krankenkasse.
Außerdem wird noch vermerkt, ob es sich bei der Anfrage um einen Erstkontakt handelt und durch wen die Anfrage gestellt wird. Darüber hinaus wird der Typ der Anfrage unterschieden. Hier bietet das bisher verwendete Formular die Möglichkeit, durch Ankreuzen zwischen häuslicher Pflege, Menüdienst, Hospizarbeit, einem SGB XI

Termin[14], Mobiler Sozialer Dienst, Hausnotruf, Fahrdienst, Beratung und Sonstiges zu unterscheiden. Mehrfachauswahlen sind an dieser Stelle üblich, da viele Kunden z.B. einen ambulanten Pflegedienst und einen Hausnotrufdienst in Anspruch nehmen. Der Leistungsträger der ambulanten Versorgung kann durch Ankreuzen eines der Felder SGB XI, SGB V, BSHG (Bundes Sozialhilfe Gesetz) oder Selbstzahler ausgewählt werden. Außerdem werden noch bekannte Diagnosen bzw. pychosoziale Aufälligkeiten, Angaben zur Versorgungssituation, der Hilfewunsch mit Art, Dauer und Häufigkeit sowie Angaben über den Dienst, an den der Kunde vermittelt wird, als Freitext erfaßt. Nachdem die Stammdaten und Leistungswünsche des Kunden erfaßt worden sind, wird mit der eigentlichen Koordinierung begonnen. Diese hat für alle Dienste, außer für ambulante Pflegedienste, den gleichen Ablauf.
In diesem Standardablauf wählt der Koordinator zusammen mit dem Kunden einen Dienstanbieter aus. Die Auswahl des Dienstanbieters hängt von den Kenntnissen des Koordinators ab. Da die meisten Servicezentren einem Träger wie z.B. den Johannitern zugeordnet sind wird zuerst über die zu diesem Träger gehörenden Dienste beraten. Nach der Auswahl wird der gewählte Dienst angerufen und der Name und die Adresse des Kunden an diesen Dienst weitergegeben. Hiermit ist die Koordination an dieser Stelle beendet. Alle weiteren Schritte, wie der Vertragsabschluß oder die Bestimmung des genauen Leistungsumfangs entziehen sich der Aufsicht des Koordinators. Hervorzuheben ist jedoch, daß jeder vom Kunden angeforderte Dienst die Stammdaten des Kunden erneut aufnimmt und nicht auf die vom Koordinator bereits erfaßten Datenmengen zurückgreift.

4.1.3 Koordinierung mit ambulanter Pflege

Der Koordinierungsablauf ändert sich, wenn der Kunde nach Leistungen von ambulanten Pflegediensten fragt. In diesem Fall wird zusätzlich das Formular *persönliches Leistungsprofil* zusammen mit dem Kunden ausgefüllt. Dieses Formular ist wie eine Tabelle aufgebaut. Darin sind auf der Hochachse die 26 vom SGB XI vorgegebenen Leistungskomplexe sowie deren Einzelpreise aufgeführt. Die Längsachse enthält die Tage einer Woche mit je einer Spalte für Vormittags, Mittags, Nachmittags,

[14] Ein SGB XI Termin muß von pflegenden Angehörigen mit einem ambulanten Pflegedienst vereinbart werden, um die Versorgungssituation des Pflegebedürftigen zu kontrollieren.

Abends und Nachts. Außerdem gibt es noch Spalten für die Anzahl der Leistungen sowie den Betrag für eine Leistung und Felder für den Gesamtbetrag der Leistungen für eine Woche, den Betrag, den die Pflegeversicherung zahlt und den sich daraus ergebenden Restbetrag, den der Kunde aufbringen muß.
Das persönliche Leistungsprofil gibt dem Kunden einen Überblick über die Leistungen, die er angefordert hat. Außerdem kann es zu groben Einschätzung der für die Pflege anfallenden Kosten verwendet werden.
Als nächsten Schritt in der Koordination fragt der Koordinator telefonisch bei den ambulanten Pflegediensten nach, ob diese die im persönlichen Leistungsprofil angegebenen Leistungen erbringen können. Im Normalfall wird zuerst bei dem der Koordinierungsstelle angeschlossen Dienst angefragt. Sind dort keine Kapazitäten mehr frei, wird ein Dienst angerufen, mit dem der Koordinator schon zusammengearbeitet hat.
Wenn ein ambulanter Pflegedienst gefunden ist, der in der Lage ist, die Versorgung zu übernehmen, werden der Name und die Adresse des Kunden an diesen Dienst weitergegeben. Damit ist auch hier die Arbeit des Koordinators beendet. Auch bei den ambulanten Pflegediensten entziehen sich alle weiteren Schritte dem Einfluß des Koordinators und es werden alle Stammdaten des Kunden von dem ausgewählten Dienst erneut aufgenommen. Außerdem wird beim Erstbesuch des ambulanten Pflegedienstes erneut ein *persönliches Leistungsprofil* erstellt, wobei das bereits vom Koordinator erstellte nicht beachtet wird. Abbildung 4-1 gibt eine Übersicht über den oben beschriebenen Ist-Abauf.

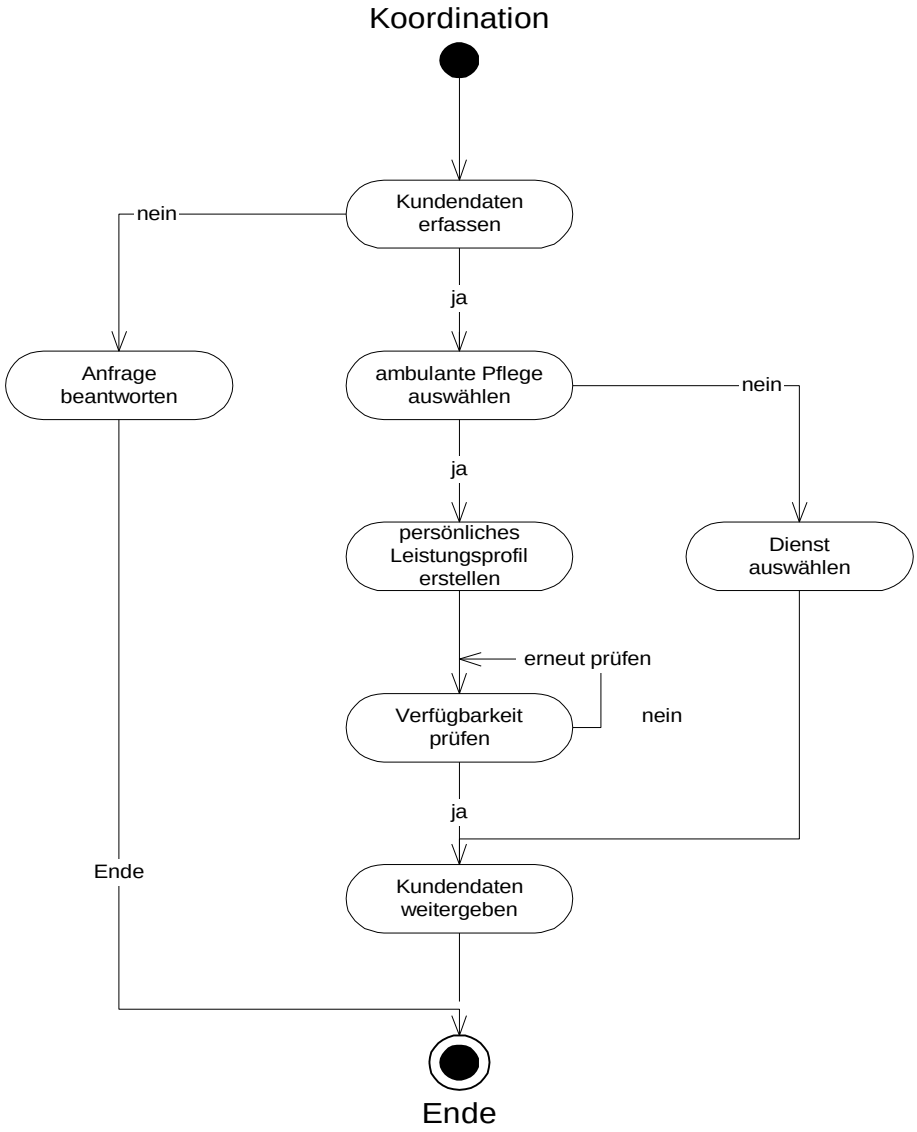


Abbildung 4-1 Ist-Ablauf einer Koordination

4.1.4 Zusammenfassung

Zur Zeit kommen die vom Koordinator zur Beratung und Koordination benutzten Informationen größtenteils aus dem Wissensschatz des Koordinators. Nur in Ausnahmefällen gibt es eine Leistungsbeschreibung von Diensten. Außerdem werden die Daten, die der Koordinator aufnimmt, wie z.B. die Stammdaten des Kunden oder das persönliche Leistungsprofil nicht weiter verwendet. Statt dessen werden diese Daten mehrfach aufgenommen bzw. erstellt.

Ergänzend muß festgehalten werden, daß in Servicezentren natürlich auch Leistungen aus anderen Bereichen, wie z.B. der stationären Pflege koordiniert werden, welche aber in dieser Arbeit nicht behandelt werden.

4.2 Beschreibung des Soll-Zustandes

In diesem Abschnitt wird ein softwareunterstützter Koordinierungsablauf beschrieben. Auch bei einer softwareunterstützten Koordination werden zuerst, sofern es sich nicht nur um eine einfache Anfrage handelt, die Stammdaten des Kunden erfaßt. Dies wird in einer Anfragemaske getan, welche dieselben Felder wie das Formular „Kundenanfrage“ enthält. Die Kundenanfrage enthält neben den Stammdaten des Kunden noch weitere Angaben, wie z.B. die Diensttypen über die der Kunde beraten werden möchte.
Die eigentliche Koordinierung hat einen ähnlichen Ablauf wie die Koordinierung ohne Softwareunterstützung. Bei der einfachen Koordinierung, ohne ambulante Pflege, kann der Koordinator nun die Dienste aus dem Servicekatalog auswählen und dem Kunden detaillierte Auskünfte über alle Dienste in einem Bereich sowie deren Leistungsangebote und Kosten geben. Nach der Auswahl eines Dienstes wird die Kundenanfrage mit den bereits eingegebenen Stammdaten des Kunden an den gewählten Dienst geschickt werden. In der Kundenanfrage sollten die Daten so aufbereitet sein, daß der empfangende Dienst sie direkt weiterverwenden kann. Auch bei diesem Konzept muß sich der Dienst anschließend beim Kunden melden und mit ihm einen Vertrag schließen.

4.2.5 Koordinierung mit ambulanter Pflege

Wie bei der herkömmlichen Lösung ist auch bei der softwareunterstützten Lösung die Auswahl von Leistungen der ambulanten Pflege mit zusätzlichen Eingaben verbunden. Auch hier füllt der Koordinator zusammen mit dem Kunden eine Maske aus, welche die Daten für das persönliche Leistungsprofil erfaßt. Das persönliche Leistungsprofil des Kunden und ein Teil seiner Stammdaten werden zu einer Leistungsanfrage zusammengestellt. Diese wird an alle angeschlossenen Dienste gesendet, um die Verfügbarkeit der Leistungen zu erfragen. Die Leistungsanfrage sollte die Daten in einer solchen Form beinhalten, daß der Pflegedienst diese weiterverwenden kann und nicht, wie in der Praxis üblich, komplett neu aufnehmen muß. Außerdem ist darauf zu achten, daß die Leistungsanfrage nur anonymisierte Kundendaten enthält.

Nach dem Erhalt einer solchen Leistungsanfrage haben die Dienste nun die Gelegenheit, diese zu prüfen und ein Leistungsangebot zurückzuschicken. Dieses Leistungsangebot sollte neben dem Zeitpunkt an dem die Pflege durchgeführt werden soll, auch einen Termin für ein Erstgespräch enthalten. Ein Problem bei den ambulanten Pflegediensten ist neben der generellen Verfügbarkeit der Leistungen auch die Zeit, zu der die Leistungen erbracht werden können. Die meisten Pflegebedürftigen möchten gerne zu einer bestimmten Zeit gepflegt werden, diese Zeit paßt aber nicht unbedingt in die Zeitplanung des Pflegedienstes hinein. Daher ist es denkbar, daß die eingehenden Leistungsangebote nach der Zeit, wann die Leistungen erbracht werden können ausgewählt werden. Im nächsten Schritt des Koordinierungsablaufes wählt der Koordinator mit dem Kunden einen Pflegedienst aus den eingegangenen Leistungsangeboten aus. Dieser erhält daraufhin genau wie bei der normalen Koordinierung die gesamten Daten des Kunden, also die Kundenanfrage. Alle weiteren Schritte, wie das Erstgespräch und der Vertragsabschluß, bleiben auch hier dem Dienst überlassen.

Der in Abbildung 4-2 dargestellte Ablauf der softwareunterstützten Koordinierung von Pflegedienstleistungen kann in die Dokumente *Kundenanfrage*, persönliches *Leistungsprofil*, *Leistungsanfrage* und *Leistungsangebot* aufgeteilt werden. Diese Dokumente haben die Aufgabe, die für eine Koordinierung benötigten Datenfelder zusammen zu fassen, welche die Grundlage für die Erstellung der Datenobjekte des Prototyps bilden.

Darüber hinaus können sie auch als Grundlage für spätere Workflow-Ansätze verwendet werden.

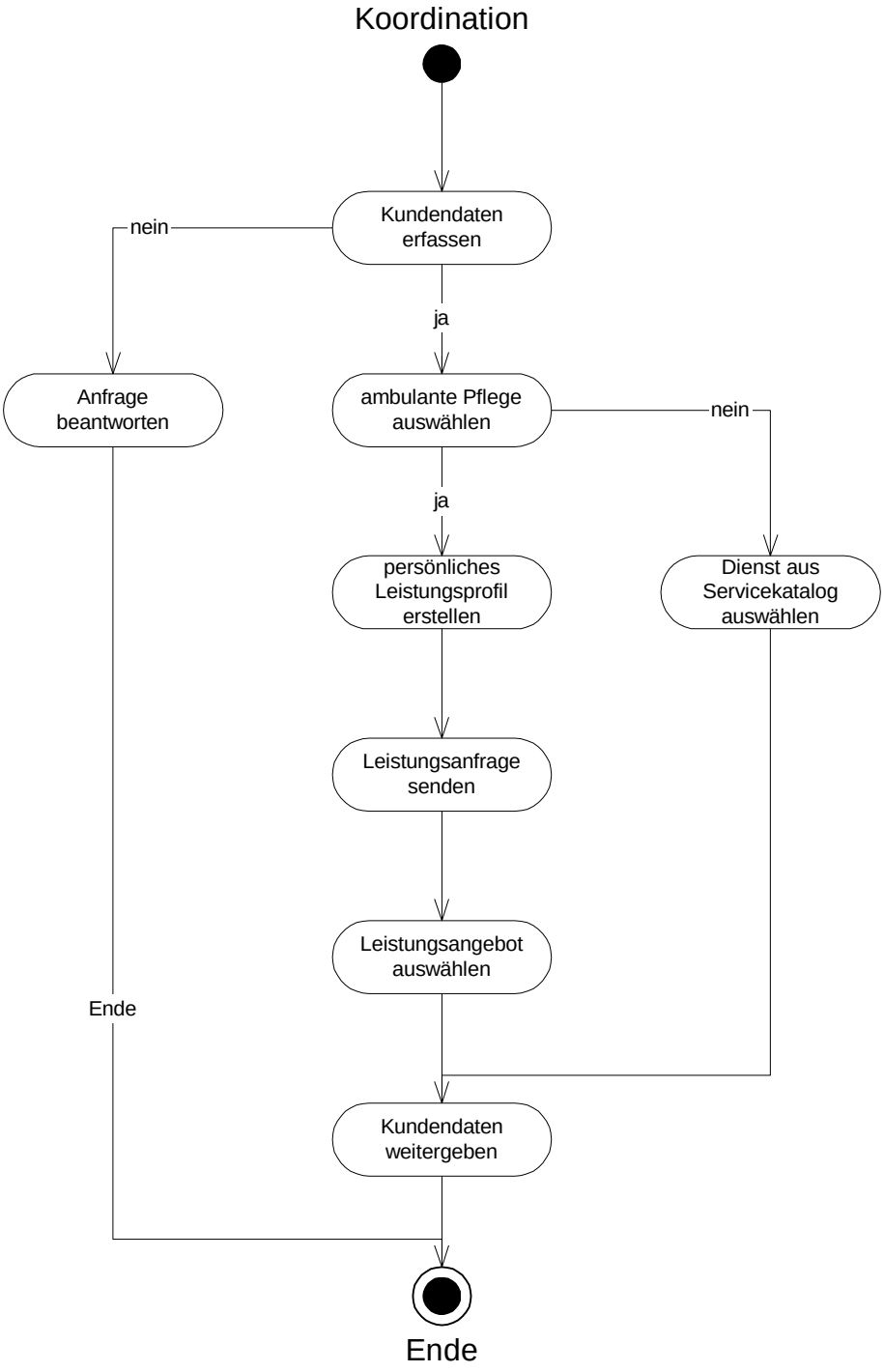


Abbildung 4-2 Soll-Ablauf einer Koordination

4.3 Kundenanfrage

Das erste Dokument, das im Laufe einer Koordinierung ausgefüllt wird, ist die Kundenanfrage. Sie wird ausgefüllt, wenn ein Kunde eine Anfrage an den Koordinator stellt, die nicht sofort beantwortet werden kann und somit weitere Arbeitsschritte erfordert. Die Kundenanfrage kommt im softwareunterstützten sowie im nicht softwareunterstützten Konzept der Koordinierung vor und hat die Aufgabe die Stammdaten des Kunden sowie Angaben über Leistungswünsche und Versicherungssituation aufzunehmen.

4.3.1 Daten in der Kundenanfrage

Das Dokument *Kundenanfrage* muß die folgenden Daten enthalten:

- Nachname
- Vorname
- Straße
- Hausnummer
- PLZ
- Ort
- Geburtsdatum
- Familienstand
- Nationalität
- Konfession
- Bezeichnung der Kranken- oder Pflegekasse
- Mitgliedsnummer des Kunden in der Kranken- oder Pflegekasse
- Namen behandelnder Ärzte
- Angaben über die Leistungsträger der ambulanten Versorgung (SGB V, BSHG(Bundes Sozialhilfe Gesetz), SGB XI, privat)
- Bemerkung (Freitext)

4.4 Persönliches Leistungsprofil

Das persönliche Leistungsprofil wird verwendet, um die Leistungswünsche und Bedürfnisse des Kunden für die ambulante Pflege zu ermitteln. Außerdem ist es damit

möglich, dem Kunden einen groben Überblick über die entstehenden Kosten zu geben. Auch dieses Dokument kommt in beiden Ansätzen zum Einsatz.

4.4.1 Daten im persönlichen Leistungsprofil

Das persönliche Leistungsprofil enthält folgende Daten:

- Die vom SGB XI vorgegeben Leistungskomplexe
- Den Preis jedes Leistungskomplexes
- Angaben, an welchem Wochentag ein Leistungskomplex benötigt wird
- Angaben, zu welchen Tageszeiten an jedem Wochentag ein Leistungskomplex benötigt wird
- Angaben, wie oft ein Leistungskomplex insgesamt pro Woche benötigt wird
- Die Summe der Kosten aller gewählten Leistungen

4.5 Leistungsanfrage

Das Dokument *Leistungsanfrage* ist eine Kombination der Dokumente *Kundenanfrage* und *persönliches Leistungsprofil*. Es wird zur Nachfrage der Verfügbarkeit der Leistungen bei der Koordinierung von ambulanten Pflegediensten verwendet und kommt nur im softwareunterstützten Konzept zum Einsatz.

4.5.1 Daten in der Leistungsanfrage

Das Dokument *Leistungsanfrage* enthält zum einen die gesamten Daten des persönlichen Leistungsprofils und zum anderen die folgenden Daten des Kunden:

- Straße
- Ort
- Familienstand
- Religion
- Nationalität
- Pflegestufe

Ein Teil der Stammdaten des Kunden wurde weggelassen, um eine Anonymisierung zu erreichen.

4.6 Leistungsangebot

Das Leistungsangebot ist das Dokument, welches die an der Versorgung eines Kunden interessierten Dienste an den Koordinator zurückschicken. Es soll hauptsächlich eine verbindliche Auskunft darüber geben, ob ein Dienst in der Lage ist, einen Kunden zu versorgen. Auch dieses Dokument ist nur im Sollkonzept zu finden.

4.6.1 Daten im Leistungsangebot

Das Leistungsangebot eines Dienstes enthält nur sehr wenige Daten. Dies sind:

- Name des Dienstes
- Nummer der Leistungsanfrage, auf die sich das Angebot bezieht
- Bemerkung

4.7 Anforderungen an den Softwareprototyp

Der im Laufe dieser Arbeit entstehende Softwareprototyp soll neben der Umsetzung des im Soll-Konzept beschriebenen Koordinierungsablaufs auch die Beratung von Pflegebedürftigen und deren Angehörigen unterstützen. Daraus ergeben sich im Groben folgende Anwendungsfälle für das Programm:

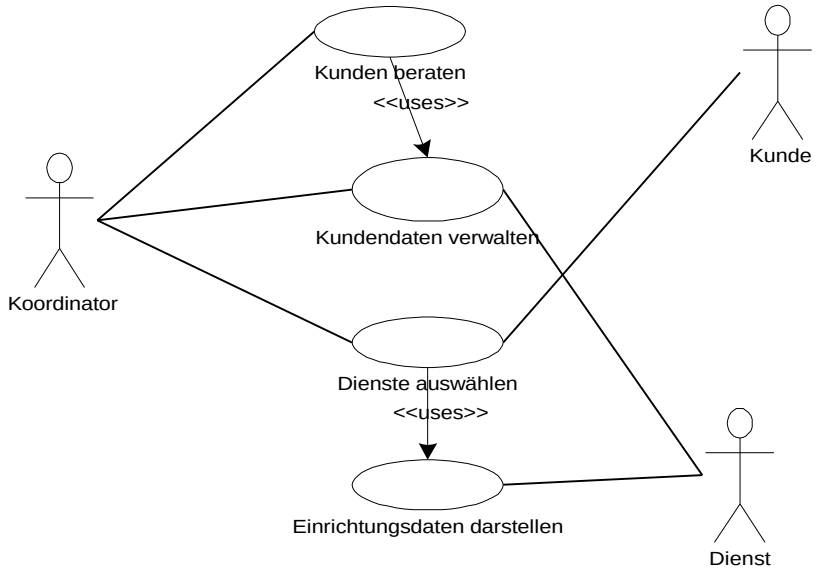


Abbildung 4-3 UseCase Diagramm 1 des Softwareprototyps

Wie das UseCase Diagramm in Abbildung 4-3 darstellt, sollen drei Akteure mit dem Programm arbeiten. Dies sind erstens der Koordinator, welchen das Programm bei der Beratung der Kunden, bei der Verwaltung der Kundendaten sowie bei der Auswahl der Dienste unterstützen soll. Als zweites die Kunden, welche die Software zusammen mit dem Koordinator benutzen sollen um z.B. einen Dienst auszuwählen. Den dritten Akteur bildeten die Dienste, welche am Koordinierungsprozeß teilnehmen. Diese sollen bei der Darstellung ihrer Einrichtungsdaten sowie bei der Verwaltung der Kundendaten unterstützt werden.

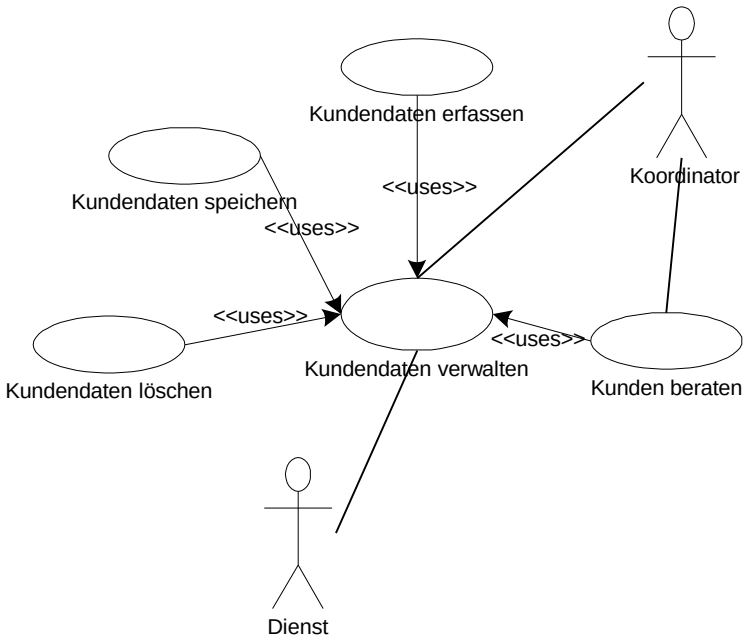


Abbildung 4-4 UseCase Diagramm 2 des Softwareprototyps

Abbildung 4-4 stellt dar, daß sich der Anwendungsfall *Kundendaten verwalten* noch durch die Anwendungsfälle *Kundendaten löschen*, *Kundendaten erfassen*, *Kundendaten speichern* sowie *Kunden beraten* verfeinern läßt.
Außerdem kann das UseCase Diagramm 1 des Softwareprototyps um den Anwendungsfall *Daten senden* erweitert werden, der die Möglichkeit Daten zwischen dem Koordinator und den einzelnen Diensten zu versenden darstellt. Abbildung 4-5 zeigt, daß der allgemeine UseCase *Daten senden* wiederum durch die Anwendungsfälle *Kundendaten senden*, *Leistungsanfrage senden* sowie *Leistungsangebot senden* erweitert wird. Darüber hinaus sollte das Programm noch Funktionen für die Erstellung

eines persönlichen Leistungsprofils sowie für die Erfassung der Einrichtungsdaten enthalten.

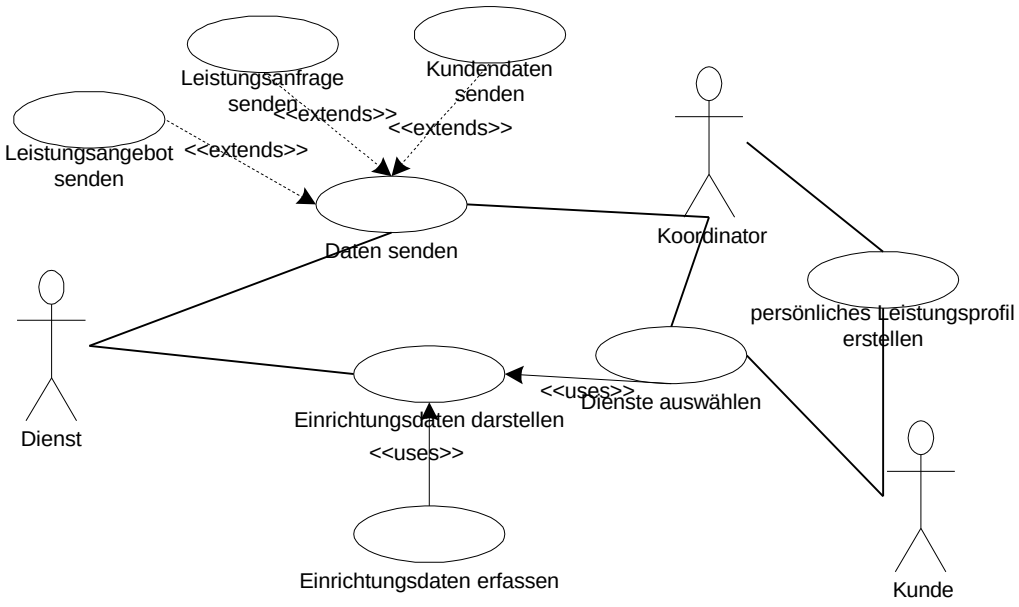


Abbildung 4-5 UseCase Diagramm 3 des Softwareprototyps

5 Der Servicekatalog

Der als Bestandteil des Softwareprototyps entwickelte Servicekatalog ist als zentrale Datenbasis für die Beratung sowie für die Koordinierung anzusehen.
Seine Hauptaufgabe besteht darin, die Informationen über die dem System angeschlossenen Einrichtungen in einer flexiblen und einfachen Form darzustellen. Darüber hinaus soll er den Diensten die Möglichkeit geben, sich auf einfache, ansprechende Weise zu präsentieren.
Außerdem soll der Koordinator schnell und einfach auf die von ihm gesuchten Informationen zugreifen können.
Ein weiterer Grund für die Notwendigkeit des Servicekatalogs als zentrale Datenbasis ist, daß der Koordinator eine einheitliche Repräsentation aller an das System angeschlossenen Dienste benötigt, um eine objektive Beratung und Koordinierung durchführen zu können. Der Aufbau eines Servicekatalogs für einen Stadtteil oder Bezirk ist zwar mit erheblichem Aufwand verbunden und dürfte auch auf einigen Widerstand stoßen, da er einen direkten Vergleich der Angebote und Preise von verschiedenen Diensten ermöglicht. Sein Nutzen für die Beratung und Koordinierung ist aber leicht ersichtlich.
Ein einheitlicher Servicekatalog hat den Vorteil, daß dem Koordinator jetzt vollständige Informationen über jeden Dienst zur Verfügung stehen. Die an ein solches System angeschlossenen Dienste sollten schon aus Wettbewerbsgründen daran interessiert sein, daß ihre Daten vollständig und aktuell sind, um nicht gegenüber Konkurrenten benachteiligt zu werden. Dies ermöglicht dem Koordinator auf ständig aktuelle Informationen zuzugreifen und für seine Kunden das optimale Leistungspaket zusammenzustellen. Außerdem haben aktuelle Daten auch für die Dienste den Vorteil, daß potentielle Kunden sowie der Koordinator direkt über neue Angebote oder Besonderheiten informiert werden.
Die Struktur der Daten des Servicekatalogs muß fest vorgegeben werden, da zuviel Freiheit bei der Gestaltung der Datenstruktur jeden Dienst dazu motivieren würde, seine eigenen Strukturen abzubilden, was sehr schnell zu Unübersichtlichkeit führt. Diese Unübersichtlichkeit würde ein effektives Suchen und Navigieren in diesem Katalog verhindern.

5.1 Daten im Servicekatalog

Da sich der Servicekatalog wie auch der gesamte Prototyp in den Rahmen der anderen Prototypen des SWIFT Projektes einfügen soll, wurde bei dessen Erstellung darauf geachtet, auf bestehende Datenstrukturen, Modelle und Klassen zurückzugreifen.
Im Mittelpunkt des Servicekatalogs stehen die Dienste mit ihren Kontaktinformationen und Leistungen. Daher sind für den Servicekatalog folgende Daten jedes Dienstes relevant:

- Der Name des Dienstes
- Eine Kurzbeschreibung des Dienstes
- Eine Beschreibung des Dienstes
- Der Typ des Dienstes (Mahlzeitendienst, Hausnotrufdienst oder ambulanter Pflegedienst)
- Name des Trägers des Dienstes
- Der Leistungskatalog des Dienstes
- Sonstige Informationen

Darüber hinaus werden zu jedem Dienst noch beliebig viele Adressen, E-Mail Adressen, Telefonnummern gespeichert. Dieses Konzept stammt aus dem bereits existierenden SWIFT Personen-Modul und wurde um die Möglichkeit zusätzlich beliebig viele Bankverbindungen und Ansprechpartner speichern zu können erweitert. Der Leistungskatalog kann für jeden Dienst mit Leistungskatalog-Modul von SWIFT erstellt und verändert werden.

Alle oben aufgeführten Daten werden im Prototyp in einer Microsoft Access97 Datenbank abgelegt. Abbildung 5-1 zeigt das Datenmodell des Servicekatalogs. Die Haupttabelle in diesem Datenmodell ist die Tabelle *Service*. Sie enthält neben den Datenfeldern für den Namen, die Kurzbeschreibung, die Beschreibung und Sonstige Informationen auch ein Datenfeld, welches eine eindeutige ID für jeden Dienst enthält. Mit Hilfe dieser ID können dem Dienst seine Adressen, Telefonnummern, Bankverbindungen, E-Mail Adresse und Kontaktpersonen aus den Tabellen *Adress*, *PhoneNumber*, *BankAccount*, *EMail* und *Contact* zugeordnet werden. Darüber hinaus enthält die Tabelle *Service* noch zwei Fremdschlüssel, mit denen jedem Dienst der Typ des Dienstes aus der Tabelle *ServiceTypes* und ein Träger aus der Tabelle *Facility*

zugeordnet werden können. Außerdem wird noch die ID des mit dem Leistungskatalog-Modul erstellten Leistungskatalogs gespeichert. Der Leistungskatalog selbst wird in der Datenbank des Leistungskatalog-Moduls gespeichert. Der Zugriff auf die Datenbanken mit Java erfolgt über die ODBC Schnittstelle.
Neben den in der Datenbank gespeicherten Informationen des Servicekatalogs wird noch zu jedem Dienst eine HTML Seite im Dateisystem gespeichert. Mit dieser Seite hat jeder Dienst die Möglichkeit, seine Präsentation zu beeinflussen. HTML wurde aus dem Grund der sehr einfachen Handhabung sowie der großen Verbreitung und Akzeptanz gewählt. Zudem bieten HTML Seiten sehr einfach die Möglichkeit Bilder und Graphiken zu verwenden, was oft ansprechender wirkt als eine reine Textdarstellung.

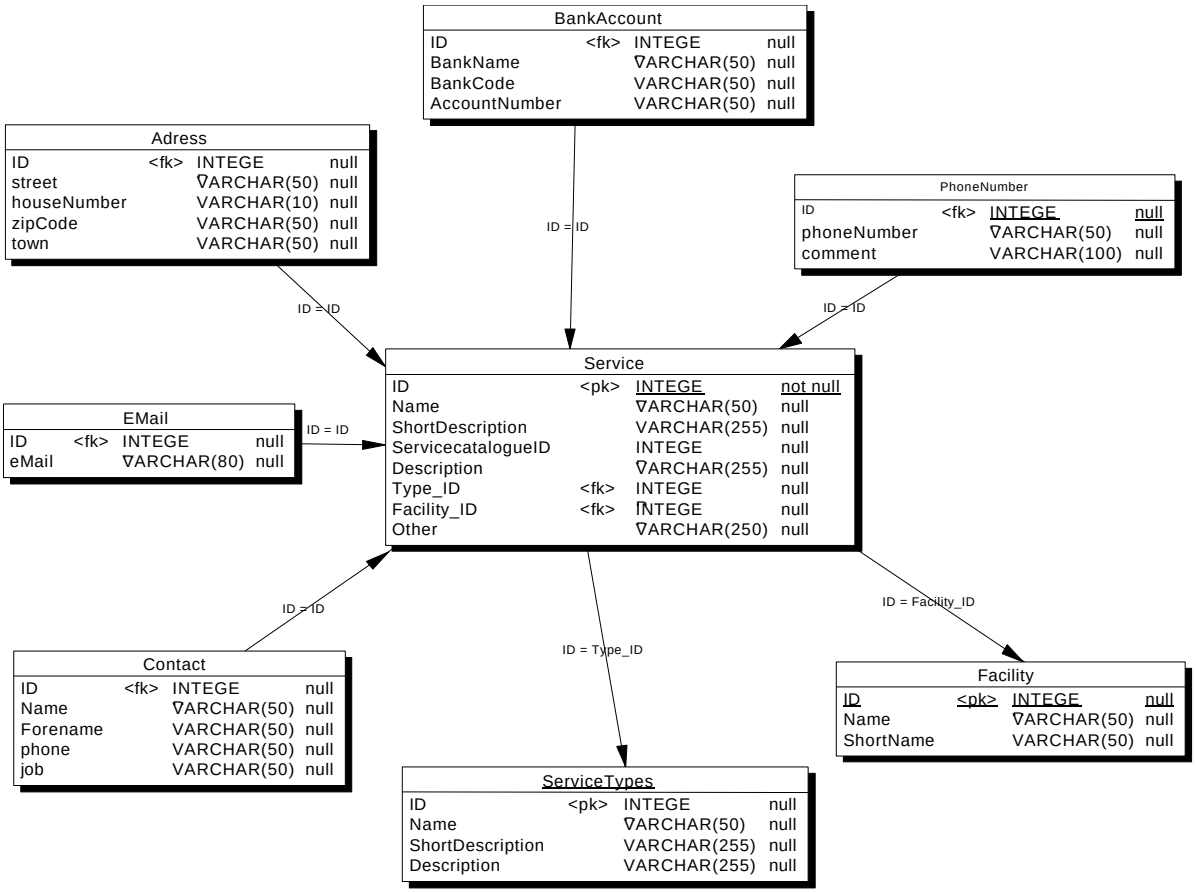


Abbildung 5-1 Datenmodell des Servicekatalogs

5.2 Darstellung des Servicekatalogs

Die Darstellung des Servicekatalogs geschieht mit Hilfe einer Baumstruktur. Dies hat zum Vorteil, daß ein schnelles und einfaches Navigieren in dieser Baumstruktur möglich ist, und daß sich sehr leicht Teilbäume ausgliedern und für andere Zwecke verwenden lassen.
Darüber hinaus haben Baumstrukturen den Vorteil, daß man sie sehr leicht erweitern kann.
Jeder Knoten dieses Baumes repräsentiert einen Teil der Daten des Datenmodells des Servicekatalogs.

5.2.2 Beschreibung der Baumstruktur

Die Baumstruktur des Servicekatalogs besteht aus fünf Ebenen. Die erste Ebene oder Wurzel dieser Struktur bildet der übergeordnete Begriff *Alle Dienste*. Dieser dient nur als Sammelbegriff und enthält keine weiteren Informationen.
Auf der zweiten Ebene des Baumes befinden sich die verschiedenen Diensttypen. Dies sind in diesem Prototyp die drei untersuchten Diensttypen Hausnotrufdienste, Mahlzeitendienste sowie ambulante Pflegedienste. Die Knoten dieser Ebene stellen allgemeine Informationen über die einzelnen Diensttypen bereit wie z.B. eine allgemeine Beschreibung über Hausnotrufdienste, welche Leistungen die Dienste dieses Typs normalerweise anbieten oder worauf bei der Auswahl eines Dienstes besonders zu achten ist.
Unterhalb jedes Diensttyps, auf der dritten Ebene, sind alle an das System angeschlossenen Dienste eines Typs zu finden. Dabei stellen die Knoten der dritten Ebene allgemeine Informationen über jeden einzelnen Dienst bereit.
Auf der vierten Ebene, unterhalb der einzelnen Dienste, befinden sich zwei weitere Baumknoten, sowie ein Blatt, die alle weiteren für den Koordinator oder die Kunden relevanten Daten des Dienstes darstellen. Dies sind die Kontaktinformationen, der Leistungskatalog sowie Sonstige Informationen. Der Knoten *Kontaktinformationen* enthält die Ansprechpartner des jeweiligen Dienstes, sowie vier Blätter der fünften Ebene, welche die Adressen, E-Mail Adressen, Telefonnummern und Bankverbindungen darstellen. Der Knoten *Leistungskatalog* enthält den mit dem Leistungskatalog-Modul erstellten Leistungskatalog jedes Dienstes. Dieser wird

wiederum als Baumstruktur unterhalb des Leistungskatalog Knotens eingebunden. Das Blatt *Sonstiges* stellt Raum für Freitext bereit, der z.B. für Urlaubshinweise oder besondere Angebote genutzt werden kann.
Da die gesamten Daten sowie die Baumstruktur des Servicekatalogs direkt aus einer Datenbank ausgelesen werden, sind selbst strukturelle Änderungen am Katalog sehr leicht möglich. So ist es z.B. durch einfaches Einfügen eines Datensatzes in die Tabelle *ServiceTypes* möglich einen komplett neuen Typ von Diensten mit in den Katalog aufzunehmen. Die Darstellung des Baumes ist also von den Daten unabhängig.
Abbildung 5-2 zeigt eine Übersicht über die gesamte Baumstruktur des Servicekatalogs.

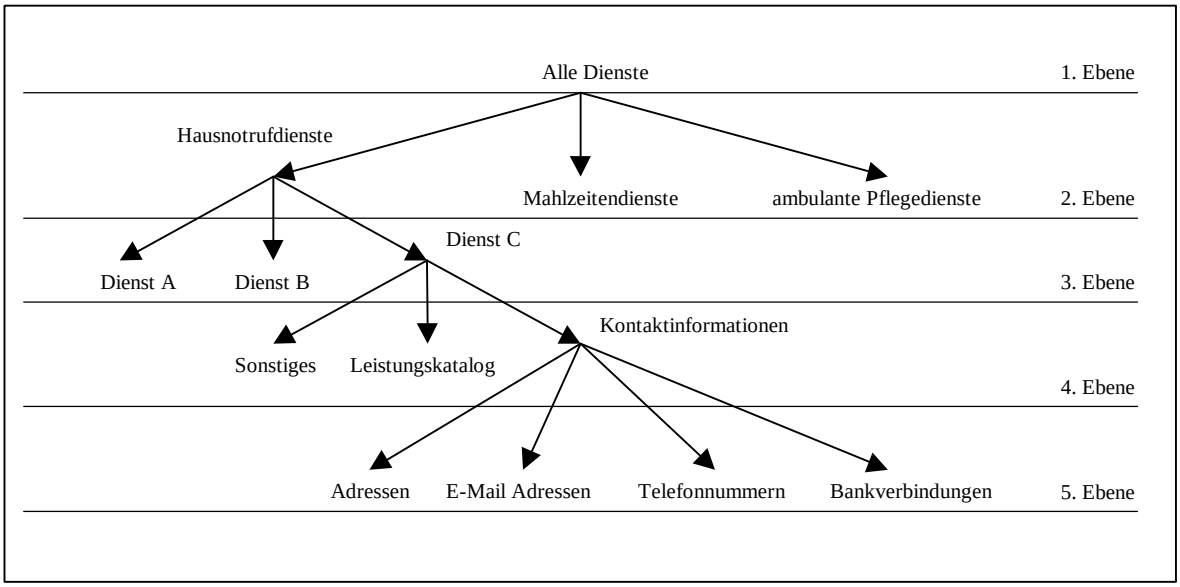


Abbildung 5-2 Schema der Baumstruktur des Servicekatalogs

5.3 Umsetzung des Servicekatalogs in Java

Das Ziel dieser Umsetzung ist es, den Servicekatalog innerhalb des Softwareprototyps darzustellen.
Für diese Darstellung der Baumstruktur in Java wurde die Standardklasse *javax.swing.JTree* gewählt. Diese stellt die Grundfunktionalität eines Baumes in Java zur Verfügung. Die Knoten in einem *JTree* bestehen normalerweise aus Objekten vom Typ *javax.swing.tree.DefaultMutableTreeNode*. Dies sind Standardobjekte, die entweder Blätter oder Knoten des Baumes repräsentieren können. Um die speziellen Anforderungen des Servicekatalogs zu erfüllen, wurden zwei Datenklassen entwickelt, die von *DefaultMutableTreeNode* abgeleitet wurden.

Dies sind die Klasse *com.server.ServiceItemNode* und die Klasse *com.server.ServiceNode*. Sie erweitern die Eigenschaften ihrer Oberklasse, sich in einem *JTree* verwenden zu lassen, um einige zusätzliche Datenfelder, die für das Anzeigen der Informationen über einen Dienst erforderlich sind. Die Klasse *ServiceItemNode* stellt einen einfachen Baumknoten oder ein Blatt zur Verfügung und erweitert ihre Oberklasse nur um einige Textfelder, die den Inhalt des Knotens oder Blattes aufnehmen. Sie wird im Servicekatalog für die Darstellung der Ebenen 1,2,4 und 5 verwendet. Die Klasse *ServiceNode* wird für die Repräsentation der Dienste auf Ebene 3 im Katalog verwendet. Diese Klasse enthält ein Objekt vom Typ *com.server.Service*, welches die eigentlichen Daten eines Dienstes enthält. Außerdem enthält es mehrere Objekte der Klasse *ServiceNode* und baut aus ihnen die gesamte Struktur des Baumes auf, die unterhalb der dritten Ebene liegt. Dadurch ist es möglich z.B. durch Hinzufügen eines Objektes des Typs *ServiceNode* einen neuen Dienst mit allen zu ihm gehörenden Angaben in den Baum zu integrieren.

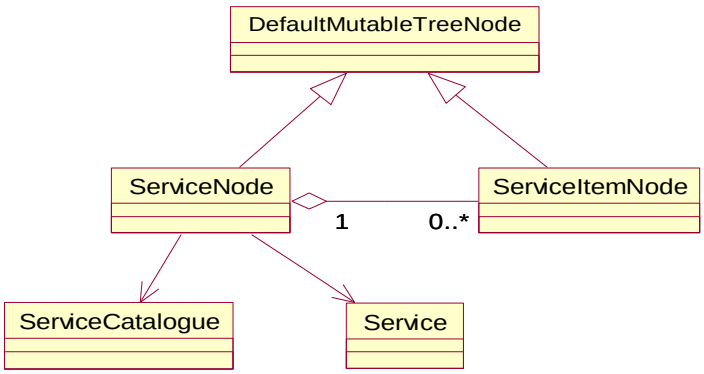


Abbildung 5-3 Klassendiagramm des Servicekatalogs

Abbildung 5-3 zeigt, daß jeder ServiceNode zudem ein Objekt vom Typ *servicecatalogue.ServiceCatalogue* enthält. Dieses Objekt enthält wiederum den mit dem Leistungs- und Tätigkeitskatalog-Modul erstellten Leistungskatalog des Dienstes. Da die Darstellung dieses Leistungskatalogs ebenfalls auf einem *JTree* basiert, kann dieser automatisch unterhalb des Leistungskatalog-Knotens in den Servicekatalog eingefügt werden.

6 Beschreibung des Programms

Als Prototyp für die Darstellung der Koordinierungsabläufe wurden ein Server- und ein Clientprogramm entwickelt. Dies sind der Koordinationsserver, welcher die Abläufe und Tätigkeiten auf der Seite des Koordinators unterstützt und der Koordinationsclient, der die Möglichkeiten der einzelnen Dienste repräsentiert. Diese beiden Programme kommunizieren über eine Java RMI Verbindung miteinander. Die Kommunikation ist so ausgelegt, daß ein Server mit beliebig vielen Clientprogrammen gleichzeitig kommunizieren kann. Für die Erstellung der Masken wurden die *Swing* Klassen von Java verwendet.
Dieses Kapitel beschreibt die Implementation der beiden Programmteile in Java.

6.1 Beschreibung des Koordinationsservers

Die Funktionen des Prototyps sind möglichst nah an den in Kapitel 4 beschriebenen Soll-Abläufen für eine Koordinierung angelehnt. Die Hauptfunktionen des Koordinationsservers sind auf die fünf Masken Servicekatalog, Kundenanfrage, Hausnotrufdienste, Mahlzeitendienste und ambulante Pflegedienste aufgeteilt. Dabei hat die Maske *Servicekatalog* die Aufgabe, eine Gesamtansicht des Servicekataloges wie er in Kapitel 5 beschrieben wird darzustellen. Die Maske *Kundenanfrage* dient zum Eingeben der Kundendaten sowie zum Auswählen der gewünschten Leistungen. Außerdem gibt es für jeden der drei für die Koordinierung zur Auswahl stehenden Diensttypen eine weitere Maske. Diese stellen die für die Auswahl eines Dienstes des jeweiligen Typs benötigten Daten sowie die Funktionalität einen Kunden zu koordinieren bereit. Darüber hinaus stellt der Koordinierungsserver die RMI Server Funktionen zur Verfügung, welche vom Koordinierungsclient benutzt werden, um z.B. Kundendaten oder Leistungsanfragen zu empfangen. Die fünf Hauptmasken sind in Form von Karteikarten im Programmfenster des Koordinationsservers angeordnet. Das hat den Vorteil, daß jederzeit ein wechseln der Maske möglich ist, und daß die Hauptfunktionen des Programmes leicht zu überblicken sind und nicht erst in einer Menüstruktur gesucht werden müssen. Die Masken werden durch einige Dialoge in ihrer Funktionalität unterstützt.
Nachfolgend werden die einzelnen Masken mit ihren Funktionen und Daten näher erklärt.

6.1.3 Servicekatalog

Für die Beratung der Kunden wird eine Gesamtansicht aller an das System angeschlossenen Dienste benötigt. Diese wird im Softwareprototyp durch die Maske *Servicekatalog* umgesetzt. Sie ist in eine rechte und eine linke Seite unterteilt. Auf der linken Seite wird die gesamte Baumstruktur des Servicekatalogs angezeigt. Nach dem Programmstart sind nur die beiden ersten Ebenen des Baumes sichtbar. Alle weiteren Ebenen können durch Doppelklicken des entsprechenden Eintrages oder durch Einfachklicken der Pluszeichen neben den Einträgen sichtbar gemacht werden. Das Verhalten dieses Baumes entspricht größtenteils dem des Windows-Explorers. Ein Einfachklick auf einen Eintrag bewirkt, daß die entsprechenden Informationen zu diesem Eintrag auf der rechten Seite der Servicekatalog-Maske angezeigt werden. Die einzelnen hier angezeigten Informationen werden in Kapitel 5 beschrieben. Links neben den Einträgen für die einzelnen Dienste (Ebene 3) befinden sich Ikonen in Form von Bällen.

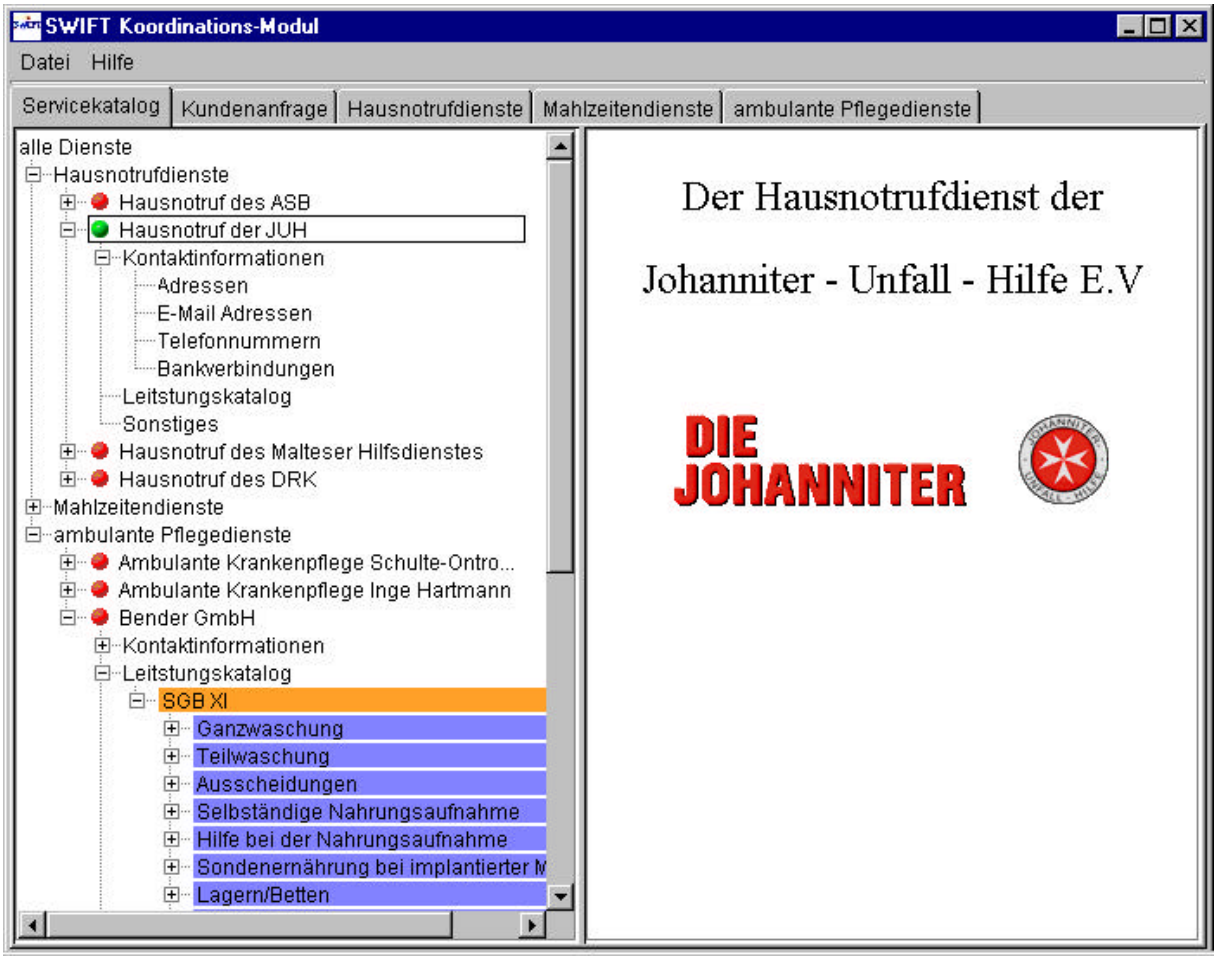


Abbildung 6-1 Maske Servicekatalog

Diese Bälle sind normalerweise rot und wechseln ihre Farbe zu grün, wenn sich der entsprechende Dienst mit seinem Clientprogramm an den Koordinationsserver anmeldet. Dadurch hat der Koordinator eine Übersicht über die Dienste, die zur Zeit mit dem Programm arbeiten. Wenn der Dienst sich wieder abmeldet, wird der Ball neben seinem Eintrag im Servicekatalog wieder rot. Siehe Abbildung 6-1.
Da in Kapitel 5 bereits die für die Darstellung des Servicekatalogs verwendeten Klassen beschrieben wurden, werden an dieser Stelle nur die Datenklassen des Servicekatalogs beschrieben. Die Hauptaufgabe des Servicekatalogs ist die Darstellung der Informationen über die dem System angeschlossenen Dienste. Diese Informationen werden in Objekten der Klasse *person.Service* gespeichert. Jeder Dienst wird im Servicekatalog durch ein Objekt dieses Typs repräsentiert. Die Objekte der Klasse *Service* haben folgende Attribute:

- Name
- Typ des Dienstes
- Beschreibung
- Kurzbeschreibung
- Sonstige Informationen
- Name des Trägers des Dienstes
- Leistungskatalog des Dienstes

Außerdem werden jedem Service-Objekt noch beliebig viele Adressen, E-Mail Adressen, Telefonnummern, Bankverbindungen und Ansprechpartner zugeordnet. Abbildung 6-2 zeigt, daß diese Daten durch Objekte der Klassen *person.Adress*, *person.EMail*, *person.PhoneNumber*, *person.BankAccount* und *person.Contact* repräsentiert werden. Der Großteil der Klassenstruktur stammt aus dem SWIFT Personen-Modul und wurde für die Verwendung im Koordinationsserver erweitert.

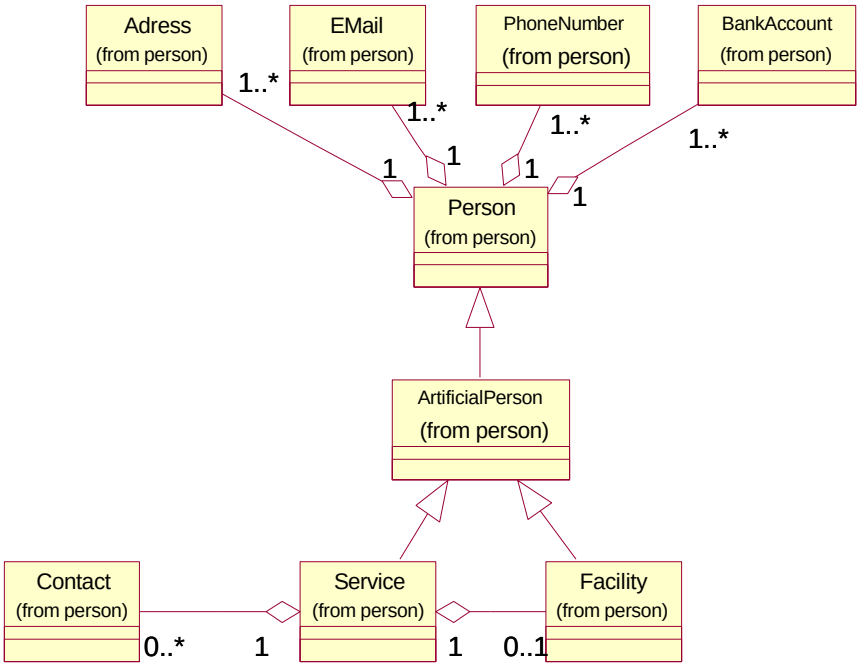


Abbildung 6-2 Datenklassen des Servicekatalogs

6.1.4 Kundenanfrage

Eine der Hauptaufgaben des Programmes, die Erfassung der Kundendaten, wird durch die in der Maske *Kundenanfrage* implementierten Funktionen erfüllt. Sie enthält die in Kapitel 4 erarbeiteten und beschriebenen Datenfelder, welche die für die Koordinierung eines Kunden erforderlichen Daten aufnehmen. Die Kundenanfrage basiert auf einem Objekt vom Typ *person.CareClient*. Dieser Objekttyp stammt aus dem Personen-Modul von SWIFT und wurde für diese Arbeit erheblich erweitert und verändert. Zu diesen Erweiterungen zählt das Hinzufügen von Attributen sowie von Funktionen, die z.B. das Löschen und Speichern dieser Objekte in einer Datenbank ermöglichen. Außerdem wurden noch einige Verknüpfungen zu bereits bestehenden oder neu erstellten Objekttypen hinzugefügt, so daß ein CareClient-Objekt nun über folgende Attribute verfügt:

- Name
- Vorname
- Familienstand
- Nationalität

- Geburtsdatum
- Geburtsort
- Religion
- Bemerkung

Darüber hinaus sind jedem CareClient-Objekt noch beliebig viele Adressen, E-Mail Adressen, Telefonnummern, Bankverbindungen, Ärzte, Versorgungseinrichtungen sowie die Dienstleistungen, die nachgefragt wurden, zugeordnet. Diese Daten werden durch Objekte der Klassen *person.Adress*, *person.EMail*, *person.PhoneNumber*,

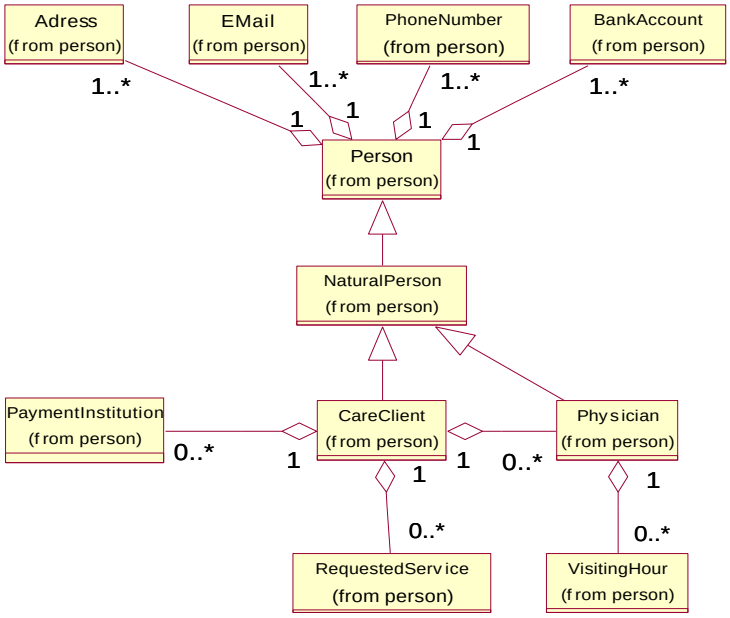


Abbildung 6-3 Datenklassen der Kundenanfrage

person.BankAccount, *person .Physician*, *person.PaymentInstitution* sowie *person.RequestedService* repräsentiert. Das Klassendiagramm in Abbildung 6-3 zeigt eine Übersicht über diese Klassen und deren Verknüpfungen.

Für die Darstellung der Daten des CareClient-Objekts in der Maske *Kundenanfrage* werden verschiede graphische Elemente von Java verwendet. Am oberen Rand dieser Maske befindet sich eine Schaltflächenleiste oder Toolbar mit folgenden vier Schaltflächen:

- Kundendaten suchen:

  Die Kundendaten suchen-Schaltfläche soll benutzt werden, wenn ein Kunde den Koordinator wiederholt aufsucht, um die bereits eingegebenen Daten des Kunden in

der Datenbank zu suchen. Bei Betätigung dieser Schaltfläche wird ein Dialog angezeigt, der einerseits die Angabe von Suchparametern in den Feldern *Name*, *Vorname* und *Geburtsdatum* erlaubt und andererseits eine Liste mit den gefundenen Einträgen aus der Datenbank anzeigt. Siehe Abbildung 6-4. Durch Betätigen der Schaltfläche *Suchen* wird ein neuer Suchvorgang ausgelöst. Danach kann aus der Liste mit den Ergebnissen einfach der gesuchte Kunde ausgewählt werden und dessen Daten mit Hilfe der Schaltfläche *Daten übernehmen* in die Kundenanfrage übernommen werden.

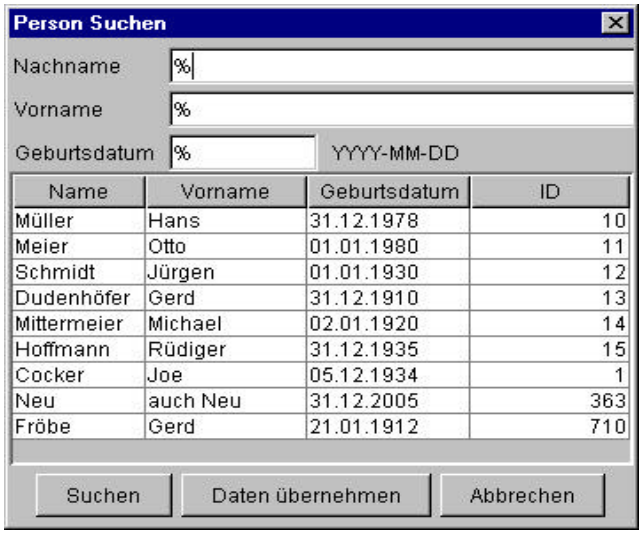


Abbildung 6-4 Person suchen-Dialog

- Dienste:

  Die Schaltfläche *Dienste* ermöglicht es dem Koordinator, die Diensttypen auszuwählen oder zu ändern, über welche der aktuell in der Kundenanfrage angezeigte Kunde informiert und an die er vermittelt werden möchte. Bei Betätigung dieser Schaltfläche erscheint ein Dialog, in dem die gewünschten Diensttypen gewählt werden können. Hier stehen zur Zeit nur die drei für diese Arbeit untersuchten Diensttypen Hausnotrufdienste, Mahlzeitendienste sowie ambulante Pflegedienste zur Auswahl. Wie in Abbildung 6-5 zu sehen ist kann in diesem Dialog neben den Diensten noch der Leistungsträger der ambulanten Versorgung gewählt werden. Mit Betätigung der Speichern-Schaltfläche werden die gewählten Angaben den Kundendaten (dem *CareClient* Objekt) hinzugefügt. Eine weitere Aufgabe dieses Dialoges ist es, den Status eines Kunden anzuzeigen. Mit Status sind Informationen gemeint, die dem Koordinator z.B. anzeigen, ob der

Kunde mit dem Dienst, an den er vermittelt wurde, einen Vertrag abgeschlossen hat oder ob es dabei Probleme gab.

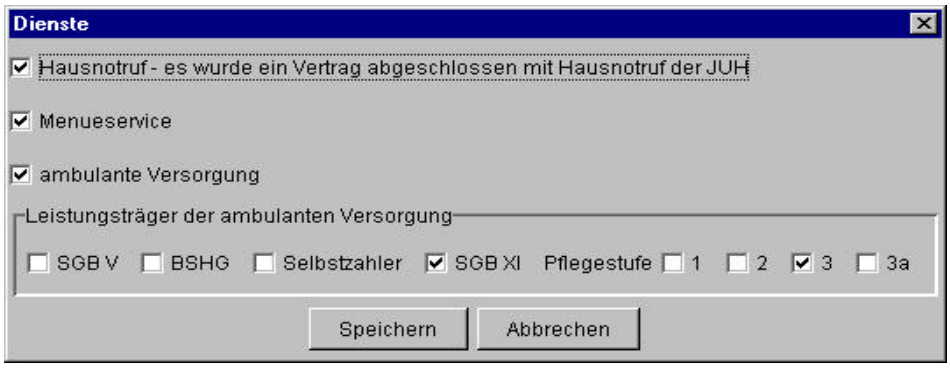


Abbildung 6-5 Dienste-Dialog

- Ärzte:

  Die Ärzte-Schaltfläche dient dazu, die in das Programm eingegebenen Ärzte einem Kunden zuzuordnen. Bei Betätigung dieser Schaltfläche wird ebenfalls ein Dialog geöffnet. Im oberen Teil dieses Dialoges befindet sich eine Liste, in der die dem Kunden zugeordneten Ärzte angezeigt werden. Wird in dieser Liste ein Arzt ausgewählt, werden in den Fünf darunter befindlichen Karteikarten die Stammdaten des gewählten Arztes angezeigt. Am unteren Ende des Dialoges befinden sich Schaltflächen, um Ärzte aus der Liste zu löschen, die Daten zu speichern oder um Ärzte hinzuzufügen. Siehe Abbildung 6-6. Bei Betätigung der Hinzufügen-Schaltfläche wird ein weiterer Dialog angezeigt, der den gleichen Aufbau und die gleichen Funktionen hat wie der Dialog *Person suchen*.

Abbildung 6-6 Ärzte-Dialog

- Kranken/Pflegekassen:

  Die Kranken/Pflegekassen-Schaltfläche hat die gleichen Funktionen wie die Ärzte-Schaltfläche. Sie dient dazu, die im Programm eingegebenen Kranken- oder Pflegekassen einem Kunden zuzuordnen.

Unterhalb der Toolbar in der Kundenanfrage befindet sich der Bereich für die Eingabe der persönlichen Daten des Kunden. Daran schließt sich ein Bereich für Kontaktinformationen an. Hier können die Adressen, E-Mail Adressen, Telefonnummern und Bankverbindungen des Kunden eingegeben werden. Dieser Kontaktinformationen-Bereich stammt aus dem SWIFT Personen-Modul und wird mit leichten Veränderungen an verschiedenen Stellen in diesem Programm verwendet. Unterhalb des Kontaktinformationen-Bereichs befindet sich ein größeres Freitextfeld, in dem Bemerkungen zum Kunden eingegeben werden können. Darunter sind noch einmal vier Schaltflächen angeordnet, die folgende Funktionen haben:

- Koordinierung:

  Die Schaltfläche *Koordinierung* überprüft bei Betätigung, welche Dienste für den aktuell angezeigten Kunden im Dienste-Dialog gewählt wurden und springt in Abhängigkeit von dieser Auswahl zur nächsten Maske. Dies kann entweder die Maske *Hausnotrufdienste*, *Mahlzeitendienste* oder *ambulante Pflegedienste* sein. Diese Funktionalität soll den Koordinator bei seinem Arbeitsfluß unterstützen, da dieser nicht darauf achten muß, welche Diensttypen der Kunde gewählt hat, sondern direkt an die Stelle im Programm dirigiert wird, an der der Koordinierungsprozeß weitergehen soll.
- neuer Kunde:

  Die Schaltfläche *neuer Kunde* löscht alle Eingabefelder und erstellt ein neues Objekt vom Typ *CareClient*. Das bedeutet, daß alle Eingaben und Änderungen in den Dialogen und in den Eingabefeldern, die seit der letzten Betätigung der Speichern-Schaltfläche durchgeführt wurden, gelöscht werden.
- Daten speichern:

  Mit dieser Schaltfläche speichert man das aktuelle CareClient-Objekt in der Datenbank.

- Daten löschen:

  Die Schaltfläche *Daten löschen* löscht das aktuell angezeigte CareClient-Objekt aus der Datenbank.

Abbildung 6-7 zeigt die Maske *Kundenanfrage*. Der normale Arbeitsablauf in dieser Maske sollte so aussehen, daß der Koordinator die Daten des Kunden aufnimmt oder aus der Datenbank lädt und diese verändert. Danach sollte die Schaltfläche *Koordinierung* betätigt werden. Daraufhin wird die Maske angezeigt, auf der die nächsten Arbeitsschritte für die Koordinierung durchzuführen sind.

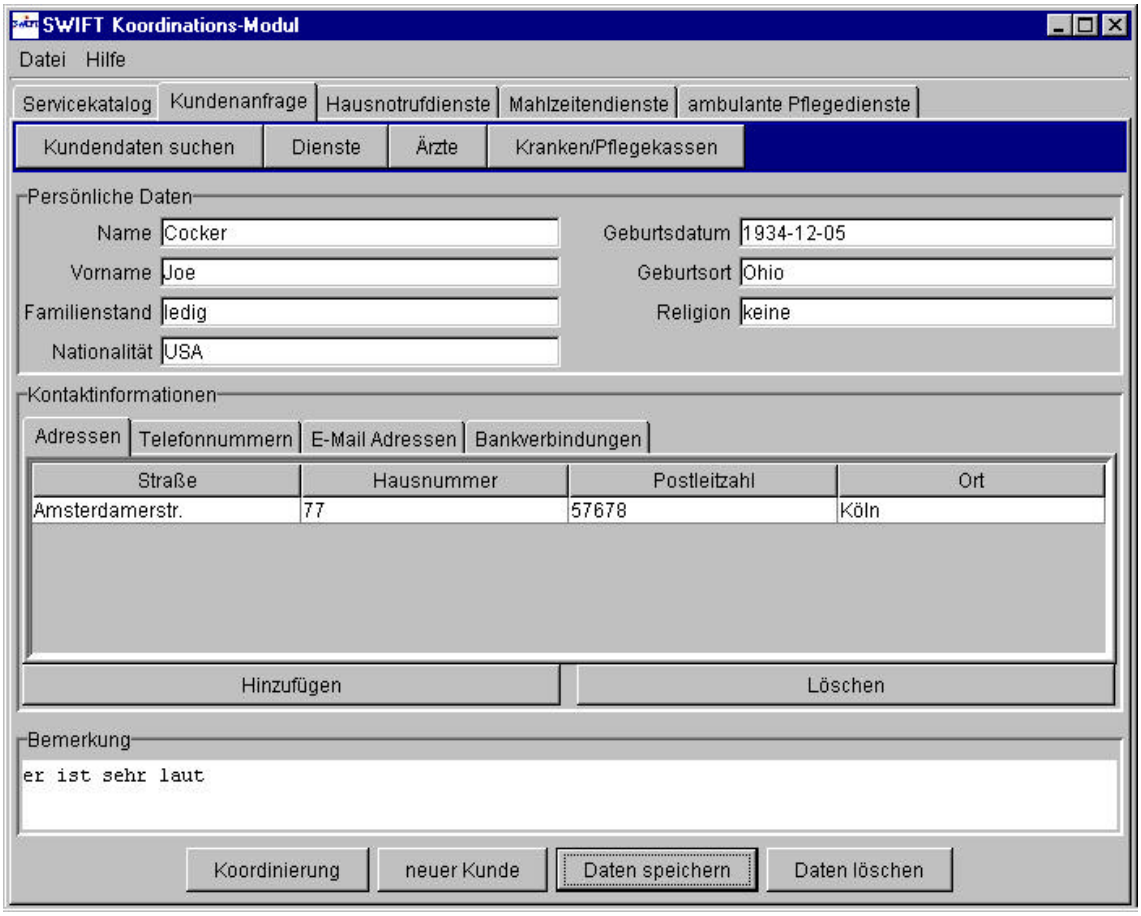


Abbildung 6-7 Maske Kundenanfrage

6.1.5 Hausnotrufdienste

Die Hausnotrufdienste Maske hat bei der Koordinierung die Aufgabe, dem Koordinator die Möglichkeit zu geben, einen Hausnotrufdienst auszuwählen. Dazu ist es nötig, daß dem Benutzer die Daten, welche die Auswahlentscheidung beeinflussen, zur Verfügung stehen. Die Darstellung dieser Daten wird über eine Teildarstellung des Servicekatalogs erreicht. In dieser Teildarstellung werden nur die Hausnotrufdienste angezeigt. Darüber hinaus steht die gesamte weitere Funktionalität des Servicekatalogs zur Verfügung. Der Aufbau dieser Maske ist dem der Servicekatalog-Maske sehr ähnlich. Auf der linken Seite befindet sich die Baumstruktur mit allen Hausnotrufdiensten, während auf der rechten Seite die Informationen zu den einzelnen Diensten in Abhängigkeit des in der Baumstruktur gewählten Dienstanbieters angezeigt werden. Darüber hinaus befinden sich am unteren Rand der Karteikarte zwei Schaltflächen. Siehe Abbildung 6-8.

- Auswählen:

  Die Schaltfläche *Auswählen* wird benutzt, um dem in der Baumstruktur ausgewählten Hausnotrufdienst die Daten des Kunden zu senden. Bei Betätigung dieser Schaltfläche wird ein Dialog angezeigt, der dem Benutzer mitteilt, welcher Dienstanbieter gewählt wurde, und welcher nach einer Bestätigung für das Senden der Daten fragt. Wird diese Abfrage mit ja beantwortet, werden die Daten des Kunden - also das aktuelle CareClient-Objekt - über die RMI Verbindung direkt an den gewählten Dienstanbieter gesendet.
- Weiter:

  Die Schaltfläche *Weiter* hat die Aufgabe, den Koordinator weiter durch den Koordinierungsprozeß zu führen, indem sie genau wie die Schaltfläche *Koordination* in der Maske *Kundenanfrage* überprüft, welche weiteren Dienste für eine Koordinierung ausgewählt wurden und anschließend die entsprechende Maske anzeigt.

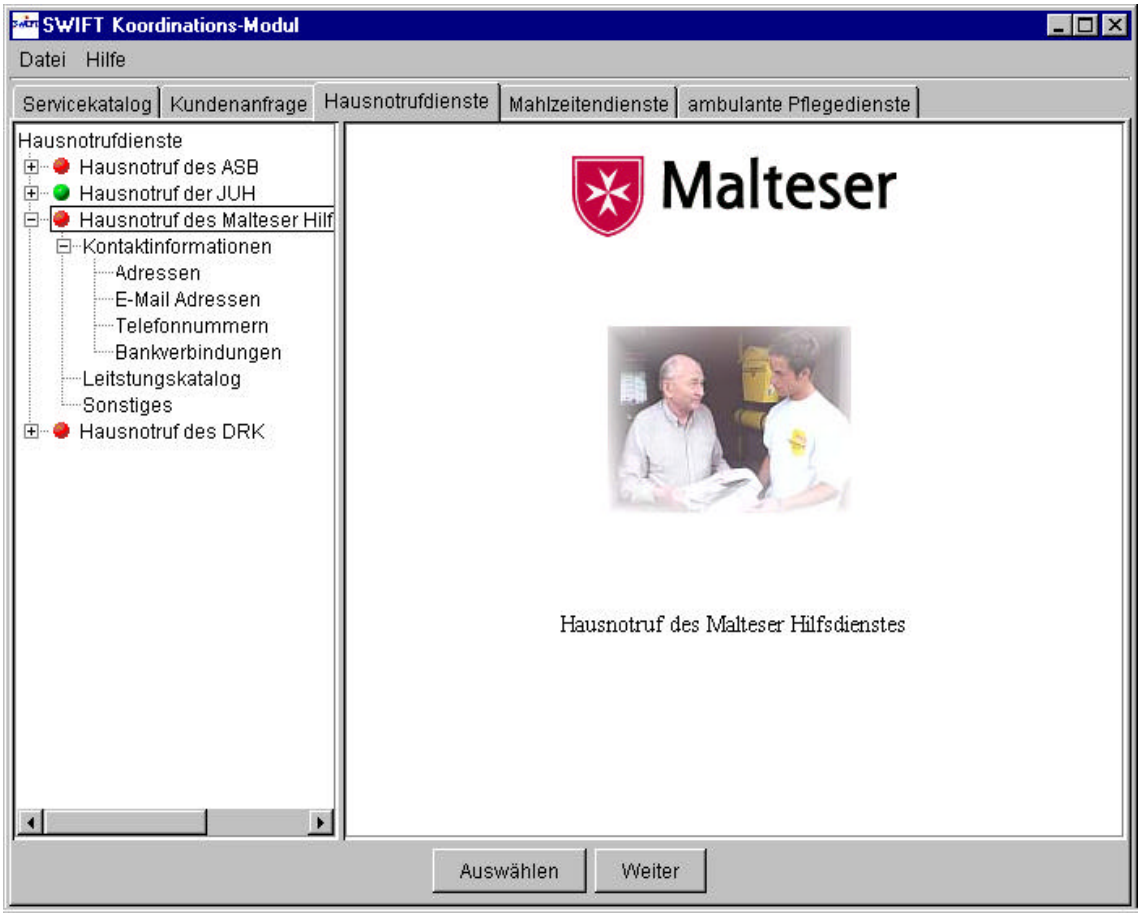


Abbildung 6-8 Maske Hausnotrufdienste

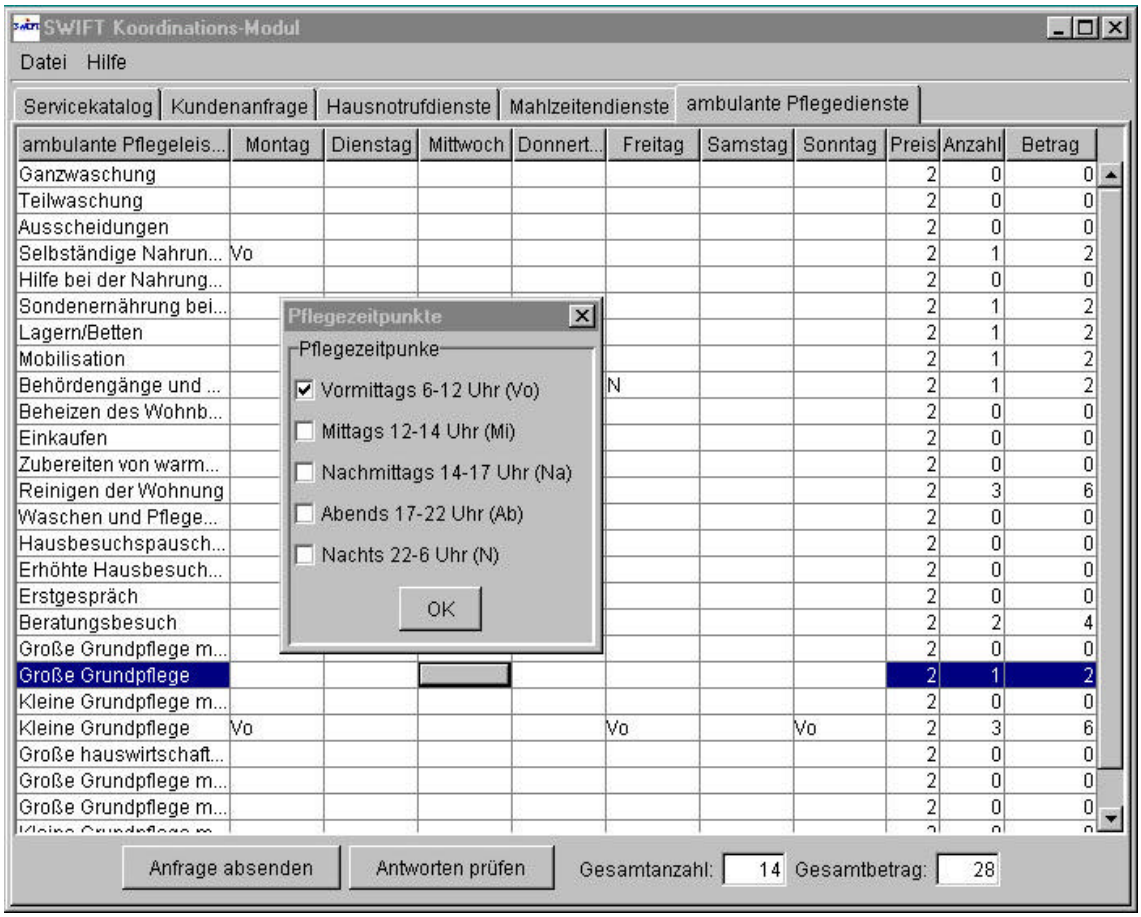
6.1.6 Mahlzeitendienste

Für die Koordinierung von Mahlzeitendiensten steht die Maske *Mahlzeitendienste* zur Verfügung. Sie hat den gleichen Aufbau und die gleichen Funktionen wie die Maske *Hausnotrufdienste*. Diese beiden Masken unterscheiden sich nur durch die Daten mit denen sie arbeiten.

6.1.7 ambulante Pflegedienste

Die Maske *ambulante Pflegedienste* wird für die Koordinierung der Leistungen von ambulanten Pflegediensten benötigt. Bei diesem Vorgang wird der eigentlichen Auswahl eines Dienstanbieters noch eine Überprüfung der Verfügbarkeit der Leistungen vorangestellt. Dazu wird zuerst mit Hilfe der sich in dieser Maske befindlichen Tabelle ein persönliches Leistungsprofil für den Kunden erstellt. Auf der Hochachse dieser Tabelle befinden sich die zur Auswahl stehenden Leistungen, aus denen das persönliche Leistungsprofil zusammengestellt werden kann. Auf der

Längsachse befinden sich je eine Spalte für jeden Tag einer Woche sowie eine Spalte für die Einzelpreise der Leistungen, für die Anzahl der gewählten Leistungen sowie für die Summe der Einzelpreise. Durch Anklicken eines Feldes dieser Tabelle wird ein Dialog sichtbar, in dem für jeden Tag jede Leistung zu fünf verschieden Zeitpunkten gewählt werden kann. Hinter jedem Zeitpunkt ist ein Kürzel angegeben, welches, falls dieser Zeitpunkt gewählt wurde, bei Verlassen des Dialoges in der Tabelle angezeigt wird. Siehe Abbildung 6-9.



ambulante Pflegeleis...	Montag	Dienstag	Mittwoch	Donnert...	Freitag	Samstag	Sonntag	Preis	Anzahl	Betrag
Ganzwaschung								2	0	0
Teilwaschung								2	0	0
Ausscheidungen								2	0	0
Selbständige Nahrun...	Vo							2	1	2
Hilfe bei der Nahrung...								2	0	0
Sondenernährung bei...								2	1	2
Lagern/Betten								2	1	2
Mobilisation								2	1	2
Behördengänge und ...					N			2	1	2
Beheizen des Wohnb...								2	0	0
Einkaufen								2	0	0
Zubereiten von warm...								2	0	0
Reinigen der Wohnung								2	3	6
Waschen und Pflege...								2	0	0
Hausbesuchspausch...								2	0	0
Erhöhte Hausbesuch...								2	0	0
Erstgespräch								2	0	0
Beratungsbesuch								2	2	4
Große Grundpflege m...								2	0	0
Große Grundpflege								2	1	2
Kleine Grundpflege m...								2	0	0
Kleine Grundpflege	Vo				Vo		Vo	2	3	6
Große hauswirtschaft...								2	0	0
Große Grundpflege m...								2	0	0
Große Grundpflege m...								2	0	0

Abbildung 6-9 Maske ambulante Pflegedienste

Dadurch läßt sich für eine Woche exemplarisch der Leistungsbedarf des Kunden genau erfassen. Das so erstellte persönliche Leistungsprofil kann anschließend durch Betätigen der Schaltfläche *Anfrage absenden* zu einer Leistungsanfrage zusammengestellt werden. Dafür wird aus einem Teil der Daten des aktuellen CareClient-Objekts ein Objekt der Klasse *com.server.SecureCareClient* erstellt. Das neu erstellte *SecureCareClient* Objekt wird zu einem Objekt des Typs *com.server.ServiceProfile* hinzugefügt, welches außerdem noch die Daten des persönlichen Leistungsprofils enthält. Dieses

ServiceProfile-Objekt wird nun über die RMI Verbindung an alle ambulanten Pflegedienste gesendet. Jeder Dienstanbieter hat dadurch die anonymisierten Daten sowie die Leistungswünsche des Kunden vorliegen und kann entscheiden, ob er diesen Kunden versorgen kann oder nicht. Dies geschieht mit Hilfe einer Nachricht, die an den Koordinator zurückgesendet wird. In dieser Nachricht erklärt sich der Dienst verbindlich bereit, die Leistungswünsche des Kunden zu erfüllen oder teilt dem Koordinator mit, daß die Leistungsanfrage nicht erfüllt werden kann. Jeder Dienstanbieter, der sich bereit erklärt, die gewünschten Leistungen zu erbringen, wird automatisch in eine Liste mit eingegangen Antworten aufgenommen. Diese Liste ist nach Eingang der Antworten sortiert. Das bedeutet, wer zuerst antwortet, steht als erster auf der Liste. Durch Betätigen der Schaltfläche *Antworten prüfen* erscheint ein Dialog auf dem Bildschirm, der, sofern schon Antworten eingegangen sind, diese Liste anzeigt und die Möglichkeit bietet, einen Dienstanbieter auszuwählen und ihm die gesamten Daten des Kunden zu senden. Damit ist der Koordinierungsablauf beendet und kann erneut begonnen werden.

6.2 Beschreibung des Koordinationsclients

Der Koordinationsclient ist das Gegenstück zum Koordinierungsserver. Er stellt die Funktionen zur Verfügung, die ein Dienstanbieter benötigt, um mit dem Koordinator interagieren zu können. In diesem Prototyp identifiziert sich jeder Client gegenüber dem Server mit seinem Namen (dem Namen des Dienstanbieters, den er repräsentiert). Nach dem Programmstart verbindet sich der Client mit dem Koordinationsserver und empfängt die Daten seines Dienstes. Der Hauptgrund für diese Vorgehensweise ist, daß der Dienstanbieter so überprüfen kann, ob die für seinen Dienst angezeigten Informationen im Servicekatalog des Koordinators korrekt sind. Die empfangen Daten werden in der in Abbildung 6-10 dargestellten Maske *Einrichtungsdaten* angezeigt. Der Koordinationsclient stellt genau wie der Koordinierungsserver auch seine Masken in Karteikarten dar. Die Maske *Einrichtungsdaten* ist in drei Abschnitte eingeteilt. Oben in der Maske befinden sich die allgemeinen Daten über den Dienstanbieter. Dies sind:

- Der Name
- Die Kurzbeschreibung
- Der Träger des Dienstes

- Der Typ des Dienstes

Darunter befinden sich die Kontaktinformationen, die aus Adressen, Telefonnummern, E-Mail Adressen, Bankverbindungen sowie Kontaktpersonen bestehen. Unterhalb der Kontaktinformationen befindet sich noch ein Freitextfeld in das sonstige Informationen geschrieben werden können. Außerdem enthält die Maske noch eine Schaltfläche, mit der die Einrichtungsdaten zurück an den Koordinationsserver gesendet werden können. Wird diese Schaltfläche betätigt, wird dem Koordinator eine Meldung angezeigt, daß dieser Dienstanbieter seine Daten aktualisiert hat. Zudem werden im Serverprogramm alle Baumstrukturen aktualisiert.

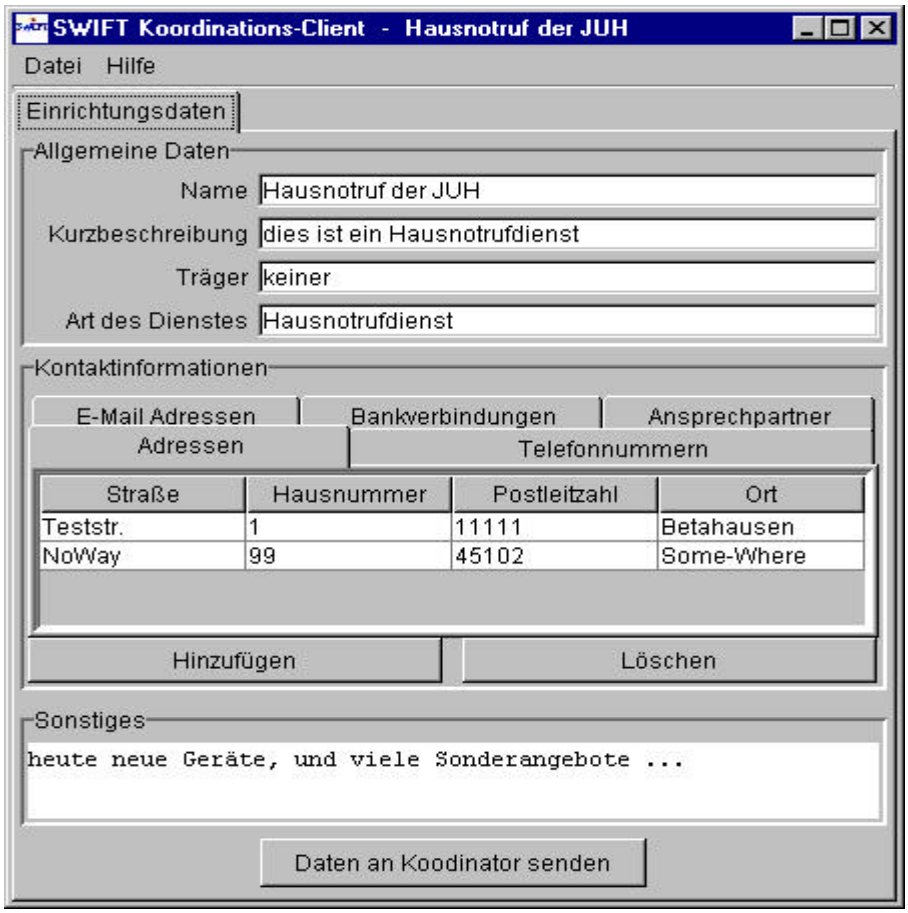


Abbildung 6-10 Maske Einrichtungsdaten

Neben der Maske *Einrichtungsdaten* kann der Koordinationsclient noch zwei weitere Arten von Masken darstellen. Dies sind die Masken des Typs Kundendaten und Leistungsanfrage.

6.2.1 Kundendaten

Die Maske *Kundendaten* stellt die Daten eines Kunden dar. Nach dem Programmstart des Koordinationsclients ist kein Exemplar dieser Maske vorhanden. Erst wenn im Laufe eines Koordinierungsprozeßes Kundendaten an den Dienst gesendet werden, wird diese Maske hinzugefügt. Da es ja durchaus normal ist, daß einem Dienst mehrere Kunden zugeordnet werden, können von dieser Maske beliebig viele Exemplare angezeigt werden. Die einzelnen Masken können leicht auseinander gehalten werden, da der Nachname des Kunden mit in die Beschriftung jeder Karteikarte aufgenommen wird. Die Kundendaten-Maske hat einen ähnlichen Aufbau wie die Kundenanfrage-Maske im Koordinationsserver, da sie ebenfalls die Daten eines Objektes der Klasse *CareClient* anzeigt. Sie unterscheiden sich hauptsächlich durch den Aufbau der Schaltflächenleiste am unteren Ende der Maske. Siehe Abbildung 6-11. In der Kundendaten-Maske besteht diese aus drei Schaltflächen:

- Nachricht an Koordinator

  Mit dieser Schaltfläche ist es möglich, vorformulierte Nachrichten, die den Status des Kunden betreffen an den Koordinationsserver und damit den Koordinator zu senden. Bei Betätigung dieser Schaltfläche erhält der Koordinator eine Meldung, aus der hervorgeht, welcher Dienstanbieter mit welchem Kunden welche Vereinbarung getroffen hat. Neben der Meldung, die in Form eines Pop-Up-Dialoges erscheint können, die so versendeten Informationen vom Koordinator im Dialog *Dienste* der Kundenanfrage-Maske verifiziert werden.
- Daten speichern

  Diese Schaltfläche hat noch keine Funktion, könnte aber dazu verwendet werden die empfangenen Kundendaten in einer lokalen Datenbank auf der Seite des Dienstanbieters zu speichern.
- Daten löschen

  Die Schaltfläche *Daten löschen* löscht die Maske *Kundendaten* aus dem Koordinationsclient und mit ihr auch die empfangen Datenobjekte.

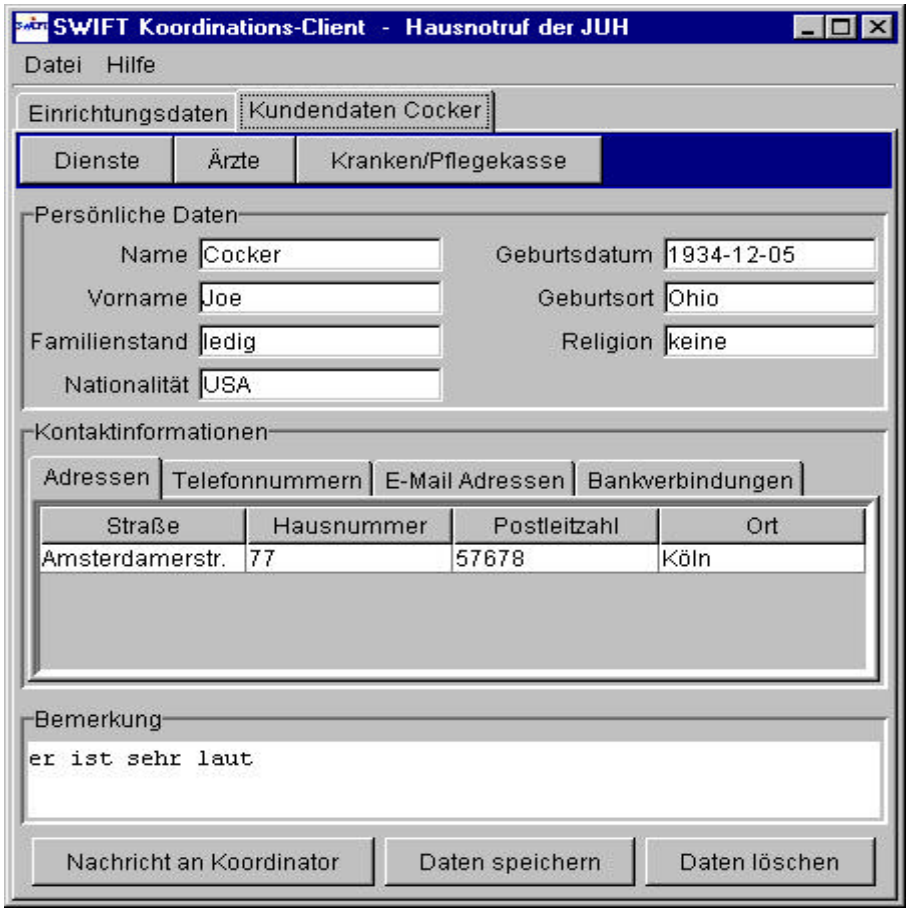


Abbildung 6-11 Maske Kundendaten

6.2.2 Leistungsanfrage

Die in Maske Leistungsanfrage hat die Aufgabe, die vom Koordinator im Falle der Koordination von ambulanten Pflegeleistungen abgesendeten Leistungsanfragen darzustellen. Sie ist genau wie die Kundendaten-Maske nach dem Programmstart nicht vorhanden und wird erst hinzugefügt, wenn eine Leistungsanfrage vom Koordinationsserver an die ambulanten Pflegedienste gesendet wird. Auch von diesem Maskentyp können beliebig viele Exemplare angezeigt werden. Da Leistungsanfragen nur an Dienste des Typs *ambulante Pflegedienste* gerichtet sind, kann diese Maske nur angezeigt werden, wenn der angemeldete Dienst von diesem Typ ist. Der Aufbau dieser Maske ist dem der Maske *ambulante Pflegedienste* im Koordinationsserver sehr ähnlich, da in beiden die gleichen Daten dargestellt werden. Die Hauptunterschiede zwischen diesen Masken sind zum einen, daß sich die Tabelle in der Leistungsanfrage-Maske nicht verändern läßt und zum anderen, daß die Schaltflächenleiste am unteren Ende der Maske einen anderen Aufbau hat. Abbildung 6-12 zeigt, daß diese

Schaltflächenleiste den gleichen Aufbau hat, wie die Schaltflächenleiste in der Maske *Kundendaten*. Sie besteht aus folgenden Schaltflächen:

- Nachricht an Koordinator

  Mit Betätigung dieser Schaltfläche kann der Dienstanbieter dem Koordinator mitteilen, ob er in der Lage ist, die aktuelle Kundenanfrage zu übernehmen. Dies geschieht mit Hilfe vorformulierter Nachrichten, die an den Koordinationsserver gesendet werden. Dadurch erhält der Koordinator eine Meldung aus der hervorgeht, welcher Dienstanbieter sich gerade meldet. Neben dieser Meldung, wird der Name des Dienstes noch in die Antwortenliste des Koordinationsservers eingetragen.
- Daten speichern

  Diese Schaltfläche hat noch keine Funktion.
- Daten löschen

  Die Schaltfläche *Daten löschen* löscht die Maske *Leistungsanfrage* aus dem Koordinationsclient und mit ihr auch die empfangen Datenobjekte.

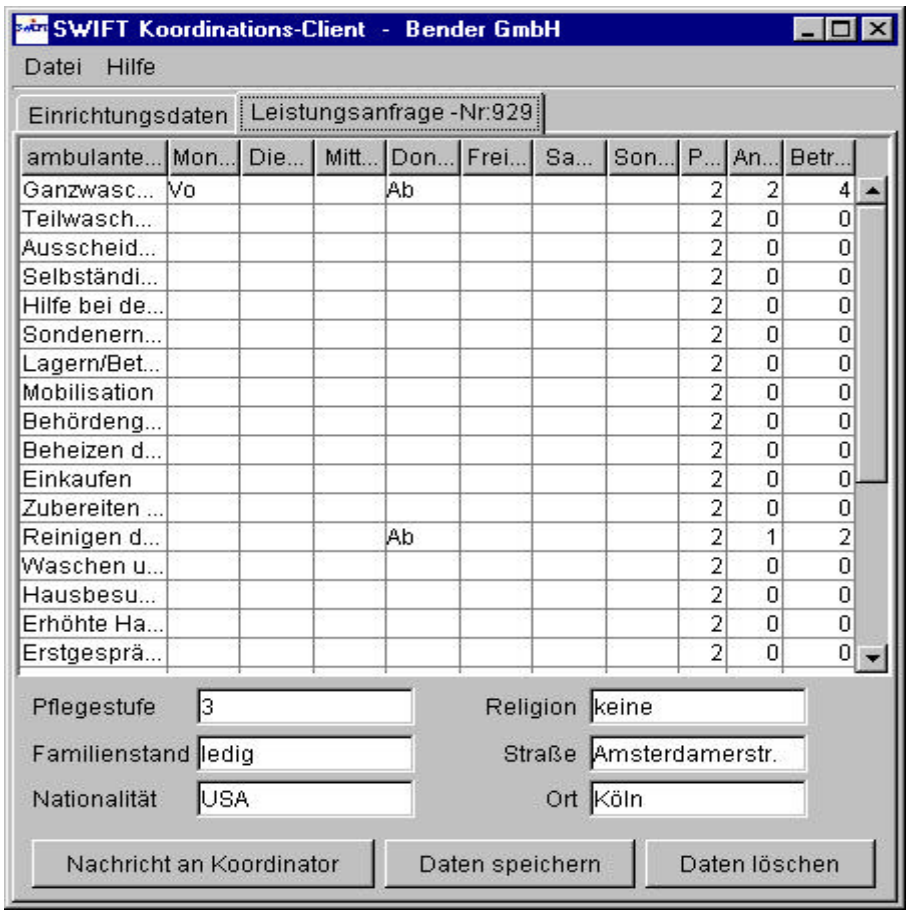


Abbildung 6-12 Maske Leistungsanfrage

6.3 Datenmodell

Für die Datenhaltung in diesem Programm wird eine Microsoft Access Datenbank in der Version Access97 verwendet. Dabei ist zu bemerken, daß nur der Koordinationsserver über eine Verbindung zur Datenbank verfügt, und daß der Koordinationsclient alle benötigten Daten über die RMI Verbindung erhält. Die Datenbankzugriffe werden über die JDBC-ODBC Bridge von Java abgewickelt. Der Hauptgrund für den Einsatz der Access Datenbank ist, daß der Prototyp hauptsächlich zur Demonstration des Konzeptes benötigt wird und daher auf möglichst vielen verschiedenen Systemen lauffähig sein sollte. Da Access Datenbanken sehr weit verbreitet sind, und da das Programm keine besonderen Ansprüche an die Datenbank stellt, bot sich diese Lösung an. Außerdem ermöglicht es die Verwendung von OBDC, jede andere ODBC-fähige Datenbank zu benutzen. Dazu muß lediglich das Datenmodell konvertiert werden, es sind jedoch keine Änderungen am Programm notwendig.

Durch die Verwendung einer relationalen Datenbank zusammen mit einer objektorientierten Programmiersprache hat der Programmierer die Aufgabe, die Serialisierung der Objekte in die einzelnen Tabellen selbst zu übernehmen. Daraus ergibt sich, daß einzelne Objekte die ihrerseits wieder Listen oder Vektoren von Objekten enthalten auf mehrere Tabellen aufgeteilt werden müssen. Die Datenbank enthält daher folgende Tabellen:

- ID:

  Diese Tabelle wird zur Generierung der eindeutigen Objekt ID für jedes Datenobjekt verwendet, da die Access Datenbank keine Sequenzen oder Ähnliches kennt. Sie enthält nur eine Spalte vom Typ *Integer*, die immer fortgeschrieben wird, wenn ein neues Objekt erzeugt wird.
- Adress, BankAccount, EMail, PhoneNumber:

  Diese Tabellen speichern die Adressen, Bankverbindungen, E-Mail Adressen sowie Telefonnummern, die z.B. in den Objekten des Typs *CareClient* oder *Service* verwendet werden. Die Speicherung und Darstellung dieser im Programm im Bereich Kontaktinformationen verwendeten Daten wird über Klassen aus dem Personenmodul von SWIFT abgewickelt. Dabei enthalten diese Tabellen keinen Primärschlüssel, statt dessen werden n-zu-m Beziehungen, wie sie z.B. zwischen

Objekten des Typs *CareClient* und des Typs *Adress* normal sind, durch Mehrfacheinträge in der Tabelle *Adress* dargestellt. Obwohl diese Art der Zuordnung gegen die erste Normalform verstößt, wurde dieses Modell aus Gründen der Kompatibilität zu den bestehenden Modulen von SWIFT beibehalten.

- CareClient:

  In dieser Tabelle werden die einfachen Datenfelder der *CareClient* Objekte gespeichert. Dies sind Name, Vorname, Geburtstag, Geburtsort, Pflegestufe, Familienstand, Nationalität, Religion, Bemerkung und ID. Mit Hilfe des Primärschlüssels ID können jedem *CareClient* Datensätze aus den Tabellen *Adress*, *BankAccount*, *EMail*, *PhoneNumber* zugeordnet werden. Zusätzlich können einem CareClient-Objekt noch mit Hilfe der Zuordnungstabellen *CareClientPaymentInstitution*, *CareClientPhysician*, *CareClientRequestedService* beliebig viele Datensätze aus den Tabellen *PaymentInstitution*, *Physician* sowie *RequestedService* zugeordnet werden.
- PaymentInstitution:

  Die Tabelle *PaymentInstitution* nimmt die Daten der PaymentInstitution-Objekte auf, welche die Kranken- und Pflegekassen repräsentieren. Sie enthält neben der ID noch Felder für den Namen, eine Beschreibung sowie für ein Kürzel.
- Physician:

  Hier werden die Informationen über die in das Programm eingegebenen Ärzte gespeichert. Diese Informationen werden in Objekten der Klasse *Physician* verwendet. Die Tabelle enthält die Felder ID, Name und Vorname. Mit Hilfe der ID können wiederum Adressen, E-Mail Adressen, Bankverbindungen, Telefonnummern sowie Sprechstunden zugeordnet werden.
- VisitingHour:

  Diese Tabelle enthält die Sprechstunden, welche den Ärzten zugeordnet werden können. Auch hier wird das Modell der Zuordnung aus dem Personenmodul verwendet.
- CareClientPaymentInstitution:

  Diese Tabelle dient dazu, einem Kunden beliebig viele Kranken- und Pflegekassen zuzuordnen.

- CareClientPhysician:
  In dieser Tabelle stehen die Zuordnungen der Ärzte zu den Kunden.
- CareClientRequestedService:
  Die Tabelle *CareClientRequestedService* enthält die Zuordnungen der Diensttypen, an die ein Kunde koordiniert werden möchte, zu den Kunden. Außerdem enthält sie noch ein Feld, welches die Zahlungsart der ambulanten Versorgung aufnimmt sowie ein Feld für den Status der Koordinierung. Damit ist es möglich, für jeden Diensttyp, an den ein Kunde koordiniert wird einen Status abzuspeichern, der dem Koordinator z.B. Auskunft darüber gibt, ob ein Kunde mit einem Dienst einen Vertrag abgeschlossen hat oder nicht.
- Service:
  Diese Tabelle enthält, wie in Kapitel 5 beschrieben, die Daten über die einzelnen Einrichtungen, die innerhalb des Servicekatalogs dargestellt werden.
- Contact:
  Hier werden die Ansprechpartner zu den einzelnen Diensten gespeichert. Da diese Informationen ebenfalls zu den Kontaktinformationen gehören, werden diese genau wie z.B. die Adressen gespeichert.
- ServiceTypes:
  Hier werden die Diensttypen gespeichert, die für eine Koordination zur Auswahl stehen. Diese Tabelle enthält nur den Namen, eine Beschreibung sowie die ID jedes Dienstes. In diesem Prototyp enthält diese Tabelle nur die drei untersuchten Diensttypen.

Eine vollständige Abbildung aller hier erklärten Tabellen befindet in Anhang B.

6.4 Kommunikation zwischen Client und Server

Die Kommunikation zwischen dem Koordinationsserver und dem Koordinationsclient wird über Java RMI ausgeführt. Wie in Kapitel 3 beschrieben wird für eine RMI Kommunikation ein Schnittstellendefinition und eine Klasse, welche die in der Schnittstellendefinition definierten Methoden implementiert, benötigt. Im Koordinationsserver übernimmt das Interface *com.server.ServerInterface* die Schnittstellendefinition und die Klasse *com.server.Server* die Implementation der Methoden.

Für die einfache Client-Server-Verbindung würden diese beiden Dateien völlig ausreichen. Damit ist es möglich, daß der Client auf dem Serverobjekt Methoden aufruft, in denen er z.B. ein Objekt als Parameter mitgibt, und als Rückgabewert ein anderes Objekt erhält. Dies dürfte für die meisten Anwendungen genügen, stößt allerdings an seine Grenzen, wenn z.B. der Server auf dem Client ein Meldungsfenster öffnen soll. Es gibt zwei verbreitete Methoden, dieses Problem zu umgehen. Die eine ist das regelmäßige Anfragen des Clients bei seinem Server ob es z.B. neue Nachrichten gibt. Dies hat den Nachteil, daß das Netzwerk ständig belastet wird, und daß auch das Clientprogramm ständig beschäftigt ist. Die andere Möglichkeit ist, die in dieser Arbeit verwendete Call-Back Methode. Hierbei wird neben dem eigentlichen Server auch der Koordinationsclient als Server verwendet. Um dies zu erreichen, wird im Clientprogramm auch ein Remote-Objekt definiert und implementiert. Dies geschieht mit dem Interface *com.client.ClientInterface* und der Klasse *com.client.Client*.
Damit nun der Koordinationsserver einen Methodenaufruf auf dem Remote-Objekt des Clients durchführen kann, muß bei der Anmeldung des Clients eine Referenz auf dessen Remote-Objekt an den Server übergeben werden. Diese Referenzen auf das jeweilige Remote-Objekte müssen für jeden Client gespeichert werden, um später gezielt kommunizieren zu können.
Neben den oben erwähnten Klassen für die Schnittstellendefinition und Implementation werden auf beiden Seiten noch Klassen bereitgehalten, die es ermöglichen, bestimmte Methoden der Remote-Objekte in einem separaten Thread[15] aufzurufen. Dies ist notwendig, da sonst z.B. der Client, wenn er eine Methode auf dem Server aufruft, blockiert wird, bis der Aufruf komplett abgearbeitet ist. Dieses Blockieren ist besonders auffällig, wenn im Laufe eines solchen Methodenaufrufes ein Dialog geöffnet wird, der eine Bestätigung des Benutzers fordert. Im Koordinationsserver wird dieser Thread von der Klasse *com.server.ServerContainerThread* bereitgehalten. Auf der Seite des Koordinationsclients übernimmt diese Aufgabe die Klasse *com.client.ClientContainerThread*. Abbildung 6-13 stellt das Klassendiagramm der RMI Klassen dar.

[15] Ein Thread ist ein kleiner eigenständig ablaufender Prozeß, damit ist es ermöglicht mehrere Programmteile gleichzeitig zu bearbeiten.

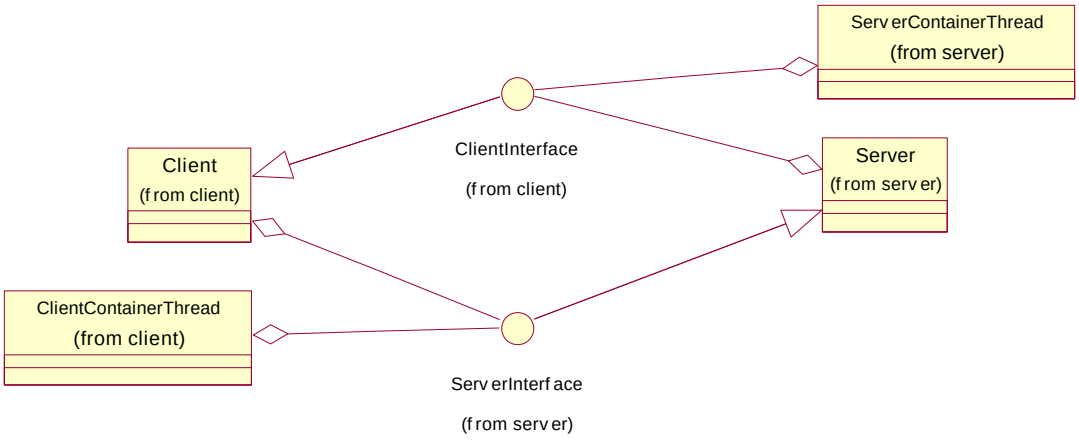


Abbildung 6-13 Klassendiagramm der RMI Klassen

6.5 Sicherheit im Prototyp

Die Aspekte der Sicherheit spielen gerade bei verteilten Programmen eine große Rolle, da hier die Angriffsmöglichkeiten wesentlich größer sind als bei anderen Systemen. Da in dem in diesem Prototyp verfolgten Konzept der Koordinierung Kundendaten direkt zwischen Server und Client verschickt werden, ist hier eine Sicherung der Daten erforderlich. Diese Sicherung verfolgt die in Kapitel 3 vorgestellten Schutzziele der Vertraulichkeit und Integrität.

6.5.1 Verschlüsseln von Java-Objekten

Java bietet mit der Klasse *javax.crypto.SealedObject* die Möglichkeit, Objekte, welche über RMI versendet werden sollen, sehr einfach zu verschlüsseln und damit das Schutzziel der Vertraulichkeit zu erreichen. Die Klasse *SealedObject* ist nicht Bestandteil der Standard Java Klassenbibliotheken sondern wird mit der Java Cryptography Extension (JCE) ausgeliefert. Diese erweiterte Klassenbibliothek ist frei unter [Sun00] verfügbar. Die Verschlüsselung von Objekten findet im Prototyp bei der Versendung der Kundendaten an einen Dienstanbieter exemplarisch Anwendung. Für die Verschlüsselung wird zunächst ein Schlüsselpaar benötigt. Dieses kann, wenn es z.B. von einer Zertifizierungsstelle bezogen wurde, direkt aus dem Dateisystem geladen werden oder wie in diesem Prototyp selbst mit Hilfe der Klasse *KeyPairGenerator* aus dem *java.security* Paket generiert werden. Die Verschlüsselung erfolgt beim Sender mit dem öffentlichen Schlüssel und die Entschlüsselung beim Empfänger mit dem privaten

Schlüssel des Empfängers. Um nun ein Objekt zu verschlüsseln - also ein normales Objekt in ein *SealedObject* umzuwandeln - , muß zunächst ein Objekt der Klasse *javax.crypto.Cipher* erzeugt werden. Dieses *Cipher* Objekt wird mit dem öffentlichen Schlüssel des Empfängers und der auszuführenden Aktion (Verschlüsseln) initialisiert. Danach wird einfach ein neues *SealedObject* mit dem ursprünglichen Objekt und dem initialisierten *Cipher* Objekt als Parameter erzeugt. Das so erzeugte *SealedObject* kann nun direkt über RMI versendet werden.
Zum Entschlüsseln muß wieder ein *Cipher* Objekt erzeugt und mit dem privaten Schlüssel des Empfängers sowie der auszuführenden Aktion (entschlüsseln) initialisiert werden. Danach muß die *getObject* Methode des verschlüsselten Objektes mit dem zuvor initialisierten *Cipher* Objekt als Parameter aufgerufen werden. Diese Methode gibt dann direkt das unverschlüsselte Ausgangsobjekt zurück.

6.5.2 Signieren von Java-Objekten

Um das Schutzziel der Integrität zu erreichen können Java-Objekte signiert werden. Dabei werden zu versendende Objekte mit einer digitalen Unterschrift versehen. Auch für dieses Verfahren wird ein Schlüsselpaar benötigt, da die Unterschrift oder Signatur vom Absender mit dessen privatem Schlüssel erzeugt und vom Empfänger mit dem öffentlichen Schlüssel des Absenders verifiziert wird. Für das Signieren von Objekten bietet das zu den Standardbibliotheken von Java gehörende *java.security* Paket die Klasse *SignedObject* an. Um ein Objekt zu signieren, muß zunächst eine Instanz der Klasse *java.security.Signature* erzeugt und deren *getInstance* Methode aufgerufen werden. Diese Methode benötigt als Parameter den Namen des Algorithmus, der für die Erzeugung der Signatur verwendet werden soll. Danach kann einfach ein neues *SignedObject* erzeugt werden, welches das zu signierende Objekt, den privaten Schlüssel des Absenders und das zuvor erstellte Signature-Objekt als Parameter benötigt. Bei der Erstellung eines *SignedObjects* wird eine Kopie des original Objekts zusammen mit einer digitalen Signatur in dem neuen Objekt hinterlegt, so daß alle weiteren Änderungen am original Objekt keine Auswirkungen auf die Kopie mehr haben.
Um zu überprüfen, ob ein so signiertes Objekt verändert wurde muß beim Empfänger zuerst wieder ein Signature-Objekt erzeugt und mit dem Namen des entsprechenden Algorithmus initialisiert werden. Danach kann die Methode *verify* des empfangen

SignedObjects aufgerufen werden. Sei benötigt als Parameter den öffentlichen Schlüssel des Senders sowie das zuvor erstellte Signature-Objekt. Wenn die Signatur unbeschädigt ist, gibt diese Methode den Wert *true* zurück und das original Objekt kann über die *getContent* Methode aus dem *SignedObject* herausgelöst werden. Zu beachten ist bei der Überprüfung der Signatur, daß ein korrektes Ergebnis nur mit dem richtigen öffentlichen Schlüssel erzielt werden kann.

Da die Konvertierung der über RMI zu versendenden Objekte in den Typ *SealedObject* nur davor schützt, daß die Objekte von Dritten gelesen werden können, aber nicht gewährleistet, daß die Objekte auf dem Transportweg nicht verändert wurden, müssen diese Objekte zur Erreichung des Schutzzieles der Integrität auch signiert werden. In [Gong98] wird davor gewarnt, blind verschlüsselte Objekte zu signieren, da deren Inhalt nicht ersichtlich ist. Statt dessen wird empfohlen, zu versendende Objekte zuerst zu signieren und dann zu verschlüsseln.

6.6 Fazit der Programmierung

Zusammenfassend kann gesagt werden, daß die Programmiersprache Java2 in der Version 1.3 soweit ausgereift ist, daß die Erstellung von Applikationen ohne Probleme möglich ist. Die sehr umfangreichen Klassenbibliotheken, die für Java existieren sind eine große Hilfe, da es für alle Standardprobleme bereits gute Lösungen gibt. Auch die in dieser Arbeit verwendete Entwicklungsumgebung der JDeveloper ist in der Version 1.3 so sicher, daß es nur noch äußerst selten zu Systemabstürzen kommt. Das einzige Problem bei der Verwendung von Java stellen immer noch die hohen Anforderungen an die Rechner dar, da diese z.B. ein Ausführen des Koordinationsservers auf älteren Rechnern verhindern.

7 Möglichkeiten des Servicekatalogs

Da das Konzept des Servicekatalogs sehr mächtig ist, aber innerhalb des Prototyps nur ein kleiner Teil der Möglichkeiten gezeigt wird, werden nachfolgend weitere Verwendungsmöglichkeiten aufgezeigt.

7.1 Darstellung im Internet

Durch die rasante Entwicklung und Verbreitung des Internets sind heute Dinge selbstverständlich, die vor zwei Jahren noch als utopisch und unrealistisch galten. So ist es z.B. normal, daß sich auch ältere Menschen für das Internet interessieren, und daß Internet-Kurse für diese Zielgruppe ausgebucht sind.
Daher kann eine zukünftige Darstellung des Servicekatalogs im Internet älteren Menschen oder deren Angehörigen die Möglichkeit geben, sich vor einem Besuch in einem Servicezentrum über das Dienste-Angebot im ihrem Bereich zu informieren.
Die für den Aufbau und die Darstellung des Servicekatalogs innerhalb des Koordinationsmoduls verwendeten Java-Technologien können ohne großen Aufwand für eine Darstellung im Internet verwendet werden. Dazu müßten nur die entsprechenden Programmteile in Java-Applets eingebaut werden. Die für die Repräsentation der einzelnen Dienstanbieter innerhalb des Servicekataloges verwendete HTML Seite unterstützt diesen Ansatz.

Desweiteren ist es auch denkbar, daß in Zukunft Dienstleistungen von älteren Menschen direkt im Internet gebucht werden können, und daß eine Kommunikation mit einem Koordinator oder mit einzelnen Dienstanbietern ebenfalls über dieses Medium läuft.

7.2 Datenaustausch der Dienste untereinander

Ein zentraler Servicekatalog kann auch für den Datenaustausch der einzelnen Dienste untereinander verwendet werden. Hierbei ist es z.B. denkbar, daß ein ambulanter Pflegedienst mit Hilfe des Servicekatalogs alle Dienste, die mit einem bestimmten Kunden einen Vertrag abgeschlossen haben, darüber informiert, daß dieser Kunde in Urlaub ist, oder daß er nach einem Schwächeanfall in ein Krankenhaus eingeliefert wurde.

Ein großflächig angelegter Servicekatalog könnte auch dazu dienen, Kundendaten auszutauschen falls ein Kunde umzieht und damit den Einsatzbereich seiner Pflegedienste verläßt.

7.3 Reporting

Eine weitere Möglichkeit für eine Nutzung des Servicekatalogs ist in der Erweiterung der bestehenden Datenstruktur des Servicekatalogs zu sehen. Durch Hinzufügen von Daten, welche die Verteilung der Kunden auf die einzelnen Dienste angeben, kann kontrolliert werden, wie sich die Kunden der Koordinierungsstelle auf die einzelnen Dienstanbieter verteilen. Daraus können Rückschlüsse auf das Angebot der Dienste gezogen sowie die Gleichbehandlung aller Dienstanbieter kontrolliert werden.

7.4 Dokumentenfluß im Pflegeprozeß

Der Servicekatalog kann eine zentrale Rolle für den Dokumentenfluß im Pflegeprozeß einnehmen. Zum einen ermöglicht er es als zentrale Datenbank sämtliche anfallenden Dokumente zentral und für alle zugänglich (je nach Berechtigung) aufzubewahren. Dies könnte für alle Beteiligten Vorteile, wie z.B. ein gemeinsames Sicherungskonzept oder Einsparungen im Wartungsaufwand und bei Hardwarekosten mit sich bringen.
Neben diesen technischen Vorteilen kann eine zentrale Datenhaltung auch zur Nachverfolgung von Dokumenten eingesetzt werden. So könnte z.B. kontrolliert werden, ob das Krankenhaus *A* die Kundenakte eines bestimmten Kunden schon bereitgestellt oder bearbeitet hat. Eine solche Dokumentenverfolgung kann dazu dienen spätere Workflow Ansätze vorzubereiten.

7.5 XML

Mit XML[16] und dem damit verbundenen XML-EDI[17] bieten sich ganz neue Möglichkeiten für den Datenaustausch zwischen dem Servicekatalog und anderen Programmen. Durch eine Umsetzung der Struktur des Servicekatalogs in die beschreibende Sprache XML können sehr leicht Schnittstellen zu beliebigen anderen Programmen geschaffen werden. Dadurch ist es möglich, Altdatenbestände in den Servicekatalog zu integrieren. Durch die zusätzliche Verwendung einer Datenbank in

[16] Extensible Markup Language
[17] Electronic Data Interchange

der die Bedeutung der einzelnen XML-Tags eindeutig definiert und beschrieben ist, können sogar erste Schritte hin zu einer Standardisierung des Datenaustauschs zwischen Programmen aus dem Pflegebereich gemacht werden. Dies ist möglich, da es zur Zeit für diesen Bereich noch keinen Standard für den Datenaustausch gibt.

8 Validierung des Prototyps

In diesem Kapitel sollte die Validierung des Prototyps mit möglichen späteren Benutzern oder mit Personen, die tagtäglich Koordinationsaufgaben wahrnehmen, beschrieben werden. Leider bot sich durch den Ausstieg eines entscheidenden Projektpartners aus dem SWIFT Projekt diese Möglichkeit nicht. Daher können hier nur die während der zahlreichen Präsentationen des Prototyps gemachten Erfahrungen und Anmerkungen beschrieben werden. Eine vollständige Validierung muß später im Rahmen des SWIFT Projekts erfolgen.

8.1 GUI

Bei Präsentationen des Prototyps konnte festgestellt werden, daß die Benutzeroberfläche des Programms leicht und fast ohne zusätzliche Erklärungen zu bedienen ist. Der Koordinationsserver startet mit der Maske *Servicekatalog*, in welcher die in Kapitel 5 beschriebene Baumstruktur dargestellt wird. Benutzer, die mit den Windows Explorer vertraut sind, können diese intuitiv bedienen und die gesuchten Informationen leicht finden. Da alle Masken des Prototyps aus Standardkomponenten aufgebaut sind, gilt diese intuitive Bedienbarkeit auch für alle übrigen Masken. Durch aussagekräftig beschriftete Schaltflächen können alle angebotenen Funktionen leicht gefunden werden.

8.2 Funktionen

Der Funktionsumfang scheint trotz der auf drei Diensttypen begrenzten Koordinierung ausreichend zu sein, um die dahinterstehenden Konzepte zu verdeutlichen. Die Möglichkeit den Prototyp um weitere Diensttypen für eine Koordinierung zu erweitern ist Bestandteil des Konzeptes. Daher sind alle Dialoge so ausgelegt, daß sie genügend Raum für weitere Diensttypen bieten. Durch die Verwendung von Karteikarten zur Darstellung der Masken, welche die für die Koordinierung zur Auswahl stehenden Dienste beinhalten, ist es durch Hinzufügen einer solchen Maske möglich einen neuen Diensttyp mit in den Koordinierungsablauf einzubeziehen. Darüber hinaus bietet das Programm mit der Speicherung der Statusinformationen zu jedem Kunden Funktionen an, die über die im Soll-Ablauf definierten Anforderungen hinaus gehen.

Die in der Maske *Kundenanfrage* integrierten Datenbankfunktionen reichen aus, um alle anfallenden Kundendaten effizient verwalten zu können.

8.3 Abläufe

Eine objektive Aussage über die dargestellten Ablaufe könnte nur ein Koordinator treffen, der den Prototyp bei der täglichen Arbeit einsetzt. Eine solche Überprüfung muß aus den oben genannten Gründen zu einem späteren Zeitpunkt nachgeholt werden.

8.4 Fazit

Zusammenfassend muß festgehalten werden, daß es sich bei diesem Programm nur um einen ersten Prototyp handelt, der im Laufe einer zyklischen Entwicklung verbessert, erweitert und getestet werden muß. Dieser Prototyp ist nicht als marktreife vollständige Softwarelösung anzusehen.
Dennoch können die erarbeiteten Konzepte als Grundlage für eine noch zu entwickelnde Software dienen. Für eine Erweiterung des Prototyps müssen besonders die Datenfelder im Servicekatalog erweitert werden, um die verschiedenen Angebote der einzelnen Dienstanbieter besser darstellen zu können. Einen vollständigen Ansatz hierfür bietet das *BISS-System* der Firma *Synergie*, allerdings beschränkt sich dieses System ausschließlich auf die Unterstützung der Beratung von Pflegebedürftigen und läßt den Bereich der Koordinierung unbeachtet.

9 Zusammenfassung und Ausblick

Der in dieser Arbeit entstandene Prototyp zeigt, daß die für eine Umsetzung der hier vorgestellten Konzepte benötigten technischen Voraussetzungen bereits vorhanden sind, und daß sie ohne größere Probleme eingesetzt werden können.
Da diese Konzepte jedoch auf dem Vertrauen und den Willen zur Kooperation aller am Pflegeprozeß beteiligten Gruppen basieren, muß an dieser Stelle bei einer zukünftigen Umsetzung begonnen werden. Desweiteren ist im Pflegebereich der Umgang mit neueren Technologien, wie Computern oder Handhelds noch lange nicht so alltäglich wie in anderen Bereichen. Daher wird es noch einige Zeit benötigen bis die Ängste im Umgang mit diesen Technologien abgebaut und sie voll akzeptiert sind. Erst dann kann auch damit gerechnet werden, daß solche Konzepte, wie daß hier vorgestellte zum Einsatz kommen.

Für eine Weiterentwicklung des Prototyps sollte dieser in der nächsten Ausbaustufe um ein Kunden-Modul erweitert werden.
Damit wäre es möglich, daß Dienstleistungen von Kunden direkt gebucht werden können, und daß eine Kommunikation mit einem Koordinator oder mit einzelnen Dienstanbietern komplett über das Programm abgewickelt werden kann. Um bei einem solchen Ansatz den für ältere Menschen oft sehr wichtigen persönlichen Kontakt zum Gesprächspartner nicht unbeachtet zu lassen, könnte die Kommunikation durch die bereits ausgereifte *Web-Cam* Technik unterstützt werden. Die Programmiersprache Java bietet mit dem *Java Media Framework* (JMF) eine weitreichende Unterstützung dieser Technik an, was die Integration in bestehende Programme sehr erleichtern würde.

Da das Internet immer mehr an Bedeutung gewinnt, sollte der darauffolgende Schritt zur Erweiterung des Konzeptes in Richtung auf eine Darstellung im Internet abzielen.

Anhang A

Quellcode des Prototyps:

Neben den nachfolgend abgebildeten Klassen benötigt das in dieser Arbeit entstandene Programm weitere Klassen aus dem Personen- sowie aus dem Leistungskatalog-Modul SWIFT Systems für seine Funktion. Außerdem wird die in Kapitel 6 beschriebe Datenbank für eine Ausführung benötigt.

Die zum Koordinationsmodul gehörenden Klassen befinden sich in den Paketen *com.server* sowie *com.client*. Das Paket *com.server* enthält die Klassen des Koordinationsservers und das Paket *com.client* die Klassen des Koordinationsclients.

Die Klasse CoordinationModule

Dies ist die Startklasse für den Koordinationsserver.

```
// Copyright (c) 2000
package com.server;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
/**
 * Starter class for the
<code>CoordinationModule</code> Server.
 * <P>
 * @author Martin Braches
 */
public class CoordinationModule {

  private MainFrame frame = new MainFrame();
   /**
    * Constructor
    */
  public CoordinationModule() {
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    frame.setLocation((screenSize.width - frameSize.width) /2,
(screenSize.height - frameSize.height)/2);  // centered
    frame.setLocation((screenSize.width - frameSize.width) -5,
(screenSize.height - frameSize.height)-25);  // lower right
    frame.addWindowListener(new WindowAdapter() { public
void windowClosing(WindowEvent e) { System.exit(0); } });
    frame.setVisible(true);
  }

  /**
    * main
    * @param args
    */
  public static void main(String[] args) {
    try  {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFe
elClassName());
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    new CoordinationModule();

  }
}
```

Die Klasse MainFrame

Diese Klasse erzeugt das Hauptfenster des Koordinationsservers, welches als Rahmen für die einzelnen in Kapitel 6 beschriebenen Masken dient.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
import com.client.ClientInterface;
import person.*;
import java.rmi.registry.*;
import java.rmi.Naming;
import java.net.*;
/**
 * A Swing-based top level window class.
```

```
 * It represents the main-window of the
CoordinationModule Server
 * <P>
 * @author Martin Braches
 */
public class MainFrame extends JFrame {

  private ServiceTree serviceTree = new ServiceTree();
  private Vector serviceVector =
serviceTree.getServiceVector();
  private Vector messageVector = new Vector();
  private CareClient careClient = null;
  private ClientRequestPanel crp = null;
  private CoordinationPanel scp = null;
  private CoordinationPanel cop1 = null;
  private CoordinationPanel cop2 = null;
  private CoordinationPanel3 cop3 = null;
  private Server server = null;

  // GUI elements
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JMenuBar menuBar1 = new JMenuBar();
  JMenu menuFile = new JMenu();
  JMenuItem menuFileExit = new JMenuItem();
  JMenu menuHelp = new JMenu();
  JMenuItem menuHelpAbout = new JMenuItem();
  JTabbedPane jTabbedPane1 = new JTabbedPane();
  BorderLayout borderLayout2 = new BorderLayout();
  JSplitPane jSplitPane1 = new JSplitPane();
  /**
   * Constructs a new instance.
   */
  public MainFrame() {
    super();
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    this.setTitle("SWIFT Koordinations-Modul");
    this.setIconImage(new
ImageIcon("Com/Images/Swift2.gif").getImage());
    this.getContentPane().setLayout(borderLayout1);
    this.setSize(new Dimension(700, 550));
    jPanel1.setLayout(borderLayout2);
    menuFile.setText("Datei");
    menuFileExit.setText("Ende");
    menuFileExit.addActionListener(new
ActionListener() {
      public void actionPerformed(ActionEvent e) {
        fileExit_ActionPerformed(e);
      }
    });
    menuHelp.setText("Hilfe");
    menuHelpAbout.setText("Über..");
    menuHelpAbout.addActionListener(new
ActionListener() {
      public void actionPerformed(ActionEvent e) {
        helpAbout_ActionPerformed(e);
      }
    });
    menuFile.add(menuFileExit);
    menuBar1.add(menuFile);
    menuHelp.add(menuHelpAbout);
    menuBar1.add(menuHelp);
    this.setJMenuBar(menuBar1);
    this.getContentPane().add(jPanel1, BorderLayout.CENTER);
    jPanel1.add(jTabbedPane1, BorderLayout.CENTER);
    careClient = new CareClient();
    crp = new ClientRequestPanel(careClient,this);
    scp =  new
CoordinationPanel(serviceTree,careClient,this,"ServiceCatalogu
ePanel","","");
    cop1 = new
CoordinationPanel(serviceTree,careClient,this,"Hausnotrufdienst
e","Mahlzeitendienste","ambulante Pflegedienste");
    cop2 = new
CoordinationPanel(serviceTree,careClient,this,"Mahlzeitendienst
e","ambulante Pflegedienste","");
    cop3 = new CoordinationPanel3(careClient,this);
    jTabbedPane1.addTab("Servicekatalog", scp);
    jTabbedPane1.addTab("Kundenanfrage", crp);
    jTabbedPane1.addTab("Hausnotrufdienste", cop1);
    jTabbedPane1.addTab("Mahlzeitendienste", cop2);
    jTabbedPane1.addTab("ambulante Pflegedienste", cop3);
    rmi();
  }
  /**
   * Starts the RMI Registry and registers the serverInterface
<B>MyService</B> at port 1099.
   */
  public void rmi() {
    try {
      System.out.println("Starting RMI");
      server = new Server(this);
      LocateRegistry.createRegistry(1099);

      String host = "";
      try {
        host = InetAddress.getLocalHost().toString();
      }
      catch(Exception m) {
        host = "";
        System.out.println("Ein Fehler  " +m);
      }
      host = host.substring(0,host.indexOf("/"));

      Naming.rebind("rmi://"+host+":1099/MyService", server);
      System.out.println("rebind ok");
    }
    catch (Exception e) {
      System.out.println("Ein Fehler  " +e);
    }
  }
  /**
   * Performs the action defined for "File|Exit".
   * @param e
   */
  void fileExit_ActionPerformed(ActionEvent e) {
    System.exit(0);
  }
  /**
   * Performs the action defined for "Help|About".
   * @param e
   */
  void helpAbout_ActionPerformed(ActionEvent e) {
      //System.out.println("ok");
      JOptionPane.showMessageDialog(this,"SWIFT
Koordinations-Modul \n       ©2000", "Über..",
JOptionPane.PLAIN_MESSAGE);
  }
  /**
   * Returns the serviceVector.
   */
  public Vector getServiceVector()  {
    return this.serviceVector;
  }
  /**
```

```
   * Returns the messageVector.
   */
  public Vector getMessageVector()  {
    return this.messageVector;
  }
  /**
   * Sets the messageVector.
   * <P>
   * @param messageVector
   */
  public void getMessageVector(Vector messageVector)
{
    this.messageVector=messageVector;
  }
  /**
   * Sets the careClient to all GUI Panels and this frame.
   * <P>
   * @param careClient
   */
  public void setCareClient(CareClient careClient) {
    this.careClient=careClient;
    cop1.setCareClient(this.careClient);
    cop2.setCareClient(this.careClient);
    cop3.setCareClient(this.careClient);
  }
  /**
   * Tells the server to send the messages for the service
<B>serviceName</B>.
   * <P>
   * @param serviceName
   */
  public void sentMessages(String serviceName) {
    ServerContainerThread sct = new
ServerContainerThread("test",this.server,serviceName);
    sct.start();
    //server.sentMessages(serviceName);
  }
  /**
   * Tells all panels to refresh their tree.
   */
  public void refreshTree() {
    scp.refreshTree();
    cop1.refreshTree();
    cop2.refreshTree();
  }
  /**
   * Shows a message-dialog.
   * <P>
   * @param object - The object of the message
   * @param serviceName - The name of the sender
   * @param message - The message
   */
  public void showMessage(Object object, String serviceName,
String message) {
  //public void showMessage(CareClient careClient, String
serviceName, String message) {
    System.out.println(object.getClass().getName());
    if (object.getClass().getName().equals("person.CareClient")) {
      CareClient careClient = (CareClient)object;
      JOptionPane.showMessageDialog(this,message+"\nmit : "
+careClient.getForename()+"
"+careClient.getName(),"Nachricht -
"+serviceName,JOptionPane.PLAIN_MESSAGE);
      String newMessage = " - "+message+" mit "+serviceName;
      //System.out.println(newMessage);
      careClient.storeRequestedServiceStatus(serviceName
,newMessage);
    }
    else
if(object.getClass().getName().equals("com.server.SecureCareCl
ient")) {
      JOptionPane.showMessageDialog(this,message,"Nachricht -
"+serviceName,JOptionPane.PLAIN_MESSAGE);
      System.out.println(message);
      if (!message.endsWith("nicht übernehmen."))
        cop3.addServiceToList(serviceName);
    }
  }
  /**
   * Shows a message-dialog and tells the user that a service has
changed his data.
   * Also tells the service to stores his data into the database and
tells all panels to refresh their trees.
   * <P>
   * @param service -
   */
  public void setService(Service service) {
    JOptionPane.showMessageDialog(this,"Der Dienst -
"+service.getName()+ " - hat seine Daten geändert!","Daten
wurden geändert!",JOptionPane.PLAIN_MESSAGE);
    service.storeInDB();

    scp.refreshService(service);
    cop1.refreshService(service);
    cop2.refreshService(service);

    refreshTree();
  }
}
```

Die Klasse CoordinationPanel

Diese Klasse wird zur Darstellung der Masken *Servicekatalog*, *Hausnotrufdienste* und *Mahlzeitendienste* verwendet. Die Unterscheidung, welche Maske dargestellt werden soll, erfolgt über Parameter im Constructor der Klasse.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.text.*;
import javax.swing.tree.*;
import java.awt.*;
import java.util.Vector;
import javax.swing.event.*;
import java.awt.event.*;
import servicecatalogue.*;
import servicecatalogue.explorer.*;
import person.*;
import java.net.URL;
/**
 * A Swing-based gerneric panel class.
 * <P>
 * @author Martin Braches
 */
public class CoordinationPanel extends JPanel {

  private CareClient careClient        = null;
  private MainFrame mainFrame          = null;
  private ServiceTree serviceTree      = null;
  private String myName                = "";
  private String nextName1             = "";
```

```
  private String nextName2             = "";

  // GUI elements
  JSplitPane jSplitPane1               = new JSplitPane();
  BorderLayout borderLayout1           = new
BorderLayout();
  JPanel jPanel1                     = new JPanel();
  JButton selectButton                 = new JButton();
  JButton nextButton                   = new JButton();
  JTextArea jTextArea1                 = new JTextArea();
  JTree jTree1                       = null;
  JScrollPane jScrollPane1             = new JScrollPane();
  JScrollPane jScrollPane2             = new JScrollPane();
  JTextPane textPane                   = new JTextPane();
  /**
   * Constructs a new instance.
   * <P>
   * @param serviceTree - The serviceTree for this
panel
   * @param careClient - The careClient for this panel
   * @param mainFrame - The mainframe for this panel
   * @param myName - The name for this panel
   * @param nextName1 - The name for the second
panel
   * @param nextName2 - The name for the third panel
   */
  public CoordinationPanel(ServiceTree
serviceTree,CareClient careClient,MainFrame
mainFrame, String myName, String nextName1 ,String
nextName2)  {
    super();
    this.serviceTree = serviceTree;
    this.careClient = careClient;
    this.mainFrame = mainFrame;
    this.myName=myName;
    this.nextName1=nextName1;
    this.nextName2=nextName2;
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    this.setLayout(borderLayout1);
    selectButton.setText("Auswählen");
    nextButton.setText("Weiter");
    if (myName.equals("ServiceCataloguePanel"))
      jTree1 = serviceTree.getTree(0);
    else if (myName.equals("Hausnotrufdienste"))
      jTree1 = serviceTree.getTree(1);
    else if (myName.equals("Mahlzeitendienste"))
      jTree1 = serviceTree.getTree(2);
    nextButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        nextButton_actionPerformed(e);
      }
    });
    selectButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        selectButton_actionPerformed(e);
      }
    });
    jTree1.addTreeSelectionListener(new
TreeSelectionListener() {
      public void valueChanged(TreeSelectionEvent e) {
        treeListener(e);
      }
    });
    jPanel1.add(selectButton, null);
    jPanel1.add(nextButton, null);
    jScrollPane1.getViewport().add(jTree1);
    jScrollPane2.getViewport().add(textPane, null);
    jSplitPane1.add(jScrollPane1, JSplitPane.LEFT);
    jSplitPane1.add(jScrollPane2, JSplitPane.RIGHT);
    jSplitPane1.setDividerLocation(200);
    this.add(jSplitPane1, BorderLayout.CENTER);
    if (!myName.equals("ServiceCataloguePanel"))
      this.add(jPanel1, BorderLayout.SOUTH);
    ServiceItemNode sin =
((ServiceItemNode)(jTree1.getModel()).getRoot());
    jTree1.setSelectionPath(new TreePath(sin));
  }
  /**
   * Listens for selection in the serviceTree and displays the
corresponding content
   * or HTML page on the left side of the JSplitPane.
   */
  public void treeListener(TreeSelectionEvent e)  {
    if (!jTree1.isSelectionEmpty()) {
      String name =
jTree1.getLastSelectedPathComponent().getClass().getName();
      //System.out.println (name);
      if (name.equals("com.server.ServiceItemNode")) {
        ServiceItemNode node =
(ServiceItemNode)jTree1.getLastSelectedPathComponent();
        //System.out.println(node.getName());
        Object tempValue = node.getParent();
        if (node.isRoot())
          selectPage(node.getName());
        else if(tempValue instanceof ServiceItemNode) {
          if (((ServiceItemNode)node.getParent()).isRoot()) {
            selectPage(node.getName());
          }
          else {
            ServiceItemNode serviceItemNode =
(ServiceItemNode)node.getParent();
            ServiceNode serviceNode =
(ServiceNode)serviceItemNode.getParent();
            Service service = (Service)serviceNode.getService();
            displayContactInfos(node.getName(),service);
          }
        }
        else if (node.getName().equals("Kontaktinformationen"))  {
          ServiceNode serviceNode =
(ServiceNode)node.getParent();
          Service service = (Service)serviceNode.getService();
          displayContacts(node.getName(),service);
        }
        else if (node.getName().equals("Leitstungskatalog"))  {
          ServiceNode serviceNode =
(ServiceNode)node.getParent();
          Service service = (Service)serviceNode.getService();
          displayServiceCatalogues(node.getName(),service);
        }
        else if (node.getName().equals("Sonstiges"))  {
          ServiceNode serviceNode =
(ServiceNode)node.getParent();
          Service service = (Service)serviceNode.getService();
          displayOther(node.getName(),service.getOther());
        }
        else { //if(tempValue instanceof ServiceNode) {
          //ServiceNode node = (ServiceNode)node.getParent();
          displayInfo(node.getName());
        }
      }
      else if (name.equals("servicecatalogue.Node")) {
        Node node =
(Node)jTree1.getLastSelectedPathComponent();
      }
```

```
    else if (name.equals("com.server.ServiceNode"))  {
      ServiceNode node =
(ServiceNode)jTree1.getLastSelectedPathComponent();
      selectPage(node.getName());
      //System.out.println(node.getName());
    }
   }
   else  {
    ServiceItemNode sin =
((ServiceItemNode)(jTree1.getModel()).getRoot());
    jTree1.setSelectionPath(new TreePath(sin));
   }
  }
  /**
   * Used to display simple text on the left side of the
JSplitPane.
   */
  public void displayInfo(String nodeInfo)  {
   try {
    System.out.println("in displayInfo "+nodeInfo);
    Document doc = textPane.getDocument();
    doc.remove(0,doc.getLength());
    doc = textPane.getDocument();

doc.insertString(doc.getLength(),nodeInfo,textPane.get
Style("new") );
   }
   catch (Exception x) {
    System.out.println("Fehler in displayInfo "+ x);
   }
  }
  /**
   * Used to display the contact information of a service
on the left side of the JSplitPane.
   */
  public void displayContactInfos(String nodeInfo,
Service service)  {
   try {
    initStylesForTextPane(textPane);
    Document doc = textPane.getDocument();
    doc.remove(0,doc.getLength());
    doc = textPane.getDocument();

doc.insertString(doc.getLength(),nodeInfo+":",textPane.
getStyle("new") );
    String newLine = "\n";
    if (nodeInfo.equals("Adressen")) {
     Vector adresses = service.getAdresses();
     doc = textPane.getDocument();
     doc.insertString(doc.getLength(),"\n\nAdressen für
: ",textPane.getStyle("boldNew2") );
     String tempString =service.getName()+"\n";
     doc = textPane.getDocument();

doc.insertString(doc.getLength(),tempString,textPane.g
etStyle("largeNew2") );
      for (int i=0;i<adresses.size();i++) {
       Adress adress = (Adress)adresses.elementAt(i);

       doc = textPane.getDocument();
       doc.insertString(doc.getLength(),"Straße :
",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),adress.getStreet(),text
Pane.getStyle("largeNew") );

       doc = textPane.getDocument();
       doc.insertString(doc.getLength()," 
Hausnummer : ",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),adress.getHouseNumber()+"\n"
,textPane.getStyle("largeNew") );

       doc = textPane.getDocument();
       doc.insertString(doc.getLength(),"Ort :
",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),adress.getTown(),textPane.getS
tyle("largeNew") );

       doc = textPane.getDocument();
       doc.insertString(doc.getLength()," 
Postzeitzahl :
",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),adress.getZipCode()+"\n",textP
ane.getStyle("largeNew") );

      }
     }
     else if (nodeInfo.equals("E-Mail Adressen")) {
      Vector eMails = service.getEMails();
      doc = textPane.getDocument();
      doc.insertString(doc.getLength(),"\n\nE-Mail Adressen für :
",textPane.getStyle("boldNew2") );
      String tempString =service.getName()+"\n";
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),tempString,textPane.getStyle("l
argeNew2") );
      for (int i=0;i<eMails.size();i++) {
       EMail eMail = (EMail)eMails.elementAt(i);

       doc = textPane.getDocument();
       doc.insertString(doc.getLength(),"E-Mail Adresse :
",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),eMail.getEMail()+"\n",textPan
e.getStyle("largeNew") );
      }
     }
     else if (nodeInfo.equals("Telefonnummern")) {
      Vector phoneNumbers = service.getPhoneNumbers();
      doc = textPane.getDocument();
      doc.insertString(doc.getLength(),"\n\nTelefonnummern für
: ",textPane.getStyle("boldNew2") );
      String tempString =service.getName()+"\n";
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),tempString,textPane.getStyle("l
argeNew2") );
      for (int i=0;i<phoneNumbers.size();i++) {
       PhoneNumber phoneNumber =
(PhoneNumber)phoneNumbers.elementAt(i);

       doc = textPane.getDocument();
       doc.insertString(doc.getLength(),"Nummer :
",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),phoneNumber.getPhoneNumbe
r(),textPane.getStyle("largeNew") );

       doc = textPane.getDocument();
       doc.insertString(doc.getLength()," 
Kommentar :
",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),phoneNumber.getComment()+
"\n",textPane.getStyle("largeNew") );
```

```
      }
     }
     else if (nodeInfo.equals("Bankverbindungen")) {
      Vector bankAccounts =
service.getBankAccounts();
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),"\n\nBankverbindunge
n für : ",textPane.getStyle("boldNew2") );
      String tempString =service.getName()+"\n";
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),tempString,textPane.g
etStyle("largeNew2") );
      for (int i=0;i<bankAccounts.size();i++) {
       BankAccount bankAccount =
(BankAccount)bankAccounts.elementAt(i);

       doc = textPane.getDocument();
       doc.insertString(doc.getLength(),"Name der Bank
: ",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),bankAccount.getBank
Name(),textPane.getStyle("largeNew") );

       doc = textPane.getDocument();
       doc.insertString(doc.getLength()," Bankleitzahl
: ",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),bankAccount.getBank
Code()+"\n",textPane.getStyle("largeNew") );

       doc = textPane.getDocument();
       doc.insertString(doc.getLength(),"Kontonummer
: ",textPane.getStyle("boldNew") );
       doc = textPane.getDocument();

doc.insertString(doc.getLength(),bankAccount.getAcco
untNumber()+"\n",textPane.getStyle("largeNew") );

      }
     }
    }
    catch (Exception x) {
     System.out.println("Fehler in displayContacts "+ x);
    }
  }
  /**
   * Used to display the contacts of a service on the left
side of the JSplitPane.
   */
  public void displayContacts(String nodeInfo, Service
service)  {
    try {
     initStylesForTextPane(textPane);
     Document doc = textPane.getDocument();
     doc.remove(0,doc.getLength());
     doc = textPane.getDocument();

doc.insertString(doc.getLength(),nodeInfo+":",textPane.
getStyle("new") );
     String newLine = "\n";
     Vector contacts = service.getContacts();
     doc = textPane.getDocument();

doc.insertString(doc.getLength(),"\n\nAnsprechpartner
für : ",textPane.getStyle("boldNew2") );
     String tempString =service.getName()+"\n";
     doc = textPane.getDocument();

doc.insertString(doc.getLength(),tempString,textPane.getStyle("l
argeNew2") );

     for (int i=0;i<contacts.size();i++) {
      Contact contact = (Contact)contacts.elementAt(i);

      doc = textPane.getDocument();
      doc.insertString(doc.getLength(),"Vorname :
",textPane.getStyle("boldNew") );
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),contact.getForename(),textPane
.getStyle("largeNew") );

      doc = textPane.getDocument();
      doc.insertString(doc.getLength()," Nachname :
",textPane.getStyle("boldNew") );
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),contact.getName()+"\n",textPa
ne.getStyle("largeNew") );

      doc = textPane.getDocument();
      doc.insertString(doc.getLength(),"Telefon :
",textPane.getStyle("boldNew") );
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),contact.getPhone(),textPane.get
Style("largeNew") );

      doc = textPane.getDocument();
      doc.insertString(doc.getLength()," Tätigkeit :
",textPane.getStyle("boldNew") );
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),contact.getJob()+"\n",textPane.
getStyle("largeNew") );
     }
    }
    catch (Exception x) {
     System.out.println("Fehler in displayContacts "+ x);
    }

  }
  /**
   * Used to display the service-catalogues of a service on the left
side of the JSplitPane.
   */
  public void displayServiceCatalogues(String nodeInfo, Service
service)  {
    try {
     initStylesForTextPane(textPane);
     Document doc = textPane.getDocument();
     doc.remove(0,doc.getLength());
     doc = textPane.getDocument();

doc.insertString(doc.getLength(),nodeInfo+":",textPane.getStyle
("new") );
     String newLine = "\n";
    }
    catch (Exception x) {
     System.out.println("Fehler in displayContent "+ x);
    }
  }
  /**
   * Used to display the other information about a service on the
left side of the JSplitPane.
   */
  public void displayOther(String nodeInfo, String other)  {
    try {
     initStylesForTextPane(textPane);
     Document doc = textPane.getDocument();
```

```
      doc.remove(0,doc.getLength());
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),nodeInfo+":",textPane.
getStyle("new") );
      String newLine = "\n";
      String tempString = newLine + other;
      doc = textPane.getDocument();

doc.insertString(doc.getLength(),tempString,textPane.g
etStyle("large") );
    }
    catch (Exception x) {
      System.out.println("Fehler in displayOther "+ x);
    }
  }
  /**
   * Loads a HTML page from the file system.
   * <P>
   * @param name - The name of the page to be loaded.
Sample: com/doc/<B>name</B>.html
   */
  public void selectPage(String name) {
    String s = null;
    String page = name;
    try {
      s =  "file:" + "com/doc/" + page+".html";
      //System.out.println(s);
      URL myURL = new URL(s);
      displayURL(myURL);
    }
    catch (Exception e) {System.err.println("Couldn't
create URL: " + s + e);
    }
  }
  /**
   * Displays an URL in the textPane.
   * <P>
   * @param url - The URL to be displayed
   */
  public void displayURL(URL url) {
    try {
      textPane.setPage(url);
    }
    catch (Exception e) {
      System.err.println("Attempted to read a bad URL: "
+ url);
    }
  }
  /**
   * Tells the tree to refresh.
   */
  public void refreshTree() {
    jTree1.repaint();
  }
  /**
   * Updates the data for a service in the tree.
   * <P>
   * @param service - the Service to be updated in the
tree
   */
  public void refreshService(Service service)  {
    if (myName.equals("ServiceCataloguePanel")) {
      try {
        String serviceName = service.getName();
        TreeModel tm = jTree1.getModel();
        ServiceItemNode root =
(ServiceItemNode)tm.getRoot();
        for (int i=0;i< root.getChildCount();i++) {
          ServiceItemNode group =
(ServiceItemNode)root.getChildAt(i);
          for (int a=0;a< group.getChildCount();a++) {
            ServiceNode serviceNode =
(ServiceNode)group.getChildAt(a);
            if (serviceNode.name.equals(serviceName)) {
              service.setConnected(true);
              serviceNode.setService(service);
            }
          }
        }
      }
      catch (Exception x) {
        System.out.println("Fehler in
ServiceCataloguePanel.refreshService "+ x);
      }
    }
    else {
      try {
        String serviceName = service.getName();
        TreeModel tm = jTree1.getModel();
        ServiceItemNode root = (ServiceItemNode)tm.getRoot();
        for (int i=0;i< root.getChildCount();i++) {
          ServiceNode serviceNode =
(ServiceNode)root.getChildAt(i);
          System.out.println("refreshService  "+serviceName);
          if (serviceNode.name.equals(serviceName)) {
            service.setConnected(true);
            serviceNode.setService(service);
          }
        }
      }
      catch (Exception x) {
        System.out.println("Fehler in
CoordinationPanel.refreshService "+ x);
      }
    }
  }
  /**
   * Initalizes some text-styles for a given JTextPane.
   * <P>
   * @param textPane - The textPane
   */
  protected void initStylesForTextPane(JTextPane textPane) {
    //Initialize some styles.
    Style def =
StyleContext.getDefaultStyleContext().getStyle(StyleContext.D
EFAULT_STYLE);
    Style regular = textPane.addStyle("regular", def);
    StyleConstants.setFontFamily(def, "SansSerif");

    Style s = textPane.addStyle("italic", regular);
    StyleConstants.setItalic(s, true);

    s = textPane.addStyle("boldNew", regular);
    StyleConstants.setBold(s, true);
    StyleConstants.setFontSize(s, 12);

    s = textPane.addStyle("boldNew2", regular);
    StyleConstants.setBold(s, true);
    StyleConstants.setFontSize(s, 14);

    s = textPane.addStyle("small", regular);
    StyleConstants.setFontSize(s, 10);

    s = textPane.addStyle("large", regular);
    StyleConstants.setFontSize(s, 16);

    s = textPane.addStyle("largeNew", regular);
    StyleConstants.setFontSize(s, 12);

    s = textPane.addStyle("largeNew2", regular);
    StyleConstants.setFontSize(s, 14);

    s = textPane.addStyle("new", regular);
    StyleConstants.setFontSize(s, 24);
```

```
    StyleConstants.setAlignment(s,
StyleConstants.ALIGN_JUSTIFIED);
    StyleConstants.setUnderline(s, true);

    s = textPane.addStyle("icon", regular);
    StyleConstants.setAlignment(s,
StyleConstants.ALIGN_CENTER);
    StyleConstants.setIcon(s, new
ImageIcon("images/Pig.gif"));

    s = textPane.addStyle("button", regular);
    StyleConstants.setAlignment(s,
StyleConstants.ALIGN_CENTER);
    JButton button = new JButton(new
ImageIcon("images/sound.gif"));
    button.setMargin(new Insets(0,0,0,0));
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        Toolkit.getDefaultToolkit().beep();
      }
    });
    StyleConstants.setComponent(s, button);
  }
  /*
   * Sets the serviceTree for this Panel.
   * <P>
   * @param serviceTree -
   */
  public void setServiceTree(ServiceTree serviceTree)
{
    this.serviceTree = serviceTree;
  }
  /*
   * Sets the careClient for this Panel.
   * <P>
   * @param careClient -
   */
  public void setCareClient(CareClient careClient) {
    this.careClient=careClient;
  }
  /**
   * Shows an optionDialog and asks for sending a
message to a services.
   * <P>
   * @param node - A ServiceNode : it's containig the
service which should receive the message
   */
  public void showDialog(ServiceNode node)  {
    Vector messageVector =
mainFrame.getMessageVector();
    Object[] options = { "Ja", "Nein" };
    Service service= (Service)node.getService();
    if (markAsCoordinated(service.getName())) {
      int selectedOption =
JOptionPane.showOptionDialog(this, "Sie haben den
Dienst  "+node.name+" gewählt.\nWollen Sie die
Kundendaten an diesen Dienst senden ?",
"Dienstauswahl",

JOptionPane.DEFAULT_OPTION,
JOptionPane.WARNING_MESSAGE,null, options,
options[0]);

//System.out.println("selectedOption"+selectedOption);
      if (selectedOption == 0)  {
        Message message = new
Message(careClient,service,"Ein Kunde hat Leistungen
Ihres Dienstes angefordert !\nWollen Sie die Daten
dieses Kunden empfangen ?");
        messageVector.add(message);
        if (service.getConnected())
          mainFrame.sentMessages(service.getName());
      }
    }
  }
  /**
   * If the select-button is pressed this mehtod determines the
selected node and calls showDialog.
   * <P>
   * @see com.server.CoordinationPanel#showDialog
   */
  void selectButton_actionPerformed(ActionEvent e) {
    if (!jTree1.isSelectionEmpty()) {
      if
(jTree1.getLastSelectedPathComponent().getClass().getName().
equals("com.server.ServiceNode"))  {
        ServiceNode node =
(ServiceNode)jTree1.getLastSelectedPathComponent();
        textPane.setText("Gewählt wurde "+node.name);
        showDialog(node);
      }
      else {
        if
(jTree1.getLastSelectedPathComponent().getClass().getName().
equals("com.server.ServiceItemNode"))  {
          ServiceItemNode node =
(ServiceItemNode)jTree1.getLastSelectedPathComponent();
          if (!node.isRoot()) {

while(!(((Object)node.getParent())).getClass().getName().equals
("com.server.ServiceNode"))  {
            node = (ServiceItemNode)node.getParent();
          }
          ServiceNode node2 = (ServiceNode)node.getParent();
          showDialog(node2);
          textPane.setText("Gewählt wurde "+node2.name);
        }
      }
    }
  }
  /**
   * If the next-button is pressed this mehtod determines if there
are any other services to be
   * coordinated and calls the corresponding
<B>CoodinatoinPanel</B>.
   */
  void nextButton_actionPerformed(ActionEvent e) {
    int i = mainFrame.jTabbedPane1.getSelectedIndex();
    Vector requestedServiceVector =
careClient.getRequestedServices();
    if (!requestedServiceVector.isEmpty())  {
      for (int a=0;a<requestedServiceVector.size();a++) {
        RequestedService tempService =
(RequestedService)requestedServiceVector.elementAt(a);
        if (tempService.getName().equals(nextName1)) {
          mainFrame.jTabbedPane1.setSelectedIndex(i+1);
          break;
        }
        else if (tempService.getName().equals(nextName2))  {
          mainFrame.jTabbedPane1.setSelectedIndex(i+2);
          break;
        }
      }
      //JOptionPane.showMessageDialog(this,"Es wurden KEINE
weiteren Dienste für eine Koordinierung
gewählt","Koordinierung",JOptionPane.ERROR_MESSAGE);
    }
    else
      JOptionPane.showMessageDialog(this,"Es wurden KEINE
Dienste für eine Koordinierung
gewählt","Koordinierung",JOptionPane.ERROR_MESSAGE);
  }
  /**
   * Sets a status for a requestedService.
   * <P>
```

```
   * @param serviceName - The name of the service
   */
  public boolean markAsCoordinated(String
serviceName) {
    boolean status = false;
    Vector requestedServiceVector =
careClient.getRequestedServices();
    if (!requestedServiceVector.isEmpty())  {
      for (int i=0;i<requestedServiceVector.size();i++) {
        RequestedService tempRequestedService =
(RequestedService)requestedServiceVector.elementAt(i
);
       if
(tempRequestedService.getName().equals(myName)) {
         tempRequestedService.setStatus("  - koordiniert an Dienst
-  "+serviceName);
         status = true;
        }
       }
     }
    if (!status)
      JOptionPane.showMessageDialog(this,"Es wurde kein
"+myName+" für eine Koordinierung
gewählt","Koordinierung",JOptionPane.ERROR_MESSAGE);
    return status;
  }

}// end of file
```

Die Klasse Message

Dies ist eine reine Datenklasse, die nur dazu dient verschiedene Objekte zur Datenübertragung zusammenzufassen.

```
// Copyright (c) 2000
package com.server;

import person.*;
import java.io.Serializable;
import java.sql.*;

/**
 * A data class for messages.
 * <P>
 * @author Martin Braches
 */
public class Message extends Object implements
Serializable {

  private CareClient careClient;
  private Service service;
  private String message;
  private Timestamp timestamp;
  private ServiceProfile serviceProfile;
  private int messageType=0;

  /**
   * Constructor
   */
  public Message() {
  }
  /**
   * Constructor
   * <P>
   * @param careClient - the object of the message
   * @param service - the receiver
   * @param message - the message
   */
  public Message(CareClient careClient, Service
service, String message) {
    this.careClient=careClient;
    this.service=service;
    this.message=message;
  }
  /**
   * Constructor
   * <P>
   * @param service - the receiver
   * @param message - the message
   * @param serviceProfile - the object of the message
   */
  public Message(Service service, String message ,ServiceProfile
serviceProfile) {
    this.service=service;
    this.message=message;
    this.serviceProfile=serviceProfile;
  }

  public CareClient getCareClient() {
    return this.careClient;
  }

  public void setCareClient(CareClient careClient)  {
    this.careClient=careClient;
  }

  public Service getService(){
    return this.service;
  }

  public void setService(Service service) {
    this.service=service;
  }

  public String getMessage()  {
    return this.message;
  }

  public void setMessage(String message){
    this.message=message;
  }

  public int getMessageType()  {
    return this.messageType;
  }

  public void setMessageType(int messageType){
    this.messageType=messageType;
  }

  public String toString()  {
    return this.message;
  }

  public ServiceProfile getServiceProfile()  {
    return this.serviceProfile;
  }

  public void setServiceProfile(ServiceProfile serviceProfile){
```

```
   this.serviceProfile=serviceProfile;
  }
}
```

Die Klasse Server

Sie implementiert die RMI-Server Funktionen, die in der Klasse *ServerInterface* definiert werden.

```
 // Copyright (c) 2000
package com.server;

import javax.swing.*;
import com.client.ClientInterface;
import java.security.*;
import javax.crypto.*;
import java.lang.Runnable;
import java.util.Vector;
import person.*;
import java.io.*;

/**
 * A class which represents the RMI functionality for
the server.
 * <P>
 * @author Martin Braches
 */
public class Server extends
java.rmi.server.UnicastRemoteObject implements
ServerInterface, Runnable{

  private MainFrame mainFrame=null;
  private boolean encode = false;
  private PrivateKey serverPrivateKey = null;
  private PublicKey serverPublicKey = null;
  private Cipher rsa = null;
  private Thread keyThread = null;
  private Vector serviceVector;
  /**
   * Constructor
   * <P>
   * @param mainFrame
   */
  public Server(MainFrame mainFrame) throws
java.rmi.RemoteException {
    super();
    this.mainFrame= mainFrame;
    if (encode) {
      keyThread = new Thread(this, "Key");
      keyThread.start();
    }
    serviceVector = mainFrame.getServiceVector();
  }
  /**
   * An unused method - usable as pattern for decryption
of sealed-objects.
   * <P>
   * @param serviceName
   * @param object
   */
  public boolean setObject(String serviceName, Object
object) throws java.rmi.RemoteException {
    boolean status=true;
    try {
      if (encode) {
        SealedObject so = (SealedObject)object;

rsa.init(Cipher.DECRYPT_MODE,serverPrivateKey);
        String test3 = (String)so.getObject(rsa);
      }
    }
    catch (Exception e) {
      System.out.println("Fehler in Server.setObject "+e);
    }
    return status;
  }
  /**
   * Traverses the message-vector an returns a careClient to the
calling service.
   * Optionaly with encryption.
   * <P>
   * @param receiverName
   */
  public Object getCareClient(String receiverName) throws
java.rmi.RemoteException {
    try {
      Vector messageVector = mainFrame.getMessageVector();
      //System.out.println("messageVector  ="+messageVector);
      for(int i=0;i<messageVector.size();i++)  {
        Message message = (Message)messageVector.elementAt(i);
        if (message.getService() != null) {
          Service service = (Service)message.getService();
          CareClient careClient =
(CareClient)message.getCareClient();
          String serviceName = service.getName();
          //System.out.println("name 99"+serviceName+"
"+name);
          if (serviceName.equals(receiverName)) {
            if (encode) {
              Security.addProvider(new
au.net.aba.crypto.provider.ABAProvider());
              rsa = Cipher.getInstance("RSA","ABA");

rsa.init(Cipher.ENCRYPT_MODE,service.getPublicKey());
              SealedObject sealedObject = new
SealedObject((Serializable)careClient,rsa);
              return sealedObject;
            }
            else  {
              messageVector.removeElementAt(i);
              return careClient;
            }
          }
        }
      }
    }
    catch (Exception e) {
      System.out.println("Fehler in Server.getObject "+e);
    }
    return null;
  }
  /**
   * Traverses the message-vector an returns a serviceProfile to
the calling service.
   * Optionaly with encryption.
   * <P>
   * @param receiverName
   */
  public Object getServiceProfile(String receiverName) throws
java.rmi.RemoteException {
    try {
      Vector messageVector = mainFrame.getMessageVector();
      Message message = new Message();
```

```
     for(int i=0;i<messageVector.size();i++)  {
       message = (Message)messageVector.elementAt(i);
       if (message.getService() != null) {
         Service service = (Service)message.getService();
         String serviceName = service.getName();
         ServiceProfile serviceProfile
=(ServiceProfile)message.getServiceProfile();
         if (serviceName.equals(receiverName)) {
           if (encode) {
             Security.addProvider(new
au.net.aba.crypto.provider.ABAProvider());
             rsa = Cipher.getInstance("RSA","ABA");

rsa.init(Cipher.ENCRYPT_MODE,service.getPublicKe
y());
             SealedObject sealedObject = new
SealedObject((Serializable)serviceProfile,rsa);
             return sealedObject;
           }
           else  {
             messageVector.removeElementAt(i);
             return serviceProfile;
           }
         }
       }
     }
   }
   catch (Exception e) {
     System.out.println("Fehler in
Server.getServiceProfile "+e);
   }
   return null;
  }
  /**
   * Registers a new Client to the server.
   * Also tells all trees to refresh themselves.
   * <P>
   * @param name - the name of the client
   * @param clientInterface - used for callBacks to the
client
   * @param clientPublicKey - used to encrypt objects
for this client
   */
  public boolean register(String serviceName,
ClientInterface clientInterface, PublicKey
clientPublicKey)  {
    Service service;
    try  {
      for(int i=0;i<serviceVector.size();i++)  {
       ServiceItemNode serviceItemNode =
(ServiceItemNode)serviceVector.elementAt(i);
       Vector childServicesVector =
serviceItemNode.getChildServicesVector();
       for(int a=0;a<childServicesVector.size();a++)  {
         ServiceNode serviceNode =
(ServiceNode)childServicesVector.elementAt(a);
         service = serviceNode.getService();
         String serviceName2 = service.getName();
         int typeID = service.getTypeID().intValue();
         if (serviceName2.equals(serviceName)) {
           if (!service.getConnected())  {
             //System.out.println("serviceName
"+serviceName+"  name "+name);
             service.setConnected(true);
             service.setClientInterface(clientInterface);
             if (encode)
service.setPublicKey(clientPublicKey);
             sentMessages(serviceName);
             mainFrame.refreshTree();
             if (encode)

clientInterface.setServerKey(serverPublicKey);
             return true;
           }
         }
       }
     }
    }
    catch(Exception xc) {
      System.out.println("Fehler in Register "+ xc);
    }
    return false;
  }
  /**
   * Unregisters a Client from the server.
   * Also tells all trees to refresh themselves.
   * <P>
   * @param name - the name of the client
   */
  public boolean disconnect(String serviceName) {
    boolean status=true;
    try  {
      for(int i=0;i<serviceVector.size();i++)  {
       ServiceItemNode serviceItemNode =
(ServiceItemNode)serviceVector.elementAt(i);
       Vector childServicesVector =
serviceItemNode.getChildServicesVector();
       for(int a=0;a<childServicesVector.size();a++)  {
         ServiceNode serviceNode =
(ServiceNode)childServicesVector.elementAt(a);
         Service service = serviceNode.getService();
         String serviceName2 = service.getName();
         if (serviceName2.equals(serviceName)) {
           service.setConnected(false);
           service.setClientInterface(null);
           service.setPublicKey(null);
           mainFrame.refreshTree();
           return status;
         }
       }
      }
    }
    catch(Exception xc) {
      System.out.println("Fehler in disconnect " +xc);
    }
    status=false;
    return status;
  }
  /**
   * Generate a Key pair for a RSA algorithm.
   * It creates a public/private key pair with the length
   * the length of the parameter <code>keysize</code>.
   * <P>
   * @param keysize
   */
  public void generateKey(int length) {
    try {
      System.out.print("erzeuge Schlüssel...");
      KeyPairGenerator kpg =
KeyPairGenerator.getInstance("RSA"); //=> Keygenerator
erzeugen
      System.out.print(".");
      kpg.initialize(length, new SecureRandom());   //2048 dauert
sehr lange  //=> Keygenerator initialisieren
      System.out.print(".");
      KeyPair kp = kpg.generateKeyPair();                    //=>
Schlüsselpaar erzeugen
      System.out.print(".");
      serverPrivateKey = kp.getPrivate();                      //=>
gibt den privaten Schlüssel aus
      serverPublicKey = kp.getPublic();                        //=>
gibt den öffentl. Schlüssel aus
      System.out.println("Beendet !!");
      /* lets add a "save" Provider */
      Security.addProvider(new
au.net.aba.crypto.provider.ABAProvider());
```

```
      rsa = Cipher.getInstance("RSA","ABA");
      }
    catch (Exception e)
    {
      System.out.println("Fehler in generateKey" + e);
    }
  }
  /**
   * If Key - Gereation is enabled it could be run as
Thread.
   * This is the run Method of the Thread.
   */
  public void run() {
    System.out.print("running!");
    generateKey(512);
  }
  /**
   * Enables a Client to call the sentMessages method.
   * <P>
   * @param serviceName
   * @see com.server.Server#sentMessages
   */
  public boolean getMessages(String serviceName)
throws java.rmi.RemoteException {
    try {
      return sentMessages(serviceName);
    }
    catch (Exception ex)  {
      System.out.println("Fehler in getMessages" +ex);
    }
    return false;
  }
  /**
   * Sents all messages to a Client.
   * <P>
   * @param serviceName
   */
  public boolean sentMessages(String serviceName)  {
    boolean status = false;
    Vector messageVector =
mainFrame.getMessageVector();
    /** Hier werden die Nachrichten für den Dienst mit
dem Namen *serviceName* gesendet **/
    for(int i=0;i<messageVector.size();i++)  {
      Message message =
(Message)messageVector.elementAt(i);
      //Message newMessage = new
Message(message.getCareClient(),message.getService()
,message.getMessage());
      if (message.getService() != null) {
        Service service = (Service)message.getService();
        String name = service.getName();
        if (name.equals(serviceName)) {
          try {

((ClientInterface)service.getClientInterface()).setMessage(messa
ge.getMessage(),message.getMessageType());
          }
          catch (Exception ex)  {
            System.out.println("Fehler in Server.sentMessages "+ex);
          }
        }
      }
    }
    return status;
  }
  /**
   * Used to sent a Message from Client to Server.
   * <P>
   * @param secureCareClient -
   * @param serviceName -
   * @param message -
   * <P>
   * @see com.server.MainFrame#setService
   */
  public void setMessage(Object secureCareClient, String
serviceName, String message) throws java.rmi.RemoteException
{
    mainFrame.showMessage(secureCareClient, serviceName,
message );
  }
  /**
   * Used to sent the ServiceData back to Server.
   * <P>
   * @param service
   */
  public void setService(Service service) throws
java.rmi.RemoteException {
    mainFrame.setService(service);
  }
  /**
   * Used to get the ServiceData from Server.
   * <P>
   * @param serviceName
   */
  public Service getService(String serviceName) throws
java.rmi.RemoteException  {
    Service service = new Service(serviceName);
    return service;
  }
}
```

Die Klasse ServerInterface

Diese Klasse definiert die Methoden, welche dem Koordinationsclient über RMI zur Verfügung stehen.

```
// Copyright (c) 2000
package com.server;

import java.rmi.*;
import com.client.ClientInterface;
import person.*;

import java.security.*;
/**
 * The RMI ServerInterface. It defines the methodes
which could be called on the Server via RMI.
 * <P>
 * @author Martin Braches
 */
public interface ServerInterface extends java.rmi.Remote {

  /** Used to register as Client. */
  public boolean register(String serviceName, ClientInterface
clientInterface, PublicKey clientPubicKey) throws
java.rmi.RemoteException;
  /** Used to unregister as Client. */
  public boolean disconnect(String serviceName) throws
java.rmi.RemoteException;
  /** Used to tell the Server to sent the Messages for this
Service.*/
```

```
  public boolean getMessages(String serviceName)
throws java.rmi.RemoteException;
  /** Used to sent a Message from Client to Server. */
  public void setMessage(Object secureCareClient,
String serviceName, String message) throws
java.rmi.RemoteException;
  /** Used to get the CareClient Data. */
  public Object getCareClient(String receiverName)
throws java.rmi.RemoteException;
  /** Used to get the ServiceProfile. */
  public Object getServiceProfile(String receiverName)
throws java.rmi.RemoteException;
  /** Used to sent the ServiceData back to Server. */
  public void setService(Service service) throws
java.rmi.RemoteException;
  /** Used to get the ServiceData from Server. */
  public Service getService(String serviceName) throws
java.rmi.RemoteException;
  /** Unused until now. */
  public boolean setObject(String serviceName, Object object)
throws java.rmi.RemoteException;

}
```

Die Klasse ServerContainerThread

Sie wird verwendet, wenn der Koordinationsserver auf dem Koordinationsclient Methoden aufruft, die eine Reaktion des Benutzers erfordern. Da der Koordinationsserver blockieren würde, bis der Benutzer des Client reagiert, werden diese Methoden in einem eigenen Thread ausgeführt.

```
// Copyright (c) 2000
package com.server;
/**
 * A simpel thrad class which is used to call methodes
on the Server object in an separate thread.
 * <P>
 * @author Martin Braches
 */
public class ServerContainerThread extends Thread {

  private Server server = null;
  private String serviceName = "";
  /**
   * Constructor
   */
  public ServerContainerThread(String threadName,
Server server, String serviceName) {
    super(threadName);
    this.server = server;
    this.serviceName = serviceName;
  }

  public void run() {
    try {
       server.sentMessages(serviceName);
    }
    catch (Exception e) {
      System.out.println("Fehler in ServerContainerThread.run "+
e);
    }
  }
}
```

Die Klasse ServiceTree

Sie stellt die Baumstruktur des Servicekatalogs dar.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.tree.*;
import java.util.Vector;
import servicecatalogue.*;
import servicecatalogue.explorer.*;
import SwiftConstants;
import generic.*;
import java.sql.*;
import person.*;

/**
 * A class that represents the serviceTree.
 * <P>
 * @author Martin Braches
 */
public class ServiceTree {

  private JDBCEngine db=new JDBCEngine("com");
  private JTree jTree1 = null;
  private Vector serviceTypesVector = null;
  private Vector serviceVector = null;
  private boolean debug = true;
  private JDBC myCatalogues = new JDBC("ServCatalogue");
  private Vector serviceCatalogues =
myCatalogues.getServiceCatalogues();

  /**
   * Constructor
   */
  public ServiceTree() {
    serviceTypesVector = getServiceTypes();
    for(int i=0;i<serviceTypesVector.size();i++)  {
      ServiceItemNode serviceItemNode =
(ServiceItemNode)serviceTypesVector.elementAt(i);

serviceItemNode.setChildServicesVector(getServices(serviceIte
mNode.getID()));
    }
  }
  /**
   * Assembles the serviceTree from a database
```

```
  */
  public void initTree(int root)  {
    ServiceItemNode newRoot=null;
    int newRootID  = 0;
    if (root == 0)  {
      newRoot = new ServiceItemNode("alle Dienste");
      for(int i=0;i<serviceTypesVector.size();i++)  {
        ServiceItemNode serviceItemNode =
(ServiceItemNode)serviceTypesVector.elementAt(i);
        newRoot.add(serviceItemNode);
        Vector childServicesVector =
serviceItemNode.getChildServicesVector();
        for(int a=0;a<childServicesVector.size();a++)  {
          ServiceNode serviceNode =
(ServiceNode)childServicesVector.elementAt(a);
          serviceItemNode.add(serviceNode);
        }
      }
    }
    else  {
      for(int i=0;i<serviceTypesVector.size();i++)  {
        newRoot =
(ServiceItemNode)serviceTypesVector.elementAt(i);
        newRootID = newRoot.getID();
        if (newRootID == root)
          break;
      }
      Vector childServicesVector =
newRoot.getChildServicesVector();
      for(int a=0;a<childServicesVector.size();a++)  {
        ServiceNode serviceNode =
(ServiceNode)childServicesVector.elementAt(a);
        newRoot.add(serviceNode);
      }
    }
    jTree1 = new JTree(newRoot);
    jTree1.setCellRenderer(new ServiceTreeRenderer());

jTree1.getSelectionModel().setSelectionMode(TreeSele
ctionModel.SINGLE_TREE_SELECTION);
  }
  /**
   * Returns a Tree with a specific RootNode
   */
  public JTree getTree(int root)  {
    initTree(root);
    return jTree1;
  }
  /**
   * Returns a Vector containing all ServiceTypes from
the DB
   * each type is represented by an object of
ServiceItemNode.
   */
  public Vector getServiceTypes() {
    Vector serviceTypesVector = new Vector();
    try {
      db.connect();
      String query = "select * from ServiceTypes;";
      ResultSet rset = db.getSQLResult(query);
      if (rset != null)  {
        while(rset.next()) {
          ServiceItemNode serviceItemNode = new
ServiceItemNode(rset.getString(2),rset.getInt(1));
          serviceTypesVector.addElement(serviceItemNode);
        }
      }
      db.disconnect();
    }
    catch (Exception e) {
      System.out.println("Fehler in getAllServices: " +
e.getMessage());
    }
    return serviceTypesVector;
  }
  /**
   * Returns a Vector containing all Services of a specific type
from the DB
   * each service is represented by an object of Service.
   */
  public Vector getServices(int type) {
    Vector serviceNodeVector = new Vector();
    try {
      db.connect();
      String query = "select * from Service WHERE
TypeID="+type+";";
      ResultSet rset = db.getSQLResult(query);
      //if (debug) System.out.println(query+rset.wasNull());
      if (rset != null)  {
        while(rset.next()) {
          int ID = rset.getInt(1);
          //if (debug) System.out.println("ID = "+ID);
          Service service = new Service(ID);
          service.setDataFromDB(service.getID().intValue());
          ServiceNode serviceNode= new ServiceNode(service);
          int scatID = service.getServicecatalogueID().intValue();
          if (scatID != 0) {
            for(int i=0;i<serviceCatalogues.size();i++)  {
              ServiceCatalogue scat =
(ServiceCatalogue)serviceCatalogues.elementAt(i);
              if (scat.getCatalogueId() == scatID)  {
                //s.setServiceCatalogue(scat);
                serviceNode.addScatNode(scat.getAnchor());
                continue;
              }
            }
          }
          serviceNodeVector.addElement(serviceNode);
        }
      }
      db.disconnect();
    }
    catch (Exception e) {
      System.out.println("Fehler in getServices: " +
e.getMessage());
    }
    return serviceNodeVector;
  }
  /**
   * Returns the ServiceVector.
   */
  public Vector getServiceVector()  {
    return this.serviceTypesVector;
  }

}// end of file
```

Die Klasse ServiceTreeRenderer

Diese Klasse sorgt für unterschiedliche Darstellungen der einzelnen Ebenen des Servicekatalogs auf dem Bildschirm.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.tree.*;
import javax.swing.ImageIcon;
import java.awt.*;
import servicecatalogue.*;
import servicecatalogue.explorer.*;
import person.*;
/**
 * A class which extends
ServiceCatalogueTreeCellRenderer.
 * <P>
 * @author Martin Braches
 */
public class ServiceTreeRenderer extends
ServiceCatalogueTreeCellRenderer {
  ImageIcon icon1;
  ImageIcon icon2;
  ImageIcon icon3;

 /**
  * Constructor
  */
  public ServiceTreeRenderer() {
    icon1 = new
ImageIcon("Com/Images/ball1_green.gif");
    icon2 = new
ImageIcon("Com/Images/ball1_blue.gif");
    icon3 = new
ImageIcon("Com/Images/ball1_red.gif");
  }

  public Component
getTreeCellRendererComponent(JTree tree,Object
value,boolean sel,boolean expanded,boolean leaf,int
row,boolean hasFocus) {
    this.hasFocus=hasFocus;
                        this.selected=sel;
                        nodeContent=null;
    //System.out.println(value.getClass().getName());
    if (value instanceof ServiceNode) {
      ServiceNode serviceNode = (ServiceNode)value;
      //System.out.println(serviceNode.getUserObject());
      String name =(String)serviceNode.getUserObject();
      boolean connected =
((Service)serviceNode.getService()).getConnected();
      if (connected)
        setIcon(icon1);
      else
        setIcon(icon3);
      setText(name);

    }
                        else if (value instanceof Node) { // =>
value is not null!
                                renderNode=(Node)value;

            nodeContent=renderNode.getContent();
      setIcon(null);
                        }
    else if (value instanceof ServiceItemNode) { // => value is not
null!
                                ServiceItemNode serviceNode
= (ServiceItemNode)value;
      Object tempValue = serviceNode.getParent();
      if (tempValue instanceof ServiceItemNode) {
        ServiceItemNode tempServiceNode =
(ServiceItemNode)tempValue;
        if (tempServiceNode.isRoot())
          setIcon(null); //icon2);
      }
      else if (serviceNode.isRoot())
        setIcon(null);
      else
        setIcon(null);
      //System.out.println("w"+serviceNode.getUserObject());
      String name =(String)serviceNode.getUserObject();
      setText(name);
                        }
                        if (nodeContent==null ) {

            //setToolTipText("Leistungskataloge");
      setToolTipText(null);
                        }
                        else{
                                setText("
"+nodeContent.toString());

            setToolTipText(nodeContent.toString());
                        }
                        calcForegroundColor();
                        calcBackgroundColor();

            setComponentOrientation(tree.getComponentOrientat
ion());
                        return this;
  }
}
```

Die Klasse ServiceItemNode

Sie repräsentiert die einfachen Knoten des Servicekatalogs, die nur wenige Daten enthalten.

```
// Copyright (c) 2000
package com.server;

import javax.swing.tree.*;
import java.util.Vector;

/**
 * A class which extends DefaultMutableTreeNode with a few
textFields.
 * Used for treeNodes containig only a few data
 * <P>
```

```
 * @author Martin Braches
 */
public class ServiceItemNode extends
DefaultMutableTreeNode {

  String name = "";
  String status = "";
  String toolTip = "";
  String info="";
  String content="";
  int ID=0;
  private Vector childServicesVector;
  /**
   * Constructor
   */
  public ServiceItemNode() {
    super();
    this.childServicesVector = new Vector();
  }
  /**
   * Constructor
   * <P>
   * @param name
   */
  public ServiceItemNode(String name) {
    super(name);
    this.name=name;
    this.childServicesVector = new Vector();
  }
 /**
   * Constructor
   * <P>
   * @param name
   * @param ID
   */
  public ServiceItemNode(String name,int ID) {
    super(name);
    this.name=name;
    this.ID=ID;
    this.childServicesVector = new Vector();
  }
  /**
   * Constructor
   * <P>
   * @param name
   * @param tip
   */
  public ServiceItemNode(String name,String toolTip) {
    super(name);
    this.name=name;
    this.toolTip=toolTip;
    this.childServicesVector = new Vector();
  }
  /**
   * Constructor
   * <P>
   * @param name
   * @param toolTip
   * @param info
   */
  public ServiceItemNode(String name,String toolTip,String
info) {
    super(name);
    this.name=name;
    this.toolTip=toolTip;
    this.info = info;
    this.childServicesVector = new Vector();
  }

  public String getInfo() {
    return this.info;
  }
  public void setInfo(String inf) {
    this.info = inf;
  }
  public String getContent() {
    return this.content;
  }
  public void setContent(String content) {
    this.content = content;
  }

  public String getName() {
    return this.name;
  }

  public int getID() {
    return this.ID;
  }

  public Vector getChildServicesVector()  {
    return this.childServicesVector;
  }

  public void setChildServicesVector(Vector
childServicesVector)    {
    this.childServicesVector=childServicesVector;
  }
}
```

Die Klasse ServiceNode

Diese Klasse stellt die Baumknoten für die Darstellung der einzelnen Dienstanbieter im Servicekatalog bereit.

```
// Copyright (c) 2000
package com.server;

import person.*;
import java.util.Vector;
import servicecatalogue.*;
import javax.swing.tree.*;

/**
 * A class which extends DefaultMutableTreeNode with
a few textFields and a service object.
 * Also contains about seven subNodes.
 * <P>
 * @author Martin Braches
 */
public class ServiceNode extends DefaultMutableTreeNode {

  private Vector personVector = new Vector();
  private ServiceCatalogue sc = new ServiceCatalogue();
  private String description = "";
  private String specialOffer = "";
  private Service service = null;
  String name = "";
  String status = "";
```

```
  String toolTip = "";
  String info= "";
  int servicecatalogueID = 0;
  int number = 0;
  ServiceItemNode scatNode = new
ServiceItemNode("Leitstungskatalog");
  ServiceItemNode contactNode = new
ServiceItemNode("Kontaktinformationen");
  ServiceItemNode adressNode = new
ServiceItemNode("Adressen"); // und
Ansprechpartner");
  ServiceItemNode eMailNode = new
ServiceItemNode("E-Mail Adressen");
  ServiceItemNode phoneNode = new
ServiceItemNode("Telefonnummern");
  ServiceItemNode bankAccoundNode = new
ServiceItemNode("Bankverbindungen");
  ServiceItemNode otherNode = new
ServiceItemNode("Sonstiges");
  /**
   * Constructor
   */
  public ServiceNode() {
    super();
  }
  /**
   * Constructor
   * <P>
   * @param name
   */
  public ServiceNode(String name) {
    super(name);
    this.name=name;
  }
 /**
   * Constructs a serviceNode for a service
   * <P>
   * @param service
   */
  public ServiceNode(Service service) {
    super(service.getName());
    this.service=service;
    this.name = service.getName();
    initTree();
  }

  public void setServiceCatalogue(ServiceCatalogue scat)  {
    this.sc = scat;
  }

  public void initTree()  {
    contactNode.add(adressNode);
    contactNode.add(eMailNode);
    contactNode.add(phoneNode);
    contactNode.add(bankAccoundNode);
    this.add(contactNode);
    this.add(scatNode);
    this.add(otherNode);
  }

  public void addContactNode(Node node) {
    contactNode.add(node);
  }

  public void addScatNode(Node node) {
    scatNode.add(node);
  }

  public void addOhterNode(Node node) {
    otherNode.add(node);
  }

  public void setDescription(String description) {
    this.description = description;
  }
  public String getDescription() {
    return description;
  }

  public void setService(Service service)  {
    this.service=service;
  }

  public Service getService() {
    return this.service;
  }

  public String getName() {
    return this.name;
  }

}
```

Die Klasse ClientRequestPanel

Diese Klasse stellt die Maske *Kundenanfrage* dar.

```
// Copyright (c) 2000
package com.server;

import person.*;
import java.util.Vector;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import oracle.jdeveloper.layout.*;
import javax.swing.event.*;

/**
 * A GUI class which represents the Data for a
<code>ClientRequest</code> in a JPanel.
 * <P>
 * @author Martin Braches
 */
public class ClientRequestPanel extends GenericPanelGUI {

  private ContactInformationPanel contactData;
            private PersonalDataPanel personalData;
            private CareClient careClient;
  private MainFrame mainFrame = null;
  private CareClientPhysicianDialog ccpd  = null;
  private CareClientRequestedServicesDialog ccrsd = null;
  private CareClientPaymentInstitutionDialog ccpid = null;
  // GUI Elements
  BorderLayout borderLayout1 = new BorderLayout();
  BorderLayout borderLayout2 = new BorderLayout();
  JToolBar jToolBar1 = new JToolBar();
  JButton searchButton = new JButton();
  JButton requestedServicesButton = new JButton();
  JButton physicianButton = new JButton();
  JButton healthInsuranceButton = new JButton();
  JPanel topPanel = new JPanel();
  JPanel bottomPanel = new JPanel();
```

```
  JScrollPane remarkScrollPane = new JScrollPane();
  JTextArea remarkTextArea = new JTextArea();
  JPanel bottomButtonPanel = new JPanel();
  JButton saveButton = new JButton();
  JButton deleteButton = new JButton();
  JButton coordinationButton = new JButton();
  JButton newClientButton = new JButton();
  /**
   * Constructs a new instance.
   */
  public ClientRequestPanel(CareClient
careClient,MainFrame mainFrame) {
    this.careClient=careClient;
    this.mainFrame = mainFrame;
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
                              personalData = new
PersonalDataPanel(careClient.getName(),careClient.get
Forename(),

careClient.getBirthday(),careClient.getTownOfBirth(),c
areClient.getMartialStatus(),

careClient.getNationality(),careClient.getReligion());
    contactData = new
ContactInformationPanel(careClient.getAdresses(),care
Client.getEMails(),

careClient.getPhoneNumbers(),careClient.getBankAcco
unts());
    topPanel.setLayout(borderLayout1);
    topPanel.add(personalData,BorderLayout.NORTH);
    topPanel.add(contactData, BorderLayout.CENTER);
    searchButton.setLabel("Kundendaten suchen");
    searchButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        searchButton_actionPerformed(e);
      }
    });
    requestedServicesButton.setText("Dienste");
    requestedServicesButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        requestedServicesButton_actionPerformed(e);
      }
    });
    physicianButton.setText("Ärzte");
    physicianButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        physicianButton_actionPerformed(e);
      }
    });

healthInsuranceButton.setText("Kranken/Pflegekassen"
);
    healthInsuranceButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        healthInsuranceButton_actionPerformed(e);
      }
    });
    coordinationButton.setText("Koordinierung");
    coordinationButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        coordinationButton_actionPerformed(e);
      }
    });
    newClientButton.setText("neuer Kunde");
    newClientButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        newClientButton_actionPerformed(e);
      }
    });
    saveButton.setText("Daten speichern");
    saveButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        saveButton_actionPerformed(e);
      }
    });
    deleteButton.setText("Daten löschen");
    deleteButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        deleteButton_actionPerformed(e);
      }
    });
    jToolBar1.setName("Kundenanfrage Toolbar");
    jToolBar1.setBackground(SystemColor.textHighlight);
    jToolBar1.add(searchButton, null);
    jToolBar1.add(requestedServicesButton,null);
    jToolBar1.add(physicianButton, null);
    jToolBar1.add(healthInsuranceButton, null);
    Border remarkScrollPaneBevelBorder = new BevelBorder(1);
    Border remarkScrollPaneBorder = new
TitledBorder(remarkScrollPaneBevelBorder,"Bemerkung");
    remarkScrollPane.setBorder(remarkScrollPaneBorder);
    remarkScrollPane.getViewport().add(remarkTextArea);
    remarkScrollPane.setPreferredSize(new Dimension(500,70));
    bottomButtonPanel.add(coordinationButton, null);
    bottomButtonPanel.add(newClientButton, null);
    bottomButtonPanel.add(saveButton, null);
    bottomButtonPanel.add(deleteButton, null);
    bottomPanel.setLayout(borderLayout2);
    bottomPanel.add(remarkScrollPane, BorderLayout.CENTER);
    bottomPanel.add(bottomButtonPanel,
BorderLayout.SOUTH);
    this.add(jToolBar1, BorderLayout.NORTH);
    this.add(topPanel, BorderLayout.CENTER);
    this.add(bottomPanel, BorderLayout.SOUTH);
  }
  /**
   * Shows a SearchPersonDialog.
   * <p>
   * @see com.server.SearchPersonDialog
   */
  void searchButton_actionPerformed(ActionEvent e) {
    SearchPersonDialog spd = new
SearchPersonDialog(this,null,"Person Suchen",true);
    spd.search();
    spd.show();
  }
  /**
   * Shows a CareClientRequestedServicesDialog.
   * <p>
   * @see com.server.CareClientRequestedServicesDialog
   */
  void requestedServicesButton_actionPerformed(ActionEvent e)
{
    ccrsd = new
CareClientRequestedServicesDialog(careClient,null,"Dienste",tr
ue);
    ccrsd.show();
```

```
  }
  /**
   * Shows a CareClientPhysicianDialog.
   * <p>
   * @see com.server.CareClientPhysicianDialog
   */
  void physicianButton_actionPerformed(ActionEvent
e) {
    ccpd = new
CareClientPhysicianDialog(careClient,null,"Ärzte",true
);
    ccpd.show();
  }
  /**
   * Shows a CareClientPaymentInstitutionDialog.
   * <p>
   * @see
com.server.CareClientPaymentInstitutionDialog
   */
  void
healthInsuranceButton_actionPerformed(ActionEvent
e) {
    ccpid = new
CareClientPaymentInstitutionDialog(careClient,null,"K
assen",true);
    ccpid.show();
  }
  /**
   * Determines which Panel should be selected if the
Next button is pressed and displays it.
   *
   */
  void
coordinationButton_actionPerformed(ActionEvent e) {
    int i = mainFrame.jTabbedPane1.getSelectedIndex();
    Vector tempRS = careClient.getRequestedServices();
    if (!tempRS.isEmpty())  {
      for (int a=0;a<tempRS.size();a++) {
        RequestedService tempService =
(RequestedService)tempRS.elementAt(a);
        if
(tempService.getName().equals("Hausnotrufdienste")) {
          mainFrame.jTabbedPane1.setSelectedIndex(i+1);
          break;
        }
        else if
(tempService.getName().equals("Mahlzeitendienste"))
{
          mainFrame.jTabbedPane1.setSelectedIndex(i+2);
          break;
        }
        else if (tempService.getName().equals("ambulante
Pflegedienste"))  {
          mainFrame.jTabbedPane1.setSelectedIndex(i+3);
          break;
        }
      }
    }
    else
      JOptionPane.showMessageDialog(this,"Es wurden
KEINE Dienste für eine Koordinierung
gewählt","Koordinierung",JOptionPane.ERROR_MES
SAGE);
  }
  /**
   * Creates a new CareClient.
   */
  void newClientButton_actionPerformed(ActionEvent
e) {
    this.careClient = new CareClient();
    showCareClient();
  }
  /**
   * Saves the CareClient data.
   */
  void saveButton_actionPerformed(ActionEvent e) {
    setCareClientData();
    saveCareClient();
  }
  /**
   * Deletes the CareClient.
   */
  void deleteButton_actionPerformed(ActionEvent e) {
    deleteCareClient();
  }
  /*
   * Creates a new CareClient with a specific ID from the
Database and shows it in the GUI components.
   */
  public void setCareClient(int ID) {
    //System.out.println(ID);
    CareClient cl = new CareClient(ID);
    cl.setDataFromDB(ID);
    this.careClient = cl;
    showCareClient();
    mainFrame.setCareClient(cl);
  }
  /**
   * Shows the CareClient Data in the GUI-Elements.
   */
  public void showCareClient() {
    personalData.setName(careClient.getName());
    personalData.setForename(careClient.getForename());
    personalData.setMartialStatus(careClient.getMartialStatus());
    personalData.setNationality(careClient.getNationality());
    personalData.setBirthday(careClient.getBirthday());
    personalData.setTownOfBirth(careClient.getTownOfBirth());
    personalData.setReligion(careClient.getReligion());
    personalData.undoChanges();

    contactData.setAdresses(careClient.getAdresses());

contactData.setPhoneNumbers(careClient.getPhoneNumbers());
    contactData.setEMails(careClient.getEMails());
    contactData.setBankAccounts(careClient.getBankAccounts());
    contactData.undoChanges();
    remarkTextArea.setText(careClient.getRemark());
  }
  /**
   * Gets the Data from the GUI-Elements and stores it into the
CareClient.
   */
  public void setCareClientData() {
    personalData.applyChanges();
    careClient.setName(personalData.getName());
    careClient.setForename(personalData.getForename());
    careClient.setMartialStatus(personalData.getMartialStatus());
    careClient.setNationality(personalData.getNationality());
    careClient.setBirthday(personalData.getBirthday());
    careClient.setTownOfBirth(personalData.getTownOfBirth());
    careClient.setReligion(personalData.getReligion());

    contactData.applyChanges();
    careClient.setAdresses(contactData.getAdresses());

careClient.setPhoneNumbers(contactData.getPhoneNumbers());
    careClient.setEMails(contactData.getEMails());
    careClient.setBankAccounts(contactData.getBankAccounts());

    careClient.setRemark(remarkTextArea.getText());
  }
  /**
   * Tells the CareClient to store his Data in the Database.
   */
  public void saveCareClient() {
    int status = careClient.storeInDB();
```

```
    if (status !=2) {
      if (status==0)
        JOptionPane.showMessageDialog(this,"Daten
erfolgreich gespeichert","Daten speichern",
        JOptionPane.INFORMATION_MESSAGE);
      else
        JOptionPane.showMessageDialog(this,"Fehler
beim speichern der Daten","Daten speichern",
        JOptionPane.ERROR_MESSAGE);
    }
  }
  /**
   * Tells the CareClient to delete his Data from the
Database
   */
  public void deleteCareClient() {
    int status = careClient.deleteFromDB();
    if (status !=2) {
      if (status==0)
        JOptionPane.showMessageDialog(this,"Daten erfolgreich
gelöscht","Daten löschen",
        JOptionPane.INFORMATION_MESSAGE);
      else
        JOptionPane.showMessageDialog(this,"Fehler beim
löschen der Daten","Daten löschen",
        JOptionPane.ERROR_MESSAGE);
    }
  }
}
```

Die Klasse SearchPersonDialog

Die Klasse *SearchPersonDialog* stellt den Person suchen-Dialog bereit.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.awt.*;
import oracle.jdeveloper.layout.*;
import java.awt.event.*;
import generic.JDBCEngine;
import java.sql.*;
import java.util.*;

/**
 * A Swing-based dialog class.
 * Used to search Persons.
 * <P>
 * @author Martin Braches
 */
public class SearchPersonDialog extends JDialog {

  // GUI Elements
  JPanel jPanel1 = new JPanel();
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel2 = new JPanel();
  JPanel jPanel3 = new JPanel();
  JScrollPane jScrollPane1 = new JScrollPane();
  SearchDBConnect db = new SearchDBConnect();
  SearchTableModel model = new SearchTableModel();
  JTable jTable1 = new JTable(model);
  JButton searchButton = new JButton();
  JButton takeDataButton = new JButton();
  JButton cancelButton = new JButton();
  XYLayout xYLayout1 = new XYLayout();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel2 = new JLabel();
  JLabel jLabel3 = new JLabel();
  JTextField jTextField1 = new JTextField();
  JTextField jTextField2 = new JTextField();
  JTextField jTextField3 = new JTextField();
  JLabel jLabel4 = new JLabel();
  ClientRequestPanel parent = null;
  int selectedRow = 0;
  String table = "CareClient";
  JLabel jLabel5 = new JLabel();
  /**
   * Constructs a new instance.
   * @param parent
   * @param title
   * @param modal
   */
  public SearchPersonDialog(Frame parent, String title, boolean
modal) {
    super(parent, title, modal);
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new instance
   * @param parent1 - A Frame
   * @param parent - A ClientRequestPanel
   * @param title
   * @param modal
   */
  public SearchPersonDialog(ClientRequestPanel parent, Frame
parent1, String title, boolean modal) {
    super(parent1, title, modal);
    this.parent = parent;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new non-modal unparented instance with a
blank title.
   */
  public SearchPersonDialog() {
    this(null, "", false);
  }
  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jPanel1.setLayout(borderLayout1);
    jPanel2.setLayout(xYLayout1);
    jPanel1.setPreferredSize(new Dimension(350, 300));
    jPanel1.setMinimumSize(new Dimension(200, 300));
    searchButton.setText("Suchen");
```

```
    searchButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        searchButton_actionPerformed(e);
      }
    });
    takeDataButton.setText("Daten übernehmen");
    takeDataButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        takeDataButton_actionPerformed(e);
      }
    });
    cancelButton.setText("Abbrechen");
    cancelButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cancelButton_actionPerformed(e);
      }
    });

jTable1.setSelectionMode(ListSelectionModel.SINGLE
_SELECTION);
    ListSelectionModel rowSM =
jTable1.getSelectionModel();
    rowSM.addListSelectionListener(new
ListSelectionListener() {
      public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting()) return;
          ListSelectionModel lsm
=(ListSelectionModel)e.getSource();
          if (lsm.isSelectionEmpty()) {
            //nothing
          }
          else {
            selectedRow = lsm.getMinSelectionIndex();
          }
        }
      });
    jLabel1.setText("Nachname");
    jLabel2.setText("Vorname");
    jLabel3.setText("Geburtsdatum");
    jTextField1.setText("%");
    jTextField2.setText("%");
    jTextField3.setText("%");
    jLabel5.setText("YYYY-MM-DD");
    getContentPane().add(jPanel1);
    jPanel1.add(jPanel2, BorderLayout.NORTH);
    jPanel2.add(jLabel1, new XYConstraints(2, 5, 86,
22));
    jPanel2.add(jLabel2, new XYConstraints(3, 32, 86,
22));
    jPanel2.add(jLabel3, new XYConstraints(3, 59, 86,
22));
    jPanel2.add(jTextField1, new XYConstraints(90, 5,
273, 23));
    jPanel2.add(jTextField2, new XYConstraints(90, 32,
273, 23));
    jPanel2.add(jTextField3, new XYConstraints(90, 59,
84, 23));
    jPanel2.add(jLabel4, new XYConstraints(221, 52, -1,
-1));
    jPanel2.add(jLabel5, new XYConstraints(182, 62, -1,
-1));
    jPanel1.add(jPanel3, BorderLayout.SOUTH);
    jPanel3.add(searchButton, null);
    jPanel3.add(takeDataButton, null);
    jPanel3.add(cancelButton, null);
    jPanel1.add(jScrollPane1, BorderLayout.CENTER);
    jScrollPane1.getViewport().add(jTable1, null);
  }
  /**
     * Makes the Dialog visible. If the dialog and/or its owner
     * are not yet displayable, both are made displayable.  The
     * dialog will be validated prior to being made visible.
     * If the dialog is already visible, this will bring the dialog
     * to the front.
     * <p>
     * If the dialog is modal and is not already visible, this call
will
     * not return until the dialog is hidden by calling
<code>hide</code> or
     * <code>dispose</code>. It is permissible to show modal
dialogs from
     * the event dispatching thread because the toolkit will ensure
that
     * another event pump runs while the one which invoked this
method
     * is blocked.
     * @see Component#hide
     * @see Component#isDisplayable
     * @see Component#validate
     * @see java.awt.Dialog#isModal
     */
  public void show() {
    //TODO:  Override this java.awt.Dialog method
    this.toFront();
    this.setResizable(false);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    this.setLocation((screenSize.width - frameSize.width)/2,
(screenSize.height - frameSize.height)/2);
    //this.setVisible(true);
    super.show();
  }
  /**
   * Disposes this dialog.
   */
  void cancelButton_actionPerformed(ActionEvent e) {
    this.dispose();
  }
  /**
   * Adds the choosen Person to the parent ClientRequestPanel.
   */
  void takeDataButton_actionPerformed(ActionEvent e) {
    if (jTable1.getSelectedRowCount() > 0)  {
      int ID
=((Integer)model.getValueAt(selectedRow,3)).intValue();
      parent.setCareClient(ID);
      this.dispose();
    }
  }
  /**
   * Calls the search method.
   * <P>
   * @see com.server.SearchPersonDialog#search
   */
  void searchButton_actionPerformed(ActionEvent e) {
    search();
   }
  /**
   * Sents a query to the database and displays the result.
   */
  public void search() {
    String user = "";
    String password = "";
    String host = "com";
```

```
    String erg = db.openConnection(user,password,host);
    String name = jTextField1.getText();
    String foreName = jTextField2.getText();
    String birthday = jTextField3.getText();
    String birthdayString = "' and birthday Like
DateValue('"+birthday+"');";
    if (birthday.equals("%"))
      birthdayString = ";";
    String query = "Select name,foreName,birthday,ID
from "+table+" where name Like'"+ name +
                "' and foreName Like'"+ foreName +
birthdayString;
    //System.out.println(query);
    erg = db.createStatement(query);
    System.out.println(erg);
    evaluateResultSet();
  }
  /**
   * Traverses the resultSet and adds the content to a
JTable.
   */
  void evaluateResultSet()  {
    try {
      model.clear();
      ResultSetMetaData rsmd = db.rset.getMetaData();
      int columns = rsmd.getColumnCount();
      //System.out.println(columns+ "  " );
      if (columns >= 1) {
       //for (int i=1;i<=columns;i++)
        //model.addColumn(rsmd.getColumnName(i));
        model.addColumn("Name");
        model.addColumn("Vorname");
        model.addColumn("Geburtsdatum");
        model.addColumn("ID");
       if (db.rset != null)  {
        while(db.rset.next()) {
          Vector rowVector = new Vector();
          for (int i=1;i<=columns;i++)
            rowVector.addElement(db.rset.getObject(i));
          model.addRow(rowVector);
          //System.out.println(rowVector+ "  " );
        }
       }
       jTable1.sizeColumnsToFit(0);
       }

    }
    catch (Exception e) {
      System.out.println("Fehler in beim Auswerten des
ResultSets: "+e);
    }
  }
}
```

Die Klassse SearchDBConnect

Diese Klasse stellt die für das Suchen in der Datenbank benötigten Methoden bereit.

```
// Copyright (c) 2000
package com.server;

import java.sql.*;
/**
 * A simple Database access class.
 * <P>
 * @author Martin Braches
 */
public class SearchDBConnect {

  String DBInit_status ="Fehler bei Anmeldung!";
  Connection conn = null;
  ResultSet rset;
  DatabaseMetaData dbmd;
  /**
   * Constructor
   */
  public SearchDBConnect() {
  }
  /**
   * Opens a connection to a Database.
   * Currentliy it uses the Sun JDBC-ODBC as driver.
   * This class is also usable ORACLE DBMS - whith
simple changes.
   * <P>
   * @param name - the User Name
   * @param passwort - the Password
   * @param db - the Name of the database
   */
  public String openConnection(String name, String
passwort, String db){
    String connString = "jdbc:odbc:" + db;
// ODBC/JDBC Bridge
    //String connString = "jdbc:oracle:thin:@" + db;
// Thin Oracel Driver
    //String connString = "jdbc:oracle:oci8::@" + db;
// Thick Oracle Driver
    try{
      Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");   // Laden
des JDBC-Treibers
      //DriverManager.registerDriver(new
oracle.jdbc.driver.OracleDriver());  // Laden des Oracle-JDBC-
Treibers
      //Class.forName ("oracle.jdbc.driver.OracleDriver");
// oder das -> ist beides gleich - nur hier wird der Treiber erst zur
Laufzeit geladen
      // Anmeldung an der Datenbank
      conn = DriverManager.getConnection (connString, name,
passwort);
      dbmd = conn.getMetaData();
      DBInit_status = "User " + name + " ist angemeldet auf " +
db;
    }
    catch (Exception e) {
      return DBInit_status = "Fehler in beim Erstellen der DB-
Verbindung: "+e.getMessage();
    }
  return DBInit_status;
  }
  /**
   * Closes a Database-Connection.
   */
  public String closeConnection(){
    try{
      conn.close();
    }
    catch (Exception e) {
      return "Fehler in Schließen der DB-Verbindung:
"+e.getMessage();
    }
  return "Verbindung erfolgreich geschlossen !!   schönen Tag
noch";
```

```
  }
  /**
    * Sents a Query to a Database and returns a status
string.
    * The resultset is reachable about the rset variable.
    */
  public String createStatement(String query)  {
    try {
      Statement stmt = conn.createStatement();
      rset = stmt.executeQuery(query);
      //stmt.close();
    }
    catch (Exception e) {
      return "Fehler beim Ausführen des Befehls:
"+e.getMessage();
    }
  return "";
  }
  /**
    * Requests all Tabels from a Database.
    * The resultset is reachable about the rset variable.
    * <P>
    * Only for Oracle DBMS - does not work with MS-
Access.
    * <P>
    * @see
com.server.SearchDBConnect#getTablesAccess
    */
  public String getTables(String user) {
    String[] types ={"TABLE"};
    try {
      rset = dbmd.getTables(null,user,"%",types);
    }
    catch (Exception e) {
      return "Fehler beim Ausführen von alle Tabellen:
"+e.getMessage();
    }
  return "OK";
  }
  /**
    * Requests all Tabels from a Database.
    * The resultset is reachable about the rset variable.
    * <P>
    * Only for MS-Access.
    * <P>
    * @see com.server.SearchDBConnect#getTables
    */
  public String getTablesAccess(String user) {
    String[] types ={"TABLE"};
    try {
      rset = dbmd.getTables(null,null,null,types);
    }
    catch (Exception e) {
      return "Fehler beim Ausführen von alle Tabellen:
"+e.getMessage();
    }
  return "OK";
  }
  /**
   * Returns the variable rset.
   */
  public ResultSet getResultSet() {
    return this.rset;
  }
}
```

Die Klasse SearchTableModel

Sie erweitert die Klasse *AbstractTableModel* um die für die Darstellung der Suchergebnisse benötigten Funktionen.

```
// Copyright (c) 2000
package com.server;

import java.util.Vector;
import java.util.ArrayList;
import java.text.DateFormat;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableModel;
/**
 * A TabelModel for the Search*  classes.
 * <P>
 * @author Martin Braches
 */
public class SearchTableModel extends
AbstractTableModel {

  Vector columnNames= new Vector() ;
  Vector data= new Vector();
  /**
   * Constructor
   */
  public SearchTableModel() {
  }
  /**
   * Sets the data and the column names for the tabel
model.
   * <P>
   * @param data - The data Vector
   * @param columnNames - The column names Vector
   */
  public void setDataVector(Vector data,Vector columnNames)
{
    this.columnNames = columnNames;
    this.data=data;
    this.fireTableStructureChanged();
    //System.out.println(data);
  }
  /**
   * Returns the column count for the tabel model.
   */
  public int getColumnCount() {
    return columnNames.size();
  }
  /**
   * Returns the row count for the tabel model.
   */
  public int getRowCount() {
    return data.size();
  }
  /**
   * Returns the column name for the column col.
   * <P>
   * @param col
   */
  public String getColumnName(int col) {
    return (String)columnNames.elementAt(col);
  }
  /**
```

```
   * Returns the value at row , col for this table.
   * <P>
   * @param row
   * @param col
   */
  public Object getValueAt(int row, int col) {
      Vector _ret=(Vector)data.get(row);
   return _ret.elementAt(col);
  }
  /**
   * Returns the class for column col.
   * <P>
   * @param col
   */
  public Class getColumnClass(int col) {
    try {
      return getValueAt(0, col).getClass();
    }
    catch (Exception e){}
    return null;
  }
  /**
   * Sets the column name at col.
   * <P>
   * @param name
   * @param col
   */
  public void setColumnName(String name,int col) {
    columnNames.setElementAt(name,col);
  }
  /**
   * Sets the all column name.
   * <P>
   * @param columnNames
   */
  public void setColumnNames(Vector columnNames) {
    this.columnNames=columnNames;
  }
  /**
   * Adds a column to the tabel model.
   * <P>
   * @param name
   */
  public void addColumn(String name) {
    columnNames.addElement(name);
    this.fireTableStructureChanged();
  }
  /**
   * Adds a row to the tabel model.
   * <P>
   * @param rowVector
   */
  public void addRow(Vector rowVector) {
    data.add(rowVector);
    this.fireTableStructureChanged();
  }
  /**
   * Sets the value at row, col to value.
   * <P>
   * @param value
   * @param row
   * @param col
   */
  public void setValueAt(Object value, int row, int col) {
    ((Vector)data.elementAt(row)).setElementAt(value,col);
    this.fireTableCellUpdated(row,col);
  }
  /**
   * Clears the row and column vector.
   */
  public void clear() {
    data.clear();
    columnNames.clear();
  }
}
```

Die Klasse CareClientRequestedServicesDialog

Sie repräsentiert den Dienste-Dialog in der Maske Kundenanfrage.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.util.Vector;
import java.awt.BorderLayout;
import java.awt.event.*;
import person.*;
/**
 * A Swing-based dialog class.
 * It represents the <code>RequestedServices</code>
for a specific <code>CareClient</code>
 * <P>
 * @author Martin Braches
 */
public class CareClientRequestedServicesDialog
extends JDialog {

  private CareClient careClient = null;

  // GUI Elements
  private BorderLayout borderLayout1 = new
BorderLayout();
  private JPanel jPanel1 = new JPanel();
  private JPanel jPanel2 = new JPanel();
  private JCheckBox jCheckBox1 = new JCheckBox();
  private JCheckBox jCheckBox2 = new JCheckBox();
  private JCheckBox jCheckBox3 = new JCheckBox();
  private JCheckBox jCheckBox4 = new JCheckBox();
  private JLabel jLabel1 = new JLabel();
  private JCheckBox jCheckBox5 = new JCheckBox();
  private JCheckBox jCheckBox6 = new JCheckBox();
  private JCheckBox jCheckBox7 = new JCheckBox();
  private JCheckBox jCheckBox11 = new JCheckBox();
  private JPanel jPanel4 = new JPanel();
  private JButton saveButton = new JButton();
  private JButton cancelButton = new JButton();
  private JPanel jPanel3 = new JPanel();
  private BorderLayout borderLayout2 = new BorderLayout();
  private JPanel jPanel5 = new JPanel();
  private JCheckBox jCheckBox8 = new JCheckBox();
  private JCheckBox jCheckBox9 = new JCheckBox();
  private JCheckBox jCheckBox10 = new JCheckBox();
  private GridLayout gridLayout1 = new GridLayout();
  private String status1="";
  private String status2="";
  private String status3="";
  /**
   * Constructs a new instance.
   * @param parent
```

```
   * @param title
   * @param modal
   */
  public CareClientRequestedServicesDialog(Frame
parent, String title, boolean modal) {
    super(parent, title, modal);
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new instance.
   * @param careClient - The CareClient which
RequestedServices will be shown
   * @param parent
   * @param title
   * @param modal
   */
  public CareClientRequestedServicesDialog(CareClient
careClient,Frame parent, String title, boolean modal) {
    super(parent, title, modal);
    this.careClient=careClient;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new non-modal unparented instance
with a blank title.
   */
  public CareClientRequestedServicesDialog() {
    this(null, "", false);
  }
  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jPanel1.setLayout(borderLayout1);
    jPanel1.setPreferredSize(new Dimension(530, 200));
    jPanel1.setMinimumSize(new Dimension(200, 200));
    jCheckBox1.setText("SGB V");
    jCheckBox2.setText("BSHG");
    jCheckBox3.setText("Selbstzahler");
    jCheckBox4.setText("SGB XI");
    jLabel1.setText("Pflegestufe");
    jCheckBox5.setText("1");
    jCheckBox6.setText("2");
    jCheckBox7.setText("3");
    jCheckBox11.setText("3a");
    jPanel2.setEnabled(false);
    disableAll();
    saveButton.setText("Speichern");
    saveButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        saveButton_actionPerformed(e);
      }
    });
    cancelButton.setText("Abbrechen");
    cancelButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cancelButton_actionPerformed(e);
      }
    });
    jPanel4.add(saveButton, null);
    jPanel4.add(cancelButton, null);
    Border border1 = new BevelBorder(1);
    Border border2 = new TitledBorder(border1,"Leistungsträger
der ambulanten Versorgung");
    gridLayout1.setRows(3);
    jPanel5.setLayout(gridLayout1);
    jCheckBox11.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox11_actionPerformed(e);
      }
    });
    jCheckBox7.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox7_actionPerformed(e);
      }
    });
    jCheckBox6.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox6_actionPerformed(e);
      }
    });
    jCheckBox5.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox5_actionPerformed(e);
      }
    });
    jCheckBox4.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox4_actionPerformed(e);
      }
    });
    jCheckBox3.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox3_actionPerformed(e);
      }
    });
    jCheckBox2.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox2_actionPerformed(e);
      }
    });
    jCheckBox1.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox1_actionPerformed(e);
      }
    });
    jCheckBox10.setText("ambulante Versorgung");
    jCheckBox10.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox10_actionPerformed(e);
      }
    });
    jCheckBox9.setText("Menueservice");
    jCheckBox9.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jCheckBox9_actionPerformed(e);
      }
    });
    jCheckBox8.setText("Hausnotruf");
    jCheckBox8.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
```

```
      jCheckBox8_actionPerformed(e);
    }
  });
  jPanel3.setLayout(borderLayout2);
  jPanel2.setBorder(border2);
  jPanel2.add(jCheckBox1, null);
  jPanel2.add(jCheckBox2, null);
  jPanel2.add(jCheckBox3, null);
  jPanel2.add(jCheckBox4, null);
  jPanel2.add(jLabel1, null);
  jPanel2.add(jCheckBox5, null);
  jPanel2.add(jCheckBox6, null);
  jPanel2.add(jCheckBox7, null);
  jPanel2.add(jCheckBox11, null);
  jPanel1.add(jPanel5, BorderLayout.CENTER);
  jPanel5.add(jCheckBox8, null);
  jPanel5.add(jCheckBox9, null);
  jPanel5.add(jCheckBox10, null);
  jPanel1.add(jPanel3, BorderLayout.SOUTH);
  jPanel3.add(jPanel4, BorderLayout.SOUTH);
  jPanel3.add(jPanel2, BorderLayout.NORTH);
  getContentPane().add(jPanel1);
  setRequestedServices();
 }
 /**
  * Shows this dialog and centers it on screen.
  */
 public void show() {
  this.toFront();
  this.setResizable(false);
  //Center the window
  Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
  Dimension frameSize = this.getSize();
  if (frameSize.height > screenSize.height) {
   frameSize.height = screenSize.height;
  }
  if (frameSize.width > screenSize.width) {
   frameSize.width = screenSize.width;
  }
  this.setLocation((screenSize.width -
frameSize.width)/2, (screenSize.height -
frameSize.height)/2);
  super.show();
 }
 /**
  * Disables all checkboxes in this dialog.
  */
 void disableAll() {
  jCheckBox1.setEnabled(false);
  jCheckBox2.setEnabled(false);
  jCheckBox3.setEnabled(false);
  jCheckBox4.setEnabled(false);
  jCheckBox1.setSelected(false);
  jCheckBox2.setSelected(false);
  jCheckBox3.setSelected(false);
  jCheckBox4.setSelected(false);
  jLabel1.setEnabled(false);
  jCheckBox5.setEnabled(false);
  jCheckBox6.setSelected(false);
  jCheckBox7.setEnabled(false);
  jCheckBox11.setEnabled(false);
  jCheckBox5.setSelected(false);
  jCheckBox6.setEnabled(false);
  jCheckBox7.setSelected(false);
  jCheckBox11.setSelected(false);
 }
 /**
  * Enables all checkboxes in this dialog.
  */
 void enableAll() {
  jCheckBox1.setEnabled(true);
  jCheckBox2.setEnabled(true);
  jCheckBox3.setEnabled(true);
  jCheckBox4.setEnabled(true);
  jLabel1.setEnabled(true);
  jCheckBox5.setEnabled(true);
  jCheckBox7.setEnabled(true);
  jCheckBox6.setEnabled(true);
  jCheckBox11.setEnabled(true);
 }
 /**
  * Enables the checkboxes in dependence of the
requestedServices of the CareClient
  */
 void setRequestedServices() {
  Vector requestedServiceVector =
careClient.getRequestedServices();
  for (int a=0;a<requestedServiceVector.size();a++) {
   RequestedService tempRS
=(RequestedService)requestedServiceVector.elementAt(a);
   System.out.println(tempRS.getName());
   if (tempRS.getName().equals("Hausnotrufdienste")) {
    jCheckBox8.setSelected(true);
    status1=tempRS.getStatus();
    jCheckBox8.setText(jCheckBox8.getText()+status1);
   }
   if (tempRS.getName().equals("Mahlzeitendienste")) {
    jCheckBox9.setSelected(true);
    status2=tempRS.getStatus();
    jCheckBox9.setText(jCheckBox9.getText()+status2);
   }
   if (tempRS.getName().equals("ambulante Pflegedienste")) {
    jCheckBox10.setSelected(true);
    status3=tempRS.getStatus();
    jCheckBox10.setText(jCheckBox10.getText()+status3);
    enableAll();
    if (tempRS.getPaymentInstitution().equals("SGB XI")) {
     jCheckBox4.setSelected(true);
     int careClassification = careClient.getCareClassification();
     if (careClassification == 1)
      jCheckBox5.setSelected(true);
     if (careClassification == 2)
      jCheckBox6.setSelected(true);
     if (careClassification == 3)
      jCheckBox7.setSelected(true);
     if (careClassification == 4)
      jCheckBox11.setSelected(true);
    }
    if (tempRS.getPaymentInstitution().equals("Selbstzahler"))
     jCheckBox3.setSelected(true);
    if (tempRS.getPaymentInstitution().equals("BSHG"))
     jCheckBox2.setSelected(true);
    if (tempRS.getPaymentInstitution().equals("SGB V"))
     jCheckBox1.setSelected(true);
   }
  }
 }
 /**
  * Disposes this dialog.
  */
 void cancelButton_actionPerformed(ActionEvent e) {
  this.dispose();
 }
 /**
  * Determines which checkbox is checked and sets the choosen
RequestedServices to the CareClient.
  */
 void saveButton_actionPerformed(ActionEvent e) {
  Vector requestedServiceVector = new Vector();
  if (jCheckBox8.isSelected()) {
   RequestedService rs = new RequestedService(1);
   rs.setName("Hausnotrufdienste");
   rs.setPaymentInstitution("Selbstzahler");
   rs.setStatus(status1);
   requestedServiceVector.add(rs);
```

```
    }
    if (jCheckBox9.isSelected())  {
      RequestedService rs = new RequestedService(2);
      rs.setName("Mahlzeitendienste");
      rs.setPaymentInstitution("Selbstzahler");
      rs.setStatus(status2);
      requestedServiceVector.add(rs);
    }
    if (jCheckBox10.isSelected())  {
      RequestedService rs = new RequestedService(3);
      rs.setName("ambulante Pflegedienste");
      if (jCheckBox4.isSelected())  {
        rs.setPaymentInstitution("SGB XI");
        if (jCheckBox5.isSelected())
          careClient.setCareClassification(1);
        else if (jCheckBox6.isSelected())
          careClient.setCareClassification(2);
        else if (jCheckBox7.isSelected())
          careClient.setCareClassification(3);
        else if (jCheckBox11.isSelected())
          careClient.setCareClassification(4);
        else careClient.setCareClassification(0);
      }
      else if (jCheckBox3.isSelected())
        rs.setPaymentInstitution("Selbstzahler");
      else if (jCheckBox2.isSelected())
        rs.setPaymentInstitution("BSHG");
      else if (jCheckBox1.isSelected())
        rs.setPaymentInstitution("SGB V");
      else
        rs.setPaymentInstitution(" ");
        rs.setStatus(status3);
      requestedServiceVector.add(rs);
    }

careClient.setRequestedServices(requestedServiceVecto
r);
     careClient.storeRequestedServiceIDs();
     this.dispose();
   }
   /**
    * Does nothing.
    */
   void jCheckBox8_actionPerformed(ActionEvent e) {

   }
   /**
    * Does nothing.
    */
   void jCheckBox9_actionPerformed(ActionEvent e) {

   }
   /**
    * Triggers some simple Checkbox checking logic.
    */
   void jCheckBox10_actionPerformed(ActionEvent e) {
    if (jCheckBox10.isSelected() == false)  {
      disableAll();
    }
    else  {
      jCheckBox1.setEnabled(true);
      jCheckBox2.setEnabled(true);
      jCheckBox3.setEnabled(true);
      jCheckBox4.setEnabled(true);
    }
   }
   /**
    * Triggers some simple Checkbox checking logic..
    */
   void jCheckBox1_actionPerformed(ActionEvent e) {
    if (jCheckBox1.isSelected() == true)  {
      jCheckBox2.setSelected(false);
      jCheckBox3.setSelected(false);
      jCheckBox4.setSelected(false);
      jCheckBox4_actionPerformed(e);
    }
   }
   /**
    * Triggers some simple Checkbox checking logic.
    */
   void jCheckBox2_actionPerformed(ActionEvent e) {
    if (jCheckBox2.isSelected() == true)  {
      jCheckBox1.setSelected(false);
      jCheckBox3.setSelected(false);
      jCheckBox4.setSelected(false);
      jCheckBox4_actionPerformed(e);
    }
   }
   /**
    * Triggers some simple Checkbox checking logic.
    */
   void jCheckBox3_actionPerformed(ActionEvent e) {
    if (jCheckBox3.isSelected() == true)  {
      jCheckBox1.setSelected(false);
      jCheckBox2.setSelected(false);
      jCheckBox4.setSelected(false);
      jCheckBox4_actionPerformed(e);
    }
   }
   /**
    * Triggers some simple Checkbox checking logic.
    */
   void jCheckBox4_actionPerformed(ActionEvent e) {
    if (jCheckBox4.isSelected() == true)  {
      jCheckBox1.setSelected(false);
      jCheckBox2.setSelected(false);
      jCheckBox3.setSelected(false);

      jLabel1.setEnabled(true);
      jCheckBox5.setEnabled(true);
      jCheckBox6.setEnabled(true);
      jCheckBox7.setEnabled(true);
      jCheckBox11.setEnabled(true);
      jCheckBox5.setSelected(false);
      jCheckBox6.setSelected(false);
      jCheckBox7.setSelected(false);
      jCheckBox11.setSelected(false);
    }
    else  {
      jLabel1.setEnabled(false);
      jCheckBox5.setEnabled(false);
      jCheckBox6.setSelected(false);
      jCheckBox7.setEnabled(false);
      jCheckBox11.setEnabled(false);
      jCheckBox5.setSelected(false);
      jCheckBox6.setEnabled(false);
      jCheckBox7.setSelected(false);
      jCheckBox11.setSelected(false);
    }
   }
   /**
    * Triggers some simple Checkbox checking logic.
    */
   void jCheckBox5_actionPerformed(ActionEvent e) {
    if (jCheckBox5.isSelected() == true)  {
      jCheckBox6.setSelected(false);
      jCheckBox7.setSelected(false);
      jCheckBox11.setSelected(false);
    }
   }
   /**
    * Triggers some simple Checkbox checking logic.
    */
   void jCheckBox6_actionPerformed(ActionEvent e) {
    if (jCheckBox6.isSelected() == true)  {
      jCheckBox7.setSelected(false);
```

```
    jCheckBox5.setSelected(false);
    jCheckBox11.setSelected(false);
   }
  }
  /**
   * Triggers some simple Checkbox checking logic.
   */
  void jCheckBox7_actionPerformed(ActionEvent e) {
   if (jCheckBox7.isSelected() == true)  {
    jCheckBox5.setSelected(false);
    jCheckBox6.setSelected(false);
    jCheckBox11.setSelected(false);
   }
  }
  /**
   * Triggers some simple Checkbox checking logic.
   */
  void jCheckBox11_actionPerformed(ActionEvent e) {
   if (jCheckBox11.isSelected() == true)  {
    jCheckBox5.setSelected(false);
    jCheckBox6.setSelected(false);
    jCheckBox7.setSelected(false);
   }
  }
}
```

Die Klasse CareClientPhysicianDialog

Diese Klasse sorgt für die Darstellung des Ärzte-Dialoges.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import person.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.sql.*;
import java.util.*;

/**
 * A Swing-based dialog class.
 * It represents the <code>Physicians</code> for a
specific <code>CareClient</code>
 * <P>
 * @author Martin Braches
 */
public class CareClientPhysicianDialog extends JDialog
{

  private ContactInformationPanel contactData;
                private PhysiciansGUI physicianData;
                private CareClient careClient = null;

  // GUI Elements
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JButton addButton = new JButton();
  JButton closeButton = new JButton();
  JButton removeButton = new JButton();
  JButton saveButton = new JButton();
  /**
   * Constructs a new instance.
   * @param parent
   * @param title
   * @param modal
   */
  public CareClientPhysicianDialog(Frame parent,
String title, boolean modal) {
    super(parent, title, modal);
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new instance.
   * @param careClient - The CareClient which Physicians will
be shown
   * @param parent
   * @param title
   * @param modal
   */
  public CareClientPhysicianDialog(CareClient careClient,
Frame parent, String title, boolean modal) {
    super(parent, title, modal);
    this.careClient = careClient;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new non-modal unparented instance with a
blank title.
   */
  public CareClientPhysicianDialog() {
    this(null, "", false);
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jPanel1.setLayout(borderLayout1);
    jPanel1.setPreferredSize(new Dimension(500, 300));
    jPanel1.setMinimumSize(new Dimension(200, 300));
    addButton.setText("Hinzufügen");
    addButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        addButton_actionPerformed(e);
      }
    });
    closeButton.setText("Abbrechen");
    closeButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        closeButton_actionPerformed(e);
      }
    });
    physicianData = new
PhysiciansGUI(careClient.getPhysicians());
    physicianData.undoChanges();
    physicianData.setPreferredSize(new Dimension(150, 100));
    physicianData.setMinimumSize(new Dimension(150, 100));
    contactData = new ContactInformationPanel();
    contactData.removeButtons();
```

```
    contactData.addVisitingHours();
    physicianData.setContactData(contactData);
    removeButton.setText("Löschen");
    saveButton.setText("Speichern");
    saveButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        saveButton_actionPerformed(e);
      }
    });
    removeButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        removeButton_actionPerformed(e);
      }
    });
    getContentPane().add(jPanel1);
    jPanel1.add(physicianData, BorderLayout.NORTH);
    jPanel1.add(contactData, BorderLayout.CENTER);
    jPanel1.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(addButton, null);
    jPanel2.add(saveButton, null);
    jPanel2.add(removeButton, null);
    jPanel2.add(closeButton, null);
  }
  /**
   * Shows this dialog and centers it on screen.
   */
  public void show() {
    this.toFront();
    this.setResizable(false);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    this.setLocation((screenSize.width -
frameSize.width)/2, (screenSize.height -
frameSize.height)/2);
    super.show();
  }
  /**
   * Shows a SearchPhysicianDialog.
   * <p>
   * @see com.server.SearchPhysicianDialog
   */
  void addButton_actionPerformed(ActionEvent e) {
    SearchPhysicianDialog spd = new
SearchPhysicianDialog(this,null,"Arzt Suchen",true);
    spd.search();
    spd.show();
  }
  /**
   * Saves the ID's of the choosen Physicians in the CareClient
object and disposes this dialog.
   */
  void saveButton_actionPerformed(ActionEvent e) {
    careClient.storePhysicianIDs();
    this.dispose();
  }
  /**
   * Removes a Physician from the CareClient.
   */
  void removeButton_actionPerformed(ActionEvent e) {
    boolean isOk = physicianData.removePhysician();
    if (isOk) {

careClient.removePhysician(physicianData.getSelectedIndex());
      physicianData.undoChanges();
    }
  }
  /**
   * Disposes this dialog.
   */
  void closeButton_actionPerformed(ActionEvent e) {
    this.dispose();
  }
  /**
   * Creates a new Physician with a specific ID from Database
and adds it to a CareClient.
   * Is also responsible for telling the GUI to show the correct
Data.
   */
  public void addPhysician(int ID)  {
    Physician tempPh = new Physician(ID);
    tempPh.setDataFromDB(ID);
    careClient.addPhysician(tempPh);

physicianData.addPhysician(tempPh.getName(),tempPh.getFore
name());
    physicianData.undoChanges();
  }
}
```

Die Klasse SearchPhysicianDialog

Diese Klasse ermöglicht durch die Darstellung eines Dialoges das Suchen von Ärzten in der Datenbank.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.awt.*;
import oracle.jdeveloper.layout.*;
import java.awt.event.*;
import generic.JDBCEngine;
import java.sql.*;
import java.util.*;

/**
 * A Swing-based dialog class.
 * Used to search Physicians.
 * <P>
 * @author Martin Braches
 */
public class SearchPhysicianDialog extends JDialog {

  // GUI Elements
  JPanel jPanel1 = new JPanel();
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel2 = new JPanel();
  JPanel jPanel3 = new JPanel();
```

```
  JScrollPane jScrollPane1 = new JScrollPane();
  SearchDBConnect db = new SearchDBConnect();
  SearchTableModel model = new SearchTableModel();
  JTable jTable1 = new JTable(model);
  JButton searchButton = new JButton();
  JButton takeDataButton = new JButton();
  JButton cancelButton = new JButton();
  XYLayout xYLayout1 = new XYLayout();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel2 = new JLabel();
  JTextField jTextField1 = new JTextField();
  JTextField jTextField2 = new JTextField();
  CareClientPhysicianDialog parent = null;
  int selectedRow = 0;
  String table = "Physician";
  /**
   * Constructs a new instance.
   * @param parent
   * @param title
   * @param modal
   */
  public SearchPhysicianDialog(Frame parent, String
title, boolean modal) {
    super(parent, title, modal);
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new instance
   * @param parent1 - A Frame
   * @param parent - A CareClientPhysicianDialog
   * @param title
   * @param modal
   */
  public
SearchPhysicianDialog(CareClientPhysicianDialog
parent, Frame parent1, String title, boolean modal) {
    super(parent1, title, modal);
    this.parent = parent;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new non-modal unparented instance
with a blank title.
   */
  public SearchPhysicianDialog() {
    this(null, "", false);
  }
  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jPanel1.setLayout(borderLayout1);
    jPanel2.setLayout(xYLayout1);
    jPanel1.setPreferredSize(new Dimension(350, 300));
    jPanel1.setMinimumSize(new Dimension(200, 300));
    searchButton.setText("Suchen");
    searchButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        searchButton_actionPerformed(e);
      }
    });
    takeDataButton.setText("Daten übernehmen");
    takeDataButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        takeDataButton_actionPerformed(e);
      }
    });
    cancelButton.setText("Abbrechen");
    cancelButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cancelButton_actionPerformed(e);
      }
    });

jTable1.setSelectionMode(ListSelectionModel.SINGLE_SELEC
TION);
    ListSelectionModel rowSM = jTable1.getSelectionModel();
    rowSM.addListSelectionListener(new ListSelectionListener()
{
      public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting()) return;
          ListSelectionModel lsm
=(ListSelectionModel)e.getSource();
          if (lsm.isSelectionEmpty()) {
            // nothing
          }
          else {
            selectedRow = lsm.getMinSelectionIndex();
          }
        }
      });
    jLabel1.setText("Nachname");
    jLabel2.setText("Vorname");
    jTextField1.setText("%");
    jTextField2.setText("%");
    getContentPane().add(jPanel1);
    jPanel1.add(jPanel2, BorderLayout.NORTH);
    jPanel2.add(jLabel1, new XYConstraints(2, 5, 86, 22));
    jPanel2.add(jLabel2, new XYConstraints(3, 32, 86, 22));
    jPanel2.add(jTextField1, new XYConstraints(90, 5, 273, 23));
    jPanel2.add(jTextField2, new XYConstraints(90, 32, 273,
23));
    jPanel1.add(jPanel3, BorderLayout.SOUTH);
    jPanel3.add(searchButton, null);
    jPanel3.add(takeDataButton, null);
    jPanel3.add(cancelButton, null);
    jPanel1.add(jScrollPane1, BorderLayout.CENTER);
    jScrollPane1.getViewport().add(jTable1, null);
  }
  /**
     * Makes the Dialog visible. If the dialog and/or its owner
     * are not yet displayable, both are made displayable.  The
     * dialog will be validated prior to being made visible.
     * If the dialog is already visible, this will bring the dialog
     * to the front.
     * <p>
     * If the dialog is modal and is not already visible, this call
will
     * not return until the dialog is hidden by calling
<code>hide</code> or
     * <code>dispose</code>. It is permissible to show modal
dialogs from
     * the event dispatching thread because the toolkit will ensure
that
     * another event pump runs while the one which invoked this
method
     * is blocked.
     * @see Component#hide
     * @see Component#isDisplayable
     * @see Component#validate
     * @see java.awt.Dialog#isModal
```

```
   */
  public void show() {
    //TODO:  Override this java.awt.Dialog method
    this.toFront();
    this.setResizable(false);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    this.setLocation((screenSize.width -
frameSize.width)/2, (screenSize.height -
frameSize.height)/2);
    //this.setVisible(true);
    super.show();
  }
  /**
   * Disposes this dialog.
   */
  void cancelButton_actionPerformed(ActionEvent e) {
    this.dispose();
  }
  /**
   * Adds the choosen Physician to the parent
CareClientPhysicianDialog.
   */
  void takeDataButton_actionPerformed(ActionEvent e)
{
    if (jTable1.getSelectedRowCount() > 0)  {
      int ID
=((Integer)model.getValueAt(selectedRow,2)).intValue
();
      parent.addPhysician(ID);
      this.dispose();
    }
  }
  /**
   * Calls the search method.
   * <P>
   * @see com.server.SearchPhysicianDialog#search
   */
  void searchButton_actionPerformed(ActionEvent e) {
    search();
  }
  /**
   * Sents a query to the database and displays the result.
   */
  public void search()  {
    String user = "";
    String password = "";
    String host = "com";
    String erg = db.openConnection(user,password,host);

    String name = jTextField1.getText();
    String foreName = jTextField2.getText();
    String query = "Select name,foreName,ID from "+table+"
where name Like'"+ name +
                "' and foreName Like'"+ foreName +"';";
    //System.out.println(query);
    erg = db.createStatement(query);
    System.out.println(erg);
    evaluateResultSet();
  }
  /**
   * Traverses the resultSet and adds the content to a JTable.
   */
  void evaluateResultSet()  {
    try {
      model.clear();
      ResultSetMetaData rsmd = db.rset.getMetaData();
      int columns = rsmd.getColumnCount();
      //System.out.println(columns+ "  " );
      if (columns >= 1) {
        //for (int i=1;i<=columns;i++)
          //model.addColumn(rsmd.getColumnName(i));
          model.addColumn("Name");
          model.addColumn("Vorname");
          model.addColumn("ID");
        if (db.rset != null)  {
          while(db.rset.next()) {
            Vector rowVector = new Vector();
            for (int i=1;i<=columns;i++)
              rowVector.addElement(db.rset.getObject(i));
            model.addRow(rowVector);
            //System.out.println(rowVector+ "  " );
          }
        }
       jTable1.sizeColumnsToFit(0);
       }

    }
    catch (Exception e) {
      System.out.println("Fehler in beim Auswerten des
ResultSets: "+e);
    }
  }
}
```

Die Klasse CareClientPaymentInstitutionDialog

Sie stellt den Kranken/Pflegekassen-Dialog zur Verfügung.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import person.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.sql.*;
import java.util.*;

/**
 * A Swing-based dialog class.
 * It represents the <code>PaymentInstitutions</code> for a
specific <code>CareClient</code>
 * <P>
 * @author Martin Braches
 */
public class CareClientPaymentInstitutionDialog extends
JDialog {

  private ContactInformationPanel contactData;
            private PaymentInstitutionsGUI
paymentInstitutionData;;
```

```
        private CareClient careClient = null;

  // GUI Elements
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JButton addButton = new JButton();
  JButton closeButton = new JButton();
  JButton removeButton = new JButton();
  JButton saveButton = new JButton();
  /**
   * Constructs a new instance.
   * @param parent
   * @param title
   * @param modal
   */
  public CareClientPaymentInstitutionDialog(Frame
parent, String title, boolean modal) {
    super(parent, title, modal);
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new instance.
   * @param careClient - The CareClient which
PaymentInstitutions will be shown
   * @param parent
   * @param title
   * @param modal
   */
  public CareClientPaymentInstitutionDialog(CareClient
careClient, Frame parent, String title, boolean modal) {
    super(parent, title, modal);
    this.careClient = careClient;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new non-modal unparented instance
with a blank title.
   */
  public CareClientPaymentInstitutionDialog() {
    this(null, "", false);
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jPanel1.setLayout(borderLayout1);
    jPanel1.setPreferredSize(new Dimension(500, 300));
    jPanel1.setMinimumSize(new Dimension(200, 300));
    addButton.setText("Hinzufügen");
    addButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        addButton_actionPerformed(e);
      }
    });
    closeButton.setText("Abbrechen");
    closeButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        closeButton_actionPerformed(e);
      }
    });
    paymentInstitutionData = new
PaymentInstitutionsGUI(careClient.getPaymentInstitutions());
    paymentInstitutionData.undoChanges();
    paymentInstitutionData.setPreferredSize(new Dimension(150,
100));
    paymentInstitutionData.setMinimumSize(new
Dimension(150, 100));
    contactData = new ContactInformationPanel();
    contactData.removeButtons();
    paymentInstitutionData.setContactData(contactData);
    removeButton.setText("Löschen");
    saveButton.setText("Speichern");
    saveButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        saveButton_actionPerformed(e);
      }
    });
    removeButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        removeButton_actionPerformed(e);
      }
    });
    getContentPane().add(jPanel1);
    jPanel1.add(paymentInstitutionData, BorderLayout.NORTH);
    jPanel1.add(contactData, BorderLayout.CENTER);
    jPanel1.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(addButton, null);
    jPanel2.add(saveButton, null);
    jPanel2.add(removeButton, null);
    jPanel2.add(closeButton, null);
  }
  /**
   * Shows this dialog and centers it on screen.
   */
  public void show() {
    this.toFront();
    this.setResizable(false);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    this.setLocation((screenSize.width - frameSize.width)/2,
(screenSize.height - frameSize.height)/2);
    super.show();
  }
  /**
   * Shows a SearchPaymentInstitutionDialog.
   * <p>
   * @see com.server.SearchPaymentInstitutionDialog
   */
  void addButton_actionPerformed(ActionEvent e) {
    SearchPaymentInstitutionDialog spid = new
SearchPaymentInstitutionDialog(this,null,"Kasse Suchen",true);
    spid.search();
    spid.show();
  }
  /**
   * Saves the ID's of the choosen PaymentInstitutions in the
CareClient object and disposes this dialog.
   */
  void saveButton_actionPerformed(ActionEvent e) {
    careClient.storePaymentInstitutionIDs();
```

```
    this.dispose();
  }
  /**
   * Removes a PaymentInstitution from the CareClient.
   */
  void removeButton_actionPerformed(ActionEvent e) {
    boolean isOk =
paymentInstitutionData.removePaymentInstitution();
    if (isOk) {

careClient.removePaymentInstitution(paymentInstitutio
nData.getSelectedIndex());
      paymentInstitutionData.undoChanges();
    }
  }
  /**
   * Disposes this dialog.
   */
  void closeButton_actionPerformed(ActionEvent e) {
    this.dispose();
  }
  /**
   * Creates a new PaymentInstitution with a specific ID from
Database and adds it to a CareClient.
   * Is also responsible for telling the GUI to show the correct
Data.
   */
  public void addPaymentInstitution(int ID)  {
    PaymentInstitution tempPh = new PaymentInstitution(ID);
    tempPh.setDataFromDB(ID);
    careClient.addPaymentInstitution(tempPh);

paymentInstitutionData.addPaymentInstitution(tempPh.getName
(),tempPh.getDescription(),tempPh.getShortName());
    paymentInstitutionData.undoChanges();
  }

}
```

Die Klasse SearchPaymentInstitutionDialog

Sie ermöglicht das Suchen und Hinzufügen von Kranken- und Pflegekassen zu den Kundendaten.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.awt.*;
import oracle.jdeveloper.layout.*;
import java.awt.event.*;
import generic.JDBCEngine;
import java.sql.*;
import java.util.*;

/**
 * A Swing-based database search dialog class.
 * Used to search PaymentInstitutions.
 * <P>
 * @author Martin Braches
 */
public class SearchPaymentInstitutionDialog extends
JDialog {

  // GUI Elements
  JPanel jPanel1 = new JPanel();
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel2 = new JPanel();
  JPanel jPanel3 = new JPanel();
  JScrollPane jScrollPane1 = new JScrollPane();
  SearchDBConnect db = new SearchDBConnect();
  SearchTableModel model = new SearchTableModel();
  JTable jTable1 = new JTable(model);
  JButton searchButton = new JButton();
  JButton takeDataButton = new JButton();
  JButton cancelButton = new JButton();
  XYLayout xYLayout1 = new XYLayout();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel2 = new JLabel();
  JTextField jTextField1 = new JTextField();
  JTextField jTextField2 = new JTextField();
  CareClientPaymentInstitutionDialog parent = null;
  int selectedRow = 0;
  String table = "PaymentInstitution";
  /**
   * Constructs a new instance.
   * @param parent
   * @param title
   * @param modal
   */
  public SearchPaymentInstitutionDialog(Frame parent, String
title, boolean modal) {
    super(parent, title, modal);
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new instance
   * @param parent1 - A Frame
   * @param parent - A CareClientPaymentInstitutionDialog
   * @param title
   * @param modal
   */
  public
SearchPaymentInstitutionDialog(CareClientPaymentInstitutionD
ialog parent,Frame parent1, String title, boolean modal) {
    super(parent1,title, modal);
    this.parent = parent;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new non-modal unparented instance with a
blank title.
   */
  public SearchPaymentInstitutionDialog() {
    this(null, "", false);
```

```
  }
  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jPanel1.setLayout(borderLayout1);
    jPanel2.setLayout(xYLayout1);
    jPanel1.setPreferredSize(new Dimension(350, 300));
    jPanel1.setMinimumSize(new Dimension(200, 300));
    searchButton.setText("Suchen");
    searchButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        searchButton_actionPerformed(e);
      }
    });
    takeDataButton.setText("Daten übernehmen");
    takeDataButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        takeDataButton_actionPerformed(e);
      }
    });
    cancelButton.setText("Abbrechen");
    cancelButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cancelButton_actionPerformed(e);
      }
    });

jTable1.setSelectionMode(ListSelectionModel.SINGLE
_SELECTION);
    ListSelectionModel rowSM =
jTable1.getSelectionModel();
    rowSM.addListSelectionListener(new
ListSelectionListener() {
      public void valueChanged(ListSelectionEvent e) {
        if (e.getValueIsAdjusting()) return;
          ListSelectionModel lsm
=(ListSelectionModel)e.getSource();
          if (lsm.isSelectionEmpty()) {
            // nothing
          }
          else {
            selectedRow = lsm.getMinSelectionIndex();
          }
        }
      });
    jLabel1.setText("Name");
    jLabel2.setText("Kürzel");
    jTextField1.setText("%");
    jTextField2.setText("%");
    getContentPane().add(jPanel1);
    jPanel1.add(jPanel2, BorderLayout.NORTH);
    jPanel2.add(jLabel1, new XYConstraints(2, 5, 86,
22));
    jPanel2.add(jLabel2, new XYConstraints(3, 32, 86,
22));
    jPanel2.add(jTextField1, new XYConstraints(90, 5,
273, 23));
    jPanel2.add(jTextField2, new XYConstraints(90, 32,
273, 23));
    jPanel1.add(jPanel3, BorderLayout.SOUTH);
    jPanel3.add(searchButton, null);
    jPanel3.add(takeDataButton, null);
    jPanel3.add(cancelButton, null);
    jPanel1.add(jScrollPane1, BorderLayout.CENTER);
    jScrollPane1.getViewport().add(jTable1, null);
  }
  /**
     * Makes the Dialog visible. If the dialog and/or its
owner
     * are not yet displayable, both are made displayable.  The
     * dialog will be validated prior to being made visible.
     * If the dialog is already visible, this will bring the dialog
     * to the front.
     * <p>
     * If the dialog is modal and is not already visible, this call
will
     * not return until the dialog is hidden by calling
<code>hide</code> or
     * <code>dispose</code>. It is permissible to show modal
dialogs from
     * the event dispatching thread because the toolkit will ensure
that
     * another event pump runs while the one which invoked this
method
     * is blocked.
     * @see Component#hide
     * @see Component#isDisplayable
     * @see Component#validate
     * @see java.awt.Dialog#isModal
     */
  public void show() {
    this.toFront();
    this.setResizable(false);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    this.setLocation((screenSize.width - frameSize.width)/2,
(screenSize.height - frameSize.height)/2);
    super.show();
  }
  /**
   * Disposes this dialog.
   */
  void cancelButton_actionPerformed(ActionEvent e) {
    this.dispose();
  }
  /**
   * Adds the choosen PaymentInstitution to the parent
CareClientPaymentInstitutionDialog.
   */
  void takeDataButton_actionPerformed(ActionEvent e) {
    if (jTable1.getSelectedRowCount() > 0)  {
      int ID
=((Integer)model.getValueAt(selectedRow,2)).intValue();
      parent.addPaymentInstitution(ID);
      this.dispose();
    }
  }
  /**
   * Calls the search method.
   * <P>
   * @see com.server.SearchPaymentInstitutionDialog#search
   */
  void searchButton_actionPerformed(ActionEvent e) {
    search();
  }
  /**
   * Sents a query to the database and displays the result.
   */
  public void search()  {
    String user = "";
    String password = "";
    String host = "com";
    String erg = db.openConnection(user,password,host);
    String name = jTextField1.getText();
```

```
    String shortName = jTextField2.getText();
    String query = "Select name,shortName,ID from
"+table+" where name Like'"+ name +
                "' and shortName Like'"+ shortName +"';";
    erg = db.createStatement(query);
    System.out.println(erg);
    evaluateResultSet();
  }
  /**
   * Traverses the resultSet and adds the content to a
JTable.
   */
  void evaluateResultSet()  {
    try {
      model.clear();
      ResultSetMetaData rsmd = db.rset.getMetaData();
      int columns = rsmd.getColumnCount();
      //System.out.println(columns+ "  " );
      if (columns >= 1) {
        //for (int i=1;i<=columns;i++)
          //model.addColumn(rsmd.getColumnName(i));
          model.addColumn("Name");
          model.addColumn("Kürzel");
          model.addColumn("ID");
        if (db.rset != null)  {
          while(db.rset.next()) {
            Vector rowVector = new Vector();
            for (int i=1;i<=columns;i++)
              rowVector.addElement(db.rset.getObject(i));
            model.addRow(rowVector);
            //System.out.println(rowVector+ "  " );
          }
        }
       jTable1.sizeColumnsToFit(0);
      }
    }
    catch (Exception e) {
      System.out.println("Fehler in beim Auswerten des
ResultSets: "+e);
    }
  }
}
```

Die Klasse CoordinationPanel3

Diese Klasse stellt die Maske *ambulante Pflegedienste* im Koordinationsserver dar.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.table.*;

import java.awt.*;
import person.*;
import java.awt.event.*;

import java.io.*;
import java.net.URL;
import generic.*;
import java.sql.*;

import java.util.Vector;
import servicecatalogue.*;
import servicecatalogue.explorer.*;
import SwiftConstants;
import generic.*;
import javax.swing.event.TableModelEvent;
import javax.swing.event.TableModelListener;

/**
 * A Swing-based panel class.
 * <P>
 * @author Martin Braches
 */
public class CoordinationPanel3 extends JPanel
implements TableModelListener {

  private CareClient careClient = null;
  private MainFrame mainFrame = null;
  private Vector answerList = new Vector();
  private ServiceProfileCellEditor
serviceProfileCellEditor = null;
  private ServiceProfileCellRenderer
serviceProfileCellRenderer = null;
  private ServiceProfileTableModel tableModel = new
ServiceProfileTableModel();
  private CareTimesDialog ctd = new
CareTimesDialog(null,this,"Pflegezeitpunkte",true);
  private AnswerListDialog ald = null;

  // GUI elements
  JScrollPane jScrollPane1 = new JScrollPane();
  BorderLayout borderLayout1 = new BorderLayout();
  FlowLayout flowLayout1 = new FlowLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JButton jButton1 = new JButton();
  JButton jButton2 = new JButton();
  JTable jTable1 = new JTable(tableModel);
  JButton button = new JButton();
  JTextField jTextField1 = new JTextField();
  JLabel jLabel1 = new JLabel();
  JTextField jTextField2 = new JTextField();
  JLabel jLabel2 = new JLabel();
  boolean[] dialogValues;
  /**
   * Constructs a new instance.
   * <P>
   * @param careClient - The careClient for this panel
   * @param mainFrame - The mainframe for this panel
   */
  public CoordinationPanel3(CareClient careClient,MainFrame
mainFrame) {
    super();
    this.careClient = careClient;
    this.mainFrame = mainFrame;
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    this.setLayout(borderLayout1);
    JDBC myCatalogues = new JDBC("ServCatalogue");
    Vector serviceCatalogues =
myCatalogues.getServiceCatalogues();
```

```
    ServiceItemNode rootNode = new
ServiceItemNode("Leistungen");
    ServiceCatalogue scat =
(ServiceCatalogue)serviceCatalogues.elementAt(0);
    Node scatRootNode = (Node)scat.getAnchor();
    Vector columnNames = new Vector();
    columnNames.addElement("ambulante
Pflegeleistungen");
    columnNames.addElement("Montag");
    columnNames.addElement("Dienstag");
    columnNames.addElement("Mittwoch");
    columnNames.addElement("Donnertag");
    columnNames.addElement("Freitag");
    columnNames.addElement("Samstag");
    columnNames.addElement("Sonntag");
    columnNames.addElement("Preis");
    columnNames.addElement("Anzahl");
    columnNames.addElement("Betrag");
    tableModel.setColumnNames(columnNames);
    int childCount =scatRootNode.getChildCount();
    for(int i=0;i<scatRootNode.getChildCount();i++) {
      Node node = (Node)scatRootNode.getChildAt(i);
      Vector rowVector = new Vector();
      boolean[] booleanArray=
{false,false,false,false,false};
      rowVector.addElement(node.getContent());
      rowVector.addElement(booleanArray);
      rowVector.addElement(booleanArray);
      rowVector.addElement(booleanArray);
      rowVector.addElement(booleanArray);
      rowVector.addElement(booleanArray);
      rowVector.addElement(booleanArray);
      rowVector.addElement(booleanArray);
      rowVector.addElement(new Integer(2));   // hier
Preise ändern
      rowVector.addElement(new Integer(0));
      rowVector.addElement(new Integer(0));
      tableModel.addRow(rowVector);
    }
    tableModel.addTableModelListener(this);
    TableColumn col0 =
jTable1.getColumn(tableModel.getColumnName(0));
    col0.setPreferredWidth(180);
    serviceProfileCellEditor = new
ServiceProfileCellEditor(button,this);
    serviceProfileCellRenderer = new
ServiceProfileCellRenderer(true);
    TableColumn col1 =
jTable1.getColumn(tableModel.getColumnName(1));
    col1.setCellEditor(serviceProfileCellEditor);
    col1.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col2 =
jTable1.getColumn(tableModel.getColumnName(2));
    col2.setCellEditor(serviceProfileCellEditor);
    col2.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col3 =
jTable1.getColumn(tableModel.getColumnName(3));
    col3.setCellEditor(serviceProfileCellEditor);
    col3.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col4 =
jTable1.getColumn(tableModel.getColumnName(4));
    col4.setCellEditor(serviceProfileCellEditor);
    col4.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col5 =
jTable1.getColumn(tableModel.getColumnName(5));
    col5.setCellEditor(serviceProfileCellEditor);
    col5.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col6 =
jTable1.getColumn(tableModel.getColumnName(6));
    col6.setCellEditor(serviceProfileCellEditor);
    col6.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col7 =
jTable1.getColumn(tableModel.getColumnName(7));
    col7.setCellEditor(serviceProfileCellEditor);
    col7.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col8 =
jTable1.getColumn(tableModel.getColumnName(8));
    col8.setPreferredWidth(40);
    TableColumn col9 =
jTable1.getColumn(tableModel.getColumnName(9));
    col9.setPreferredWidth(50);
    TableColumn col10 =
jTable1.getColumn(tableModel.getColumnName(10));
    button.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        boolean[] tempArray =
(boolean[])jTable1.getValueAt(jTable1.getSelectedRow(),jTable
1.getSelectedColumn());
        //System.out.println("temp"+temp);

ctd.setValues(tempArray,jTable1.getSelectedRow(),jTable1.getS
electedColumn());
        ctd.show();
      }
    });
    ListSelectionModel listModel=jTable1.getSelectionModel();

              listModel.setSelectionMode(ListSelectionModel.SIN
GLE_SELECTION);
    jScrollPane1 = new JScrollPane();
    jButton1.setText("Anfrage absenden");
    jButton1.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton1_actionPerformed(e);
      }
    });
    jButton2.setText("Antworten prüfen");
    jButton2.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton2_actionPerformed(e);
      }
    });
    jTextField1.setEditable(false);
    jTextField1.setHorizontalAlignment(JTextField.RIGHT);
    jTextField1.setPreferredSize(new Dimension(40,21));
    jLabel1.setText("Gesamtanzahl:");
    jTextField2.setEditable(false);
    jTextField2.setHorizontalAlignment(JTextField.RIGHT);
    jTextField2.setPreferredSize(new Dimension(40,21));
    jLabel2.setText("Gesamtbetrag:");
    //jPanel2.setLayout(flowLayout1);
    jPanel2.setPreferredSize(new Dimension(280,30));
    jPanel2.add(jLabel1, null);
    jPanel2.add(jTextField1, null);
    jPanel2.add(jLabel2, null);
    jPanel2.add(jTextField2, null);
    this.add(jScrollPane1, BorderLayout.CENTER);
    jScrollPane1.getViewport().add(jTable1, null);
    this.add(jPanel1, BorderLayout.SOUTH);
    jPanel1.add(jButton1, null);
    jPanel1.add(jButton2, null);
    jPanel1.add(jPanel2, null);
  }
  /**
   * Sets the CareClient for this panel.
   * <P>
   * @param careClient
   */
  public void setCareClient(CareClient careClient) {
    this.careClient=careClient;
  }
  /**
   * Returns am boolean array with the dialogValues.
   */
```

```
  public boolean[] getValues() {
    return this.dialogValues;
  }
  /**
   * Sets values from the dialog into the tabelModel.
   * <P>
   * @param dialogValues
   * @param row
   */
  public void setValues(boolean[] dialogValues,int row)
{
    int count = 0;
    for (int i=0;i<dialogValues.length;i++)  {
      if (dialogValues[i])
        count++;
    }
    int oldCount =
((Integer)tableModel.getValueAt(row,9)).intValue();
    Integer newCount = new Integer(oldCount+count);
    tableModel.setValueAt(newCount,row,9);
    tableModel.fireTableDataChanged();
    this.dialogValues=dialogValues;
  }
  /**
   * Shows an optionDialog and asks for sending a
message to all services.
   */
  public void showDialog()  {
    Vector messageVector =
mainFrame.getMessageVector();
    Object[] options = { "Ja", "Nein" };
    if (isChoosen()) {
      int selectedOption =
JOptionPane.showOptionDialog(this, "Wollen Sie die
Leistungsanfrage an alle Dienste senden ?",
"Leistungsanfrage",

JOptionPane.DEFAULT_OPTION,
JOptionPane.WARNING_MESSAGE,null, options,
options[0]);
      if (selectedOption == 0)  {
        generateMessagesForAll(3);
      }
    }
  }
  /**
   * Determines if a special service was choosen to be
coordinated.
   */
  public boolean isChoosen() {
    boolean status = false;
    Vector requestedServiceVector =
careClient.getRequestedServices();
    if (!requestedServiceVector.isEmpty())  {
      for (int i=0;i<requestedServiceVector.size();i++) {
        RequestedService tempRequestedService =
(RequestedService)requestedServiceVector.elementAt(i
);
        if
(tempRequestedService.getName().equals("ambulante
Pflegedienste")) {
          status = true;
        }
      }
    }
    if (!status)
      JOptionPane.showMessageDialog(this,"Es wurde
keine ambulante Versorgung für eine Koordinierung
gewählt","Koordinierung",JOptionPane.ERROR_MES
SAGE);
    return status;
  }
  /**
   * Shows an optionDialog and asks for sending the client data
to the selected service.
   */
  public void showSentDataDialog(String name)  {
    answerList.removeAllElements();
    Vector messageVector = mainFrame.getMessageVector();
    Object[] options = { "Ja", "Nein" };
    Service service = getServiceFromServiceVector(name);
    if (markAsCoordinated2(name)) {
      int selectedOption = JOptionPane.showOptionDialog(this,
"Sie haben den Dienst  "+name+" gewählt.\nWollen Sie die
Kundendaten an diesen Dienst senden ?", "Dienstauswahl",

JOptionPane.DEFAULT_OPTION,
JOptionPane.WARNING_MESSAGE,null, options, options[0]);
      if (selectedOption == 0)  {
        Message message = new Message(careClient,service,"Ein
Kunde hat Leistungen Ihres Dienstes angefordert !\nWollen Sie
die Daten dieses Kunden empfangen ?");
        messageVector.add(message);
        if (service.getConnected())
          mainFrame.sentMessages(name);
      }
    }
  }
  /**
   * Sets a status for a requestedService.
   * <P>
   * @param serviceName - The name of the service
   */
  public boolean markAsCoordinated2(String serviceName) {
    boolean status = false;
    Vector requestedServiceVector =
careClient.getRequestedServices();
    if (!requestedServiceVector.isEmpty())  {
      for (int i=0;i<requestedServiceVector.size();i++) {
        RequestedService tempRequestedService =
(RequestedService)requestedServiceVector.elementAt(i);
        if (tempRequestedService.getName().equals("ambulante
Pflegedienste")) {
          tempRequestedService.setStatus("  - koordiniert an Dienst
-  "+serviceName);
          status = true;
        }
      }
    }
    if (!status)
      JOptionPane.showMessageDialog(this,"Es wurde keine
ambulante Versorgung für eine Koordinierung
gewählt","Koordinierung",JOptionPane.ERROR_MESSAGE);
    return status;
  }
  /**
   * Calls showDialog.
   * <P>
   * @see com.server.CoordinationPanel3#showDialog
   */
  void jButton1_actionPerformed(ActionEvent e) {
    showDialog();
  }
  /**
   * Gernerates a new message to all services of an special type
an sends it.
   * <P>
   * @param typeID
   */
  public void generateMessagesForAll(int typeID)  {
    Message message;
    if(typeID ==3) {
      try {
        Vector serviceVector =  mainFrame.getServiceVector();
        Vector messageVector =  mainFrame.getMessageVector();
```

```
      ServiceItemNode node
=(ServiceItemNode)serviceVector.elementAt(2);
      Vector childServicesVector =
node.getChildServicesVector();
      Adress adress =
(Adress)careClient.getAdresses().elementAt(0);
      String street = adress.getStreet();
      String town = adress.getTown();
      SecureCareClient secureCareClient = new
SecureCareClient(careClient.getCareClassification(),car
eClient.getMartialStatus(),careClient.getNationality(),

careClient.getReligion(),street,town);
      ServiceProfile serviceProfile = new
ServiceProfile(secureCareClient,tableModel.getData(),t
ableModel.getColumnNames());
      for (int u=0;u<childServicesVector.size();u++)  {
        ServiceNode serviceNode =
(ServiceNode)childServicesVector.elementAt(u);
        Service service
=(Service)serviceNode.getService();
        message = new Message(service,"Es gibt eine
neue Leistungsanfrage\nWollen Sie diese empfangen
?",serviceProfile);
        message.setMessageType(3);
        messageVector.add(message);
        if (service.getConnected())
          mainFrame.sentMessages(service.getName());
      }
     }
     catch (Exception ex)  {
      System.out.println("Fehler in
CoordinationPanel3.generateMessagesForAll "+ex);
     }
   }
  }
  /**
   * Adds a service to the answerlist vector
   * <P>
   * @param serviceName
   */
  public void addServiceToList(String serviceName)  {
    answerList.addElement(serviceName);
  }
  /**
   * Shows the answerList if not empty.
   */
  void jButton2_actionPerformed(ActionEvent e) {
    if (answerList.size() != 0) {
      ald = new
AnswerListDialog(null,this,this.answerList,"Antwortlist
e",true);
      ald.show();
    }
    else
      JOptionPane.showMessageDialog(this,"Es sind noch keine
Antworten eingegangen
!","Antwortliste",JOptionPane.ERROR_MESSAGE);
  }
  /**
   * Returns a service object taken from the serviceVector.
   * <P>
   * @param name
   */
  public Service getServiceFromServiceVector(String name)  {
    Vector serviceVector =
(Vector)mainFrame.getServiceVector();
    for(int i=0;i<serviceVector.size();i++)  {
      ServiceItemNode serviceItemNode =
(ServiceItemNode)serviceVector.elementAt(i);
      Vector childServicesVector =
serviceItemNode.getChildServicesVector();
      for(int a=0;a<childServicesVector.size();a++)  {
        ServiceNode serviceNode =
(ServiceNode)childServicesVector.elementAt(a);
        Service service = serviceNode.getService();
        String serviceName = service.getName();
        if (serviceName.equals(name)) {
          return service;
        }
      }
    }
    return null;
  }
  /*
   * A method of the TableModelListener Interface to calculate
the content of the sum fields.
   */
  public void tableChanged(TableModelEvent e) {
    int sum = 0;
    int count = 0;
    int rowCount = tableModel.getRowCount();
    for (int i=0;i<rowCount;i++)  {
      count = count +
((Integer)tableModel.getValueAt(i,9)).intValue();
      sum = sum +
((Integer)tableModel.getValueAt(i,10)).intValue();
    }
    jTextField1.setText(""+count);
    jTextField2.setText(""+sum);
  }
}
```

Die Klasse AnswerListDialog

Sie wird für die Darstellung der *Antwortliste* in der Maske *ambulante Pflegedienste* verwendet.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import java.util.Vector;
/**
 * A Swing-based dialog class. Used to display the list
of answers form the clients.
 * <P>
 * @author Martin Braches
 */
public class AnswerListDialog extends JDialog {

  private CoordinationPanel3 cop3;
  private Vector answerList;

  // GUI elements
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JList jList1;
  JPanel jPanel2 = new JPanel();
  JButton selectButton = new JButton();
```

```
  JButton cancelButton = new JButton();
  /**
   * Constructs a new instance.
   * @param parent
   * @param title
   * @param modal
   */
  public AnswerListDialog(Frame
parent,CoordinationPanel3 cop3,Vector answerList,
String title, boolean modal) {
    super(parent, title, modal);
    this.cop3=cop3;
    this.answerList=answerList;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jList1 = new JList(this.answerList);

jList1.setSelectionMode(ListSelectionModel.SINGLE_
SELECTION);
    if (jList1.getComponentCount() !=0)
      jList1.setSelectedIndex(0);
    jPanel1.setLayout(borderLayout1);
    selectButton.setText("Auswählen");
    selectButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        selectButton_actionPerformed(e);
      }
    });
    cancelButton.setText("Abbrechen");
    cancelButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cancelButton_actionPerformed(e);
      }
    });
    getContentPane().add(jPanel1);
    jPanel1.add(jList1, BorderLayout.CENTER);
    jPanel1.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(selectButton, null);
    jPanel2.add(cancelButton, null);
  }
  /**
   * Shows this dialog and centers it on screen.
   */
  public void show() {
    //TODO:  Override this java.awt.Dialog method
    this.toFront();
    this.setResizable(false);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    this.setLocation((screenSize.width - frameSize.width)/2,
(screenSize.height - frameSize.height)/2);
    //this.setVisible(true);
    super.show();
  }
  /**
   * Tells CoordinationPanel3 to show a confirm dialog.
   * <P>
   * @see com.server.CoordinationPanel3#showSentDataDialog
   */
  void selectButton_actionPerformed(ActionEvent e) {
    cop3.showSentDataDialog((String)jList1.getSelectedValue());
    this.dispose();
  }
  /**
   * Disposes this dialog.
   */
  void cancelButton_actionPerformed(ActionEvent e) {
    this.dispose();
  }
}
```

Die Klasse SecureCareClient

Dies ist eine Datenklasse, welche die anonymisierten Daten des Kunden aufnimmt.

Diese Daten werden bei einer Leistungsanfrage an alle Dienste eines Typs gesendet.

```
// Copyright (c) 2000
package com.server;

import person.*;
import java.io.Serializable;
import java.sql.*;
import generic.*;

/**
 * A class that represents a secure care-client.
 * It contains only a few data about the care-client.
 * <P>
 * @author Martin Braches
 */
public class SecureCareClient extends Object
implements Serializable {
  private int careClassification;
  private String martialStatus="";
  private String nationality="";
  private String religion="";
  private String street="";
  private String town="";
  private int ID=0;

  /**
   * Constructor
   */
  public SecureCareClient() {
  }
  /**
   * Constructor
   * <P>
   * @param careClassification
```

```
   * @param martialStatus
   * @param nationality
   * @param religion
   * @param street
   * @param town
   */
  public SecureCareClient(int careClassification,String
martialStatus,String nationality,String religion,String
street,String town) {
    this.careClassification=careClassification;
    this.martialStatus=martialStatus;
    this.nationality=nationality;
    this.religion=religion;
    this.street=street;
    this.town=town;
    ID = generateID();
  }

  public int getCareClassification(){
                return this.careClassification;
        }

  public void setCareClassification(int
careClassification){
                this.careClassification =
careClassification;
        }

  public String getMartialStatus(){
                return this.martialStatus;
        }

        public void setMartialStatus(String
martialStatus){
                this.martialStatus = martialStatus;
        }

  public String getNationality(){
                return this.nationality;
        }

  public void setNationality(String nationality){
                this.nationality = nationality;
        }

  public String getReligion(){
                return this.religion;
        }

        public void setReligion(String religion){
                this.religion = religion;
        }

  public String getStreeet(){
                return this.street;
        }

        public void setStreet(String street){
                this.street = street;
        }

  public String getTown(){
                return this.town;
        }

        public void setTown(String town){
                this.town = town;
        }

  public int getID(){
                return this.ID;
        }

        public void setID(int ID){
                this.ID = ID;
        }
  /**
   * Generates a new unique ID for this object.
   */
  public int generateID()  {
    int ID =0;
    JDBCEngine db=new JDBCEngine("com");
    db.connect();
    try {
      String query = "select max(ID) from ID";
      ResultSet rset = db.getSQLResult(query);
      if (rset != null)  {
        while(rset.next()) {
          ID= rset.getInt(1);
        }
      }
      ID++;
      query = "INSERT INTO ID(ID) VALUES("+ID+");";
      db.dbUpdate(query);
      db.disconnect();
    }
    catch (Exception e) {
      System.out.println("Fehler in SecureCareClient.generateID:
" + e.getMessage());
    }
    return ID;
  }
}
```

Die Klasse ServiceProfile

Dies ist eine Datenklasse, die dazu dient, die Daten eines SecureCareClient-Objektes zusammen mit dem persönlichen Leistungsprofil an alle Dienste eines Typs zu senden.

```
// Copyright (c) 2000
package com.server;

import person.*;
import java.util.Vector;
import javax.swing.*;
import java.io.Serializable;
import java.sql.*;

/**
 * A data class for the service profile.
 * <P>
 * @author Martin Braches
 */
public class ServiceProfile extends Object implements
Serializable {

  private Vector data;
  private Vector columnNames;
  private SecureCareClient secureCareClient;
  /**
   * Constructor
   */
```

```
  public ServiceProfile(SecureCareClient
secureCareClient,Vector data,Vector columnNames) {
    this.secureCareClient=secureCareClient;
    this.data=data;
    this.columnNames=columnNames;
  }

  public void setData(Vector data)  {
     this.data=data;
  }

  public void setColumnNames(Vector columnNames)
{
    this.columnNames=columnNames;
  }

  public Vector getData()  {
     return this.data;
  }

  public Vector getColumnNames()  {
   return this.columnNames;
  }

  public void setSecureCareClient(SecureCareClient
secureCareClient)  {
    this.secureCareClient=secureCareClient;
  }

  public SecureCareClient getSecureCareClient() {
   return this.secureCareClient;
  }

  public String toString()  {
   return "ServiceProfile";
  }
}
```

Die Klasse ServiceProfileTableModel

Sie erweitert die Klasse *AbstractTableModel* um die für die Darstellung des persönlichen Leistungsprofils benötigten Funktionen.

```
// Copyright (c) 2000
package com.server;

import java.util.Vector;
import java.util.ArrayList;
import java.text.DateFormat;
import javax.swing.table.AbstractTableModel;
import javax.swing.table.TableModel;

/**
 * A table model for a JTable.
 * <P>
 * @author Martin Braches
 */
public class ServiceProfileTableModel extends
AbstractTableModel {

  Vector columnNames= new Vector() ;
  Vector data= new Vector();
  /**
   * Constructor
   */
  public ServiceProfileTableModel() {
  }

  public void setData(Vector mainVector,Vector header)
{
    this.columnNames = header;
    this.data=mainVector;
    this.fireTableStructureChanged();
  }

  public Vector getData() {
   return this.data;
  }

  public Vector getColumnNames() {
   return this.columnNames;
  }

  public int getColumnCount() {
   return columnNames.size();
  }

  public int getRowCount() {
   return data.size();
  }

  public String getColumnName(int col) {
   return (String)columnNames.elementAt(col);
  }

  public Object getValueAt(int row, int col) {
   Vector rowVector=(Vector)data.get(row);
   int count = 0;
   switch(col) {
                              case 0:
      return rowVector.elementAt(col);
     case 1:
                                          return
(boolean[])rowVector.elementAt(col);
     case 2:
                                          return
(boolean[])rowVector.elementAt(col);
     case 3:
                                          return
(boolean[])rowVector.elementAt(col);
     case 4:
                                          return
(boolean[])rowVector.elementAt(col);
     case 5:
                                          return
(boolean[])rowVector.elementAt(col);
     case 6:
                                          return
(boolean[])rowVector.elementAt(col);
     case 7:
                                          return
(boolean[])rowVector.elementAt(col);
     case 8:
                                          return
rowVector.elementAt(col);
     case 9:
       return (Integer)rowVector.elementAt(9);
     case 10:
       int a =((Integer)rowVector.elementAt(8)).intValue();
```

```
      int b
=((Integer)rowVector.elementAt(9)).intValue();
      int temp = a * b;
      return new Integer(temp);
    }
   return new String();
  }

  public Class getColumnClass(int c) {
   try {
     return getValueAt(0, c).getClass();
   }
   catch (Exception e){}
   return null;
  }

  public void setColumnName(String name,int col) {
   columnNames.setElementAt(name,col);
  }

  public void setColumnNames(Vector columnNames) {
   this.columnNames=columnNames;
  }

  public void addColumn(String name) {
   columnNames.addElement(name);
   this.fireTableStructureChanged();
  }

  public void addRow(Vector rowVector) {
   data.add(rowVector);
   this.fireTableStructureChanged();
  }

  public void setValueAt(Object value, int row, int col) {
   ((Vector)data.elementAt(row)).setElementAt(value,col);
   this.fireTableCellUpdated(row,col);
  }

  public void removeRow(int row) {
  }

  public void clear() {
   data.clear();
   columnNames.clear();
  }

  public boolean isCellEditable(int row, int col) {
   if (col < 1) {
     return false;
   }
   else if (col > 7) {
     return false;
   }
   else {
     return true;
   }
  }
}
```

Die Klasse ServiceProfileCellRenderer

Diese Klasse sorgt dafür, daß in der Tabelle für das persönliche Leistungsprofil in jeder Zelle die Kürzel für die ausgewählten Uhrzeiten angezeigt werden.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import javax.swing.border. *;
import java.awt.*;
import java.awt.event.*;
import javax.swing.table.TableCellRenderer;

/**
 * A cell renderer for a JTable.
 * <P>
 * @author Martin Braches
 */
public class ServiceProfileCellRenderer extends JLabel
implements TableCellRenderer {

  Border unselectedBorder = null;
  Border selectedBorder = null;
  boolean isBordered = true;

  public ServiceProfileCellRenderer(boolean
isBordered) {
    super();
    this.isBordered = isBordered;
    setOpaque(true); //MUST do this for background to
show up.
  }

  public Component
getTableCellRendererComponent(JTable table, Object
value,
                    boolean isSelected, boolean hasFocus,int row, int
column) {
    //System.out.println("Renderer "+value+"  "+isSelected+"
"+hasFocus+"  "+row+"   "+column);
    String temp =render((boolean[])value);
    //System.out.println(temp+"temp");
    setText(temp);
    this.setBackground(Color.white);
    return this;
  }

  public String render(boolean[] value)  {
    String retValue = "";
    if (value[0])
      retValue="Vo";
    if (value[1]) {
      if (retValue!="")
        retValue=retValue+",Mi";
      else
       retValue="Mi";
    }
    if (value[2])  {
      if (retValue!="")
        retValue=retValue+",Na";
      else
        retValue="Na";
    }
    if (value[3])  {
      if (retValue!="")
        retValue=retValue+",Ab";
      else
        retValue="Ab";
```

```
   }
   if (value[4])  {
    if (retValue!="")
      retValue=retValue+",N";
    else
      retValue="N";
   }
   return retValue;
  }
}// end of file
```

Die Klasse ServiceProfileCellEditor

Diese Klasse stellt den Pflegezeitpunkte-Dialog bereit, welcher für die Angabe der Pflegezeitpunkte in der Tabelle *persönliches Leistungsprofil* verwendet wird.

```
// Copyright (c) 2000
package com.server;

import javax.swing.*;
import java.awt.*;
import java.util.*;
import java.awt.event.*;

/**
 * A cell editor for a JTable.
 * <P>
 * @author Martin Braches
 */
public class ServiceProfileCellEditor extends
DefaultCellEditor {

  Color currentColor = null;
  CoordinationPanel3 cop3;
  public ServiceProfileCellEditor(JButton
b,CoordinationPanel3 cop3) {
    super(new JCheckBox()); //Unfortunately, the
constructor
                        //expects a check box, combo box,
                       //or text field.
    editorComponent = b;
    this.cop3 = cop3;
    setClickCountToStart(1); //This is usually 1 or 2.
    //Must do this so that editing stops when appropriate.
    b.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        fireEditingStopped();
      }
    });
  }

  protected void fireEditingStopped() {
    super.fireEditingStopped();
  }

  public Object getCellEditorValue() {
    return cop3.getValues();
  }

  public Component getTableCellEditorComponent(JTable
table,Object value,boolean isSelected,
                                                int row,int column) {
    //System.out.println("Editor "+value+"  "+isSelected+"
"+row+"   "+column);
    return editorComponent;
  }

} // end of file
```

Die Klasse CoodinationClient

Dies ist die Startklasse für den Koordinationsclient.

```
// Copyright (c) 2000
package com.client;

import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

/**
 * Starter class for the <code>CoordinationModule
Client</code>.
 * <P>
 * @author Martin Braches
 */
public class CoodinationClient {

  /**
   * Constructor
   */
  public CoodinationClient(String[] remoteServerName)
{
    String host ="localhost";
    if (remoteServerName.length != 0) {
      host =remoteServerName[0];
    }
    final ClientFrame frame = new ClientFrame(host);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = frame.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    /*********** uncomment this to center the frame on screen
***********/
    //frame.setLocation((screenSize.width - frameSize.width)/2,
(screenSize.height - frameSize.height)/2);
    frame.addWindowListener(new WindowAdapter() {
      public void windowClosing(WindowEvent e) {
        frame.close();
      }
    });
    frame.setVisible(true);
    frame.init();
```

```
  }
  /*
   * Closes the program.
   */
  public void close() {
    System.exit(0);
  }
  /**
   * main
   * @param remoteServerName -- the name of the
server to connect to
   */
  public static void main(String[] remoteServerName) {
    try  {

UIManager.setLookAndFeel(UIManager.getSystemLookAndFe
elClassName());
    }
    catch (Exception e) {
      e.printStackTrace();
    }
    new CoodinationClient(remoteServerName);
  }
}
```

Die Klasse ClientFrame

Diese Klasse erweitert die Klasse *JFrame* und stellt das Hauptfenster des Koordinationsclients dar.

```
// Copyright (c) 2000
package com.client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import com.server.*;
import person.*;
import java.util.Vector;


/**
 * A Swing-based top level window class.
 * <P>
 * @author Martin Braches
 */
public class ClientFrame extends JFrame {

  private String name="";
  private Client client=null;
  private Vector careClientVector = new Vector();
  private Service service=null;
  private CareClientPanel ccp = null;
  private ClientServicePanel csp = null;
  private String remoteServerName;

  // GUI Elements
  BorderLayout borderLayout1 = new BorderLayout();
  JTabbedPane jTabbedPane1 = new JTabbedPane();
  JMenuBar menuBar1 = new JMenuBar();
  JMenu menuFile = new JMenu();
  JMenuItem menuFileExit = new JMenuItem();
  JMenu menuHelp = new JMenu();
  JMenuItem menuHelpAbout = new JMenuItem();


  /**
   * Constructs a new instance.
   */
  public ClientFrame(String remoteServerName) {
    super();
    this.remoteServerName=remoteServerName;
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    this.getContentPane().setLayout(borderLayout1);
    this.setSize(new Dimension(450, 500));  // width - higth
    this.setTitle("SWIFT Koordinations-Client");
    this.setIconImage(new
ImageIcon("Com/Images/Swift2.gif").getImage());
    this.getContentPane().add(jTabbedPane1,
BorderLayout.CENTER);
    menuFile.setText("Datei");
    menuFileExit.setText("Ende");
    menuFileExit.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        fileExit_ActionPerformed(e);
      }
    });
    menuHelp.setText("Hilfe");
    menuHelpAbout.setText("Über..");
    menuHelpAbout.addActionListener(new ActionListener() {
      public void actionPerformed(ActionEvent e) {
        helpAbout_ActionPerformed(e);
      }
    });
    menuFile.add(menuFileExit);
    menuBar1.add(menuFile);
    menuHelp.add(menuHelpAbout);
    menuBar1.add(menuHelp);
    this.setJMenuBar(menuBar1);
    csp = new ClientServicePanel(this);
    jTabbedPane1.addTab("Einrichtungsdaten", csp);
  }
  /**
   * Calls the <code>close</code> method.
   */
  void fileExit_ActionPerformed(ActionEvent e) {
    close();
  }
  /**
   * Shows an About dialog.
   */
  void helpAbout_ActionPerformed(ActionEvent e) {
      JOptionPane.showMessageDialog(this,"SWIFT
Koordinations-Client \n      ©2000", "Über..",
JOptionPane.PLAIN_MESSAGE);
  }
  /**
   * Sets up the RMI connection to the Server and
   * gets the <code>ServiceData</code> from the Server
   */
  public void init()  {
    if (login())  {
      this.client = new Client(this);
```

```
      client.startRMI();
      csp.showService(this.getServiceDataFromServer());
    }
    else
      close();
  }
  /**
   * Shows a Login dialog where the name of the service
could be choosen.
   */
  public boolean login() {
    boolean status=false;
    Object[] possibleValues = { "Hausnotruf der JUH",
                          "Menueservice des DRK",
                          "Ambulante Krankenpflege Inge
Hartmann",
                          "Ambulante Krankenpflege
Schulte-Ontrop & Volmer",
                          "Bender GmbH",
                          "kein Dienst" };
    String name =
(String)JOptionPane.showInputDialog(this,
            "Bitte wählen Sie Ihren Dienst !",
"Anmeldung",
            JOptionPane.INFORMATION_MESSAGE,
null,
            possibleValues, possibleValues[0]);
    if (name!= null) {
      setName(name);
      this.setTitle(this.getTitle()+"  -  "+this.getName());
      status = true;
    }
    return status;
  }
  /**
   * Returns the name of this ClientFrame.
   */
  public String getName() {
    return this.name;
  }
  /**
   * Sets the name of this ClientFrame.
   */
  public void setName(String name) {
    this.name=name;
  }

  public void setService(Service service) {
    this.service=service;
  }

  public Service getService() {
    return this.service;
  }
  /**
   * Tells the Server to send the ServiceData for this
client.
   */
  public Service getServiceDataFromServer() {
    return client.getService(this.name);
  }
  /**
   * Tells the Server that this client has been closed.
   */
  public void close() {
    System.out.println("By By...");
    if (client != null)
      client.disconnect();
    this.dispose();
    System.exit(0);
  }
  /**
   * Tells the Server to send the messages for this client.
   */
  public boolean getMessages() {
    return client.getMessages();
  }
  /**
   * Shows a message dialog.
   * <p>
   * If the answer is ok a CareClientServiceProfilePanel or a
CareClientPanel
   * is added to this frame.
   */
  public void showMessage(String message,int messageType)  {
    Object[] options = { "Ja", "Nein" };
    int selectedOption = JOptionPane.showOptionDialog(this,
message, "neue Nachricht",

JOptionPane.DEFAULT_OPTION,
JOptionPane.QUESTION_MESSAGE,null, options, options[0]);
    try {
      if (selectedOption == 0)  {
        if (messageType==0) {
          CareClient careClient =
(CareClient)client.getCareClient();
          if (careClient != null) {
            CareClientPanel ccp = new
CareClientPanel(careClient,this);
            jTabbedPane1.addTab("Kundendaten
"+careClient.getName(), ccp);

jTabbedPane1.setSelectedIndex(jTabbedPane1.getComponentCo
unt()-1);
          }
          this.careClientVector.addElement(careClient);
        }
        else if (messageType==3) {
        ServiceProfile serviceProfile =
(ServiceProfile)client.getServiceProfile();
        int sccID =
((SecureCareClient)serviceProfile.getSecureCareClient()).getID(
);
        CareClientServiceProfilePanel ccspp = new
CareClientServiceProfilePanel(this,serviceProfile);
        jTabbedPane1.addTab("Leistungsanfrage -Nr:"+sccID ,
ccspp);

jTabbedPane1.setSelectedIndex(jTabbedPane1.getComponentCo
unt()-1);
        }
      }
    }
    catch (Exception e) {
      System.out.println("Fehler in showMessage "+e);
    }
  }
  /**
   * Sents a Message from Client to Server.
   */
  public void sentMessage(Object object,String message) {
    ClientContainerThread cct = new
ClientContainerThread("test",this.client,2,object,message,this.na
me);
    cct.start();
    //client.sentMessage(secureCareClient, this.name, message );
  }
  /**
   * Removes a CareClientPanel from this frame.
   */
  public void removeCareClient(CareClient
careClient,CareClientPanel careClientPanel)  {
    careClientVector.removeElement(careClient);
    jTabbedPane1.remove(careClientPanel);
  }
  /**
```

```
   * Removes a CareClientServiceProfilePanel from this
frame.
   */
  public void
removeServiceProfilePanel(CareClientServiceProfilePa
nel ccspp)  {
    jTabbedPane1.remove(ccspp);
  }
  /**
   * Sents the ServiceData back to the Server.
   */
  public void setServiceDataAtServer(Service service)  {
    ClientContainerThread cct = new
ClientContainerThread("test",this.client,1,service);
    cct.start();
    //client.setService(service);
  }

  public String getRemoteServerName() {
   return this.remoteServerName;
  }
}// end of file
```

Die Klasse Client

Diese Klasse stellt die RMI-Funktionalität auf der Seite des Koordinationsclients bereit.

```
// Copyright (c) 2000
package com.client;

import com.server.*;
import java.rmi.Naming;
import java.net.MalformedURLException;
import java.rmi.*;
import java.rmi.server.*;
import java.security.*;
import javax.crypto.*;
import java.lang.Runnable;

import person.*;

/**
 * A class which represents the RMI functionality for
the client.
 * <P>
 * @author Martin Braches
 */
public class Client implements ClientInterface,
Runnable{

  ClientFrame clientFrame=null;
  ServerInterface serverInterface = null;

  private boolean encode = false;
  // if encode == false -> these variables are not used !!
  private PublicKey serverPublicKey = null;
  private PrivateKey privateKey = null;
  private PublicKey publicKey = null;
  private Cipher rsa = null;
  private Thread keyThread = null;

  /**
   * Constructor
   */
  public Client(ClientFrame clientFrame) {
    this.clientFrame=clientFrame;
    if (encode) {
      keyThread = new Thread(this);
      keyThread.start();
    }
  }
  /**
   * Sets up and starts RMI.
   */
  public void startRMI() {
    try {

System.out.println("rmi://"+clientFrame.getRemoteServ
erName()+":1099/MyService");
      UnicastRemoteObject.exportObject( this );
      serverInterface =
(ServerInterface)Naming.lookup("rmi://"+clientFrame.getRemot
eServerName()+":1099/MyService");
      //System.out.println("noch ok");
      boolean isService =
serverInterface.register(clientFrame.getName(),this,publicKey);
      //System.out.println("register OK");
      if (!isService)  {
        System.out.println("Anmeldung gescheitert  !!!");
        clientFrame.close();
      }
    }
    catch (MalformedURLException e) {
      System.out.println("MalformedURLException  " +e);
    }
    catch (RemoteException e) {
      System.out.println("RemoteException  " +e);
    }
    catch (NotBoundException e) {
      System.out.println("NotBoundException  " +e);
    }
  }
  /**
   * Tells the server that the client with a specific name has been
disconnected.
   */
  public void disconnect()  {
    try {
      serverInterface.disconnect(clientFrame.getName());
    }
    catch (Exception e)  {
      System.out.println("ein Fehler in disconnect "+e);
    }
  }
  /**
   * Gets a <code>CareClient</code> object from the server.
   * <P>
   * This object could optionaly be encrypted.
   */
  public Object getCareClient() {
    try {
      Object object =
serverInterface.getCareClient(clientFrame.getName());
      if (object != null) {
        if (encode) {
          SealedObject sealedObject = (SealedObject)object;
          rsa.init(Cipher.DECRYPT_MODE,privateKey);
          object = (Object)sealedObject.getObject(rsa);
        }
        return object;
      }
    }
    catch (Exception e)  {
      System.out.println("ein Fehler in Client.getCareClient "+e);
    }
```

```
    return null;
  }
  /**
   * Gets a <code>ServiceProfile</code> object from
the server.
   * <P>
   * This object could optionaly be encrypted.
   */
  public Object getServiceProfile() {
    try {
      Object object =
serverInterface.getServiceProfile(clientFrame.getName(
));
      if (object != null) {
        if (encode) {
          SealedObject sealedObject =
(SealedObject)object;
          rsa.init(Cipher.DECRYPT_MODE,privateKey);
          object = (Object)sealedObject.getObject(rsa);
        }
        return object;
      }
    }
    catch (Exception e)  {
      System.out.println("ein Fehler in
Client.getServiceProfile "+e);
    }
    return null;
  }
  /**
   * Tells the Server to sent the Messages for this
Service.
   */
  public boolean getMessages() {
    try {
      return
serverInterface.getMessages(clientFrame.getName());
    }
    catch (Exception e)  {
      System.out.println("ein Fehler in
Client.getMessages "+e);
    }
    return false;
  }
  /**
   * Sents a Message from Client to Server.
   */
  public void sentMessage(Object secureCareClient,
String serviceName, String message) {
    try {
      serverInterface.setMessage(secureCareClient,
serviceName, message );
    }
    catch (Exception e) {
      System.out.println("Fehler in Client.sentMessage
"+e);
    }
  }
  /**
   * Receives a Message from the Server.
   */
  public void setMessage(String message,int
messageType) throws java.rmi.RemoteException  {
    try {
      clientFrame.showMessage(message, messageType);
    }
    catch (Exception e) {
      System.out.println("Fehler in Client.setMessage
"+e);
    }
  }
  /** Receives a the public Key from the Server.
   * <P>
   * Only for encryption.
   */
  public void setServerKey(PublicKey serverPublicKey) throws
java.rmi.RemoteException {
    this.serverPublicKey=serverPublicKey;
  }
  /**
   * Receives the ServiceData from the Server.
   */
  public Service getService(String name)  {
    try {
      return serverInterface.getService(name);
    }
    catch (Exception e) {
      System.out.println("Fehler in Client.getService  "+e);
    }
    return null;
  }
  /**
   * Sents the ServiceData back to Server.
   */
  public void setService(Service service)  {
    try {
      service.setClientInterface(this);
      serverInterface.setService(service);
    }
    catch (Exception e) {
      System.out.println("Fehler in Client.setService  "+e);
    }
  }
  /**
   * Generate a Key pair for a RSA algorithm.
   * It creates a public/private key pair with the length
   * the length of the parameter <code>keysize</code>.
   * <P>
   * @param keysize
   */
  public void generateKey(int keysize) {
    try {
      System.out.print("erzeuge Schlüssel...");
      KeyPairGenerator kpg =
KeyPairGenerator.getInstance("RSA"); //=> Keygenerator
erzeugen
      System.out.print(".");
      kpg.initialize(keysize, new SecureRandom());   //2048 dauert
sehr lange  //=> Keygenerator initialisieren
      System.out.print(".");
      KeyPair kp = kpg.generateKeyPair();                    //=>
Schlüsselpaar erzeugen
      System.out.print(".");
      privateKey = kp.getPrivate();                          //=> gibt
den privaten Schlüssel aus
      publicKey = kp.getPublic();                            //=> gibt
den öffentl. Schlüssel aus
      System.out.println("Beendet !!");

      /* lets add a "save" Provider */
      Security.addProvider(new
au.net.aba.crypto.provider.ABAProvider());
      rsa = Cipher.getInstance("RSA","ABA");
      }
    catch (Exception e)
    {
      System.out.println("Fehler in generateKey" + e);
    }
  }
  /**
   * If Key - Gereation is enabled it could be run as Thread.
   * This is the run Method of the Thread.
   */
  public void run() {
    System.out.print("running!");
    //generateKey(512);
```

```
  //startRMI();
 }
```

Die Klasse ClientInterface

Sie definiert die Methoden, die der Koordinationsserver auf dem Koordinationsclient aufrufen kann.

```
// Copyright (c) 2000
package com.client;

import java.security.*;
import java.rmi.*;
/**
 * The RMI ClientInterface. It defines the methodes
which could be called on the Client via RMI.
 * <P>
 * @author Martin Braches
 */
public interface ClientInterface extends
java.rmi.Remote {

  /** Receives a Messge from the Server. */
  public void setMessage(String message,int messageType)
throws java.rmi.RemoteException;
  /** Receives a the public Key from the Server.
   * <P>
   * Only for encryption.
   */
  public void setServerKey(PublicKey serverPublicKey) throws
java.rmi.RemoteException;

}
```

Die Klasse ClientContainerThread

Sie ist das Gegenstück zur Klasse *ServerContainerThread* auf der Seite des Clients.

```
// Copyright (c) 2000
package com.client;

import person.Service;
/**
 * A simpel thrad class which is used to call methodes
on the Client object in an separate thread.
 * <P>
 * @author Martin Braches
 */
public class ClientContainerThread extends Thread {

  private Client client = null;
  private Service service = null;
  private Object object = null;
  private String message = "";
  private String clientName = "";
  private int action = 0;
  /**
   * Constructor
   */
  public ClientContainerThread(String threadName,
Client client, int action, Service service) {
    super(threadName);
    this.client = client;
    this.service = service;
    this.action = action;
  }
  /**
   * Constructor
   */
  public ClientContainerThread(String threadName, Client client,
int action,Object object, String message, String clientName) {
    super(threadName);
    this.client = client;
    this.object = object;
    this.message = message;
    this.clientName = clientName;
    this.action = action;
  }

  public void run() {
    try {
      if (action == 1)  {
        client.setService(service);
      }
      else if (action == 2)  {
        client.sentMessage(object, clientName, message );
      }
    }
    catch (Exception e) {
      System.out.println("Fehler in ClientContainerThread.run "+
e);
    }
  }

} // end of file
```

Die Klasse ClientServicePanel

Sie dient zur Darstellung der Daten eines Dienstanbieters im Koordinationsclient.

```
// Copyright (c) 2000
package com.client;

import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import person.*;
import javax.swing.border.*;
/**
 * A GUI class which represents the Data for a
<code>Service</code> in a JPanel.
 * <P>
```

```
 * @author Martin Braches
 */
public class ClientServicePanel extends
GenericPanelGUI {

  private Service service=null;
  private ClientFrame clientFrame = null;
  private ContactInformationPanel contactData;
              private ServiceDataPanel serviceData;

  // GUI elements
  BorderLayout borderLayout1 = new BorderLayout();
  BorderLayout borderLayout2 = new BorderLayout();
  JPanel bottomPanel = new JPanel();
  JPanel bottomButtonPanel = new JPanel();
  JButton jButton1 = new JButton();
  JPanel topPanel = new JPanel();
  JScrollPane otherScrollPane = new JScrollPane();
  JTextArea otherTextArea = new JTextArea();

  /**
   * Constructs a new instance.
   */
  public ClientServicePanel(ClientFrame clientFrame) {
    this.clientFrame = clientFrame;
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jButton1.setText("Daten an Koodinator senden");
    jButton1.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        jButton1_actionPerformed(e);
      }
    });
    bottomButtonPanel.add(jButton1, null);
    Border otherScrollPaneBevelBorder = new
BevelBorder(1);
    Border otherScrollPaneBorder = new
TitledBorder(otherScrollPaneBevelBorder,"Sonstiges");
    otherScrollPane.setBorder(otherScrollPaneBorder);
    otherScrollPane.getViewport().add(otherTextArea);
    otherScrollPane.setPreferredSize(new
Dimension(500,70));
    bottomPanel.setLayout(borderLayout2);
    bottomPanel.add(otherScrollPane,
BorderLayout.CENTER);
    bottomPanel.add(bottomButtonPanel,
BorderLayout.SOUTH);
    serviceData = new ServiceDataPanel();
    serviceData.jTextField1.setEditable(false);
    serviceData.setPreferredSize(new
Dimension(500,120));
    contactData = new ContactInformationPanel();
    contactData.addContacts();
    topPanel.setLayout(borderLayout1);
    topPanel.add(serviceData,BorderLayout.NORTH);
    topPanel.add(contactData, BorderLayout.CENTER);
    this.add(topPanel, BorderLayout.CENTER);
    this.add(bottomPanel, BorderLayout.SOUTH);
  }
  /**
   * Sents the ServiceData back to Server.
   */
  void jButton1_actionPerformed(ActionEvent e) {
    setServiceData();
    clientFrame.setServiceDataAtServer(service);
  }
  /**
   * Write the Data of the <code>service</code>object into the
GUI-Elements.
   */
  public void showService(Service service) {
    this.service=service;
    if (service != null)  {
      serviceData.setName(service.getName());

serviceData.setShortDescription(service.getShortDescription());
      if(service.getTypeID().intValue() == 1)
        serviceData.setType("Hausnotrufdienst");
      else if(service.getTypeID().intValue() == 2)
        serviceData.setType("Mahlzeitendienst");
      else if(service.getTypeID().intValue() == 3)
        serviceData.setType("ambulanter Pflegedienst");
      else serviceData.setType(""+service.getTypeID());
      serviceData.setFacility("keiner"); //+service.getFacilityID());
      serviceData.undoChanges();

      contactData.setAdresses(service.getAdresses());
      contactData.setEMails(service.getEMails());
      contactData.setPhoneNumbers(service.getPhoneNumbers());
      contactData.setBankAccounts(service.getBankAccounts());
      contactData.setContacts(service.getContacts());
      contactData.undoChanges();

      otherTextArea.setText(service.getOther());
    }
  }
  /**
   * Write the Data from the GUI-Elements into the
<code>service</code> object.
   */
  public void setServiceData() {
    serviceData.applyChanges();
    service.setName(serviceData.getName());

service.setShortDescription(serviceData.getShortDescription());
    service.setFacilityID(new Integer(0));
//serviceData.getFacility()));
    if(serviceData.getType().equals("Hausnotrufdienst"))
        service.setTypeID(new Integer(1));
    else if(serviceData.getType().equals("Mahlzeitendienst"))
        service.setTypeID(new Integer(2));
    else if(serviceData.getType().equals("ambulanter
Pflegedienst"))
        service.setTypeID(new Integer(3));
    else service.setTypeID(new Integer(0));
    contactData.applyChanges();
    service.setAdresses(contactData.getAdresses());
    service.setPhoneNumbers(contactData.getPhoneNumbers());
    service.setEMails(contactData.getEMails());
    service.setBankAccounts(contactData.getBankAccounts());
    service.setContats(contactData.getContacts());

    service.setOther(otherTextArea.getText());
  }
}
```

Die Klasse ServiceDataPanel

Diese Klasse wird für die Darstellung der Kontaktinformationen eines Dienstanbieters im Clientprogramm verwendet.

```
// Copyright (c) 2000
package com.client;

import javax.swing.*;
/**
 * A GUI class which represents a part of the Data for a
<code>Service</code> in a JPanel.
 * <P>
 * @author Martin Braches
 */
public class ServiceDataPanel extends
ServiceDataResource{

	private String name;
	private String shortDescription;
  private String type;
  private String facility;
  private JOptionPane dialog;
  /**
   * Constructs a new instance.
   */
  public ServiceDataPanel()  {
			this.name="";
			this.shortDescription="";
			this.type="";
			this.facility="";

			jTextField1.setText(name);

		jTextField2.setText(shortDescription);
			jTextField3.setText(type);
			jTextField4.setText(facility);
			setEditable(true);
		}
  /**
   * Constructs a new instance.
   */
		public ServiceDataPanel(String name,String
shortDescription,String type,String facility){
			this.name=name;

		this.shortDescription=shortDescription;
    this.facility=facility;
    this.type=type;
			jTextField1.setText(name);

		jTextField2.setText(shortDescription);
    jTextField3.setText(facility);
    jTextField4.setText(type);
			setEditable(true);
		}

		public void setEditable(boolean edit){
			if(edit==true){

		jTextField1.setEditable(true);

		jTextField2.setEditable(true);

		jTextField3.setEditable(true);
			jTextField4.setEditable(true);
			}
			if(edit==false){
			jTextField1.setEditable(false);
			jTextField2.setEditable(false);
			jTextField3.setEditable(false);
			jTextField4.setEditable(false);
			}

		}

		public void applyChanges(){
			this.name=jTextField1.getText();

		this.shortDescription=jTextField2.getText();
    this.facility=jTextField3.getText();
    this.type=jTextField4.getText();
		}

		public void undoChanges(){
			jTextField1.setText(name);
			jTextField2.setText(shortDescription);
    jTextField3.setText(facility);
    jTextField4.setText(type);
		}

		public ServiceDataPanel duplicate(){
			return new
ServiceDataPanel(name,shortDescription,facility,type);
		}
		public String getName(){
			return this.name;
		}

		public String getShortDescription(){
			return this.shortDescription;
		}
  public String getFacility(){
			return this.facility;
		}
  public String getType(){
			return this.type;
		}
  public void setName(String name){
			this.name = name;
		}
		public void setShortDescription(String
shortDescription){
			this.shortDescription = shortDescription;
		}
  public void setFacility(String facility){
			this.facility = facility;
		}
  public void setType(String type){
			this.type = type;
		}
}
```

Die Klasse ServiceDataResoure

Sie enthält und initialisiert die GUI-Elemente, die in de Klasse *ServiceDataPanel* dargestellt werden.

```
// Copyright (c) 2000
package com.client;

import java.awt.*;
import javax.swing.border.*;
import javax.swing.*;
/**
 * A GUI class which represents the GUI-Elements for
a <code>ServiceDataPanel</code>.
 *
 * @see com.client.ServiceDataPanel
 * <P>
 * @author Martin Braches
 */
public class ServiceDataResource extends JPanel {
  // Instance variables
  protected JPanel jPanel1;
  protected JPanel jPanel2;
  protected JPanel jPanel3;
  protected JPanel jPanel4;
  protected JPanel jPanel5;
  protected JPanel jPanel6;
  protected JLabel jLabel1;
  protected JLabel jLabel2;
  protected JLabel jLabel3;
  protected JLabel jLabel4;
  protected JLabel jLabel5;
  protected JLabel jLabel6;
  protected JLabel jLabel7;
  protected JTextField jTextField1;
  protected JTextField jTextField2;
  protected JTextField jTextField3;
  protected JTextField jTextField4;
  protected JTextField jTextField5;
  protected JTextField jTextField6;
  protected JTextField jTextField7;

  public ServiceDataResource() {
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  private void jbInit() throws Exception {
    // Setup GUI
    this.setPreferredSize(new Dimension(538,120));
    Border thisBorder1 = new BevelBorder(1);
    Border thisBorder0 = new
TitledBorder(thisBorder1,"Allgemeine Daten");
    this.setBorder(thisBorder0);
    this.setLocation(new Point(34,38));
    this.setName("this");
    jPanel1 = new JPanel();
    jPanel1.setPreferredSize(new Dimension(89,42));
    jPanel1.setMaximumSize(new
Dimension(2147483647,2147483647));
    jPanel1.setMinimumSize(new Dimension(61,42));
    jPanel1.setName("jPanel1");
    jPanel2 = new JPanel();
    jPanel2.setPreferredSize(new Dimension(110, 34));
    jPanel2.setMinimumSize(new Dimension(52, 34));
    jPanel2.setName("jPanel2");
    jLabel1 = new JLabel();
    jLabel1.setMaximumSize(new Dimension(33,17));
    jLabel1.setMinimumSize(new Dimension(33,17));
    jLabel1.setText("Name");
    jLabel1.setHorizontalAlignment(JLabel.RIGHT);
    jLabel1.setName("jLabel1");
    jLabel1.setLayout(null);
    jLabel2 = new JLabel();
    jLabel2.setMaximumSize(new Dimension(52,17));
    jLabel2.setMinimumSize(new Dimension(52,17));
    jLabel2.setText("Kurzbeschreibung");
    jLabel2.setHorizontalAlignment(JLabel.RIGHT);
    jLabel2.setName("jLabel2");
    jLabel2.setLayout(null);
    jLabel7 = new JLabel();
    jLabel7.setMaximumSize(new Dimension(52,17));
    jLabel7.setMinimumSize(new Dimension(52,17));
    jLabel7.setText("Träger");
    jLabel7.setHorizontalAlignment(JLabel.RIGHT);
    jLabel7.setName("jLabel7");
    jLabel7.setLayout(null);
    jLabel6 = new JLabel();
    jLabel6.setMaximumSize(new Dimension(52,17));
    jLabel6.setMinimumSize(new Dimension(52,17));
    jLabel6.setText("Art des Dienstes");
    jLabel6.setHorizontalAlignment(JLabel.RIGHT);
    jLabel6.setName("jLabel6");
    jLabel6.setLayout(null);
    jLabel3 = new JLabel();
    jLabel3.setMaximumSize(new Dimension(33,17));
    jLabel3.setMinimumSize(new Dimension(33,17));
    jLabel3.setText("Beschreibung");
    jLabel3.setHorizontalAlignment(JLabel.RIGHT);
    jLabel3.setName("jLabel3");
    jLabel3.setLayout(null);
    GridLayout jPanel2Layout = new GridLayout(1,1);
    jPanel2Layout.setHgap(0);
    jPanel2Layout.setVgap(5);
    jPanel2Layout.setRows(4);
    jPanel2Layout.setColumns(1);
    jPanel2.setLayout(jPanel2Layout);
    jPanel2.add(jLabel1);
    jPanel2.add(jLabel2);
    jPanel2.add(jLabel7);
    jPanel2.add(jLabel6);
    jPanel5 = new JPanel();
    jPanel5.setPreferredSize(new Dimension(4,42));
    jPanel5.setMinimumSize(new Dimension(4,42));
    jPanel5.setName("jPanel5");
    jTextField1 = new JTextField();
    jTextField1.setPreferredSize(new Dimension(150,21));
    jTextField1.setName("jTextField1");
    jTextField1.setLayout(null);
    jTextField2 = new JTextField();
    jTextField2.setPreferredSize(new Dimension(150,21));
    jTextField2.setName("jTextField2");
    jTextField2.setLayout(null);
    jTextField3 = new JTextField();
    jTextField3.setPreferredSize(new Dimension(150,21));
    jTextField3.setName("jTextField3");
    jTextField3.setLayout(null);
    jTextField4 = new JTextField();
    jTextField4.setPreferredSize(new Dimension(150,21));
    jTextField4.setName("jTextField4");
    jTextField4.setLayout(null);
    jTextField5 = new JTextField();
    jTextField5.setPreferredSize(new Dimension(150,21));
```

```
    jTextField5.setName("jTextField5");
    jTextField5.setLayout(null);
    GridLayout jPanel5Layout = new GridLayout(1,1);
    jPanel5Layout.setHgap(0);
    jPanel5Layout.setVgap(5);
    jPanel5Layout.setRows(4);
    jPanel5Layout.setColumns(1);
    jPanel5.setLayout(jPanel5Layout);
    jPanel5.add(jTextField1);
    jPanel5.add(jTextField2);
    jPanel5.add(jTextField3);
    jPanel5.add(jTextField4);

    BorderLayout jPanel1Layout = new BorderLayout();
    jPanel1.setLayout(jPanel1Layout);
    jPanel1Layout.setHgap(5);
    jPanel1Layout.setVgap(0);
    jPanel1.add(jPanel2, BorderLayout.WEST);
    jPanel1.add(jPanel5, BorderLayout.CENTER);

    GridLayout thisLayout = new GridLayout(1,1);
    thisLayout.setHgap(10);
    thisLayout.setVgap(0);
    thisLayout.setRows(1);
    thisLayout.setColumns(2);
    this.setLayout(thisLayout);
    this.add(jPanel1);

  }
}
```

Die Klasse CareClientPanel

Sie stellt die Kundendaten im Koordinationsclient dar.

```
// Copyright (c) 2000
package com.client;

import person.*;
import java.util.Vector;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.event.*;


/**
 * A GUI class which represents the Data for a
<code>CareClient</code> in a JPanel.
 * <P>
 * @author Martin Braches
 */
public class CareClientPanel extends GenericPanelGUI
{

  private ContactInformationPanel contactData;
              private PersonalDataPanel personalData;
              private CareClient careClient;
  private ClientFrame clientFrame = null;

  // GUI Elements
  BorderLayout borderLayout1 = new BorderLayout();
  BorderLayout borderLayout2 = new BorderLayout();
  JToolBar jToolBar1 = new JToolBar();
  JButton requestedServicesButton = new JButton();
  JButton physicianButton = new JButton();
  JButton healthInsuranceButton = new JButton();
  JPanel topPanel = new JPanel();
  JPanel bottomPanel = new JPanel();
  JScrollPane remarkScrollPane = new JScrollPane();
  JTextArea remarkTextArea = new JTextArea();
  JPanel bottomButtonPanel = new JPanel();
  JButton messageButton = new JButton();
  JButton saveButton = new JButton();
  JButton deleteButton = new JButton();


  /**
   * Constructs a new instance.
   */
  public CareClientPanel(CareClient
careClient,ClientFrame clientFrame) {
    this.careClient=careClient;
    this.clientFrame = clientFrame;
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
                personalData = new
PersonalDataPanel(careClient.getName(),careClient.getForenam
e(),

careClient.getBirthday(),careClient.getTownOfBirth(),careClient
.getMartialStatus(),

careClient.getNationality(),careClient.getReligion());
    contactData = new
ContactInformationPanel(careClient.getAdresses(),careClient.ge
tEMails(),

careClient.getPhoneNumbers(),careClient.getBankAccounts());
    contactData.removeButtons();
    remarkTextArea.setText(careClient.getRemark());
    topPanel.setLayout(borderLayout1);
    topPanel.add(personalData,BorderLayout.NORTH);
    topPanel.add(contactData, BorderLayout.CENTER);

    requestedServicesButton.setText("Dienste");
    requestedServicesButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        requestedServicesButton_actionPerformed(e);
      }
    });
    physicianButton.setText("Ärzte");
    physicianButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        physicianButton_actionPerformed(e);
      }
    });
    healthInsuranceButton.setText("Kranken/Pflegekasse");
    healthInsuranceButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        healthInsuranceButton_actionPerformed(e);
      }
    });
```

```
    messageButton.setText("Nachricht an Koordinator");
    messageButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        messageButton_actionPerformed(e);
      }
    });
   saveButton.setText("Daten speichern");
    saveButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        saveButton_actionPerformed(e);
      }
    });
    deleteButton.setText("Daten löschen");
    deleteButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        deleteButton_actionPerformed(e);
      }
    });

    jToolBar1.setName("Kundenanfrage Toolbar");

jToolBar1.setBackground(SystemColor.textHighlight);

    jToolBar1.add(requestedServicesButton,null);
    jToolBar1.add(physicianButton, null);
    jToolBar1.add(healthInsuranceButton, null);

    Border remarkScrollPaneBevelBorder = new
BevelBorder(1);
    Border remarkScrollPaneBorder = new
TitledBorder(remarkScrollPaneBevelBorder,"Bemerku
ng");

remarkScrollPane.setBorder(remarkScrollPaneBorder);

remarkScrollPane.getViewport().add(remarkTextArea);
    remarkScrollPane.setPreferredSize(new
Dimension(500,70));

    bottomButtonPanel.add(messageButton, null);
    bottomButtonPanel.add(saveButton, null);
    bottomButtonPanel.add(deleteButton, null);

    bottomPanel.setLayout(borderLayout2);
    bottomPanel.add(remarkScrollPane,
BorderLayout.CENTER);
    bottomPanel.add(bottomButtonPanel,
BorderLayout.SOUTH);

    this.add(jToolBar1, BorderLayout.NORTH);
    this.add(topPanel, BorderLayout.CENTER);
    this.add(bottomPanel, BorderLayout.SOUTH);

  }
  /**
   * Shows a ClientRequestedServicesDialog.
   * <P>
   * @see com.client.ClientRequestedServicesDialog
   */
  void requestedServicesButton_actionPerformed(ActionEvent e)
{
    ClientRequestedServicesDialog rsd = new
ClientRequestedServicesDialog(careClient,null,"Dienste",true);
    rsd.show();
  }
  /**
   * Shows a ClientPhysicianDialog.
   * <P>
   * @see com.client.ClientPhysicianDialog
   */
  void physicianButton_actionPerformed(ActionEvent e) {
    ClientPhysicianDialog ccpd = new
ClientPhysicianDialog(careClient,null,"Ärzte",true);
    ccpd.show();
  }
  /**
   * Shows a ClientPaymentInstitutionDialog.
   * <P>
   * @see com.client.ClientPaymentInstitutionDialog
   */
  void healthInsuranceButton_actionPerformed(ActionEvent e) {
    ClientPaymentInstitutionDialog ccpid = new
ClientPaymentInstitutionDialog(careClient,null,"Kassen",true);
    ccpid.show();
  }
  /**
   * Shows an InputDialog and sends a Message to the
CoordinatoinModuleServer.
   * <P>
   */
  void messageButton_actionPerformed(ActionEvent e) {
    Object[] options = {"es wurde ein Vertrag abgeschlossen","es
konnte kein Vertrag abgeschlossen werden"};
    String message
=(String)JOptionPane.showInputDialog(this,"Nachricht an
Koordinator","neue
Nachricht",JOptionPane.PLAIN_MESSAGE,null,options,option
s[0]);
    if (message != null)
      clientFrame.sentMessage(careClient, message);
  }
  /**
   * This function is not yet implemented in this prototype.
   */
  void saveButton_actionPerformed(ActionEvent e) {
  }
  /**
   * Removes this CareClientPanel from his parent JTabbedPane.
   */
  void deleteButton_actionPerformed(ActionEvent e) {
    clientFrame.removeCareClient(careClient,this);
  }
}
```

Die Klasse ClientRequestedServicesDialog

Diese Klasse stellt den Dienste-Dialog für die Maske *Kundendaten* bereit.

```
// Copyright (c) 2000 .00
package com.client;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.util.Vector;
import java.awt.BorderLayout;
import java.awt.event.*;
import person.*;
/**
 * A Swing-based dialog class.
 * It represents the <code>RequestedServices</code> for a
specific <code>CareClient</code>.
 * It looks like CareClientRequestedServicesDialog but has less
functionality.
```

```
 * <P>
 * @author Martin Braches
 */
public class ClientRequestedServicesDialog extends
JDialog {

  private CareClient careClient = null;

  // GUI elements
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JCheckBox jCheckBox1 = new JCheckBox();
  JCheckBox jCheckBox2 = new JCheckBox();
  JCheckBox jCheckBox3 = new JCheckBox();
  JCheckBox jCheckBox4 = new JCheckBox();
  JLabel jLabel1 = new JLabel();
  JCheckBox jCheckBox5 = new JCheckBox();
  JCheckBox jCheckBox6 = new JCheckBox();
  JCheckBox jCheckBox7 = new JCheckBox();
  JCheckBox jCheckBox11 = new JCheckBox();
  JPanel jPanel4 = new JPanel();
  JButton saveButton = new JButton();
  JButton cancelButton = new JButton();
  JPanel jPanel3 = new JPanel();
  BorderLayout borderLayout2 = new BorderLayout();
  JPanel jPanel5 = new JPanel();
  JCheckBox jCheckBox8 = new JCheckBox();
  JCheckBox jCheckBox9 = new JCheckBox();
  JCheckBox jCheckBox10 = new JCheckBox();
  GridLayout gridLayout1 = new GridLayout();


  /**
   * Constructs a new instance.
   * @param parent
   * @param title
   * @param modal
   */
  public ClientRequestedServicesDialog(Frame parent,
String title, boolean modal) {
    super(parent, title, modal);
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  public ClientRequestedServicesDialog(CareClient
careClient,Frame parent, String title, boolean modal) {
    super(parent, title, modal);
    this.careClient=careClient;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new non-modal unparented instance
with a blank title.
   */
  public ClientRequestedServicesDialog() {
    this(null, "", false);
  }
  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jPanel1.setLayout(borderLayout1);
    jPanel1.setPreferredSize(new Dimension(530, 200));
    jPanel1.setMinimumSize(new Dimension(200, 200));
    jCheckBox1.setText("SGB V");
    jCheckBox2.setText("BSHG");
    jCheckBox3.setText("Selbstzahler");
    jCheckBox4.setText("SGB XI");
    jLabel1.setText("Pflegestufe");
    jCheckBox5.setText("1");
    jCheckBox6.setText("2");
    jCheckBox7.setText("3");
    jCheckBox11.setText("3a");
    jPanel2.setEnabled(false);
    disableAll();
    saveButton.setText("Speichern");
    saveButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //saveButton_actionPerformed(e);
      }
    });
    cancelButton.setText("Schließen");
    cancelButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        cancelButton_actionPerformed(e);
      }
    });
    jPanel4.add(cancelButton, null);
    Border border1 = new BevelBorder(1);
    Border border2 = new TitledBorder(border1,"Leistungsträger
der ambulanten Versorgung");
    gridLayout1.setRows(3);
    jPanel5.setLayout(gridLayout1);
    jCheckBox11.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox11_actionPerformed(e);
      }
    });
    jCheckBox7.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox7_actionPerformed(e);
      }
    });
    jCheckBox6.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox6_actionPerformed(e);
      }
    });
    jCheckBox5.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox5_actionPerformed(e);
      }
    });
    jCheckBox4.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox4_actionPerformed(e);
      }
    });
    jCheckBox3.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox3_actionPerformed(e);
      }
    });
    jCheckBox2.addActionListener(new
java.awt.event.ActionListener() {
```

```
      public void actionPerformed(ActionEvent e) {
        //jCheckBox2_actionPerformed(e);
      }
    });
    jCheckBox1.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox1_actionPerformed(e);
      }
    });
    jCheckBox10.setText("ambulante Versorgung");
    jCheckBox10.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox10_actionPerformed(e);
      }
    });
    jCheckBox9.setText("Menueservice");
    jCheckBox9.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox9_actionPerformed(e);
      }
    });
    jCheckBox8.setText("Hausnotruf");
    jCheckBox8.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        //jCheckBox8_actionPerformed(e);
      }
    });
    jPanel3.setLayout(borderLayout2);
    jPanel2.setBorder(border2);
    jPanel2.add(jCheckBox1, null);
    jPanel2.add(jCheckBox2, null);
    jPanel2.add(jCheckBox3, null);
    jPanel2.add(jCheckBox4, null);
    jPanel2.add(jLabel1, null);
    jPanel2.add(jCheckBox5, null);
    jPanel2.add(jCheckBox6, null);
    jPanel2.add(jCheckBox7, null);
    jPanel2.add(jCheckBox11, null);
    jPanel1.add(jPanel5, BorderLayout.CENTER);
    jPanel5.add(jCheckBox8, null);
    jPanel5.add(jCheckBox9, null);
    jPanel5.add(jCheckBox10, null);
    jPanel1.add(jPanel3, BorderLayout.SOUTH);
    jPanel3.add(jPanel4, BorderLayout.SOUTH);
    jPanel3.add(jPanel2, BorderLayout.NORTH);
    getContentPane().add(jPanel1);
    setRequestedServices();
  }
  /**
   * Shows this dialog and centers it on screen.
   */
  public void show() {
    this.toFront();
    this.setResizable(false);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    this.setLocation((screenSize.width -
frameSize.width)/2, (screenSize.height -
frameSize.height)/2);
    super.show();
  }
  /**
   * Disables all checkboxes in this dialog.
   */
  void disableAll()  {
    jCheckBox1.setEnabled(false);
    jCheckBox2.setEnabled(false);
    jCheckBox3.setEnabled(false);
    jCheckBox4.setEnabled(false);
    jCheckBox1.setSelected(false);
    jCheckBox2.setSelected(false);
    jCheckBox3.setSelected(false);
    jCheckBox4.setSelected(false);
    jLabel1.setEnabled(false);
    jCheckBox5.setEnabled(false);
    jCheckBox6.setSelected(false);
    jCheckBox7.setEnabled(false);
    jCheckBox11.setEnabled(false);
    jCheckBox5.setSelected(false);
    jCheckBox6.setEnabled(false);
    jCheckBox7.setSelected(false);
    jCheckBox11.setSelected(false);
  }
  /**
   * Enables all checkboxes in this dialog.
   */
  void enableAll()  {
    jCheckBox1.setEnabled(true);
    jCheckBox2.setEnabled(true);
    jCheckBox3.setEnabled(true);
    jCheckBox4.setEnabled(true);
    jLabel1.setEnabled(true);
    jCheckBox5.setEnabled(true);
    jCheckBox7.setEnabled(true);
    jCheckBox11.setEnabled(true);
    jCheckBox6.setEnabled(true);
  }
  /**
   * Enables the checkboxes in dependence of the
requestedServices of the CareClient
   */
  void setRequestedServices() {
    Vector requestedServiceVector =
careClient.getRequestedServices();
    for (int a=0;a<requestedServiceVector.size();a++) {
      RequestedService tempRS
=(RequestedService)requestedServiceVector.elementAt(a);
      if (tempRS.getName().equals("Hausnotruf"))  {
        jCheckBox8.setSelected(true);
        jCheckBox8.setText(jCheckBox8.getText());
      }
      if (tempRS.getName().equals("Menueservice"))  {
        jCheckBox9.setSelected(true);
        jCheckBox9.setText(jCheckBox9.getText());
      }
      if (tempRS.getName().equals("ambulante Versorgung"))  {
        jCheckBox10.setSelected(true);
        jCheckBox10.setText(jCheckBox10.getText());
        enableAll();
        if (tempRS.getPaymentInstitution().equals("SGB XI"))  {
          jCheckBox4.setSelected(true);
          int careClassification = careClient.getCareClassification();
          if (careClassification == 1)
            jCheckBox5.setSelected(true);
          if (careClassification == 2)
            jCheckBox6.setSelected(true);
          if (careClassification == 3)
            jCheckBox7.setSelected(true);
          if (careClassification == 4)
            jCheckBox11.setSelected(true);
        }
        if (tempRS.getPaymentInstitution().equals("Selbstzahler"))
          jCheckBox3.setSelected(true);
        if (tempRS.getPaymentInstitution().equals("BSHG"))
```

```
        jCheckBox2.setSelected(true);
      if (tempRS.getPaymentInstitution().equals("SGB
V"))
        jCheckBox1.setSelected(true);
    }
   }
  }
```

```
  /**
   * Disposes this dialog.
   */
  void cancelButton_actionPerformed(ActionEvent e) {
    this.dispose();
  }
}
```

Die Klasse ClientPhysicianDialog

Diese Klasse stellt den Ärzte-Dialog für die Maske *Kundendaten* bereit.

```
// Copyright (c) 2000 .00
package com.client;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import person.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.sql.*;
import java.util.*;

/**
 * A Swing-based dialog class.
 * It represents the <code>Physicians</code> for a
specific <code>CareClient</code>.
 * It looks like CareClientPhysicianDialog but has less
functionality.
 * <P>
 * @author Martin Braches
 */
public class ClientPhysicianDialog extends JDialog {

  private ContactInformationPanel contactData;
              private PhysiciansGUI physicianData;
              private CareClient careClient = null;

  // GUI elements
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JButton addButton = new JButton();
  JButton closeButton = new JButton();
  JButton removeButton = new JButton();
  JButton changeButton = new JButton();
  JButton saveButton = new JButton();
  /**
   * Constructs a new instance.
   * @param parent
   * @param title
   * @param modal
   */
  public ClientPhysicianDialog(Frame parent, String
title, boolean modal) {
    super(parent, title, modal);
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  public ClientPhysicianDialog(CareClient careClient,
Frame parent1, String title, boolean modal) {
    super(parent1, title, modal);
    this.careClient = careClient;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new non-modal unparented instance with a
blank title.
   */
  public ClientPhysicianDialog() {
    this(null, "", false);
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jPanel1.setLayout(borderLayout1);
    jPanel1.setPreferredSize(new Dimension(500, 300));
    jPanel1.setMinimumSize(new Dimension(200, 300));
    closeButton.setText("Schließen");
    closeButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        closeButton_actionPerformed(e);
      }
    });
    physicianData = new
PhysiciansGUI(careClient.getPhysicians());
    physicianData.undoChanges();
    physicianData.setPreferredSize(new Dimension(150, 100));
    physicianData.setMinimumSize(new Dimension(150, 100));

    contactData = new ContactInformationPanel();
    contactData.removeButtons();
    contactData.addVisitingHours();
    physicianData.setContactData(contactData);
    getContentPane().add(jPanel1);
    jPanel1.add(physicianData, BorderLayout.NORTH);
    jPanel1.add(contactData, BorderLayout.CENTER);
    jPanel1.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(closeButton, null);
  }
  /**
   * Shows this dialog and centers it on screen.
   */
  public void show() {
    this.toFront();
    this.setResizable(false);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
```

```
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    this.setLocation((screenSize.width -
frameSize.width)/2, (screenSize.height -
frameSize.height)/2);
    super.show();
  }
  /**
   * Disposes this dialog.
   */
  void closeButton_actionPerformed(ActionEvent e) {
    this.dispose();
  }
}
```

Die Klasse ClientPaymentInstitutionDialog

Diese Klasse stellt den Kranken/Pflegekassen-Dialog für die Maske *Kundendaten* bereit.

```
// Copyright (c) 2000
package com.client;

import javax.swing.*;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import person.*;
import javax.swing.event.ListSelectionListener;
import javax.swing.event.ListSelectionEvent;
import java.sql.*;
import java.util.*;

/**
 * A Swing-based dialog class.
 * It represents the <code>PaymentInstitutions</code>
for a specific <code>CareClient</code>.
 * It looks like CareClientPaymentInstitutionDialog but
has less functionality.
 * <P>
 * @author Martin Braches
 */
public class ClientPaymentInstitutionDialog extends
JDialog {

  private ContactInformationPanel contactData;
              private PaymentInstitutionsGUI
paymentInstitutionData;;
              private CareClient careClient = null;

  //GUI elements
  BorderLayout borderLayout1 = new BorderLayout();
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JButton addButton = new JButton();
  JButton closeButton = new JButton();
  JButton removeButton = new JButton();
  JButton changeButton = new JButton();
  JButton saveButton = new JButton();
  /**
   * Constructs a new instance.
   * @param parent
   * @param title
   * @param modal
   */
  public ClientPaymentInstitutionDialog(Frame parent,
String title, boolean modal) {
    super(parent, title, modal);
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  public ClientPaymentInstitutionDialog(CareClient careClient,
Frame parent1, String title, boolean modal) {
    super(parent1, title, modal);
    this.careClient = careClient;
    try  {
      jbInit();
      pack();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }
  /**
   * Constructs a new non-modal unparented instance with a
blank title.
   */
  public ClientPaymentInstitutionDialog() {
    this(null, "", false);
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {
    jPanel1.setLayout(borderLayout1);
    jPanel1.setPreferredSize(new Dimension(500, 300));
    jPanel1.setMinimumSize(new Dimension(200, 300));
    closeButton.setText("Schließen");
    closeButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        closeButton_actionPerformed(e);
      }
    });
    paymentInstitutionData = new
PaymentInstitutionsGUI(careClient.getPaymentInstitutions());
    paymentInstitutionData.undoChanges();
    paymentInstitutionData.setPreferredSize(new Dimension(150,
100));
    paymentInstitutionData.setMinimumSize(new
Dimension(150, 100));

    contactData = new ContactInformationPanel();
    contactData.removeButtons();
    paymentInstitutionData.setContactData(contactData);
    getContentPane().add(jPanel1);
    jPanel1.add(paymentInstitutionData, BorderLayout.NORTH);
    jPanel1.add(contactData, BorderLayout.CENTER);
    jPanel1.add(jPanel2, BorderLayout.SOUTH);
    jPanel2.add(closeButton, null);
  }
  /**
   * Shows this dialog and centers it on screen.
   */
  public void show() {
```

```
    this.toFront();
    this.setResizable(false);
    //Center the window
    Dimension screenSize =
Toolkit.getDefaultToolkit().getScreenSize();
    Dimension frameSize = this.getSize();
    if (frameSize.height > screenSize.height) {
      frameSize.height = screenSize.height;
    }
    if (frameSize.width > screenSize.width) {
      frameSize.width = screenSize.width;
    }
    this.setLocation((screenSize.width - frameSize.width)/2,
(screenSize.height - frameSize.height)/2);
    super.show();
  }
  /**
                * Disposes this dialog.
        */
  void closeButton_actionPerformed(ActionEvent e) {
    this.dispose();
  }
}
```

Die Klasse CareClientServiceProfilePanel

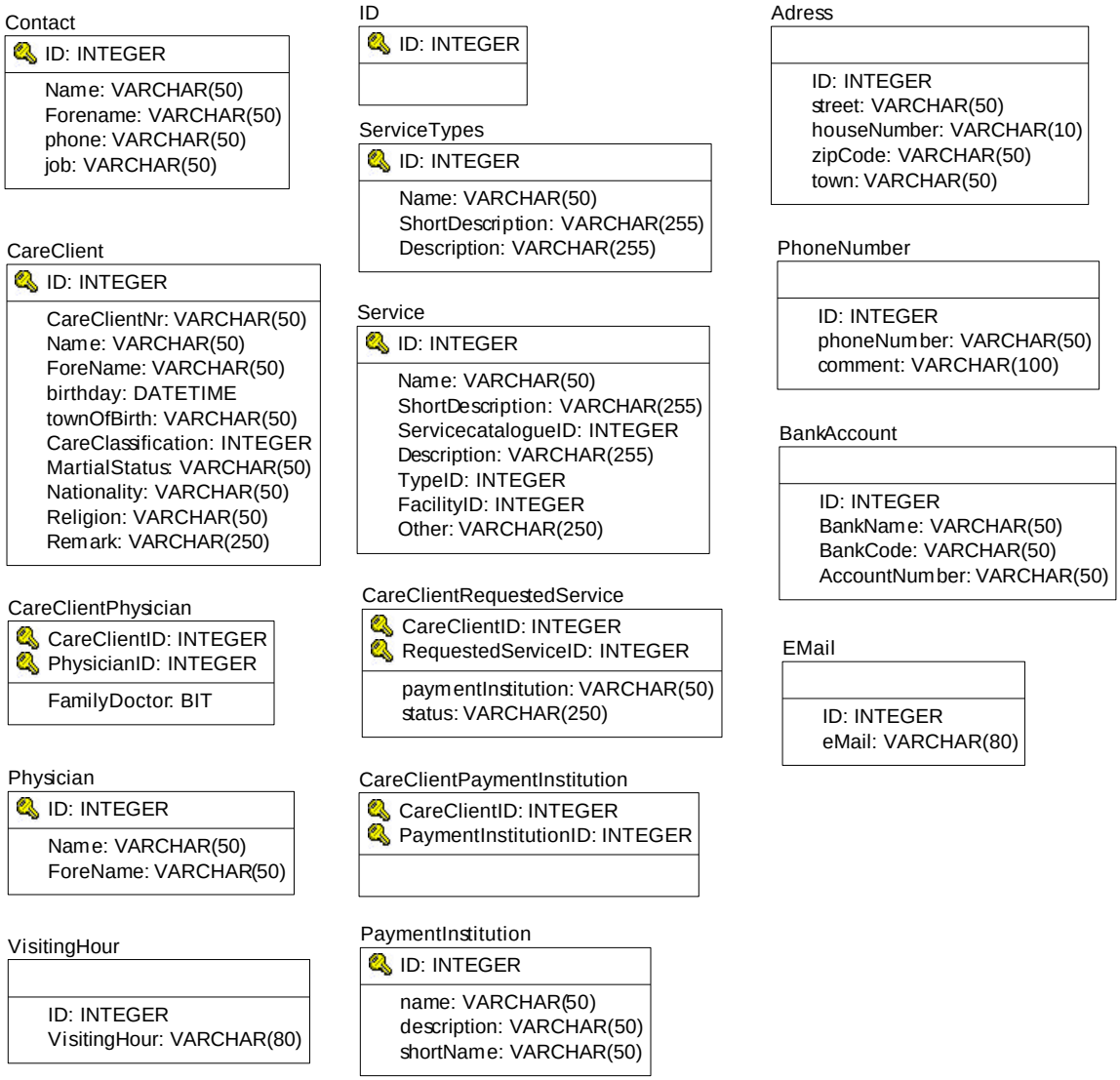
Diese Klasse stellt die Maske *Leistungsanfrage* im Koordinationsclient dar.

```
// Copyright (c) 2000
package com.client;

import com.server.*;
import person.*;
import java.util.Vector;
import javax.swing.border.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import javax.swing.table.*;
import javax.swing.event.*;
import oracle.jdeveloper.layout.*;

/**
 * A GUI class which represents the Data for a client
request in a JPanel.
 * <P>
 * @author Martin Braches
 */
public class CareClientServiceProfilePanel extends
GenericPanelGUI {

  private ClientFrame clientFrame = null;
  private ServiceProfile serviceProfile;
  private SecureCareClient secureCareClient;

  // GUI Elements
  JPanel jPanel1 = new JPanel();
  JPanel jPanel2 = new JPanel();
  JPanel jPanel3 = new JPanel();
  JScrollPane jScrollPane1 = new JScrollPane();
  JButton messageButton = new JButton();
  JButton saveButton = new JButton();
  JButton deleteButton = new JButton();
  ServiceProfileTableModel tableModel = new
ServiceProfileTableModel();
  JTable jTable1 = new JTable(tableModel);
  BorderLayout borderLayout1 = new BorderLayout();
  XYLayout xYLayout1 = new XYLayout();
  JLabel jLabel1 = new JLabel();
  JLabel jLabel2 = new JLabel();
  JLabel jLabel3 = new JLabel();
  JLabel jLabel4 = new JLabel();
  JLabel jLabel5 = new JLabel();
  JLabel jLabel6 = new JLabel();
  JTextField jTextField1 = new JTextField();
  JTextField jTextField2 = new JTextField();
  JTextField jTextField3 = new JTextField();
  JTextField jTextField4 = new JTextField();
  JTextField jTextField5 = new JTextField();
  JTextField jTextField6 = new JTextField();

  /**
   * Constructs a new instance.
   */
  public CareClientServiceProfilePanel(ClientFrame
clientFrame,ServiceProfile serviceProfile) {
    this.clientFrame = clientFrame;
    this.serviceProfile = serviceProfile;
    this.secureCareClient = serviceProfile.getSecureCareClient();
    try  {
      jbInit();
    }
    catch (Exception e) {
      e.printStackTrace();
    }
  }

  /**
   * Initializes the state of this instance.
   */
  private void jbInit() throws Exception {


tableModel.setData(serviceProfile.getData(),serviceProfile.getC
olumnNames());
    setUpTable();
    jPanel3.setLayout(borderLayout1);
    jPanel2.setPreferredSize(new Dimension(100, 80));
    jPanel2.setLayout(xYLayout1);
    jPanel2.setMinimumSize(new Dimension(100, 100));
    messageButton.setText("Nachricht an Koordinator");
    messageButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        messageButton_actionPerformed(e);
      }
    });
    saveButton.setText("Daten speichern");
    saveButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
        saveButton_actionPerformed(e);
      }
    });
    deleteButton.setText("Daten löschen");
    jLabel1.setText("Pflegestufe");
    jLabel2.setText("Familienstand");
    jLabel3.setText("Nationalität");
    jLabel4.setText("Religion");
    jLabel5.setText("Straße");
    jLabel6.setText("Ort");
    deleteButton.addActionListener(new
java.awt.event.ActionListener() {
      public void actionPerformed(ActionEvent e) {
```

```
      deleteButton_actionPerformed(e);
    }
   });
   this.add(jPanel3, BorderLayout.SOUTH);
   jPanel3.add(jPanel2, BorderLayout.NORTH);
   jPanel2.add(jLabel1, new XYConstraints(8, 3, -1, -
1));
   jPanel2.add(jLabel2, new XYConstraints(8, 29, -1, -
1));
   jPanel2.add(jLabel3, new XYConstraints(8, 54, -1, -
1));
   jPanel2.add(jLabel4, new XYConstraints(234, 3, -1, -
1));
   jPanel2.add(jLabel5, new XYConstraints(243, 29, -1,
-1));
   jPanel2.add(jLabel6, new XYConstraints(264, 54, -1,
-1));
   jPanel2.add(jTextField1, new XYConstraints(90, 1,
109, -1));
   jPanel2.add(jTextField2, new XYConstraints(90, 27,
109, -1));
   jPanel2.add(jTextField3, new XYConstraints(90, 52,
109, 21));
   jPanel2.add(jTextField4, new XYConstraints(286, 1,
109, 21));
   jPanel2.add(jTextField5, new XYConstraints(286,
27, 109, -1));
   jPanel2.add(jTextField6, new XYConstraints(286,
52, 109, 21));
   jPanel3.add(jPanel1, BorderLayout.SOUTH);
   jPanel1.add(messageButton, null);
   jPanel1.add(saveButton, null);
   jPanel1.add(deleteButton, null);
   this.add(jScrollPane1, BorderLayout.CENTER);
   jScrollPane1.getViewport().add(jTable1, null);

   if (secureCareClient.getCareClassification() == 4)
     jTextField1.setText("3a");
   else

jTextField1.setText(""+secureCareClient.getCareClassi
fication());

jTextField2.setText(secureCareClient.getMartialStatus(
));

jTextField3.setText(secureCareClient.getNationality());
   jTextField4.setText(secureCareClient.getReligion());
   jTextField5.setText(secureCareClient.getStreeet());
   jTextField6.setText(secureCareClient.getTown());
  }
  /**
   * Sets up the JTable with a
ServiceProfileCellRenderer.
   * <P>
   * @see com.server.ServiceProfileCellRenderer
   */
  public void setUpTable()  {
    TableColumn col0 =
jTable1.getColumn(tableModel.getColumnName(0));
    col0.setPreferredWidth(180);
    ServiceProfileCellRenderer
serviceProfileCellRenderer = new
ServiceProfileCellRenderer(true);
    TableColumn col1 =
jTable1.getColumn(tableModel.getColumnName(1));
    col1.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col2 =
jTable1.getColumn(tableModel.getColumnName(2));
    col2.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col3 =
jTable1.getColumn(tableModel.getColumnName(3));
    col3.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col4 =
jTable1.getColumn(tableModel.getColumnName(4));
    col4.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col5 =
jTable1.getColumn(tableModel.getColumnName(5));
    col5.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col6 =
jTable1.getColumn(tableModel.getColumnName(6));
    col6.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col7 =
jTable1.getColumn(tableModel.getColumnName(7));
    col7.setCellRenderer(serviceProfileCellRenderer);
    TableColumn col8 =
jTable1.getColumn(tableModel.getColumnName(8));
    col8.setPreferredWidth(40);
    TableColumn col9 =
jTable1.getColumn(tableModel.getColumnName(9));
    col9.setPreferredWidth(50);
    TableColumn col10 =
jTable1.getColumn(tableModel.getColumnName(10));

  }
  /**
   * Shows an InputDialog and sends a Message to the
CoordinationModuleServer.
   * <P>
   * @see com.server.Server
   */
  void messageButton_actionPerformed(ActionEvent e) {
    Object[] options = {"Unser Dienst kann die Anfrage
Nr:"+secureCareClient.getID()+" übernehmen.","Unser Dienst
kann die Anfrage Nr:"+secureCareClient.getID()+" nicht
übernehmen."};
    String message
=(String)JOptionPane.showInputDialog(this,"Nachricht an
Koordinator","neue
Nachricht",JOptionPane.PLAIN_MESSAGE,null,options,option
s[0]);
    if (message != null)
      clientFrame.sentMessage((Object)secureCareClient,
message);
  }
  /**
   * This function is not yet implemented in this prototype.
   */
  void saveButton_actionPerformed(ActionEvent e) {
  }
  /**
   * Removes this CareClientServiceProfilePanel from his parent
JTabbedPane.
   */
  void deleteButton_actionPerformed(ActionEvent e) {
    clientFrame.removeServiceProfilePanel(this);
  }

}
```

Anhang B

Übersicht über die Tabellen der Datenbank:



Anhang C

Literaturverzeichnis:

[Bmg00]	www.bmgesundheit.de/pflege/versich/vers.htm	am 26.6.2000
[ORA00]	technet.oracle.com	am 27.6.2000
[OMG00]	www.omg.org/news/about/	am 9.10.2000
[Sun00]	www.java.sun.com	am 27.6.2000

[Klie99] Thomas Klie und Roland Schmidt : **Die neue Pflege alter Menschen**, Bern 1999

[Oest97] Oestereich Bernd : **Objektorientierte Softwareentwicklung mit der Unified Modeling Language**, München 1997

[Flan98] David Flanagan : **Java in a Nutshell**, Köln 1998

[Flan98-2] David Flanagan : **Java Examples in a Nutshell**, Köln 1998

[Horstmann00] Cay S. Horstmann, Gary Cornell : **Core Java 2, Band.2, Expertenwissen**, 2000

[Gamma96] E.Gamma, R.Helm, R.Johnson, J.Vlissides **: Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software**, Bonn 1996

[Vlissides99] John Vlissides : **Entwurfsmuster anwenden**, München 1999

[Eckel96] Bruce Eckel : **Thinking in Java**, Upper Saddle River, 1996

[Quibeldey99] Klaus Quibeldey-Cirkel :**Entwurfsmuster: Design Patterns in der objektorientierten Softwaretechnik**, Berlin 1999

[Hildebrand95] Knut Hildebrand : **Informationsmanagement - wettbewerbsorientierte Informationsverarbeitung**, München 1995

[Gong98] L. Gong and R. Schemers : **Signing, Sealing, and Guarding Java Objects**. Lecture Notes in Computer Science (LNCS), Vol.1419, Springer-Verlag, 1998

[Müller97] Günter Müller, Andreas Pfitzmann : **Mehrseitige Sicherheit in der Kommunikationstechnik, Verfahren, Komponenten, Integration**, Addison Wesley Longman Verlag GmbH, 1999