INSTITUT FÜR INFORMATIK

DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Eine Architektur für SMS-basierte Dienste

Ines Riedl

Aufgabensteller:	Prof. Dr. Claudia Linnhoff-Popien
Betreuer:	Nils Reimelt (Burda) Jörg Sigmund (Burda) Axel Küpper
Abgabetermin:	8. September 2003

INSTITUT FÜR INFORMATIK
DER LUDWIG–MAXIMILIANS–UNIVERSITÄT MÜNCHEN

Diplomarbeit

Eine Architektur für SMS-basierte Dienste

Ines Riedl

Aufgabensteller: Prof. Dr. Claudia Linnhoff-Popien

Betreuer: Nils Reimelt (Burda)
Jörg Sigmund (Burda)
Axel Küpper

Abgabetermin: 8. September 2003

Hiermit versichere ich, dass ich die vorliegende Diplomarbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet habe.

München, den 8. September 2003

. . . .

(Unterschrift des Kandidaten)

Zusammenfassung

Die Beliebtheit des Short Message Service lässt immer mehr Unternehmen dazu übergehen, ihren Kunden mobile Dienste auf Basis des Formats SMS anzubieten. BURDAWIRELESS, ein New Media Kompetenzzentrum des Hubert Burda Media Verlags, besitzt eine SMS-Plattform zur Realisierung von SMS-basierten Diensten und stellt sie Redaktionen und Kunden von Burda zur Verfügung. Die Entwicklung von neuen Diensten und die Anpassung bereits realisierter Dienst ist allerdings durch den rudimentären Zustand der Plattform mit hohem Entwicklungsaufwand und komplexen Eingriffen verbunden. Die noch ausbaufähige Plattform bietet auch keinerlei Unterstützung von Push-Diensten für kundenbindende Maßnahmen.

Im Rahmen der vorliegenden Diplomarbeit soll daher aufbauend auf der ausbaufähigen Plattform eine Architektur für SMS-basierte Dienste konzipiert und implementiert werden. Einen wesentlichen Teil nimmt dabei die Modularisierung der Dienste in wiederverwendbare Basiskomponenten ein. Die Basiskomponenten erlauben eine schnelle und einfache Diensterzeugung und Dienstanpassung. Dies soll anhand der Entwicklung von komplexen Diensten durch Komposition der Komponenten gezeigt werden.

Zur Realisierung von Push-Diensten wird eine Regel-Engine entworfen. Die Regel-Engine soll eine Möglichkeit zur Regeldefinition bieten und einen Trigger-Mechanismus zur Verfügung stellen. Eine prototypische Implementierung der Basiskomponenten, der komplexen Dienste und der Regel-Engine dient als Beleg für die praktische Realisierbarkeit des entworfenen Konzepts.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einführung

1.1 Motivation

Bis Ende der 80er Jahre waren zellulare Mobilfunknetze in Europa firmenspezifische Lösungen und nicht für den Massenmarkt gedacht. Mit der Einführung europäischer Standards für digital übertragende Systeme ab 1990 hat sich die Situation markant verändert. Durch den technischen Fortschritt hat die Mobilkommunikation immer mehr an Bedeutung gewonnen und der Mobilfunk ist innerhalb kürzester Zeit zum Massenmarkt geworden. Mittlerweile gibt es in Deutschland rund 60 Millionen Mobilfunkteilnehmer, weltweit sind es über 700 Millionen.

Dabei ist die Nutzung mobiler Telefone längst nicht mehr auf das reine Telefonieren beschränkt. Mobile Telefone dienen als Terminplaner und ermöglichen über das Wireless Application Protocol (WAP) mobilen Zugang zum Internet. Der neben dem Telefonieren am häufigsten genutzte Dienst ist der Short Message Service (SMS), mittels dem kurze Textnachrichten mit bis zu 160 Zeichen verschickt werden können. Der Mitte der 90er Jahre eingeführte Short Message Service war der erste reine Datendienst für Mobilfunkteilnehmer. Trotz seiner eingeschränkten Funktionalität zeigte er eine hohe Akzeptanz und avancierte innerhalb kurzer Zeit zum bis jetzt erfolgreichsten mobilen Datendienst. Inzwischen werden allein in Deutschland Monat für Monat an die 2 Milliarden SMS-Nachrichten verschickt.

Wurde der Short Message Service anfangs vor allem von Jugendlichen zur Kommunikation untereinander genutzt, so kommt er mittlerweile immer häufiger im kommerziellen Bereich zum Einsatz. Denn gerade in diesem Bereich sind die Vorteile des Kommunikationsmediums SMS nicht von der Hand zu weisen: Der Empfänger erhält die Nachricht im Klartext, die bei akustischer Übertragung möglichen Missverständnisse bzw. Verständnisschwierigkeiten sind weitgehend ausgeschlossen. Außerdem können die gespeicherten Nachrichten bei Bedarf abgerufen werden. Ein noch größerer Vorteil liegt ähnlich wie bei dem Dienst E-Mail im asynchronen Nachrichtenempfang. Der Empfänger erhält die an ihn gesendete Nachricht in jedem Fall, unabhängig davon, womit er gerade beschäftigt ist. Selbst bei abgeschaltetem Endgerät geht die Nachricht nicht verloren, sie wird bei Wiedereinschalten in jedem Fall an den Adressaten ausgeliefert. Ein weiterer entscheidender Aspekt ist die hohe Akzeptanz. Bisher gibt es noch kein adäquates Medium, mit dem sich ähnlich viele Mitglieder unserer mobilen Gesellschaft erreichen lassen.

Beim professionellen Einsatz von SMS geht es um mehr als um die reine Nachrichten-Übermittlung.

SMS-basierte Dienste stellen zusätzliche Funktionalität bereit, sie bringen dem Benutzer einen Mehrwert. Daher spricht man auch von mobilen Mehrwertdiensten. Beispiele für solche Dienste sind:

- Benachrichtigung von Kunden (Auto nach Reparatur abholbereit, Arzttermin verschoben)
- Übermittlung von Information (Wetter, Börse, News)
- Realisierung von Preisausschreiben
- Durchführung von Meinungsumfragen

Die Abrechnung SMS-basierter Dienste erfolgt mittels sogenannter Premium-SMS. Jeder angebotene Dienst wird, abhängig von dem Wert, den er bringt, unter einer bestimmten Kurzwahlnummer publiziert. Jeder Kurzwahlnummer ist ein bestimmter Tarif zugeordnet. Durch das Senden einer SMS-Nachricht an eine Kurzwahlnummer (Premium-SMS) wird der entsprechende Betrag automatisch abgebucht.

BURDAWIRELESS, ein New Media Kompetenzzentrum des Hubert Burda Media Verlags, hat eine eigene Plattform zur Realisierung mobiler Dienste basierend auf SMS. Genutzt wird diese SMS-Plattform von Redaktionen und Kunden von Burda. Es werden bereits einige Dienste ausgeführt, wie beispielsweise Kreuzworträtsel, News-Ticker, Votings oder die bloße Weiterleitung von empfangenen SMS-Nachrichten.

Der Aufbau der Plattform ist noch sehr rudimentär. Im Moment werden nur Pull-Dienste unterstützt, die Aktion muss also immer vom Dienstnutzer ausgehen. Die Möglichkeit, mobile Teilnehmer gezielt über Aktionen oder wichtige Ereignisse zu informieren, ist nicht gegeben, ein Mechanismus zum Senden einer Push-Nachricht existiert noch nicht. Auch das Sammeln und Aufbereiten persönlicher Daten, um Interessenprofile zu erstellen, was zur gezielten Information unbedingt nötig ist, unterstützt die Plattform ebenfalls noch nicht. Ein weiteres Problem ist, dass die Entwicklung von Diensten bis jetzt immer kundenspezifisch erfolgt. Dadurch ist praktisch keine Wiederverwendung von bereits vorhandenen Applikationen möglich und das Aufsetzen neuer Dienste erweist sich aufwändig und zeitintensiv. Auch die Anpassung und Erweiterung realisierter Dienste ist weder schnell noch einfach durchzuführen.

1.2 Aufgabenstellung

Ziel der vorliegenden Arbeit ist es, aufbauend auf die bereits vorhandene SMS-Plattform bei Burda ein Modell zur Modularisierung SMS-basierter Dienste und zur Realisierung von Push-Diensten zu entwickeln und prototypisch zu implementieren. Um dies umzusetzen wird zuerst ein Katalog mit Anforderungen an eine Architektur für SMS-basierte Dienste zusammengestellt, um dann bestehende Ansätze und Architekturen danach zu bewerten. Die Modularisierung beinhaltet zwei Schritte. Der erste ist die Bestimmung und Spezifikation generischer Basiskomponenten, aus denen sich SMS-basierte Dienste zusammensetzen. Die Komposition dieser Basiskomponenten zu komplexen Diensten ist dann der nächste Schritt. Für die Realisierung von Push-Diensten ist eine Regel-Engine auf der Basis von Interessenprofilen zu entwickeln. Mit Hilfe dieser können Regeln für das Senden einer Push-Nachricht definiert werden. Durch die Angabe verschiedener Bedingungen kann festgelegt werden, welcher mobile Teilnehmer bei welchem Ereignis eine Nachricht erhält. Ausgehend von den konzeptionellen Entwürfen sind sowohl die Komponenten als auch die Regel-Engine prototypisch zu realisieren. Abschließend ist die eigene Architektur anhand des anfangs erstellten Anforderungskatalogs zu evaluieren.

1.3 Gliederung der Arbeit

Die vorliegende Arbeit gliedert sich in sieben Kapitel. In diesem ersten einführenden Kapitel wird ein kurzer Überblick über SMS-basierte Dienste gegeben und auf Problembereiche der SMS-Plattform bei Burda eingegangen. Danach folgt die Aufgabenstellung und die Gliederung der Arbeit.

Im zweiten Kapitel werden vorab die wesentlichen Grundlagen der mobilen Kommunikation erläutert. Dazu wird auf Basistechnologien wie Modulation und Multiplexverfahren eingegangen. Anschließend wird ein Überblick über die Mobilfunksysteme GSM und UMTS gegeben.

Das vierte Kapitel beginnt mit einem Katalog von Anforderungen, die eine Architektur für SMS-basierte Dienste erfüllen sollte. Anschließend werden bestehende Ansätze und Architekturen vorgestellt und mit Hilfe des Katalogs bewertet.

Das fünfte Kapitel befasst sich mit der Konzeption der Basiskomponenten und ihrer Komposition. Hier wird die Grundmenge der Basiskomponenten bestimmt und definiert, aus welchen Komponenten sich verschiedene komplexe Dienste zusammensetzen. Dieses Kapitel beinhaltet auch den Entwurf der Regel-Engine. Dies umfasst die Erstellung, der Interessenprofile, die Festlegung des Regelaufbaus sowie die Modellierung einer Triggerfunktion.

Im sechsten Kapitel wird die Implementierung der einzelnen Software-Komponenten und der Regel-Engine basierend auf den in Kapitel 5 entworfenen Konzepten beschrieben.

Das siebte Kapitel befasst sich mit einer Evaluierung der eigenen Architektur. Dazu wird die prototypische Implementierung zunächst auf ihre Funktionsfähigkeit getestet und anschließend in Analogie zu Kapitel 4 anhand des Anforderungskatalogs bewertet.

Im achten und letzten Kapitel werden die Ergebnisse der Diplomarbeit zusammengefasst und ein Ausblick auf mögliche Entwicklungen gegeben.

Kapitel 2

Grundlagen der Mobilkommunikation

2.1 Einführung

Mobil zu sein, ist ein wichtiger Aspekt im Leben unserer heutigen Gesellschaft. Drahtlose Netze und mobile Kommunikation kommen immer häufiger zum Einsatz; für bestimmte Anwendungsbereiche sind sie sogar die einzige Möglichkeit. Ohne drahtlose Netze könnte unser Wunsch, immer und überall zu kommunizieren, sicherlich nicht realisiert werden. Zunächst ist aber eine Definition der Begriffe *mobil* und *drahtlos* nötig, da sie im Folgenden immer wieder verwendet werden.

Mobilität

Unter Mobilität versteht man, dass sich jemand oder etwas bewegt. Ist dies der Nutzer eines Kommunikationssystems, so spricht man von Benutzer-Mobilität. Er kann die Dienste eines Systems an unterschiedlichen Orten nutzen, die Dienste folgen ihm also nach. Benutzer-Mobilität ermöglichen Mechanismen wie die Anrufweiterleitung von Telefonanlagen oder Computersysteme, die Nutzern immer die gleiche Oberfläche bieten unabhängig davon, welcher Computer oder Netzzugang verwendet wird.

Kann das Kommunikationsgerät an sich - mit oder ohne Nutzer - seinen Ort wechseln, so spricht man von Gerätemobilität. Dabei handelt es sich im Allgemeinen um relativ kleine, ohne maschinelle Hilfe bewegbare Geräte, die auch während der Bewegung kommunikationsfähig sind. Typische Beispiele hierfür sind Mobiltelefone und Palmsize Computers.

Den Ausdruck drahtlos wird eingesetzt, wenn es um die Kommunikationsart der Geräte geht. Von drahtloser Kommunikation spricht man, wenn ohne den Einsatz von Draht oder Glasfaser auf ein Kommunikationsnetz zugegriffen wird bzw. Daten mit einem Kommunikationspartner ausgetauscht werden. Die Daten werden mit Hilfe von elektromagnetischen Wellen durch den Raum transportiert, da kein Medium wie bei leitungsgebundener Kommunikation vorhanden ist. Für die drahtlose Kommunikation existieren zwei grundlegende Übertragungstechniken. Die eine setzt auf infrarotes Licht, die andere nutzt Funkwellen zur Übertragung. Für diese Arbeit ist nur die zweite Technik relevant, da alle vorgestellten Systeme Funkkommunikation einsetzen.

Jedes Kommunikationsgerät kann in eine der folgenden vier Kategorien eingeordnet werden, die verdeutlichen, dass mobil und drahtlos keinesfalls als gleichbedeutend angesehen werden dürfen:

- Fest und leitungsgebunden: In diese Kategorie fallen Arbeitsplatzrechner mit Festnetzanschluss. Festnetze werden aus Kostengründen und wegen ihrer hohen Leistungsfähigkeit eingesetzt.
- Mobil und leitungsgebunden: Tragbare Rechner wie Notebooks und Laptops fallen in diese Kategorie. Sie sind portabel, brauchen aber eine Leitung zur Kommunikation bzw. zum Datenaustausch.
- Fest und drahtlos: Ein Beispiel für diese Kategorie wäre ein drahtloses Computernetzwerk mit fest-installierten Rechnern. Dies ist nötig, falls eine Verkabelung nicht möglich oder nicht erwünscht ist.
- Mobil und drahtlos: In diese letzte Kategorie fallen Geräte, die portabel und ohne störendes Kabel kommunikationsfähig sind wie beispielsweise Mobiltelefone. Alle für diese Arbeit relevanten Netze unterstützen diese Art der Kommunikation und die dazugehörigen Geräte. Das wohl bekannteste Beispiel hierfür ist das GSM-Mobilfunknetz.

Mobilfunknetze in Europa/Deutschland

Um heutige Entwicklungen und existierende Mobilfunknetze besser zu verstehen, gibt der nächste Abschnitt eine kurze Übersicht über die Geschichte der Mobilfunknetze in Deutschland.

Das erste Mobilfunknetz in Deutschland war das A-Netz. Es startete 1958 und verwendete eine Trägerfrequenz von 160 MHz. Ein Gespräch konnte nur von einem mobilen Telefon aus aufgebaut werden. 1971 verzeichnete das A-Netz eine Flächendeckung von 80% und ungefähr 11.000 Kunden. Im Jahr darauf folgte das B-Netz, auch auf einer Trägerfrequenz von 160 MHz. Dieses System unterschied sich vom ersten im Wesentlichen dadurch, dass nun auch ein Anruf aus dem Festnetz an den mobilen Teilnehmer weitergeleitet werden konnte, vorausgesetzt der aktuelle Aufenthaltsort des Teilnehmers war bekannt. Das B-Netz war nicht nur in Deutschland, sondern auch in Österreich, Luxemburg und den Niederlanden verfügbar. 1979 war die Zahl der Kunden in Westdeutschland auf 13.000 gestiegen, jedoch waren die nötigen Empfangs- und Sende-Geräte immer noch sehr schwer und meistens in Autos eingebaut.

Es folgten noch einige nationale Standards, so dass zu Beginn der achtziger Jahre in Europa mehrere inkompatible analoge Mobilfunksysteme nebeneinander existierten. Deshalb einigten sich die europäischen Länder 1982 auf die Entwicklung eines paneuropäischen Standards. Dafür wurde eine Arbeitsgruppe, die Groupe Spéciale Mobile (GSM), gegründet. Das neue System sollte digital auf einer Frequenz von 900 MHz arbeiten sowie Sprach- und Datendienste anbieten.

Da digitale Systeme noch nicht verfügbar waren, folgten weitere analoge Mobilfunknetze darunter das C-Netz in Deutschland. Mit diesem System war nun auch eine Gesprächsübergabe zwischen verschiedenen Funkzellen und eine automatische Lokalisierung eines Teilnehmers im gesamten Netzbereich möglich. Außerdem wurde die Signalisierung bereits digital durchgeführt und Dienste wie Fax, E-Mail und Datenübertragung waren integriert.

Mit der Verabschiedung der Standards DECT und GSM durch ETSI (European Telecommunications Standards Institute) Anfang der neunziger Jahre wurden die ersten vollständig digitalen Systeme eingeführt. DECT (Digital Enhanced Cordless Telecommunication), welches früher auch als Digital

European Cordless Telephone bezeichnet wurde, war ein neuer Standard für schnurlose Telefone und löste ältere analoge Systeme ab. GSM, ursprünglich benannt nach der Arbeitsgruppe die es entwickelt hat, steht heute für Global System for Mobile Communications, um den Anspruch eines weltweiten Standards zu verdeutlichen. Die erste Version von GSM arbeitet bei ca. 900 MHz und wird heute in über 130 Ländern eingesetzt. Da diese Version für eine hohe Teilnehmerdichte, wie beispielsweise in Städten, nicht ausreichend war, wurde GSM in Europa auch auf der Frequenz von 1800 MHz verfügbar gemacht. Das GSM-1800-Netz ist bekannt als DCS 1800 (Digital Cellular System). Mittlerweile steht GSM auf vier verschiedenen Frequenzen zur Verfügung: Auf 450, 900, 1800 und 1900 MHz. GSM war das erste Mobilfunksystem, das den Short Message Service (SMS) integriert hatte.

Seit der Einführung von GSM hat ganz Europa ein gemeinsames System, dessen Teilnehmer ihre Telefone aber auch in Australien, Singapur oder den USA nutzen können. Die USA hingegen kämpfen immer noch mit mehreren, inkompatiblen Systemen, von denen keines eine sehr hohe Flächendeckung aufweist, außer das analoge AMPS (Advanced Mobile Phone System). Der Erfolg von GSM zeigt, dass der Ansatz eines gemeinsamen Standards eindeutig die bessere Lösung war. Mit UMTS gibt es auch schon einen neuen Standard, der Ende dieses Jahres eingeführt wird und das GSM-Mobilfunksystem mit der Zeit vollständig ablösen soll. 1998 einigten sich die Europäer auf das Universal Mobile Telecommunication System (UMTS) als europäischen Vorschlag für das IMT-2000-Programm (International Mobile Telecommunications) der International Telecommunication Union (ITU). Dieses Programm enthält Empfehlungen für einen weltweit einheitlichen Rahmen für Kommunikationssysteme im Frequenzbereich von ca. 2000 MHz [ITU].

2.2 Basistechnologien

Im Folgenden werden grundlegende Mechanismen der drahtlosen Übertragung beschrieben. Zunächst wird ein Überblick über genutzte Frequenzen gegeben, dann werden Signale und ihre Ausbreitungseigenschaften sowie Standardmodulationsverfahren vorgestellt. Da der Raum als Übertragungsmedium immer ein geteiltes Medium ist, werden Techniken zur Mehrfachnutzung eines Mediums, sogenannte Multiplexverfahren, präsentiert und ihr Zusammenhang mit den entsprechenden Zugriffsverfahren angeschnitten. Abschließend wird auf die Eigenschaften zellenbasierter Funksysteme eingegangen.

2.2.1 Frequenzen

Für die Übertragung von Daten mit Hilfe von elektromagnetischen Wellen können viele verschiedene Frequenzen genutzt werden. Dabei zeichnet sich jeder Frequenzbereich durch charakteristische Vor- und Nachteile aus. Abbildung 2.1 gibt einen groben Überblick über alle zur Datenübertragung nutzbaren Frequenzbereiche.

Funkübertragung beginnt schon bei Frequenzen von wenigen kHz. Frequenzen aus dem VLF- (Very Low Frequency) und dem Langwellenbereich (Low Frequency, LF) werden beispielsweise von U-Booten eingesetzt, da sie sich im Wasser fortpflanzen und der Krümmung der Erdoberfläche folgen. Ein paar Rundfunksender nutzen zwar noch Frequenzen im Langwellenbereich, aber primär werden weltweit für die Übertragung von Radiosendungen Mittelwellen- (Medium Frequency, MF) und Kurzwellenbereiche (High Frequency, HF) verwendet. Der heute übliche UKW-Rundfunk läuft über einen Frequenzbereich von 87,5 MHz bis 108 MHz.

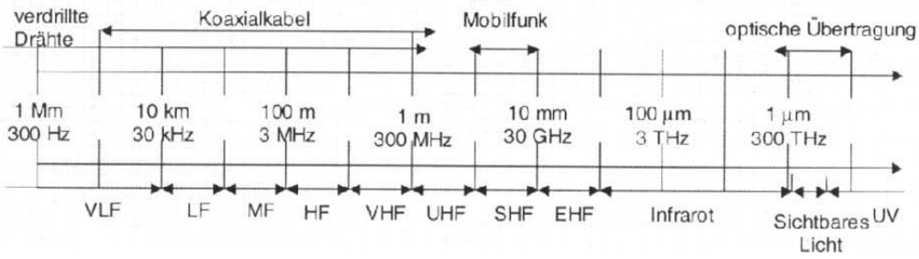


Abbildung 2.1: Frequenzspektrum

Zwischen Informationsmenge und Höhe der Trägerfrequenz gibt es einen Zusammenhang: Je mehr Information zu einem Zeitpunkt übertragen werden soll, desto höher muss die Trägerfrequenz sein. Deshalb werden Fernsehbilder in Bereichen von 174-230 MHz und 470-790 MHz übertragen. International heißen diese Frequenzbereiche Very High Frequency (VHF) und Ultra High Frequency (UHF). Im UHF-Bereich sind auch die Mobilfunksysteme angesiedelt. Das analog C-Netz nutzte Frequenzen um 450 MHz, das digitale GSM nutzt Frequenzen um 900 MHz und um 1800 MHz. VHF und vor allem UHF haben den Vorteil, dass relativ kleine Antennen eingesetzt werden können und trotzdem noch eine vergleichsweise zuverlässige Verbindung für die Mobilkommunikation möglich ist.

Zwischen 2 und 40 GHz liegt der SHF- (Super High Frequency) Bereich, der beispielsweise für Satellitenverbindungen genutzt wird. Der nächste Schritt auf der Frequenzskala führt über den Infrarot-Bereich in den Bereich der optischen Übertragung. Zur Übertragung von sichtbarem Licht kann entweder eine Glasfaser-Leitung oder eine gerichtete Laserverbindung eingesetzt werden.

Der Überblick über die Frequenzbereiche hat gezeigt, dass für die Funkübertragung nicht beliebig hohe oder niedrige Frequenzen verwendet werden können. Frequenzen sind also eine knappe Ressource und müssen staatlich reguliert werden, damit es durch Mehrfachnutzung nicht zu Interferenzen kommt. Knappe Ressourcen sind auch immer sehr begehrt. Das zeigte sich zuletzt bei der Versteigerung der UMTS-Lizenzen im Herbst 2000, für die sechs Telekommunikationsfirmen Summen in Milliarden-Höhe bezahlten.

2.2.2 Signale und Signalausbreitung

Zwei Kommunikationssysteme können nur durch die Übertragung von Signalen Daten untereinander austauschen. Signale sind also nichts anderes als die physikalische Repräsentation von Daten. Die Umwandlung von Daten in Signale und umgekehrt ist Aufgabe der Schicht 1, der Bitübertragungsschicht, im ISO/OSI-Referenzmodell.

Da die drahtlose Kommunikation auf der Übertragung elektromagnetischer Wellen beruht, werden die zu übertragenden Daten typischerweise durch ein periodisches Signal, also z.B. durch eine Sinusschwingung, dargestellt. Eine Sinusfunktion hat verschiedene veränderbare Parameter, die zur Repräsentation der Daten dienen.

Zu den Signalparametern gehören die Amplitude A, die Frequenz f und die Phasenverschiebung φ. Die Amplitude ist der Abstand der Hoch- und Tiefpunkte zum Ursprung. Unter der Frequenz versteht man die Anzahl der Schwingungen pro Zeiteinheit und die Phasenverschiebung bestimmt die Verschiebung

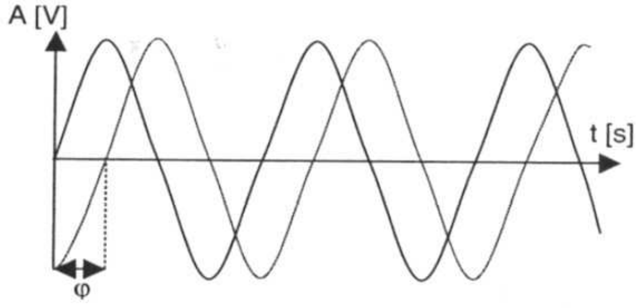


Abbildung 2.2: Sinusfunktion g(t) = A_t sin($2\pi f_t$t + φ)

des Signals in Richtung der Zeitachse relativ zum Ursprung. Alle drei Parameter können über die Zeit variiert werden.

In drahtlosen Netzen gibt es, wie die Bezeichnung schon andeutet, keinen elektrischen Leiter, der die Ausbreitung des Signals vorgibt. Die Signale müssen ohne zusätzliche Führung durch den Raum übertragen werden. Realisiert wird dies mit Hilfe von Antennen. Sie ermöglichen den Übergang elektromagnetischer Wellen von einem Leiter in den Raum und umgekehrt. Antennen selbst sind nichts anderes als nicht-isolierte elektrische Leiter, die elektromagnetische Wellen an die Umgebung abstrahlen bzw. auffangen.

Das Verhalten eines drahtlos übertragenen Signals kann nur im Vakuum, d.h. wenn weder zwischen Sender und Empfänger noch um diese Materie vorhanden ist, vorhergesagt werden: Es breitet sich gleichmäßig in alle Richtungen aus und wird quadratisch zur Entfernung immer schwächer.

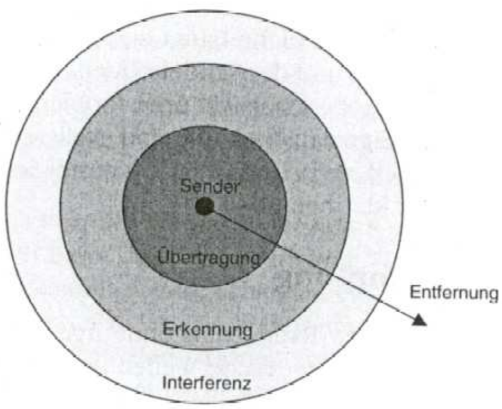


Abbildung 2.3: Bereiche um einen Sender

Durch das Abnehmen der Stärke, auch Dämpfung genannt, ergeben sich drei Bereiche um den Sender. Im Übertragungsbereich ist das Signal stark genug, um korrekt empfangen werden zu können, Sender und Empfänger also miteinander kommunizieren können. Im nächsten Bereich, dem Erkennungsbereich, kann der Empfänger zwar noch erkennen, dass ein Signal gesendet wurde, aber die Originaldaten können aufgrund der Störungen nicht mehr rekonstruiert werden. Innerhalb des Interferenzbereichs ist es dem Empfänger nicht mehr möglich, das Signal eines Senders zu erkennen, er wird es höchstens als Hintergrundrauschen wahrnehmen.

Da Übertragungen normalerweise nicht im Vakuum stattfinden, sondern Signale ihren Weg an Bergen, Tälern, Gebäuden, Bäumen, usw. vorbei finden müssen, sehen diese drei Bereiche in der Realität nicht kugelförmig, sondern wie verzerrte Polygone aus.

Neben der Dämpfung im Vakuum gibt es noch folgende Effekte bei der Signalausbreitung:

- Abschattung: Große Objekte wie Mauern, Fahrzeuge oder Bäume können ein Signal derart dämpfen, dass es hinter dem Objekt nicht mehr empfangen werden kann.
- Reflexion: Sehr große Gebäude, Berge oder die Erdoberfläche sind in der Lage, Signale zu reflektieren. Nach der Reflexion besitzt das Signal nicht mehr dieselbe Stärke, da ein Teil absorbiert wird.
- Streuung: Ein Signal kann an einem Objekt in mehrere schwächere Signale aufgespalten werden, die sich in verschiedene Richtungen ausbreiten.
- Beugung: Kanten von Hindernissen können Signale von ihrer ursprünglichen Ausbreitungsrichtung ablenken.

Ohne diese Effekte wäre eine Übertragung zwischen Sender und Empfänger, die keine Sichtverbindung haben, gar nicht möglich.

2.2.3 Modulation

Bevor digitale Daten mittels Antennen übertragen werden können, müssen sie anhand von digitaler und analoger Modulation in eine andere Form gebracht werden.

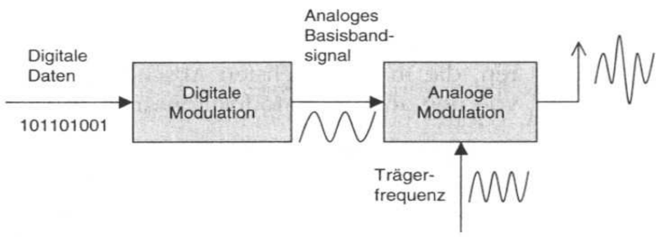


Abbildung 2.4: Modulation in einem Sender

Bei der digitalen Modulation wird der digitale Datenstrom zunächst in ein analoges Signal, das sogenannte Basisbandsignal, umgesetzt. Digitale Modulation ist immer dann nötig, wenn Daten über ein Medium übertragen werden sollen, das nur eine analoge Übertragung zulässt.

Um das generierte Basisbandsignal auf eine Frequenz zu bringen, die für die drahtlose Übertragung geeignet ist, ist zusätzlich noch eine analoge Modulation notwendig. Bei der analogen Modulation wird die Mittenfrequenz des Basisbandsignals auf die Trägerfrequenz für die Funkübertragung verschoben.

Im Allgemeinen versteht man unter Modulation die Veränderung eines Trägersignals in Abhängigkeit eines Nutzsignals. Das Trägersignal kann in den Parametern Amplitude, Frequenz und Phase verändert werden. Der Parameter, in dem das Trägersignal verändert wird, bestimmt die Art der Modulation: Amplituden-, Frequenz- oder Phasenmodulation. Je nach Art der Modulation werden die Werte des Nutzsignals entweder durch unterschiedliche Amplituden, durch unterschiedliche Frequenzen oder durch unterschiedliche Phasen im Trägersignal dargestellt. Abbildung 2.5 zeigt die Anwendung der drei Modulationsarten auf ein digitales Nutzsignal. Bei der Modulation von digitalen Signalen spricht man auch von Umtastung.

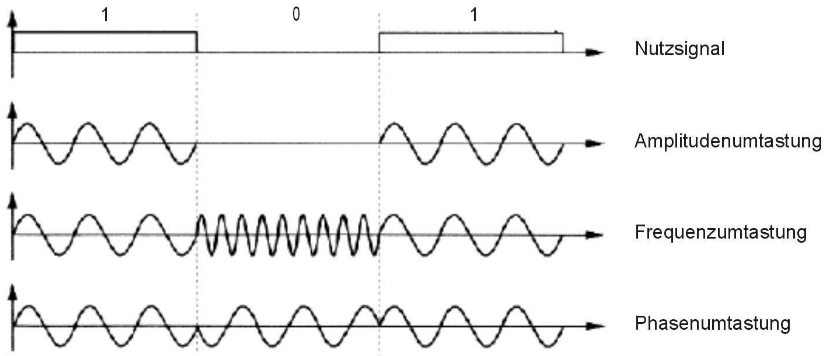


Abbildung 2.5: Schematische Darstellung der Modulationsarten

2.2.4 Multiplex- und Zugriffsverfahren

Als Multiplex bezeichnet man die mehrfache Nutzung eines Mediums durch verschiedene Nutzer. Multiplexverfahren beschreiben, wie ein gemeinsames Medium von verschiedenen Nutzern mit möglichst wenig gegenseitiger Beeinflussung genutzt werden kann.

In der drahtlosen Kommunikation gibt es vier verschiedene Multiplexverfahren, benannt nach den Dimensionen, in denen sie eingesetzt werden können: Raum-, Zeit-, Frequenz- und Code-Multiplex. Je nach Verfahren wird einem Kommunikationskanal ein bestimmter Raum zu einer bestimmten Zeit mit einer festgelegten Frequenz mit einem Code zugewiesen [Schi 00].

Raum-Multiplex (Space Division Multiplexing, SDM)
Raum-Multiplex bezeichnet die Unterteilung des geometrischen Raums in Zellen mit geeignetem Abstand, so dass die zur Übertragung verwendeten Frequenzen in jeder Zelle erneut verwendet werden können. Dieses Verfahren wird allein schon dadurch ermöglicht, dass Signale durch die Dämpfung von Natur aus eine begrenzte Reichweite haben und somit Sender mit genügend großem Abstand zur selben Zeit dieselbe Frequenz nutzen können. Jedes zellulare System basiert auf Raum-Multiplex.

Frequenz-Multiplex (Frequency Division Multiplexing, FDM)
Beim Frequenz-Multiplex wird das für die Übertragung zur Verfügung stehende Frequenzband in mehrere Frequenzbänder unterteilt, die gleichzeitig genutzt werden können.

Jedem Kanal wird eines dieser Teilfrequenzbänder zugewiesen. Um Interferenzen zwischen benachbarten Kanälen zu vermeiden, sind Schutzbänder zwischen den einzelnen Frequenzbändern nötig. Dieses Verfahren kommt seit Jahrzehnten beim Rundfunk zum Einsatz. Da jeder Radio-Sender eine andere Frequenz nutzt, können sie nebeneinander existieren, ohne sich gegenseitig zu stören.

Zeit-Multiplex (Time Division Multiplexing, TDM)
Beim Zeit-Multiplex kann ein Kanal den gesamten Frequenzbereich für eine gewisse Zeit exklusiv

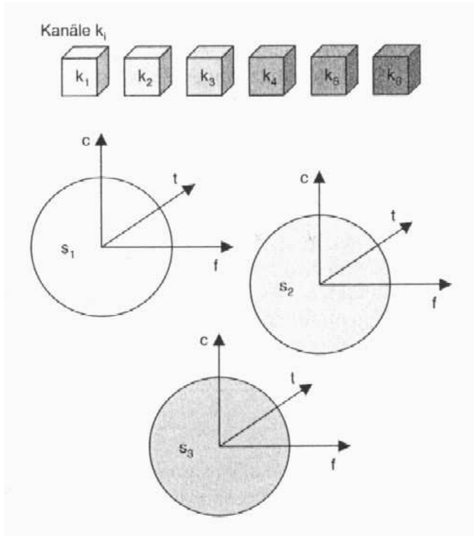


Abbildung 2.6: Raum-Multiplex

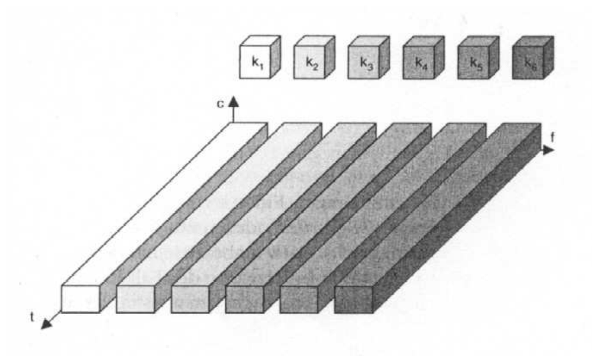


Abbildung 2.7: Frequenz-Multiplex

nutzen. Alle Sender verwenden also die gleiche Frequenz, aber nie zur selben Zeit. Als Schutzabstände dienen hier kurze Pausen vor der erneuten Belegung einer Frequenz. Obwohl Zeit-Multiplex eine präzise Synchronisation aller Sender und Empfänger erfordert, hat es gegenüber den anderen beiden Verfahren einen großen Vorteil: Die Sendezeit kann flexibel an die Anforderungen der einzelnen Sender angepasst werden.

Oft wird Zeit-Multiplex mit Frequenz-Multiplex kombiniert. Das heißt, dass jeder Kanal eine bestimmte Frequenz für eine gewisse Zeitspanne exklusiv nutzen kann. Diese Kombination kommt beispielsweise beim Mobilfunkstandard GSM zum Einsatz.

Code-Multiplex (Code Division Multiplexing, CDM)
Code-Multiplex ist im Vergleich zu den bereits vorgestellten ein relativ neues Verfahren und findet im Mobilfunk zum ersten Mal bei UMTS seine Anwendung.

Beim Code-Multiplex nutzen alle Kanäle zur selben Zeit dieselbe Frequenz. Dabei ist jedem Kanal ein eigener, spezieller Code zugeordnet, mit dem der zu sendende Datenstrom codiert wird. Der Empfänger kann dann anhand des entsprechenden Codes den an ihn gerichteten Datenstrom aus dem

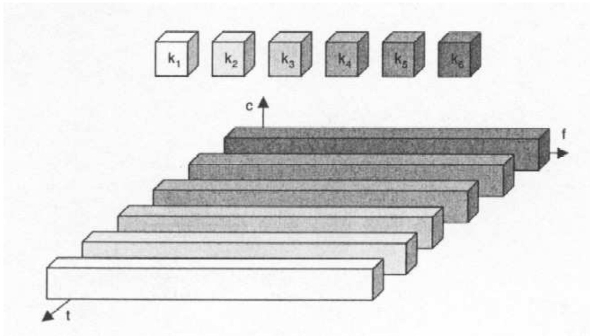


Abbildung 2.8: Zeit-Multiplex

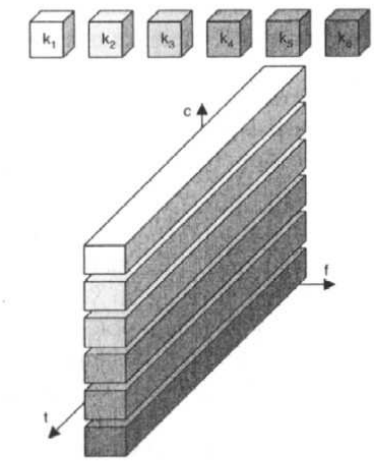


Abbildung 2.9: Code-Multiplex

Gesamtstrom herausfiltern. Die erforderlichen Schutzabstände zwischen den Kanälen werden hier durch ausreichend große Abstände der Codes im Code-Raum erreicht. Dies ist gegeben, wenn die Codes zueinander orthogonal sind, also das Skalarprodukt von je zwei verschiedenen Codes gleich 0 ist. Dieses Verfahren hat gegenüber Frequenz- und Zeit-Multiplex den Vorteil, dass der Code-Raum im Vergleich zum Frequenz-Raum riesig ist und somit die Anzahl der Kanäle fast beliebig groß sein kann. Ein großer Nachteil dieses Verfahrens ist allerdings die deutlich höhere Komplexität, die bei den Empfängern nötig ist, um den richtigen Datenstrom herauszufiltern.

Bei Multiplexverfahren geht es darum, mehrere unterschiedliche Kommunikationskanäle zu schaffen. Zugriffsverfahren sind Mechanismen, die den Zugriff mehrerer Nutzer auf diese Kommunikationskanäle regeln. Zugriffsverfahren beruhen also immer auf einem Multiplexverfahren bzw. auf einer Kombination mehrerer Multiplexverfahren. Grundsätzlich unterscheidet man vier verschiedene Arten: Mehrfachzugriff durch Raum-Multiplex (SDMA), durch Frequenz-Multiplex (FDMA), durch Zeit-Multiplex (FDMA) und durch Code-Multiplex (CDMA).

2.2.5 Zellulare Systeme

Um ihre Gesamtkapazität zu erhöhen, verwenden praktisch alle Mobilfunksysteme Raum-Multiplex. Dabei versorgt jeder Sender, auch Basisstation genannt, einen bestimmten Bereich, die sogenannte Zelle. Innerhalb von Gebäuden sind 20 m als Zellradius typisch, in Städten sind es mehrere 100 m und in ländlichen Regionen können es bis 50 km sein.

Für den Einsatz von Tausenden von Basisstationen anstatt weniger sehr leistungsstarker Sender sprechen verschiedene Gründe. Als Erstes ist hier die höhere Gesamtkapazität zu nennen. Durch den Einsatz von Raum-Multiplex können dieselben Frequenzen mehrfach verwendet werden. Wenn zwei Sender soweit voneinander entfernt sind, dass sie sich gegenseitig nicht mehr stören, können sie die gleichen Frequenzen nutzen. Aufgrund der beschränkten Anzahl der Frequenzbereiche ist auch die Zahl der Teilnehmer pro Zelle beschränkt. Werden anstatt weniger großer Zellen viele kleine Zellen eingesetzt, so erhöht sich die Zahl der möglichen Nutzer pro Flächeneinheit. Kleine Zellradien haben außerdem den Vorteil, dass der Abstand zwischen Sender und Empfänger relativ kurz ist und somit auch nur wenige Störungen des Signals auftreten können. Als weiterer wichtiger Grund ist die Robustheit zu nennen. Zellenbasierte Mobilfunksysteme sind hinsichtlich des Ausfalls einzelner Basisstationen robuster, weil stets nur eine relativ kleine Fläche betroffen ist und eventuell eine benachbarte Basisstation den Ausfall überbrücken kann.

Allerdings haben kleine Zellen auch einige Nachteile. Zum einen ist zur Koordination aller Basisstationen eine komplexe Infrastruktur nötig. Diese Infrastruktur muss neben der Lokalisierung mobiler Stationen auch eine für den mobilen Teilnehmer transparente Übergabe, falls er eine Zelle verlässt und in eine neue eintritt, unterstützen. Die Übergabe einer Verbindung zwischen einer mobilen Station und einer Basisstation an eine andere Basisstation wird auch Handover genannt. Zum anderen erfordert die geringe Anzahl verfügbarer Frequenzen eine genaue Planung, wie die Frequenzen auf die Basisstationen zu verteilen sind, damit keine Störungen bei überlappenden Interferenzbereichen auftreten.

In Modellen für Mobilfunksysteme werden meistens sechseckige Zellen verwendet, da man mit der Sechseckform eine optimale Ausnutzung der zur Verfügung stehenden Frequenzen erreicht. Da die Form der Zellen von der Geländeform, der Bebauung oder dem Wetter abhängt und nicht kreisförmig oder sechseckig ist, ist davon auszugehen, dass sich benachbarte Zellen in der Realität fast immer überlappen. Um Störungen aufgrund der Nutzung gleicher Frequenzen bei überlappenden Interferenzbereichen zu vermeiden, sollte benachbarten Zellen nie zur gleichen Zeit die gleiche Frequenz zugeordnet sein. Abbildung 2.10 zeigt zwei Möglichkeiten, dies zu realisieren.

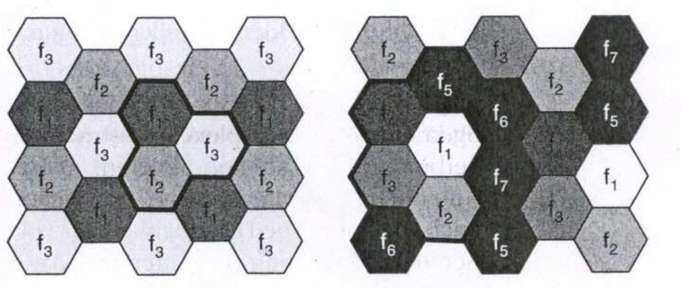


Abbildung 2.10: Zellen in 3er- und 7er-Gruppierungen

Im linken Modell werden drei Zellen zu einer Gruppierung zusammengefasst, im rechten sind es sieben Zellen. Innerhalb einer Gruppe verwendet jede Zelle eine andere Sendefrequenz. So ergibt sich ein immer wiederkehrendes Muster mit den gleichen Frequenzen.

Die Mobilfunksysteme GSM und UMTS sind Beispiele für zellulare Systeme und werden in den nächsten beiden Abschnitten vorgestellt.

2.3 GSM - Global System for Mobile Communications

GSM wird mittlerweile in über 135 Ländern eingesetzt und ist heute das erfolgreichste Mobilfunksystem weltweit. Dazu beigetragen haben Merkmale wie digitale Funkübertragung, europaweite Standardisierung, ISDN (Integrated Services Digital Network) Kompatibilität, verbesserter Schutz gegen Mithören sowie Unterstützung von Datendiensten. GSM wird als wesentlicher Fortschritt gegenüber allen Vorläufersystemen angesehen und ist ein typisches System der 2. Generation von Mobilfunksystemen. Zur 1. Generation gehören im Allgemeinen die analogen Systeme wie das C-Netz in Deutschland oder AMPS in den USA.

GSM gilt als Paradebeispiel für digitale Mobilfunksysteme, weniger wegen des kommerziellen Erfolgs, sondern weil die Systemarchitektur und viele Details von GSM anderen Systemen als Vorbild dienten. GSM eignet sich gerade deshalb so gut zur Präsentation, weil der Standard eine Vielzahl offener Schnittstellen und klar spezifizierte Netzelemente enthält. Im Folgenden wird auf Architektur, Schnittstellen, Protokolle und typische Dienste des GSM-Systems näher eingegangen.

2.3.1 Systemarchitektur

Das GSM-System besteht aus drei Teilsystemen, dem Funk-Feststationssystem (Radio Subsystem, RSS), dem Mobilvermittlungssystem (Network and Switching Subsystem, NSS) und dem Betriebs- und Wartungssystem (Operation Subsystem, OSS). Diese Teilsysteme und ihre Komponenten sind in Abbildung 2.11 in einer vereinfachten GSM-Architektur dargestellt.

Funk-Feststationssystem

Das Funk-Feststationssystem RSS enthält alle funkspezifischen Komponenten und ist über die A-Schnittstelle (durchgezogene Linien) mit dem Teilsystem NSS verbunden. Dieser Schnittstelle liegt ein leitungsvermitteltes PCM- (Pulse Code Modulation) 30-System zu Grunde. Ein PCM-30-System (2,048 Mbit/s) umfasst 32 Kanäle zu je 64 kbit/s; 30 davon sind zur Übertragung, die restlichen zwei werden zur Synchronisation und Signalisierung benötigt. Verbindungen zum OSS erfolgen über die O-Schnittstelle (gestrichelte Linien), die das Signalisierungssystem Nr. 7 (SS7) basierend auf X.25 nutzt, um Daten zwischen den beiden Teilsystemen hin und her zu transportieren.

Die einzelnen Komponenten des RSS sind:

- Feststationssystem (Base Station Subsystem, BSS): In einem GSM-Netz gibt es mehrere BSS, die alle durch einen BSC gesteuert werden. In einer BSS erfolgt die Codierung/Decodierung der Sprachdaten sowie die Anpassung der Datenraten von und zum drahtlosen Netz. Außerdem

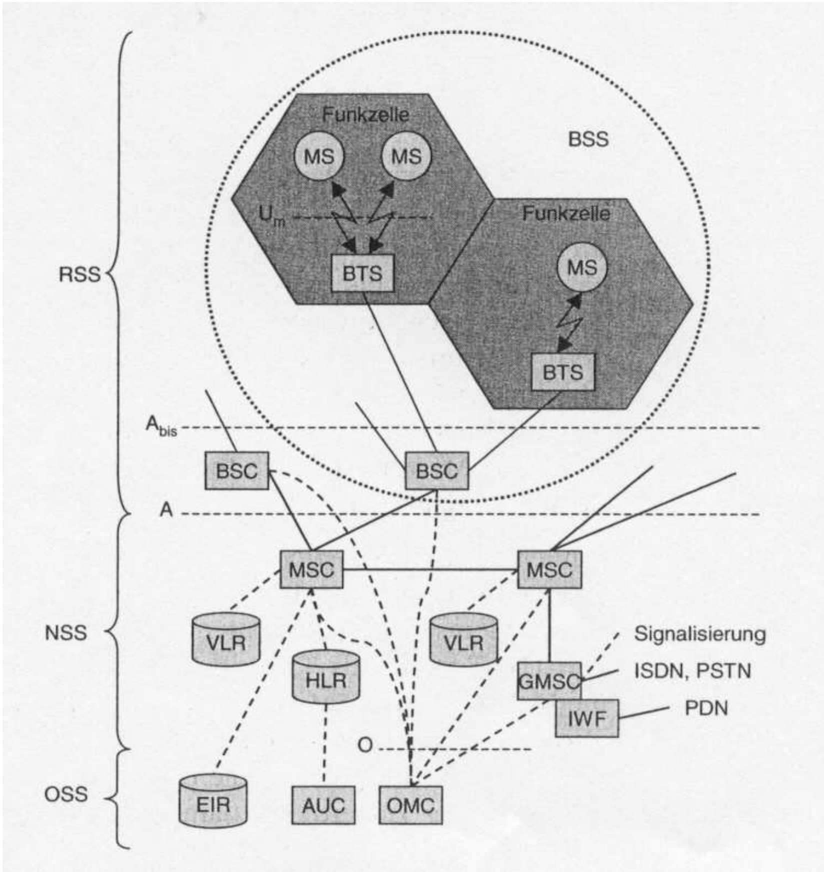


Abbildung 2.11: Funktionale Architektur eines GSM-Systems

ist ein BSS dafür zuständig, für eine ununterbrochene Funkverbindung zu einer MS zu sorgen. In einem BSS sind neben einem BSC noch mehrere BTS enthalten.

- Sende-/Empfangsstation (Base Transceiver Station, BTS): Eine BTS bildet eine Funkzelle und ist über die U_m-Schnittstelle mit den MS verbunden. Die U_m-Schnittstelle ist die sogenannte Luftschnittstelle und wird später noch genauer behandelt. Die Verbindung zum BSC geschieht über die A_{bis}-Schnittstelle, die mehrere Verbindungen zu 16 kbit/s oder 64 kbit/s umfasst. In einer BTS befinden sich Einrichtungen wie Antennen, Signalverarbeitung und Verstärker für die Übertragung. Die Radien einer GSM-Zelle reichen von 100 m bis zu 35 km.

- Feststationssteuerung (Base Station Controller, BSC): Zur Aufgabe eines BSC gehört die Verwaltung mehrerer BTS. Das beinhaltet die Reservierung von Frequenzen, die Übergabe einer Verbindung von einer BTS zu einer anderen innerhalb des BSS und die Steuerung von Rundrufen (Paging). Des weiteren ist ein BSC für die Übertragung von Daten über die A-Schnittstelle von bzw. zum Teilsystem NSS zuständig.

- Mobilstation (Mobile Station, MS): Eine MS besteht zum einen aus den zur Kommunikation mit GSM nötigen Hard- und Softwarekomponenten und zum anderen aus dem Subscriber Identity Module (SIM), eine Einheit, in der alle nutzerspezifischen Daten gespeichert sind. Jedes GSM-Endgerät hat unabhängig vom Nutzer eine eindeutige Gerätekennung, die International Mobile Equipment Identity (IMEI), anhand der es identifiziert werden kann. Mit Hilfe des SIM kann jedes GSM-Endgerät personalisiert werden. Ohne SIM kann man nur Notrufe tätigen. Zu den nutzerspezifischen Daten zählen unter anderem Typ, Seriennummer, registrierte Dienste, PIN (Personal Identity Number), PUK (PIN Unblocking Number) und Authentifizierungsschlüssel Ki.

Mobilvermittlungssystem

Das Kernstück der GSM-Architektur bildet das Mobilvermittlungssystem NSS, da es das drahtlose Netz mit den öffentlichen Partnernetzen, wie beispielsweise das Telefonnetz oder ISDN, verbindet. Das NSS ist für die Übergabe von Verbindungen zwischen zwei BSS sowie für die weltweite Lokalisierung von mobilen Teilnehmern zuständig und unterstützt Abrechnung und Roaming von Teilnehmern zwischen mehreren Netzbetreibern in verschiedenen Ländern.

Zu diesem Teilsystem gehören folgende Komponenten:

- Dienstvermittlungsstelle (Mobile Services Switching Centre, MSC): Ein MSC ist eine ISDN-Vermittlungsstelle mit hoher Leistungsfähigkeit. In den Zuständigkeitsbereich eines MSC fallen mehrere BSCs einer Region. Zu den Aufgaben eines MSC gehört der Verbindungsaufbau zu den BSCs über die A-Schnittstelle und zu anderen MSCs. Von einem MSC wird jegliche Signalisierung, die zum Verbindungsaufbau, -abbau und zur Verbindungsübergabe nötig ist, verarbeitet. Die gesamte Signalisierung beruht auf dem Signalisierungssystem Nr. 7. Ein Gateway MSC (GMSC) bietet zusätzliche Funktionen, die eine Verbindung zu anderen Festnetzen wie ISDN ermöglichen, und mit den Interworking Functions (IWF) ist ein Anschluss an öffentliche Datennetze (Public Data Networks, PDN) möglich.

- Heimatregister (Home Location Register, HLR): Im HLR, der wichtigsten Datenbank des GSM-Systems, werden alle Teilnehmerdaten gespeichert. Das sind einerseits statische Daten wie die Rufnummer (Mobile Subscriber ISDN Number, MSISDN), die freigegebenen Dienste, der Authentifizierungsschlüssel Ki und andererseits dynamische Daten wie der aktuelle Aufenthaltsort eines Teilnehmers (Location Area, LA). Ändert sich die LA eines Teilnehmers, so erfolgt automatisch eine Aktualisierung der LA-Daten im HLR. Anhand dieser eindeutigen Teilnehmerdaten kann jeder Teilnehmer im gesamten GSM-Netz lokalisiert werden.

- Besucherregister (Visitor Location Register, VLR): Jedes MSC hat sein eigenes VLR, eine sehr dynamische Datenbank. Im VLR sind alle Daten von denjenigen MS gespeichert, die sich derzeit im Einzugsbereich des MSC befinden. Wechselt eine MS ihren LA, so holt sich das für die neue LA zuständige MSC alle relevanten Daten aus dem HLR und schreibt sie in sein VLR. Somit wird eine zu häufige Aktualisierung des HLR vermieden.

Betriebs- und Wartungssystem

Das Betriebs- und Wartungssystem OSS stellt Funktionen bereit, die einen zuverlässigen Betrieb und die Wartung des gesamten GSM-Netzes ermöglichen. Die Verbindung zwischen OSS und den Komponenten von NSS geschieht über O-Schnittstelle, also über eine SS7-Signalisierung.

Das OSS besitzt folgende drei Untereinheiten:

- Betriebs- und Wartungszentrale (Operation and Maintenance Centre, OMC): Das OMC steuert und überwacht als zentrale Stelle alle anderen Netzkomponenten, mit denen es über die O-Schnittstelle (SS7 mit X.25) verbunden ist. Zu den Verwaltungsfunktionen des OMC gehören unter anderem die Überwachung des Verkehrs, das Verwalten der Teilnehmer, Gebührenabrechnung und Erstellung von Statusberichten einzelner Netzelemente.
- Authentifizierungszentrale (Authentication Centre, AuC): Das AuC enthält alle Informationen, die zum Schutz der Teilnehmeridentität und der Datenübertragung erforderlich sind. Da die Luftschnittstelle im Vergleich zu Festnetzen für Zugriffe anfälliger ist, wurden besondere Maßnahmen wie die Vergabe eines Authentifizierungsschlüssels und die Verschlüsselung der zu übertragenden Information getroffen, um unerlaubte Zugriffe zu verhindern. Das AuC speichert die dazu nötigen Schlüssel und Parameter.
- Geräteidentifikationsregister (Equipment Identity Register, EIR): Das EIR ist eine Datenbank, in der alle Gerätekennungen für einen bestimmten Netzbetreiber gespeichert sind. Über diese IMEIs führt das EIR eine weiße, eine graue und eine schwarze Liste. Die weiße Liste enthält alle gültigen IMEIs, die graue Liste alle IMEIs von Geräten mit Fehlfunktionen und die schwarze Liste beinhaltet alle IMEIs der als gestohlen gemeldeten oder aus sonstigen Gründen gesperrten Geräte. Somit kann eine illegale Nutzung einer gestohlenen MS unterbunden werden.

2.3.2 Luftschnittstelle

An der Luftschnittstelle U_m kommen viele der im Abschnitt 2.2 vorgestellten Basistechnologien zum Einsatz, da sie den Übergang zwischen mobilen Teilnehmer und GSM-Festnetz, dem PLMN (Public Land Mobile Network), realisiert. Hier erfolgt die Umwandlung der digitalen Daten in analoge Signale und die Aufteilung des Mediums in physikalische Kanäle. Des weiteren werden über diesen physikalischen Kanälen eine Reihe von logischen Kanälen definiert, um Aufgaben wie Nutzdatenübertragung oder Steuerung der Zugriffe auf diese Kanäle zu bewältigen.

Das Modulationsverfahren im GSM basiert auf der Phasenumtastung. Für die Übertragung der Daten wurde der zur Verfügung stehende Frequenzbereich in zwei Frequenzbänder getrennt, das eine für die Übertragung von der MS zur Basisstation und das andere für die umgekehrte Richtung. In GSM 900 wird jedes dieser Frequenzbänder durch FDM in 124 Kanäle von je 200 kHz Bandbreite aufgeteilt. Mit der zusätzlichen Anwendung von TDM stehen insgesamt 8*124 Kanäle pro Senderichtung zur Verfügung. Der Zugriff auf diese Kanäle basiert folglich auf einem kombinierten FDMA/TDMA-Verfahren.

Logische Kanäle entstehen, indem sie bestimmten Zeitschlitzen in physikalischen Kanälen zugeordnet werden. Das bedeutet, dass die Daten des logischen Kanals in die entsprechenden Zeitschlitze des physikalischen Kanals übertragen werden. Logische Kanäle können entweder einen Teil des Kanals oder den ganzen Kanal belegen. Angenommen man hat zwei logische Kanäle C1 und C2, wobei C1 jeden vierten und C2 jeden zweiten Zeitschlitz belegt. Die beiden logischen Kanäle können dann den gleichen physikalischen Kanal nutzen, beispielsweise nach folgendem Muster: C1 C2 x C2 C1 C2 x C2 C1 C2 ... (x kennzeichnet einen unbelegten Zeitschlitz).

In der GSM-Empfehlung wurden zwei grundlegende Arten von logischen Kanälen definiert: Verkehrskanäle und Steuerkanäle. Unter Verkehrskanäle (Traffic Channel, TCH) fallen alle logischen Kanäle, über die Nutzinformation, also Sprache oder Daten, ausgetauscht wird. Dabei unterscheidet man in

Full-Rate TCH und Half-Rate TCH. Ersterer besitzt eine Datenrate von 22,8 kbit/s und letzterer eine Datenrate von 11,4 kbit/s.

Steuerkanäle (Control Channel, CCH) dienen zur Signalisierung und zur Steuerung des Systems. Typische Aufgaben sind Verwaltung des Medienzugriffs, Unterstützung der Teilnehmermobilität und Kanalzuweisung. Steuerkanäle lassen sich in drei Gruppen unterteilen: Den Broadcast Control Channel (BCCH), der von einer BTS für Rundsendungen an alle MS ihrer Zelle genutzt wird, den Common Control Channel (CCCH), über den der Verbindungsaufbau zwischen einer MS und der BTS abgewickelt wird, und den Dedicated Control Channel (DCCH). Der DCCH besitzt selbst drei Unterkanäle, die im Gegensatz zu den beiden anderen Steuerkanälen bidirektional sind. Ist zwischen einer MS und BTS noch kein TCH aufgebaut, so muss die MS den Stand-alone Dedicated Control Channel (SDCCH) für die Signalisierung verwenden. Dies betrifft z.B. Registrierung, Authentifizierung, und Daten zum Verbindungsaufbau. Jedem TCH und SDCCH ist noch ein Slow Associated Dedicated Control Channel (SACCH) zugeordnet. Über diesen Kanal werden Systemzustandsdaten, wie Kanal- und Empfangsqualität, zwischen MS und BTS ausgetauscht. Ist ein TCH bereits vorhanden und müssen noch mehr Signalisierungsdaten übertragen werden, so kommt der Fast Associated Dedicated Control Channel (FACCH) zum Einsatz. Dies ist nötig, wenn eine große Datenmenge innerhalb kurzer Zeit ausgetauscht werden muss, wie beispielsweise bei einer Verbindungsübergabe von einer BTS zu einer anderen.

2.3.3 Dienste

Bei GSM gibt es drei unterschiedliche Arten von Diensten: Träger-, Tele- und Zusatzdienste. In Abbildung 2.12 ist das Referenzmodell für GSM-Dienste dargestellt. Über die U_m-Schnittstelle wird jede MS mit dem GSM-Festnetz verbunden. Das GSM-Festnetz wiederum ist mit einem Transitnetz, wie beispielsweise das traditionelle Telefonnetz (Public Switched Telephone Network, PSTN) oder ISDN, verknüpft. Zwischen dem Kommunikationspartner der MS, dem Endgerät TE (Terminal), und dem Transitnetz kann sich ein weiteres Netz befinden. Innerhalb der MS ist der Netzabschluss MT (Mobile Termination) für jegliche netzwerkspezifischen Aufgaben, z.B. Codierung und Medienzugriff per TDMA und FDMA, zuständig.

Trägerdienste

Zu den Trägerdiensten gehören alle Dienste, die einen Datentransport von einem Netzzugangspunkt zu einem anderen gewährleisten. Ursprünglich waren im GSM-Standard für Nichtsprachdienste nur Datenraten bis zu 9,6 kbit/s vorgesehen.

Ein Trägerdienst bietet nur reine Transportdienste, wie sie über die unteren drei Schichten des ISO/OSI-Referenzmodells definiert sind, also verbindungsorientierte leitungs- und paketvermittelte Datenübertragung. Man unterscheidet in transparente und nicht transparente Trägerdienste. Transparente Trägerdienste nutzen lediglich die Funktionen der Schicht 1 zur Datenübertragung. Die Übertragung geschieht mit konstanter Verzögerung und konstantem Durchsatz. Zur Erhöhung der Qualität werden Fehlerkorrekturverfahren eingesetzt. Abhängig von der Leistungsfähigkeit dieser Verfahren, ergeben sich Datenraten von 2,4, 4,8 oder 9,6 kbit/s.

Nicht transparente Trägerdienste nutzen zusätzlich die Protokolle der Schichten 2 und 3, um die Kommunikation durch weitergehende Fehlerkorrekturmaßnahmen abzusichern und durch Flusssteuerungs-

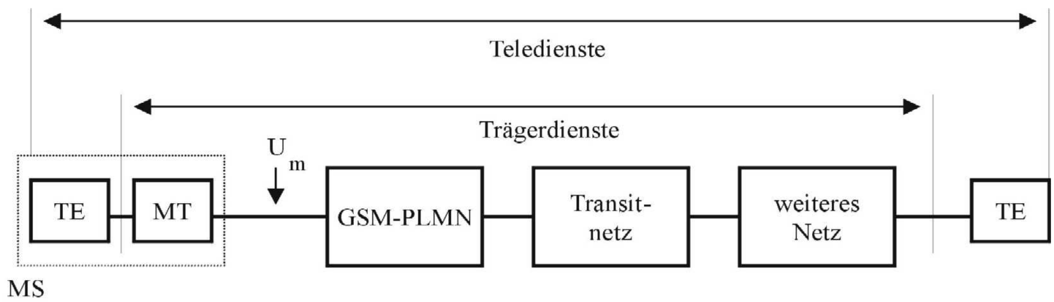


Abbildung 2.12: Träger- und Teledienste im GSM

mechanismen zu optimieren. Die nicht transparenten Dienste beruhen auf den transparenten Trägerdiensten und verwenden zusätzlich noch Mechanismen, die es ihnen ermöglichen, verloren gegangene Daten erneut zu senden.

Zur Zusammenarbeit mit traditionellen Telefonnetzen oder ISDN bietet GSM spezielle Trägerdienste, die auf den transparenten und nicht transparenten Diensten basieren.

Teledienste

Teledienste werden immer von einem TE zu einem anderen spezifiziert. Sie sind anwendungsabhängig und können alle 7 Schichten des ISO/OSI-Referenzmodells in Anspruch nehmen. In den meisten Fällen sind Teledienste sprachorientiert mit automatischer Verschlüsselung der Daten. Deshalb war es ein erklärtes Ziel von GSM, bei der Sprachübertragung eine hohe Qualität zu erreichen. Das Bereitstellen einer einheitlichen Notrufnummer im gesamten GSM-System ist ein weiterer Teledienst. Alle Netzbetreiber müssen diesen Dienst kostenlos zur Verfügung stellen. Auch Mobiltelefone ohne Vertrag, also ohne SIM, können den Notrufdienst nutzen.

Neben den sprachorientierten Diensten sind hier auch noch Datendienste wie das Gruppe-3-Fax und der Short Message Service aufzuführen. Beim Gruppe-3-Fax werden die Daten eines Fax mit Hilfe von Modems digital über ein analoges Telefonnetz übertragen. Der Short Message Service wird in Abschnitt 2.3.4 näher behandelt.

Zusatzdienste

Welche Zusatzdienste zur Verfügung gestellt werden, ist von Anbieter zu Anbieter verschieden. Typische Zusatzdienste im GSM-Netz jedoch sind Übermittlung der Teilnehmerkennung, Rufum- oder Rufweiterleitung und Konferenzschaltungen.

2.3.4 Short Message Service

Der Short Message Service (SMS) ist ein sehr einfacher Nachrichtenübermittlungsdienst, der den Austausch von Kurznachrichten zwischen einer MS und einem anderen SMS-fähigen Gerät erlaubt. Die Länge der Kurznachrichten ist beschränkt auf 160 alphanumerische Zeichen. Zur Realisierung des SMS ist ein SMSC (Short Message Service Centre) nötig, das zur Speicherung und Weiterleitung der Kurznachrichten dient. Wird eine Kurznachricht von einem SMSC zu einer MS übertragen, so bezeichnet man sie als mobile-terminated (MT), im umgekehrten Fall als mobile-originated (MO). Daher ist der SMS in zwei grundlegende Dienste unterteilt: Short Message Mobile Terminated (SM MT) und Short Message Mobile Originated (SM MO).

Bevor der genaue Ablauf dieser beiden Dienste aufgezeigt werden kann, müssen die Elemente der zugrundeliegenden Netzstruktur und ihre Funktionen näher beschrieben werden.

Netzarchitektur

Wie die Architektur in Abbildung 2.13 zeigt, erfolgt die Übertragung von SMS-Nachrichten generell von einer MS und zu einer SME (Short Message Entity) und umgekehrt. Eine SME zeichnet sich durch die Fähigkeit aus, Kurznachrichten empfangen oder versenden zu können. Sie kann sich beispielsweise in einem Festnetz, einer MS oder einem SMSC befinden.

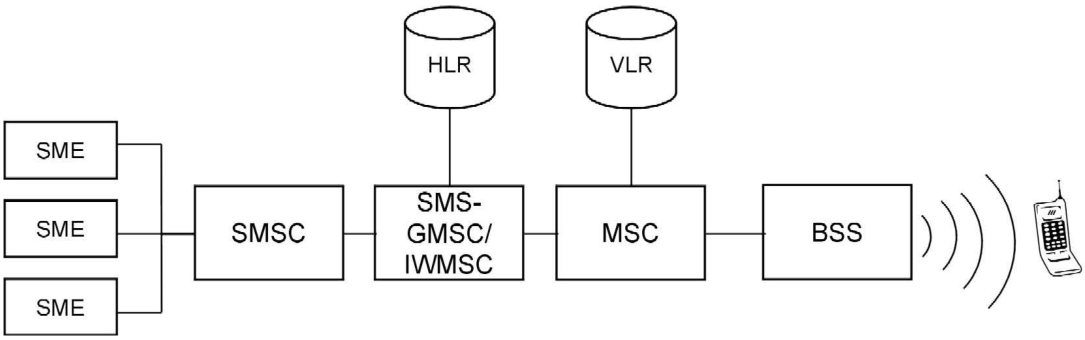


Abbildung 2.13: Grundlegende Architektur für den SMS

Das SMSC spielt, wie bereits erwähnt, eine wesentliche Rolle bei der Realisierung des SMS. Es verbindet externe SMEs mit dem GSM-Festnetz. Das SMSC ist zuständig für die Übernahme, das Speichern und das Weiterleiten von Kurznachrichten. Das Speichern von Kurznachrichten ist notwendig, wenn eine Auslieferung an den Empfänger gerade nicht möglich ist, z.B. bei ausgeschaltetem Mobiltelefon. Sobald der Empfänger wieder verfügbar ist, leitet das SMSC die Kurznachrichten weiter. Somit ist die Auslieferung von Kurznachrichten innerhalb ihrer Gültigkeitsdauer gewährleistet.

Die Aufgabe des GSM-Festnetzes besteht darin, die Kurznachrichten zwischen einem SMSC und einer MS zu übermitteln. Dies erfolgt über folgende GSM-Netzinstanzen: SMS-GMSC bzw. SMS-IWMSC, MSC, BSC und BTS. Das SMS-GMSC (Gateway MSC for Short Message Service) wird im Fall einer MT-Nachricht eingesetzt. Dieses spezielle MSC hat die Fähigkeit, Kurznachrichten von einem SMSC entgegenzunehmen, sich aus dem HLR die nötigen Routing-Informationen zu holen und die Nachrichten an das entsprechende MSC weiterzusenden. Das SMS-IWMSC (Interworking MSC for Short Message Service) kommt im umgekehrten Fall, also bei MO-Nachrichten zum Einsatz. Es ist imstande, Kurznachrichten aus dem GSM-Festnetz zu empfangen und diese an das SMSC, das für den Empfänger zuständig ist, weiterzuleiten. Der Rest der hier aufgeführten GSM-Instanzen ist bereits aus Abschnitt 2.3.1 bekannt.

Die Übertragung der Kurznachrichten zwischen den betreffenden GSM-Netzinstanzen erfolgt nicht über die normalen Datenverbindungen, sondern über freie Kapazitäten auf Signalisierungskanälen.

Dies führt zu einer besseren Auslastung dieser Kanäle und das Senden oder Empfangen von Kurznachrichten ist auch dann möglich, wenn zeitgleich eine Sprach- oder Datenübertragung stattfindet.

Protokollarchitektur

Der SMS ist ein Dienst der GSM-Vermittlungsschicht (Schicht 3). Die Vermittlungsschicht im GSM enthält drei Unterschichten, das Radio Resource Management (RR), das Mobilility Management (MM) und das Call Management (CM). RR ist die niedrigste Unterschicht und ihre Hauptaufgaben sind die Belegung, Aufrechterhaltung und Freigabe von Trägerfrequenzen. Die RR-Unterschicht ist also zuständig für die Verwaltung der physikalischen und logischen Kanäle. Die mittlere Unterschicht MM stellt Funktionen für die Registrierung, Authentifizierung, Geräteidentifikation und Aktualisierung des Aufenthaltsorts zur Verfügung. Außerdem wird hier eine temporäre Teilnehmerkennung, die Temporary Mobile Subscriber Identity (TMSI), bereitgestellt. Sie ersetzt die International Mobile Subscriber Identity (IMSI) und verschleiert somit die echte Identität eines Teilnehmers. Im Gegensatz zur IMSI gilt eine TMSI immer nur in der aktuellen LA eines VLR. Sowohl RR als auch MM bieten jeweils der nächsthöheren Schicht eine zuverlässige Datenverbindung an.

CM, die oberste Unterschicht, besteht aus drei voneinander unabhängigen Komponenten: Call Control (CC), Short Message Service (SMS) und Supplementary Service (SS). Die SS-Komponente liefert die im vorigen Abschnitt beschriebenen Zusatzdienste. Die CC-Komponente bietet eine Punk-zu-Punkt-Verbindung zwischen zwei Endgeräten an, d.h. alle höheren Schichten nutzen CC zum Verbindungsaufbau, -abbau und zum Ändern der Verbindungsparameter. Die SMS-Komponente stellt den hier beschriebenen SMS-Dienst zur Verfügung. Sie ist für das Senden von Kurznachrichten über die Steuerkanäle SDCCH und SACCH verantwortlich.

Neben der Unterschicht CM sind an der Übertragung von Kurznachrichten drei weitere Schichten beteiligt: SM-RL (Short Message Relay Layer), SM-TL (Short Message Transfer Layer) und SM-AL (Short Message Application Layer). Abbildung 2.14 gibt einen Überblick über die Schichtinstanzen der einzelnen Netzelemente, die am SMS beteiligt sind.

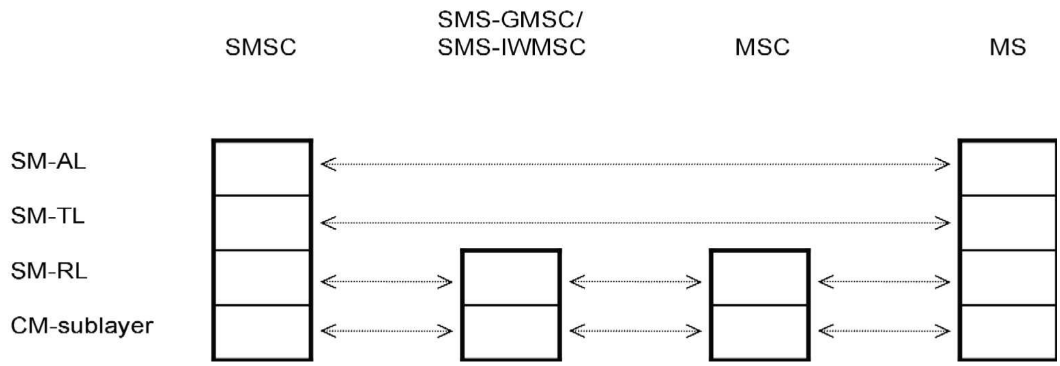


Abbildung 2.14: Protokollarchitektur für den SMS

Die oberste Schicht, SM-AL, ermöglicht das Senden von Kurznachrichten an die Partnerinstanz, das Empfangen von Kurznachrichten von der Partnerinstanz sowie das Empfangen von Übertragungsberichten, die Auskunft darüber geben, ob gesendete Kurznachrichten beim Empfänger angekommen sind oder nicht.

Die darunter liegende Schicht SM-TL kapselt jede zu übertragende Kurznachricht als Nutzdatenteil einer TPDU (Transfer Protocol Data Unit). TPDUs enthalten neben Nutzdaten auch noch Steuerinformationen und werden zwischen Partnerinstanzen der SM-TL zum Informationsaustausch transportiert. Diese Schicht bietet sechs verschiedene TPDU-Typen:

SMS-DELIVER zur Übermittlung einer Kurznachricht vom SMSC zur MS
SMS-SUBMIT zur Übermittlung einer Kurznachricht von der MS zum SMSC
SMS-STATUS-REPORT zur Übermittlung eines Berichts vom SMSC zur MS
SMS-COMMAND zur Übermittlung eines Befehls von der MS zum SMSC
SMS-DELIVER-REPORT zur Übermittlung einer positiven oder negativen Quittung an das SMSC für ein SMS-DELIVER oder SMS-STATUS-REPORT
SMS-SUBMIT-REPORT zur Übermittlung einer positiven oder negativen Quittung an die MS für ein SMS-SUBMIT oder SMS-COMMAND

SM-RL stellt das Bindeglied zwischen SM-TL und der CM-Unterschicht dar. Diese Schicht bietet zum einen Funktionen zur Übertragung von TPDUs und zum anderen Funktionen zur Benachrichtigung des SMSC über die Verfügbarkeit einer MS im GSM-Netz. SM-RL umfasst folgende Protokollelemente:

RP-MO-DATA zur Übertragung einer TPDU von der MS zum SMSC
RP-MT-DATA zur Übertragung einer TPDU vom SMSC zur MS
RP-ACK zur Bestätigung einer erfolgreichen Übertragung
RP-ERROR zur Information über das Fehlschlagen einer Übertragung
RP-ALERT-SC zur Benachrichtigung des SMSC, dass die MS wieder verfügbar ist (wird vom HLR genutzt)
RP-SM-MEMORY-AVAILABLE zur Meldung, dass die MS über freien Speicher zum Empfangen von Kurznachrichten verfügt (wird von der MS an das HLR geschickt)

RP-ACK bzw. RP-ERROR enthält im Nutzdatenteil eine SMS-DELIVER-REPORT TPDU oder eine SMS-SUBMIT-REPORT TPDU und bildet somit eine positive bzw. negative Quittung für die jeweilige Übertragung.

Die CM-Unterschicht ist für den Aufbau, die Aufrechterhaltung und den Abbau einer Verbindung zuständig. Der Verbindungsaufbau kann nur erfolgen, wenn bereits eine MM-Verbindung hergestellt ist. Über die Dienstprimitive dieser Schicht können Instanzen der SM-RL signalisieren, ob sie einen Verbindungsaufbau, -abbau oder die Übertragung ihrer RPDU wünschen. Auch hier sieht das Protokoll die Benachrichtigung der nächsthöheren Schicht über aufgetretene Fehler vor.

Short Message Mobile Terminated

Dieses Verfahren stellt alle Operationen zur Verfügung, die zur Übertragung einer TPDU von einem SMSC zu einer MS notwendig sind. Dazu gehört auch eine Rückmeldung an das SMSC, ob die Übertragung erfolgreich war oder fehlgeschlagen ist. Im Folgenden werden die einzelnen Operationen näher erläutert. Der Ablauf einer erfolgreichen Übertragung einer MT-Nachricht ist in Abbildung 2.15 dargestellt.

Operation 1: Diese Operation besteht aus zwei Teilen, der Übertragung einer Kurznachricht vom SMSC zum SMS-GMSC (1a. Message transfer) und dem Zurücksenden eines Berichts, ob die Übertragung erfolgreich war (1b. Delivery report) oder fehlgeschlagen ist (Failure report). Das SMSC erhält immer dann einen ”Failure report” , wenn eine der anderen GSM-Instanzen dem SMS-GMSC das Fehlschlagen ihrer Operation meldet.

Operation 2: Mit Hilfe dieser Operation schickt das SMS-GMSC an das HLR eine Anfrage (2. SendRoutingInfoForShortMsg). Das vom HLR gelieferte Ergebnis enthält Informationen, die zur Weiterleitung der Nachricht nötig sind, so z.B. die Adresse des MSC.

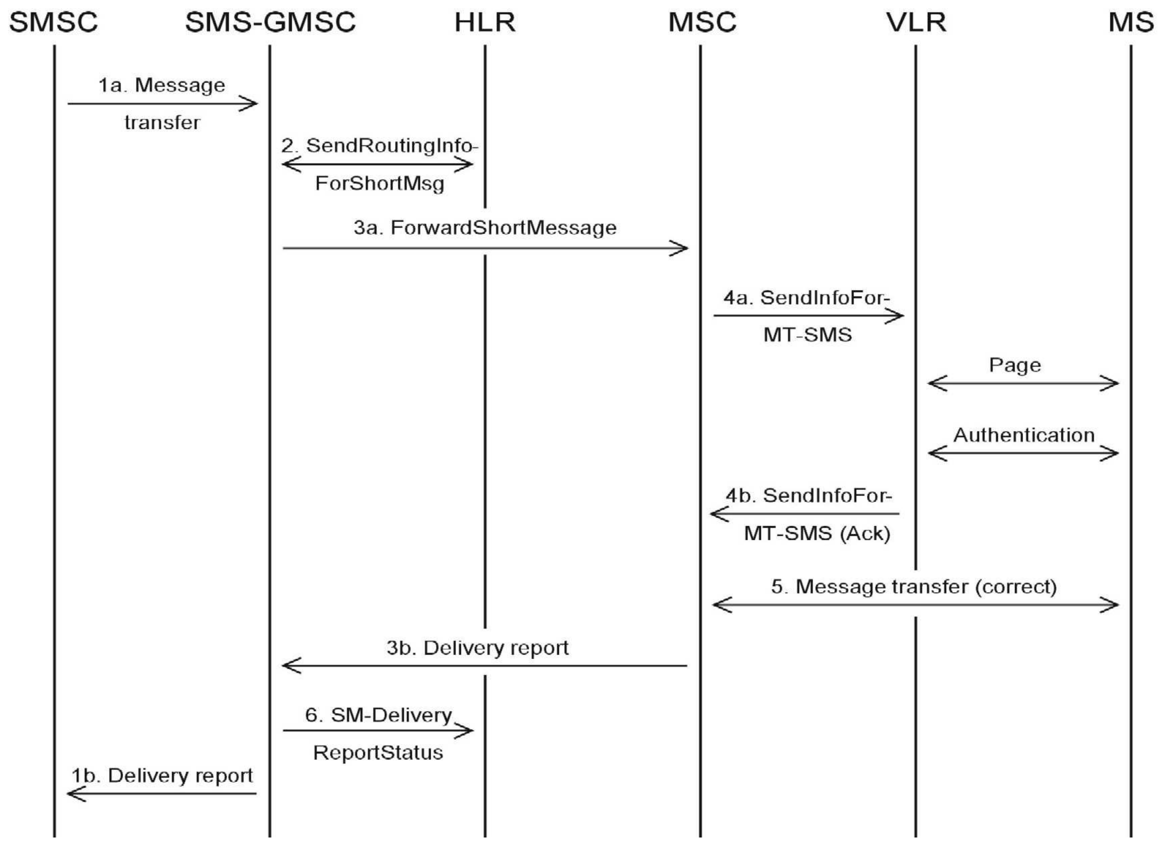


Abbildung 2.15: Ablauf des Nachrichtentransfers beim SM MT

Operation 3: Diese Operation sorgt dafür, dass das SMS-GMSC die Kurznachricht an das betreffende MSC überträgt (3a. ForwardShortMessage). Auch hier ist ein Bericht vorgesehen, der das SMS-GMSC über den Erfolg (3b. Delivery report) oder das Fehlschlagen der Übertragung an die MS informiert.

Operation 4: Diese Operation ermöglicht es dem MSC, vom VLR Informationen über den mobilen Teilnehmer, der die MT-Nachricht empfangen soll, zu erhalten (4a. SendInfoForMT-SMS). Dadurch wird ein Authentifizierungsprozess angestoßen. Bei erfolgreicher Authentifizierung wird dem MSC eine Bestätigung geschickt (4b. SendInfoForMT-SMS (Ack)). Ist dies nicht der Fall, weil beispielsweise die MS gerade nicht erreichbar ist, so erhalten alle GSM-Instanzen sukzessive einen ”Failure report” .

Operation 5: Mittels dieser Operation wird die Nachricht vom MSC zur MS übermittelt (5. Message transfer). Sollte die Verbindung während der Übertragung unterbrochen werden, so wird dem SMS-GMSC ein ”Failure report” zurückgesendet.

Operation 6: Diese Operation wird aktiviert, sobald das SMS-GMSC einen ”Failure report” empfängt, die Kurznachricht also nicht ausgeliefert werden konnte. Das HLR besitzt eine Liste mit allen SMSCs, die noch zu versendende Kurznachrichten gespeichert haben. Das SMS-GMSC überträgt damit die Adresse des SMSC und die MSISDN der MS in diese Liste (6. SM-DeliveryReportStatus). Diese Operation wird auch aktiviert, falls die Auslieferung der Kurznachricht an die MS erfolgreich war und sich das SMSC bereits in dieser Liste befindet. In diesem Fall wird die Adresse des SMSC und die MSISDN der MS aus der Liste des HLR gelöscht.

Short Message Mobile Originated

Dieses Verfahren stellt alle Operationen zur Verfügung, die zur Übertragung einer TPDU von einer MS zu einem SMSC notwendig sind. Auch hier ist ein Bericht an die MS über den Ausgang der Übertragung vorgesehen. Im Folgenden werden die einzelnen Operationen näher erläutert. Der Ablauf einer erfolgreichen Übertragung einer MO-Nachricht ist in Abbildung 2.16 dargestellt.

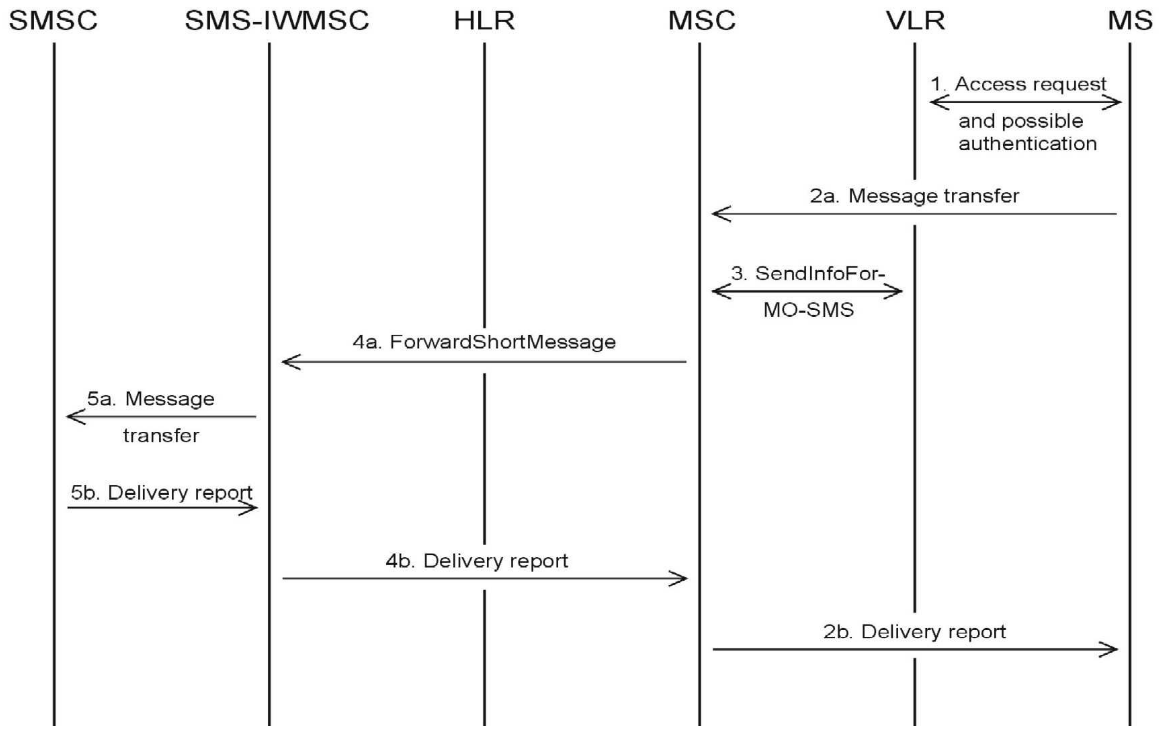


Abbildung 2.16: Ablauf des Nachrichtentransfers beim SM MO

Operation 1: Mittels dieser Operation meldet sich die MS beim GSM-Netz an und signalisiert, in welcher LA sie zur Zeit erreichbar ist.

Operation 2: Diese Operation wird verwendet, um eine Kurzmitteilung von der MS zum MSC zu übertragen (2a. Message transfer). Wird dem MSC während der ganzen Übertragung kein Fehler gemeldet, so sendet es der MS die Information, dass die Kurznachricht ausgeliefert wurde (2b. Delivery report). In jedem anderen Fall wird die MS über einen "Failure report" über das Fehlschlagen der Übertragung informiert.

Operation 3: Anhand dieser Operation kann das MSC prüfen, ob das Senden der MO-Nachricht möglich ist oder irgendwelchen Beschränkungen unterliegt (3. SendInfoForMO-SMS). Diese Informationen erhält es über das VLR.

Operation 4: Diese Operation dient dem MSC zur Weiterleitung der Kurznachricht an das SMS-IWMSC (4a. ForwardShortMessage). Der zurückgelieferte Bericht zeigt dem MSC an, ob das SMSC die Kurznachricht erhalten hat (4b. Delivery report) oder nicht (Failure report).

Operation 5: Hiermit erfolgt die Übertragung der Kurznachricht vom SMS-IWMSC zum SMSC (5a. Message transfer). Auch bei dieser Operation wird ein Bericht zurückgesendet.

2.3.5 Neue Datendienste

Als GSM entwickelt wurde, war für die damaligen Datendienste eine Bandbreite von 9,6 kbit/s vollkommen ausreichend. In den letzten Jahren zeigte sich aber immer deutlicher, dass diese Leistung den Anforderungen vieler Anwendungen wie beispielsweise Internetdiensten nicht gerecht wird. Prinzipiell gibt es zwei Möglichkeiten neue Datendienste im GSM, die eine höhere Übertragungsrate als 9,6 kbit/s aufweisen, zu realisieren. GSM arbeitet verbindungsorientiert mit 9,6 kbit/s pro Verbindung, also kann durch das Zusammenfassen mehrerer Kanäle eine höhere Bandbreite angeboten werden. Dieser Ansatz wurde von der ETSI unter dem Begriff High Speed Circuit Switched Data (HSCSD) standardisiert. Die andere Möglichkeit beruht auf der Denkweise des paketorientierten Internet und erweitert das traditionell verbindungsorientierte GSM um das Konzept der verbindungslosen Übertragung. Dieses zweite, fortschrittlichere Verfahren ist unter dem Begriff General Packet Radio Service (GPRS) standardisiert.

HSCSD

HSCSD wurde entwickelt, um dem Aufkommen größerer Datenmengen gerecht zu werden. Dieses Verfahren erweitert GSM um die Fähigkeit, gleichzeitig mehrere Verkehrskanäle zu nutzen, und erzielt somit eine höhere Datenrate. Um HSCSD einzuführen, ist lediglich eine Aktualisierung der Software in MS und MSC nötig.

HSCSD erlaubt einer MS, ein oder mehrere TCHs vom GSM-Netz anzufordern. Bei Erfolg der Anforderung erhält die MS einen oder mehrere Zeitschlitze eines 200 kHz Frequenzkanals. Dabei müssen nicht in jeder Senderichtung gleich viele Zeitschlitze belegt werden. Dies bewirkt, dass beispielsweise mehr Daten empfangen als gesendet werden können.

Durch Anpassung der Kanalcodierung an die Qualität des Funksignals ermöglicht HSCSD eine Datenrate von 14,4 kbit/s. Da maximal 4 Zeitschlitze pro Senderichtung reserviert werden können, ist HSCSD auf eine Datenrate von 57,6 kbit/s beschränkt.

Da in HSCSD verbindungsorientierte Mechanismen zum Einsatz kommen, ist es nicht für die typische Nutzung des Internet geeignet. Die Reservierung mehrerer Kanäle wäre bei dem stoßartigen Datenverkehr im Internet eine Verschwendung der Kanalkapazität. Dies schlägt sich vor allem in den Kosten der Verbindung nieder, da die reservierten Kanäle nicht von anderen Teilnehmern belegt werden können, auch wenn sie oft unbenutzt sind. Die Abrechnung der Kanäle über die Verbindungsdauer macht diese Lösung für Web-Anwendungen unattraktiv. Das hat zur Folge, dass viele Betreiber HSCSD gar nicht einführen, sondern gleich GPRS einsetzen.

GPRS

Im Gegensatz zu HSCSD wird die Datenübertragung in GPRS anforderungsgesteuert und paketorientiert durchgeführt. Die Datenströme der einzelnen Teilnehmer werden in Pakete zerlegt und gemeinsam auf verschiedenen Kanälen übertragen. Dadurch ist eine bessere Nutzung der Netzressourcen vor allem im Hinblick auf typische Web-Anwendungen möglich. Mit GPRS wird also ein Datentransferdienst angeboten, der besser zu dem verbindungslosen, paketorientierten Schema im Internet passt. Das bringt den Vorteil, dass die Abrechnung nach dem tatsächlichen Datenvolumen erfolgen kann und nicht wie bei HSCSD die Verbindungsdauer ausschlaggebend ist. Allerdings benötigt GPRS zusätzliche Hardware, eine Erweiterung der Software und ein Austausch der mobilen Endgeräte reicht in diesem Fall nicht aus.

GPRS kann traditionelle Verkehrs- und Steuerkanäle des GSM-Netzes nutzen und sieht keine Beschränkung der Datenrate vor. Diese wird nur durch die technischen Gegebenheiten im GSM-Netz beschränkt. Ein GPRS-Kanal kann einen bis acht Zeitschlitze auf einem 200 kHz Kanal belegen. Bei der Belegung der Zeitschlitze wird nicht nach einem festgelegten Muster vorgegangen, sondern nach den sich ergebenden Anforderungen. Alle Zeitschlitze können je nach Netzlast an die Nutzer verteilt werden, die beiden Senderichtungen werden völlig unabhängig voneinander behandelt. Je nach Art der Codierung kann eine Datenrate von bis zu 150 kbit/s erreicht werden.

Zur Realisierung von GPRS müssen zwei neue, im Allgemeinen als GPRS Support Nodes (GSN) bezeichnete Netzelemente in die Standardarchitektur von GSM integriert werden. Das erste neue Element ist ein Gateway GSN (GGSN) und dient als Verbindungseinheit zwischen dem GPRS-Netz und externen paketorientierten Netzen. Intern ist ein GGSN immer mit einem Serving GSN (SGSN), dem zweiten neu eingeführten Element, verbunden. Ein SGSN entspricht im Hinblick auf seine Funktionen einem MSC in einem GSM-Netz, ist also zuständig für die Weiterleitung der Datenpakete von und zu den mobilen Teilnehmern.

GPRS-Dienste ersetzen die konventionellen Träger- und Teledienste nicht, sie können zusätzlich genutzt werden. Dabei bietet GPRS zwei unterschiedliche Dienste: Point-to-Point (PTP) und Point-to-Multipoint (PTM). PTP-Dienste unterstützen die Übertragung von Paketen zwischen einem Sender und einem Empfänger. Die Übertragung kann dabei verbindungsorientiert oder verbindungslos erfolgen. PTM-Dienste erlauben die Übertragung von Paketen zwischen einem Sender und einer Empfängergruppe. Eine Empfängergruppe kann entweder aus allen Teilnehmern innerhalb einer bestimmten geographischen Region bestehen oder aus allen in einer bestimmten Teilnehmergruppe registrierten Mitgliedern.

Die Hauptvorteile von GPRS liegen in der höheren Datenrate und in der volumenabhängigen Abrechnung. Für Internet-orientierte Anwendungen ist GPRS als Ergänzung zum GSM-Netz die bestmögliche Lösung.

2.4 UMTS - Universal Mobile Telecommunications System

GSM wird sicher noch einige Jahre das bestimmende Mobilfunksystem bleiben und erst mit der Zeit von UMTS abgelöst werden. Im Folgenden wird ein grober Überblick über die Eigenschaften, Technik und Dienste des neuen Standards UMTS gegeben.

2.4.1 Anforderungen und Eigenschaften

UMTS gehört zu den Mobilfunksystemen der 3. Generation und soll zu einem weltweit einheitlichen, öffentlichen Mobilfunksystem führen. Für ein Mobilfunksystem der 3. Generation ergeben sich verschiedene Charakteristika und Anforderungen:

- Unterstützung sämtlicher Eigenschaften, die von den bestehenden Systemen angeboten werden
- Unterstützung neuer Dienste mit einer dem Festnetz ähnlichen Dienstgüte und wahlweise leitungs- oder paketvermittelte Übertragung
- Kleine, leichte und billige Endgeräte
- Hohe Kapazität für eine hohe Marktdurchdringung
- Verschiedene Bitraten (niedrige Bitraten für Sprache, Bitraten bis zu 2 Mbit/s für Multimediaanwendungen)
- Einsatz von unterschiedlich großen Zellen für Indoor- und Outdoor-Anwendungen
- Hohe Spektrumseffizienz
- Flexibles Management der Frequenzen und der Funkressourcen
- Hohe, dem Festnetz ähnliche Sicherheit

Als weitere Anforderungen sind noch zu nennen, dass die Übergabe der Verbindungen nicht nur innerhalb von UMTS, sondern auch zwischen UMTS und GSM oder Satellitennetzen möglich sein sollte, und UMTS kompatibel zu GSM-, ATM-, IP- und ISDN-basierten Netzen sein sollte. Um diesen Anforderungen gerecht zu werden, sind Fortschritte in der Übertragungstechnologie und höhere Trägerfrequenzen nötig. Für UMTS sind Frequenzen um 2000 MHz vorgesehen.

2.4.2 Aufbau und Technik

Ein UMTS-Netz besteht aus einem Kernnetz und einem Funknetz.

Das Kernnetz von UMTS ist im Grunde eine Weiterentwicklung des GSM-Festnetzes. Es umfasst verschiedene Verbindungsknoten, die die Basisstationen untereinander verknüpfen und Übergänge zu anderen Netzen wie ISDN und Internet schaffen.

Das Funknetz von UMTS hingegen ist eine vollständige Neuentwicklung. Es umfasst ganz analog zu GSM die Mobilstationen und die Basisstationen, unterscheidet sich aber grundlegend in der Funkübertragung zwischen Mobil- und Basisstation. Das UMTS-Funknetz basiert nämlich auf einer komplexen, hierarchischen Zellstruktur, die verschiedene Übertragungsgeschwindigkeiten zulässt. Je höher die Hierarchieebene, desto größer ist das Gebiet, das durch die Zelle versorgt werden kann.

In der höchsten Hierarchieebene kommen Satelliten zum Einsatz, die große, wenig besiedelte Gebiete erschließen, und somit eine Versorgung ohne explizite Infrastruktur ermöglichen. Sie bieten eine Datenrate von 144 kbit/s.

Die darunter liegenden Hierarchieebenen stellen das sogenannte UTRAN (UMTS Terrestrial Radio Access Network) dar. Jede Ebene ist zellular aufgebaut und je weiter unten sie sich in der Hierarchie befindet, desto kleiner ist der Zellradius. UTRAN besteht aus Macro-, Micro- und Pico-Zellen.

Macro-Zellen erstrecken sich über ein größeres abgegrenztes Gebiet (z.B. eine Stadt). Sie dienen dort zur flächendeckenden Grundversorgung mit einer maximalen Datenrate von 144 kbit/s bei einer Geschwindigkeit von 500 km/h.

Micro-Zellen umfassen eine Fläche von einigen Quadratkilometern und dienen zur zusätzlichen Versorgung bei dicht besiedelten Gebieten. Hier beträgt die Datenrate mindestens 384 kbit/s bei einer Geschwindigkeit von 120 km/h und an die 2 Mbit/s bei langsamer Fortbewegung.

Pico-Zellen besitzen einen Durchmesser von wenigen hundert Metern und befinden sich in Häusern, Firmen, Flughäfen oder Bahnhöfen. Nur innerhalb von Pico-Zellen ist die maximale Übertragungsrate von 2 Mbit/s bei einer Geschwindigkeit von 10 km/h verfügbar.

Nicht nur im Bezug auf das Funknetz unterscheiden sich GSM und UMTS, sondern auch in der Art, wie die Daten zwischen Mobilstation und Basisstation ausgetauscht werden. Bei UMTS erfolgt der Zugriff auf das gemeinsam genutzte Medium über ein Wideband-CDMA-Verfahren. Dabei werden die Daten durch Codes auseinander gehalten und alle Teilnehmer können zur gleichen Zeit auf demselben Frequenzband senden. Dadurch kann UMTS die vorhandenen Frequenzen viel effizienter nutzen.

Ein weiterer Unterschied liegt darin, dass bei UMTS die Zellen ihre Größe in Abhängigkeit der Anzahl der Nutzer ändern können. Jede UMTS-Zelle hat eine maximale Sendeleistung. Je mehr aktive Teilnehmer sich in einer Zelle befinden, desto weniger Leistung entfällt auf den einzelnen. Um dies zu kompensieren, wird der mögliche Abstand zur Basisstation, also der Zellradius verringert.

2.4.3 Dienste

Mit dem schnelleren und leistungsfähigeren UMTS ergeben sich ganz neue Nutzungsmöglichkeiten. Während mit GSM nur geringe Datenmengen übertragen werden konnten, stellt bei UMTS die Übertragungsrate kein Hindernis für die Entwicklung von Anwendungen und Diensten dar. Beispiele für diese Dienste sind MMS (Multimedia Message Service) oder WAP (Wireless Application Protocol). Ein Teil der Dienste ist bereits in GSM eingeführt worden und kann ins UMTS-System nahtlos übernommen werden.

WAP sorgt für eine mobile Datenkommunikation und ermöglicht Mobiltelefonen den Zugriff auf das Internet. Mit dem MMS lassen sich einfache Textnachrichten um Grafiken, Fotos, Ton- und Videoaufnahmen bereichern. Diese Dienste konnten sich in GSM aufgrund der geringen Bandbreite nicht richtig etablieren. Das soll sich mit der Einführung von UMTS ändern, da das neue System für diese Dienste eine ausreichende Bandbreite zur Verfügung stellt.

Ein bei UMTS neu eingeführter Dienst ist das Virtual Home Environment (VHE). Dieser Dienst gibt dem Nutzer die Möglichkeit, sich eine Menge von Diensten zusammenzustellen und diese in jedem Fremdnetz zu nutzen. Dienste, die im Fremdnetz eigentlich nicht angeboten werden, emuliert das VHE, so dass der Nutzer immer seine gewohnte Umgebung vorfindet.

Technologisch gesehen löst UMTS offenkundige Probleme von GSM, aber wie leistungsfähig UMTS im Alltag unter extremen Bedingungen ist und wie hoch die Akzeptanz der angebotenen UMTS-Dienste bei den Nutzern wirklich ist, wird sich erst nach der Einführung von UMTS zeigen.

Kapitel 3

Entwicklungsumgebung bei BURDAWIRELESS

Dieses Kapitel gibt einen Überblick über die Entwicklungsumgebung bei BURDAWIRELESS. Dazu wird dargestellt, wie die SMS-Plattform von BURDAWIRELESS aufgebaut ist und wie der Ablauf eines Dienstes auf dieser Plattform aussieht. Im ersten Abschnitt dieses Kapitels werden die einzelnen Komponenten der SMS-Plattform, ihre Aufgaben und ihr Zusammenspiel näher beschrieben. Im zweiten Abschnitt wird dann anhand eines typischen Szenarios aufgezeigt, welche Komponenten nacheinander durchlaufen werden vom Eingang der SMS-Nachricht bis hin zur zuständigen Applikation und welche Aktionen dort konkret durchgeführt werden. Abschließend werden Problembereiche, die die SMS-Plattform in ihrem momentanen Zustand aufweist, aufgeführt.

3.1 Architektur der SMS-Plattform

Die SMS-Plattform von BURDAWIRELESS ist über das Internet an die SMSCs der verschiedenen Netzbetreiber angeschlossen. Mit den SMSCs von D1, D2 und E-Plus ist die Plattform über ein VPN (Virtual Private Network) verbunden, mit den SMSCs von O2, Debitel und MobilCom über unverschlüsselte TCP/IP-Socket-Verbindungen.

Für den Empfang der Kurznachrichten, die von den SMSCs an die Plattform weitergeleitet werden, ist das Tool SendXMS zuständig. SendXMS unterstützt alle gängigen SMSC-Protokolle wie UCP (Universal Computer Protocol), SMPP (Short Message Peer to Peer), CIMD (Computer Interface to Message Distribution) und OIS (Open Interface Specification). Diese Protokolle haben leicht unterschiedliche Funktionalitäten und verwenden auch nicht dieselbe Zeichenkonvertierung.

Alle von der SMS-Plattform empfangenen Kurznachrichten werden zunächst in die Datenbank (MySQL) geschrieben. Die dafür angelegte Tabelle *InboundSMS* enthält als Attribute alle von den SMSCs zur Verfügung gestellten Daten sowie ein Feld, das den Status der Verarbeitung anzeigt. Tabelle 3.1 zeigt das zugrundeliegende Relationenschema.

Schlüssel dieses Schemas ist das Attribut *id*. Damit wird jeder eingegangenen Kurznachricht eine eindeutige Nummer, die automatisch beim Einfügen eines neuen Datensatzes in die Tabelle erzeugt wird,

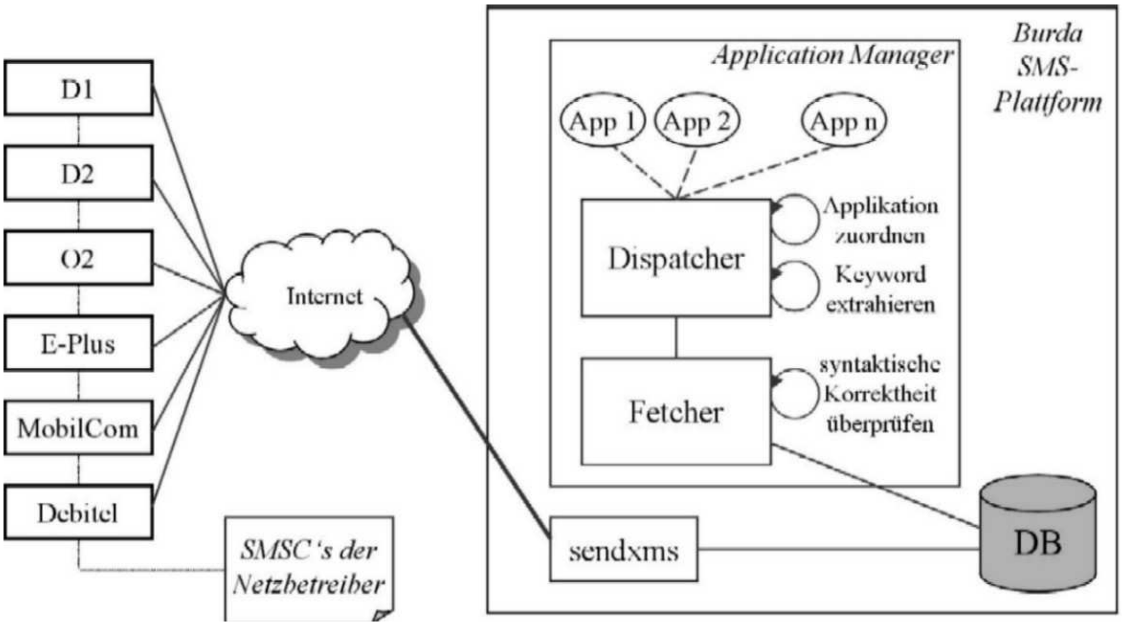


Abbildung 3.1: Architektur der SMS-Plattform

id	short_id	originating_address	datum	provider	message	status

Tabelle 3.1: Relationenschema für *InboundSMS*

zugeordnet. Das Attribut *short_id* bezeichnet die Kurzwahlnummer, an die die Kurznachricht gesendet wurde, das Attribut *originating_address* die Telefonnummer bzw. Absenderkennung des Senders und das Attribut *datum* das Datum, an dem die Kurznachricht beim SMSC eingetroffen ist. Unter textitprovider findet man den Netzbetreiber, über den die Kurznachricht gesendet worden ist, und unter *message* die eigentliche Nachricht.

Die Verarbeitung der in der Datenbank gespeicherten Kurznachrichten ist Aufgabe des *Application Managers*. Der *Application Manager* ist ein in Java entwickeltes Framework, das zur Verteilung von eingehenden Kurznachrichten auf verschiedene Applikationen dient, die abhängig vom Inhalt der Kurznachricht unterschiedliche Aktionen durchführen können.

Application Manager Framework

Das *Application Manager* Framework besteht im Wesentlichen aus zwei Komponenten, dem *Fetcher* und dem *Dispatcher*. Der *Fetcher* ist für das Einlesen von eingegangenen SMS-Nachrichten und ihrer anschließenden Übergabe an den *Dispatcher* verantwortlich. Im *Dispatcher* erfolgt die eigentliche Verarbeitung und Weiterleitung der übergebenen SMS-Nachrichten an verschiedene Applikationen. Jede Applikation realisiert dabei einen oder mehrere Dienste, die über die Plattform angeboten werden.

Fetcher

Das *Fetcher*-Package enthält das Interface *InboundSMSFetcher*, das eine Methode zum Einlesen von *InboundSMS*-Datensätzen bereitstellt. Es existieren zwei Implementierungen dieses Interfaces: *InboundDBSingleSMSFetcher* und *InboundDBSMSFetcher*. Beide Implementierungen können auf die vorher beschriebene *InboundSMS*-Tabelle zugreifen und einen oder mehrere Datensätze einlesen. Aus jedem der eingelesenen Datensätze wird eine *InboundSMS*-Instanz erzeugt, die dann vom *Dispatcher* weiter verarbeitet wird. Sind einzelne Felder des Datensatzes syntaktisch nicht korrekt, wird keine *InboundSMS*-Instanz erzeugt und der Datensatz nicht weiter verarbeitet.

Beim Erzeugen einer *InboundSMS*-Instanz werden folgende Prüfungen vorgenommen:

1. *id* muss eine Zahl größer 0 sein
2. *short_id* muss ein String mit mindestens einem Zeichen sein
3. *originating_address* muss in eine internationale Telefonnummer konvertierbar sein
4. *datum* muss ungleich null sein
5. *provider* muss ein String mit mindestens einem Zeichen sein
6. *message* muss ein String mit mindestens einem Zeichen sein

Nur die Punkte 3 und 6 können vom Absender der Kurznachricht gesetzt werden, die restlichen Punkte sind unkritisch. Die Umwandlung eines Datensatzes in eine *InboundSMS*-Instanz schlägt also nur fehl, wenn das *message*-Attribut leer ist oder die unter dem Attribut *originating_address* aufgeführte Absenderkennung nicht in eine Telefonnummer konvertiert werden kann.

Dispatcher

Die Kernkomponente des *Dispatcher*-Packages ist das Interface *SMSDispatcher* mit der Methode `dispatchSMS()`, die die übergebene *InboundSMS*-Instanz an eine Applikation weiterleitet.

Um diese Aufgabe zu erfüllen, werden dem *SMSDispatcher* ein *KeywordExtractor* und ein *KeywordApplicationMapper* zugeordnet. Der *KeywordExtractor* ist dafür zuständig, das Keyword aus dem *message*-Attribut einer *InboundSMS* zu extrahieren. Dazu müssen dem *KeywordExtractor* alle Keywords bekannt sein. Diese erhält er über den ihm zugeordneten *KeywordApplicationMapper*, der eine Liste aller Keywords verwaltet. Der *KeywordApplicationMapper* hat die Aufgabe, zu jedem Keyword, das der *KeywordExtractor* extrahiert, die passende Applikation zu finden und zurück zu liefern. Dazu wird dem *KeywordApplicationMapper* eine Konfigurationsdatei übergeben, die alle Keywords mit den zugeordneten Applikationen enthält.

Jede Applikation muss das SMSApplication-Interface mit der Methode `execute()` implementieren. Somit ist gewährleistet, dass jede Applikation die `execute()`-Methode besitzt. Diese Methode dient dazu, in der jeweiligen Applikation die Verarbeitung der *InboundSMS*, die übergeben wird anzustoßen. Wird eine Applikation vom *KeywordApplicationMapper* zurückgeliefert, so ruft der *SMSDispatcher* einfach die `execute()`-Methode dieser Applikation auf und die übergebene *InboundSMS* wird entsprechend verarbeitet. Mit dem Aufruf der `execute()`-Methode ist die Verarbeitung der Kurznachricht durch den *Dispatcher* abgeschlossen. Alle weiteren Aktionen sind abhängig von der Implementierung der einzelnen Applikationen.

3.2 Beispielszenario

Viele Zeitschriften bieten mittlerweile das Einsenden der Lösung eines Kreuzworträtsels per SMS an. Dies ist ein typischer Dienst, den BURDAWIRELESS mit Hilfe der SMS-Plattform für verschiedene Redaktionen realisiert. Im Folgenden wird der Ablauf dieses Dienstes für ein Preisrätsel in der Zeitschrift "YoungLisa" beschrieben. Dargestellt ist das Ganze in Abbildung 3.2 .

Ein Leser, der an dem Preisrätsel per SMS teilnehmen will, schickt eine Kurznachricht mit dem für die Zeitschrift eingeführten Keyword "YL" und dem von ihm gefundenen Lösungswort an die in der Zeitschrift angegebene Kurzwahlnummer. Die Kurznachricht wird von dem für diese Kurzwahlnummer zuständigen SMSC des Netzbetreibers zwischengespeichert und über die entsprechende Verbindung an die Zieladresse ausgeliefert, d.h. in diesem Fall an die SMS-Plattform von BURDAWIRELESS. Dort wird die Kurznachricht von SendXMS empfangen und erst mal in die Datenbank geschrieben, wo sie in der *InboundSMS*-Tabelle mit den entsprechenden Attributwerten gespeichert wird.

Der *Fetcher* holt den neu angelegten Datensatz aus der Datenbank und erzeugt daraus nach Feststellung der syntaktischen Korrektheit eine *InboundSMS*-Instanz. Diese wird dem *Dispatcher* übergeben, der mit Hilfe des *KeywordExtractors* das enthaltene Keyword "YL" findet und extrahiert. Anschließend tritt der *KeywordApplicationMapper* in Aktion und sucht anhand der Konfigurationsdatei die dem Keyword "YL" zugeordnete Applikation. Er stellt fest, dass die *YoungLisaApplication* dem Keyword "YL" zugeordnet ist, und liefert sie zurück. Nun ruft der *Dispatcher* die `execute()`-Methode der *YoungLisaApplication* mit der vorher erzeugten *InboundSMS*-Instanz als Parameter auf und startet somit die Verarbeitung der Kurznachricht innerhalb der Applikation.

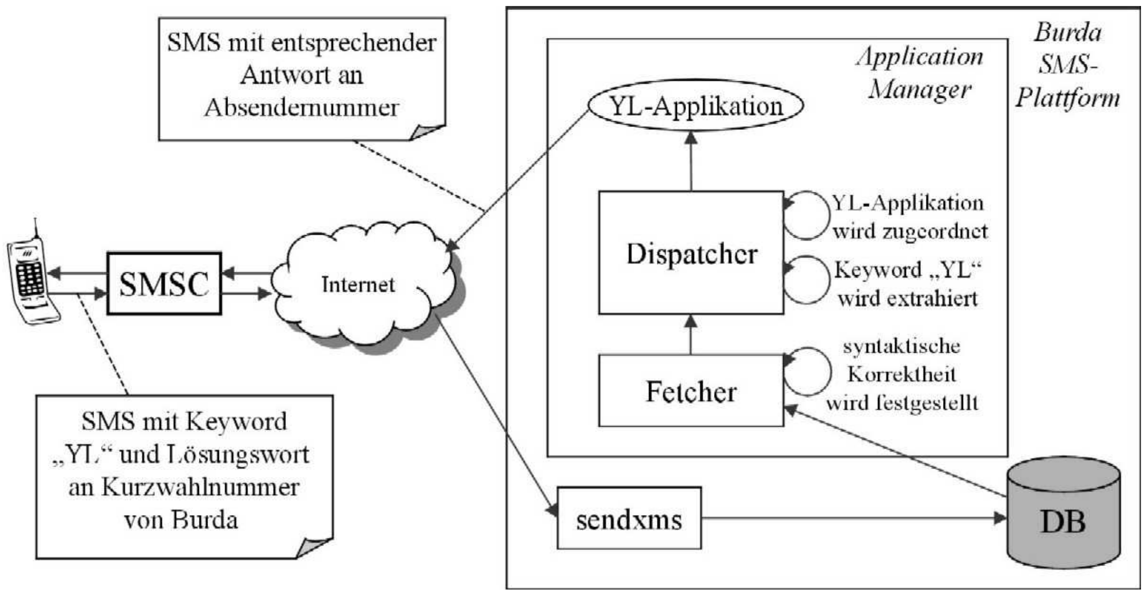


Abbildung 3.2: Beispiel eines Dienstablaufs

In der *YoungLisaApplication* wird geprüft, ob das richtige Lösungswort im Text der Kurznachricht, also im *message*-Attribut der übergebenen *InboundSMS* enthalten ist. Wird das Lösungswort gefunden, so erhält der Absender eine Nachricht, dass das von ihm gesendete Lösungswort richtig ist und er in den Gewinnpool aufgenommen wird. Ist das gesendete Lösungswort falsch, so wird der Absender auch darüber informiert und kann es, wenn er will, noch einmal versuchen.

3.3 Problembereiche

Die Architektur der SMS-Plattform und das Beispielszenario machen deutlich, dass die Plattform in manchen Bereichen noch sehr rudimentär ist. Dadurch ergeben sich verschiedene Probleme.

Der erste größere Problembereich lässt sich sehr gut an dem Beispielszenario erkennen. Die Plattform unterstützt nur Pull-Dienste. Der Nutzung eines Dienstes muss immer eine Aktion des Dienstnutzers, nämlich das Senden einer Kurznachricht, vorausgehen. Der Dienstnutzer muss in jedem Fall den Dienst aktiv anfordern. Die Dienstanbieter haben nicht einmal die Möglichkeit, von sich aus aktiv zu werden und mobile Teilnehmer beispielsweise über anstehende Aktionen, neue Angebote oder wichtige Ereignisse zu informieren.

Die Plattform bietet keinerlei Unterstützung für die Realisierung von Push-Diensten. Es kann weder definiert werden, welche Ereignisse das Senden einer Push-Nachricht auslösen, noch gibt es einen Mechanismus, der beim Eintritt eines vordefinierten Ereignisses das Senden der entsprechenden Push-Nachricht dann triggert. Außerdem gibt es keine Möglichkeit, Informationen über die Interessen und Vorlieben der Dienstnutzer zu sammeln und in Form von Interessenprofilen abzuspeichern. Das Erstellen und Verwalten von Interessenprofilen ist aber grundlegend dafür, definieren zu können, welcher Teilnehmer bei welchem Ereignis eine Push-Nachricht erhält. Diese Push-Nachricht sollte dabei für den mobilen Teilnehmer grundsätzlich einen offensichtlichen Mehrwert darstellen und ihn zu weiteren Interaktionen mit der Plattform veranlassen.

Ein weiterer Problembereich ist die Diensterzeugung an sich. Bisher wurden die Applikationen immer nur kundenspezifisch entwickelt. Das hat zum einen den Nachteil, dass Änderungen oder Erweiterungen schon realisierter Dienste oft komplexe Eingriffe in den Code nach sich ziehen. Eine Änderung des Codes hat häufig weitere Änderungen an anderen Stellen zur Folge. Das bedeutet, dass bei einer Änderung des Codes auch immer der Rest des Codes auf mögliche Fehler, die durch die Anpassung entstanden sind, überprüft werden muss. Auch Erweiterungen sind meist nicht mit einem Eingriff an einer einzigen Stelle erledigt, sondern erfordern mehrere Eingriffe an unterschiedlichen Stellen. Idealerweise sollten Erweiterungen und Änderungen bereits realisierter Dienste schnell und ohne großen Aufwand durchgeführt werden können.

Zum anderen bringt die kundenspezifische Entwicklung den Nachteil mit sich, dass neue Dienste immer von Grund auf neu aufgesetzt werden müssen. Die Dienste ähneln sich zwar in vielen Teilfunktionen, aber, da diese Teilfunktionen kundenspezifisch entwickelt wurden, ist einen direkte Wiederverwendung kaum möglich. Der Code kann nur bruchstückhaft übernommen werden und selbst dann müssen noch einzelne Programmzeilen dem Kunden entsprechend geändert werden. Diese Tatsache macht das Erzeugen neuer Dienste aufwändig und zeitintensiv. Gerade in dieser Beziehung sollte eigentlich das Gegenteil der Fall sein.

Lösungsansätze für die hier aufgeführten Probleme zu finden und diese dann prototypisch zu realisieren, ist ein wesentlicher Bestandteil dieser Arbeit.

Kapitel 4

Anforderungen und bestehende Architekturen

Im letzten Kapitel wurden ausgehend von einem typischen Szenario die Systemumgebung bei Burda und der aktuelle Zustand der SMS-Plattform beschrieben. Abschließend erfolgte eine Aufführung der Probleme, die sich nach und nach herauskristallisiert haben. Daraus lassen sich verschiedene Anforderungen, die eine Architektur für SMS-basierte Dienste erfüllen sollte, ableiten. Auf diese Anforderungen wird im ersten Teil dieses Kapitels näher eingegangen, im zweiten Teil werden dann bestehende Ansätze und Architekturen vorgestellt und mit Hilfe des Anforderungskatalogs bewertet.

4.1 Anforderungskatalog

Die einzelnen Punkte des Anforderungskatalogs ergeben sich zum einen aus den Problemen, die im letzten Kapitel aufgeführt wurden, zum anderen aus allgemeinen Anforderungen an Architekturen, die mobile bzw. SMS-basierte Dienste anbieten. Dabei spielen Erkenntnisse, die durch den Betrieb der SMS-Plattform gewonnen wurden, eine wesentliche Rolle.

4.1.1 SMS-Unterstützung

Voraussetzung für die Realisierung SMS-basierter Dienste ist natürlich die Fähigkeit, Kurznachrichten zu empfangen und zu versenden. Wie aus Kapitel 2 über die Grundlagen der Mobilkommunikation bekannt ist, muss dazu eine Verbindung zu einem SMSC bestehen. Dabei sollte die Kommunikation mit jedem SMS-fähigen Gerät, egal über welchen Netzbetreiber, möglich sein. Um dies zu gewährleisten, muss eine Anbindung an die SMSCs aller Netzbetreiber erfolgen.

Die Schwierigkeit dabei ist, dass in den SMSCs nicht ein einheitliches Protokoll, sondern mehrere verschiedene Protokolle verwendet werden. Die einzelnen Protokolle unterscheiden sich leicht in ihren Funktionalitäten und ihnen liegt auch nicht dieselbe Zeichenkonvertierung zugrunde. Die Komponente, die für den Empfang der Kurznachrichten zuständig ist, d.h. direkt mit den SMSCs verbunden ist, muss also alle SMSC-Protokolle unterstützen.

4.1.2 Verfügbarkeit

Für manche Dienste, wie beispielsweise einem Chat oder dem Abruf von Staumeldungen, ist es sehr wichtig, dass sie zu jedem Zeitpunkt genutzt werden können. Unabhängig davon, wann die Kurznachrichten eingehen, müssen die entsprechenden Aktionen, wie das Weiterleiten der Kurznachricht an den Chatpartner oder das Senden einer Kurznachricht mit den gewünschten Informationen, ausgelöst werden. Deshalb sollte eine Architektur, die solche Dienste anbietet, 24 Stunden und sieben Tage die Woche verfügbar sein.

Auftretende Fehler oder hohe Last dürfen nicht gleich zu einem Systemabsturz führen. Sollte ein Fehler auftreten, muss dies vor den Nutzern verborgen bleiben. Im Fall eines Absturzes sollte man den Fehler schnell lokalisieren und das System ohne großen Aufwand wieder zum Laufen bringen können. Die Architektur muss also eine hohe Fehlertoleranz aufweisen und auch bei extrem hoher Last zuverlässig arbeiten.

4.1.3 Einfache Dienstgenerierung

Wie bereits im letzten Kapitel beschrieben, ist die Erzeugung von neuen Diensten bei der jetzigen Plattform aufwändig und zeitintensiv. Eine der wichtigsten Anforderungen an eine Architektur für SMS-basierte Dienste ist wohl, dass gerade die Erzeugung neuer Dienste möglichst schnell und einfach zu realisieren ist.

Bei den bereits realisierten Diensten lässt sich beobachten, dass gewisse Teilfunktionen, wie das Senden einer Antwort per SMS oder das Suchen nach einem bestimmten Kennwort im Text der Kurznachricht, immer wieder benötigt werden. Diese Teilfunktionen sollten extrahiert und als generische Dienstkomponenten zur Verfügung gestellt werden. Dadurch müssen Teilfunktionen nur einmal entwickelt werden und können bei Bedarf wiederverwendet werden. Neue Dienste müssen somit nicht von Grund auf neu entwickelt werden, sondern können unter Verwendung der bereits entwickelten Teilfunktionen erstellt werden.

Existiert eine Grundmenge von generischen Dienstkomponenten, können neue Dienste ohne großen Aufwand aus diesen Komponenten zusammengesetzt werden. Die generischen Dienstkomponenten fungieren als Bausteine zur Konstruktion komplexer Dienste und machen die Entwicklung von komplexen Diensten somit effizienter und einfacher.

4.1.4 Einfache Dienstanpassung

Dienste, die schon aktiv sind, müssen oft an sich ändernde Vorgaben oder neue Ansprüche der Kunden angepasst werden. Ein Dienst beispielsweise, der ein monatlich stattfindendes Rätsel realisiert, muss auch jeden Monat dem neuen Lösungswort entsprechend angepasst werden. In manchen Fällen zieht die erforderliche Anpassung auch eine Erweiterung des Dienstes um eine oder mehrere zusätzliche Funktionen nach sich. Ein Beispiel hierfür wäre ein Informations-Dienst, der so angepasst werden soll, dass der Versand der Informationen nicht mehr nur per SMS, sondern auch per E-Mail möglich ist.

Da bei einer Architektur für SMS-basierte Dienste eine Anpassung der Dienste sehr häufig notwendig ist, sollte die Änderung oder Erweiterung von Diensten weder zeitaufwändig noch kompliziert sein.

Änderungen sollten relativ unkompliziert und schnell erledigt werden können und keine komplexen Eingriffe in den Code, womöglich noch an vielen verschiedenen Stellen, erfordern. Bei Erweiterungen sollten die neuen Funktionen möglichst einfach in den bestehenden Code integriert werden können und keine Umstrukturierung des Codes oder andere schwerwiegende Eingriffe erforderlich sein.

4.1.5 Ereignisgesteuertes Senden

Eine weitere wichtige Anforderung an eine Architektur für SMS-basierte Dienste ist, dass das Senden von SMS-Nachrichten auch durch den Dienstanbieter ausgelöst werden kann. In der Regel sind die angebotenen Dienste Pull-Dienste, der Dienstnutzer stößt durch eine Aktion, nämlich durch das Senden einer Kurznachricht an eine bestimmte Kurzwahlnummer, die Dienstausführung an. Das bedeutet aber auch, dass der Dienstnutzer von sich aus das nötige Interesse zeigen muss, die angebotenen Dienste zu nutzen.

Oft ist es jedoch von Vorteil, wenn der Dienstanbieter die Möglichkeit hat, selbst zu bestimmen, wann eine Nachricht gesendet wird, und nicht erst auf eine Initiative seitens des Nachfragers warten muss. Dadurch kann der Dienstnutzer gezielt über spezielle Angebote, anstehende Aktionen oder wichtige Ereignisse informiert werden. Eine Fluglinie könnte beispielsweise einen Dienst anbieten, der den Reisenden über Flugzeiten und Terminal informiert und ihn gegebenenfalls über einen Push-Dienst bei Verspätungen benachrichtigt.

Die Architektur sollte also neben der Realisierung von Pull-Diensten auch die Realisierung von Push-Diensten erlauben. Der Dienstanbieter sollte die Möglichkeit haben, mit Hilfe von einfachen Regeln zu definieren, welches Ereignis das Senden einer Nachricht auslöst und an welchen bzw. welche Teilnehmer die Nachricht zu senden ist.

4.1.6 Verwaltung von Benutzerinformationen

Die Forderung nach ereignisgesteuertem Senden kann nicht umgesetzt werden, ohne Informationen über die Interessen der Nutzer zu sammeln und zu verwalten. Zur Definition, welche Teilnehmer bei welchem Ereignis eine Nachricht erhalten, sind Informationen über ihre Interessen unabdingbar. Push-Nachrichten sollten für die Teilnehmer klar ersichtlich von Nutzen sein und auf keinen Fall als Belästigung empfunden werden. Will eine Zeitschrift auf eine anstehende Aktion hinweisen, so ist es sinnvoll, auch nur Personen, die zu der Zielgruppe dieser Zeitschrift gehören, zu kontaktieren. Über den plötzlichen Anstieg einer Aktie sollten dementsprechend nur Personen informiert werden, die regelmäßig Börsenkurse abrufen.

Eine Architektur, die Push-Dienste anbietet, sollte also auch das Sammeln und Speichern von personalisierten Daten erlauben. Es sollte möglich sein, die gewonnenen Informationen in Form von Interessenprofilen abzuspeichern und zu definieren, welche Informationen relevant sind und in das Profil aufgenommen werden.

4.2 Bestehende Architekturen und Ansätze

In diesem Abschnitt wird auf existierende Architekturen und Ansätze, die in irgendeiner Weise relevant für diese Arbeit sind, näher eingegangen und geprüft, in wie weit diese Architekturen und

Ansätze die einzelnen Punkte des Anforderungskatalogs unterstützen.

4.2.1 Staff Text Message Service

Dem Staff Text Message Service [CK 01] liegt ein Web-basiertes SMS System zugrunde, das den Versand von Kurznachrichten an eine bestimmte Gruppe von Teilnehmern unterstützt. Dieses System wurde entwickelt, um Dozenten die Möglichkeit zu geben, über ein Web-Frontend ihren Studenten dringende oder wichtige Informationen, beispielsweise dass eine Vorlesung verschoben wurde, per SMS zukommen zu lassen. In einer Datenbank werden alle dazu notwendigen Daten der Studenten gespeichert, der Zugriff auf diese Daten erfolgt über eine Web-Seite. Das System erlaubt den Dozenten, für jeden Kurs eine Liste mit allen Studenten, die diesen Kurs besuchen, und ihren Handynummern anzulegen und zu verwalten. Das Web-Frontend bietet eine Versand-Funktion, über die der Dozent Ergebnisse oder Bemerkungen entweder an den gesamten Kurs oder an einzelne Kursteilnehmer schicken kann.

Das System wurde unter dem Aspekt, dass nur der Versand von Kurznachrichten erforderlich ist, an das SMSC des günstigsten Anbieters angebunden. Der Empfang von Nachrichten ist gar nicht vorgesehen und auch nicht möglich. Der hier angebotene Dienst fällt zwar in die Kategorie der Push-Dienste, aber ereignisgesteuertes Senden an sich wird nicht unterstützt, da das Senden einer Nachricht nicht automatisch ausgelöst wird, sondern immer von einem Dozenten manuell angestoßen werden muss. Das Verwalten von Benutzerinformationen ist generell möglich, aber außer Name und Handynummer der Studenten werden keine weiteren Informationen abgespeichert. Weder das Erzeugen neuer Dienste noch das Anpassen bestehender Dienste waren Ziele bei der Entwicklung dieses Systems, werden also auch nicht unterstützt.

4.2.2 iMobile

iMobile [RH 01] ist eine Architektur, die mobilen Geräten erlaubt, untereinander Nachrichten auszutauschen und auf Dienste im Internet oder Datenbanken von Unternehmen zuzugreifen. iMobile fungiert als eine Art Gateway, das die von den mobilen Geräten gesendeten Nachrichten und Befehle empfängt und dann entweder auf die gewünschten Dienste und Informationen im Auftrag des Nutzers zugreift und die Ergebnisse zurückliefert oder die Nachrichten einfach an andere Geräte weiterleitet. Die Architektur von iMobile umfasst drei abstrakte Komponenten: Devlets, Infolets und Applets. Devlets sind für das Senden und Empfangen von Nachrichten zuständig. Dabei gibt es für jedes Protokoll, über das die verschiedenen mobilen Geräte mit iMobile kommunizieren können (z.B. SMS, WAP, HTTP, SMTP), ein eigenes Devlet. Infolets sind für den Zugriff auf die verfügbaren Informationen zuständig. Jedes Infolet verwendet beim Zugriff ein der Informationsquelle entsprechendes Protokoll (z.B. HTTP für Web-Seiten, JDBC für Datenbanken). Applets sind für die Weiterverarbeitung der von den Infolets gelieferten Informationen zuständig. Der Kern von iMobile, die Proxy Engine, stellt die Verbindung zwischen den drei Komponenten dar und verwaltet außerdem Geräte- und Benutzerprofile. Sie initiiert die Infolets und Applets, die nötig sind, um Anfragen eines Devlets zu bearbeiten, und wandelt die gelieferten Ergebnisse mit Hilfe der Geräteprofile in ein für das Ziel-Gerät geeignetes Format um.

Diese Architektur unterstützt den Versand und Empfang von SMS-Nachrichten, ermöglicht das Speichern von Geräte- und Benutzerinformationen und bietet eine hohe Verfügbarkeit. Allerdings werden nur Pull-Dienste angeboten, ereignisgesteuertes Senden findet keine Unterstützung. iMobile ermöglicht den Zugriff auf Dienste und Informationen, das Erzeugen von neuen Diensten ist hier nicht vorgesehen. Der Komponenten-basierte Aufbau macht sowohl die Integration neuer Dienste als auch die Erweiterung um neue Protokolle schnell und einfach.

4.2.3 OPC-SMS

OPC (Object Linking and Embedding for Process Control) ist eine standardisierte Schnittstelle, über die Überwachungssysteme mit Unternehmens- oder Datennetzen verbunden werden können. OPC stellt einen einheitlichen Mechanismus zur Verfügung, über den Nutzer mit verschiedenen OPC-Servern interagieren können und so Zugriff auf Daten erhalten, die zur Überwachung und Steuerung von wichtigen Systemen in Unternehmen oder öffentlichen Einrichtungen dienen. Weitere Informationen zu diesem Standard findet man unter [OPC]. OPC-SMS [Papa 02] hat die Funktion eines Gateways und stellt eine Verbindung zwischen GSM-Netzen und dem Internet her. Dadurch bietet es SMS-fähigen Geräten die Möglichkeit, über das Internet auf von OPC-Servern bereitgestellte Daten zuzugreifen. Ein OPC-SMS-Gateway besteht aus zwei Modulen, dem Short Message Driver, der für den Austausch von Kurznachrichten zwischen GSM-Netz und Gateway zuständig ist, und dem Proxy Server, der selbst wiederum aus dem Agent Dispatcher und einen oder mehreren Agenten besteht. Der Agent Dispatcher bildet die Schnittstelle zum Short Message Driver und leitet jede empfangene Nachricht abhängig vom Inhalt an den entsprechenden Agenten weiter. Die Agenten kommunizieren über das Internet mit den OPC-Servern, um die gewünschten Informationen zu erhalten, und liefern diese an den Agent Dispatcher zurück. OPC-SMS unterstützt dabei sowohl den Pull- als auch den Push-basierten Ansatz, d.h. ein mobiler Teilnehmer kann für ihn relevante Daten entweder durch das Senden einer Nachricht anfordern oder automatisch erhalten, sobald ein von ihm definiertes Ereignis eingetreten ist. Beispielsweise kann bei einem zu überwachenden System das Überschreiten eines vorgeschriebenen Grenzwertes das Senden einer entsprechenden Nachricht auslösen, so dass die zuständige Person sofort über diesen Zustand in Kenntnis gesetzt wird.

OPC-SMS bietet SMS-Unterstützung sowie ereignisgesteuertes Senden und kann durch Hinzufügen entsprechender Agenten um neue Dienste erweitert werden. Allerdings sind die hier angebotenen Dienste nicht beliebig, sondern beinhalten immer einen lesenden oder schreibenden Zugriff auf OPC-Daten. Da das ereignisgesteuerte Senden von den Interessen der Nutzer abhängt, ist das Verwalten von Benutzerinformationen in diesem Sinne auch nicht nötig.

4.2.4 COMPASS

COMPASS [AM 02] steht für Cooperation Model for Personalized And Situation dependent Services und bildet einen Rahmen für die Bereitstellung situationsabhängiger mobiler Datendienste. Über eine Dienstplattform, die vom Mobilfunkbetreiber unterhalten wird, gelangen Dienstanbieter an Informationen über die Nutzungssituation ihrer mobilen Kunden. Auf diese Weise können die angebotenen Dienste an den Nutzungskontext und die Bedürfnisse der Kunden angepasst werden. Da der COMPASS-Ansatz die Kooperation von Mobilfunkbetreiber und Dienstanbieter voraussetzt, integriert die Architektur von COMPASS beide Parteien und ermöglicht ihre Zusammenarbeit. Dabei ist die Aufgabe des Mobilfunkbetreibers, die aktuelle Nutzungssituation der mobilen Kunden zu erfassen und die daraus erstellten Situationsbeschreibungen an den Dienstanbieter weiterzuleiten. Der Dienstanbieter ist dafür zuständig, die vom Mobilfunkbetreiber bereitgestellten Informationen auszuwerten und die angebotenen Dienste situationsgerecht anzupassen. Die Architektur bietet die Möglichkeit, Regeln und Ereignisse zu definieren und zu prüfen, ob eine Situationsbeschreibung mit einer Regeldefinition oder Ereignisdefinition übereinstimmt. Die Regeldefinitionen dienen zur Individualisierung benutzerinitiierter Dienste, die Ereignisdefinitionen zum Aktivieren von Push-Diensten.

COMPASS bietet SMS-Unterstützung und erlaubt ereignisgesteuertes Senden und die dazu nötige Verwaltung von Benutzerinformationen. Allerdings ist letzteres völlig entkoppelt vom Dienstanbieter, das Erfassen und Speichern von Benutzerinformationen findet allein beim Mobilfunkbetreiber statt. Das Erzeugen und Anpassen der angebotenen Dienste gehört zu den Aufgaben des Dienstanbieters und ist nicht Teil der COMPASS-Architektur.

4.2.5 SAHARA

SAHARA (Service Architecture for Heterogeneous Access, Resources and Applications) [RB 02] ist eine Architektur, die entwickelt wurde, um die Komposition von Diensten verschiedener Dienstanbieter zu ermöglichen. Ein Hintergedanke dabei war, dass ein Dienstanbieter nicht alle gewünschten Dienste selbst zur Verfügung stellen muss, sondern diese durch Kombination seiner Dienste mit Diensten anderer Anbieter erstellen kann. SAHARA erlaubt Dienstanbietern, ihre Dienstinstanzen für andere verfügbar zu machen, bietet die technische Unterstützung, die nötig ist, um die verfügbaren Dienstinstanzen zu kombinieren. SAHARA stellt Funktionen zum Management der gesamten Dienstinstanzen zur Verfügung, bietet Authentifizierungs- und Abrechnungsmechanismen und führt Protokoll- und Formatanpassungen durch. Zur Komposition der Dienste können zwei unterschiedliche Modelle angewendet werden, entweder das kooperative Modell oder das Vermittler-basierte Modell. Beim kooperativen Modell interagieren die Dienstanbieter in verteilter Art und Weise, um einen zusammengesetzten Dienst anzubieten, jeder Anbieter trägt dabei einen Teil der Verantwortung. Beim Vermittler-basierten Modell nutzt ein einzelner Anbieter, der Vermittler, die Funktionalitäten, die ihm untergeordnete Dienstanbieter zur Verfügung stellen, und setzt daraus einen komplexen Dienst zusammen. Hier trägt der Vermittler die ganze Verantwortung. Welches Modell eingesetzt wird, hängt von der Komposition und dem Verhältnis, das die beteiligten Dienstanbieter zueinander haben, ab.

SAHARA macht durch die Komposition von Diensten eine schnelle und flexible Erzeugung von neuen Diensten möglich. Da jeder Dienstanbieter Angaben über die Verfügbarkeit seiner Dienstinstanzen machen muss und bei Bedarf eine ausgefallene Instanz jederzeit durch eine andere ersetzt werden kann, kann insgesamt eine hohe Verfügbarkeit gewährleistet werden. Allerdings gibt es keine explizite Unterstützung für den Short Message Service, für ereignisgesteuertes Senden und auch nicht für die Verwaltung von Benutzerinformationen.

4.2.6 Web Services

Web Services ist eine relativ neue Technologie, die eingesetzt wird, um eine vollautomatische Dienstnutzung und Dienstkomposition der im Internet angebotenen Dienste zu ermöglichen. Viele Softwarehersteller haben mittlerweile entsprechende Produkte auf den Markt gebracht oder entwickeln gerade ihre eigenen Lösungen. Bekannte Beispiele hierfür sind HP Web Services Platform, BEA Weblogic, IBM WebSphere und Microsoft .NET. Unter dem Begriff Web Service versteht man im Allgemeinen einen Dienst, der Benutzern über das Web zur Verfügung gestellt wird und dabei auf XML-basierte

Standards und HTTP zurückgreift. Web Services sind auf die automatisierte Benutzung ausgerichtet, sie sollen in einer standardisierten Weise genutzt und kombiniert werden können unabhängig vom Betriebssystem, der Programmiersprache, in der sie entwickelt worden sind, und der Web Service Engine, die sie in einem Netz verfügbar macht. Um dies zu gewährleisten, werden verschiedene Standards wie SOAP (Simple Object Access Protocol), UDDI (Universal Description, Discovery and Integration), WS-Inspection (Web Services Inspection Language) und WSDL (Web Services Description Language), eingesetzt. SOAP ist ein Kommunikationsprotokoll für verteilte Anwendungen, das mit Hilfe von XML den Austausch strukturierter und typisierter Daten ermöglicht. UDDI ist ein Verzeichnis für Dienstinformationen, das die Dienstinformationen in einer einheitlichen Datenstruktur speichert und an zentralen Stellen im Internet für Dienstnutzer zugänglich macht. Zudem bietet UDDI eine normierte Anfragesprache zur Abfrage der Dienstinformationen. Über WS-Inspection-Dokumente können Dienstanbieter ihre Dienste auf ihrem Web-Server in standardisierter Form anbieten - alternativ oder zusätzlich zur Speicherung in einem UDDI-Verzeichnis. Und mit WSDL lassen sich die zum Aufruf eines Dienstes notwendigen Informationen in einem XML-Dokument spezifizieren. Die Komposition von neuen Diensten aus bestehenden Diensten wird durch spezielle Sprachen, wie beispielsweise WSFL (Web Services Flow Language), vereinfacht.

Web Services unterstützen eine schnelle und einfache Erzeugung von neuen Diensten. Bei allen eingesetzten Standards waren sowohl Erweiterbarkeit als auch Verfügbarkeit wichtige Kriterien. Im UDDI-Verzeichnis werden zwar Daten über die Dienstanbieter gespeichert, aber eine Möglichkeit zum Verwalten von Benutzerinformationen ist nicht direkt gegeben. Außerdem wird weder der Empfang und Versand von Kurznachrichten noch ereignisgesteuertes Senden von dieser Technologie unterstützt.

4.2.7 Intelligente Netze

Unter einem intelligenten Netz (IN) versteht man weniger ein Netz, sondern eher einen architektonischen Ansatz zur Erweiterung traditioneller Telekommunikationsnetze um bestimmte Leistungen (Mehrwertdienste), die über die reine Informationsübertragung hinausgehen. Beispiele für Mehrwertdienste sind Gesprächsweiterleitung, Gesprächskostenaufteilung und Gebührenabrechnung für Informationsdienste. Die Dienste werden durch bestimmte Dienstkennzahlen (0800, 0900 usw.) aktiviert. Das Konzept des IN sieht vor, dass die zusätzlichen Leistungen eines Netzes in "Intelligenten Netzknoten" konzentriert werden. Beim IN werden also die Dienstfunktionen von den reinen Übermittlungsfunktionen getrennt. So müssen Dienste, die netzweit umzusetzen und zu überprüfen sind, nicht je Netzknoten realisiert, sondern können zentral für das Gesamtnetz erbracht werden. Die IN-Architektur stellt dafür drei verschiedene Komponenten bereit: Service Switching Point (SSP), Service Control Point (SCP) und Service Management Point (SMP). Der SSP ist eine Vermittlungsstelle im Telekommunikationsnetz und hat die Aufgabe, den Aufruf eines IN-Dienstes zu erkennen und vom entsprechenden SCP die nötigen Informationen zur Weiterleitung zu holen. Der SCP ist ein Dienststeuerungspunkt mit einer großen Datenbank für die Verwaltung der Netzpunkte, der eine leistungsfähige Anfragebearbeitung bietet. Durch den SMP werden die IN-Dienste des SCP eingerichtet, modifiziert, verwaltet und überwacht. Außerdem erfolgt hier die Einrichtung von Dienstkunden und Benutzerprofilen sowie die Erfassung von Daten für Statistiken.

Zur Standardisierung des IN-Ansatzes wurde ein dienst- und netzunabhängiges so genanntes Intelligent Network Conceptual Model (INCM) definiert. Dieses Modell besteht aus vier Ebenen:

- Dienstebene

- Global-funktionale Ebene
- Verteilt-funktionale Ebene
- Physikalische Ebene.

Die beiden oberen Ebenen sind zur Entwicklung und Einführung von IN-Diensten, die beiden unteren Ebenen konzentrieren sich auf die IN-Architektur selbst. Die Dienstebene bietet, wie der Name schon sagt, eine ausschließlich dienstorientierte Sicht. Hier werden die Dienste in Dienstmerkmale, so genannte Features, zerlegt, wobei sich ein Dienst aus einem oder mehreren Dienstmerkmalen zusammensetzen kann. In der Global-funktionalen Ebene wird ein IN als einzige steuerbare Einheit betrachtet. Diese Ebene enthält dienstunabhängige Bausteine (Service Independent Building Blocks, SIBs), die zur Dienstgenerierung dienen. Diese SIBs sind wiederverwendbar und können zwecks Diensterbringung miteinander verkettet werden. Die Verteilt-funktionale Ebene bildet eine verteilte Sicht auf ein IN mit Hilfe von Funktionalen Einheiten (Functional Entities, FEs). Die FEs stellen Funktionen der realen Bausteine der Physikalischen Ebene dar, z.B. Service Swiching Function (SSF) oder Service Control Function (SCF). Zwischen den FEs finden Informationsflüsse statt; dies ist nötig, falls mehrere FEs für einen SIB kooperieren müssen. Die Physikalische Ebene modelliert die verschiedenen physikalischen Einheiten eines IN-strukturierten Netzes mit ihren Schnittstellen und definiert die Abbildung der FEs auf diese physikalischen Einheiten.

Die Diensterbringung im Intelligenten Netz läuft im Grunde ähnlich wie bei einer SMS-Plattform ab. In beiden Fällen werden die Dienste durch die Wahl einer bestimmten Nummer aktiviert und je nach Inhalt der Kurznachricht bzw. darauffolgender Rufnummer erfolgt die Dienstausführung durch die entsprechende Applikation bzw. Dienststelle. Das Konzept des INCM unterstützt eine flexible und einfache Dienstentwicklung und -anpassung. Weiterhin bieten IN-Systeme die Möglichkeit, Benutzerprofile einzurichten, und müssen sehr hohen Verfügbarkeitsansprüchen gerecht werden, da sich Ausfälle meist auf das gesamte Netz auswirken und in manchen Fällen beträchtliche Umsatzeinbußen nach sich ziehen. SMS-Unterstützung oder ereignisgesteuertes Senden sind Anforderungen, die für die hier beschriebene Architektur irrelevant sind.

4.3 Zusammenfassung

Tabelle 4.1 listet alle Architekturen und Ansätze noch einmal auf und gibt einen Überblick darüber, welche Anforderungen im einzelnen erfüllt werden und welche nicht.

Erklärungen zur Tabelle:

SMS = SMS-Unterstützung, HV = Hohe Verfügbarkeit, EDG = Einfache Dienstgenerierung, EDA = Einfache Dienstanpassung, ES = Ereignisgesteuertes Senden, VBI = Verwaltung von Benutzerinformationen

ja = Anforderung wird erfüllt, nein = Anforderung wird nicht erfüllt, part = Anforderung wird teilweise erfüllt, kA = dazu wurden keine Angaben gemacht

System	SMS	HV	EDG	EDA	ES	VBI
Staff Text Message Service	part	kA	nein	nein	nein	part
iMobile	ja	ja	nein	ja	nein	ja
OPC-SMS	ja	kA	part	part	ja	nein
COMPASS	ja	kA	nein	nein	ja	ja
SAHARA	nein	ja	ja	nein	nein	nein
Web Services	nein	ja	ja	ja	nein	nein
Intelligente Netze	nein	ja	ja	ja	nein	ja

Tabelle 4.1: Erfüllung der Anforderungen in bestehenden Architekturen und Ansätzen

Kapitel 5

Implementierungskonzept

In diesem Kapitel werden Lösungsansätze für die in Abschnitt 3.3 aufgeführten Problembereiche vorgestellt. Bei der Lösungsfindung lag das Hauptaugenmerk darauf, den sich aus den Problemen ergebenden Anforderungen an eine Architektur für SMS-basierte Dienste (siehe Abschnitt 4.1) gerecht zu werden.

Zunächst werden in Abschnitt 5.1 Basiskomponenten, die den Aufwand bei der Diensterzeugung und der Dienstanpassung verringern sollen, beschrieben. Daran anschließend werden in Abschnitt 5.2 zwei komplexe Dienste aus den Komponenten modelliert. In Abschnitt 5.3 wird schließlich auf das entworfene Konzept für eine Regel-Engine, die das ereignisbasierte Senden von Push-Nachrichten unterstützt, näher eingegangen.

5.1 Modularisierung - Modellierung der Komponenten

Unter dem Begriff Modularisierung versteht man generell, Funktionen, die allgemein und übergreifend genutzt werden, aus den spezifischen Zusammenhängen herauszulösen und in separate Module zu packen. Somit wird Code für immer wieder gleiche Aufgaben wirklich nur einmal geschrieben und an Stellen, wo er benötigt wird, einfach eingesetzt. Eine Modularisierung bringt verschiedene Vorteile mit sich:

- Kleine Komponenten sind leichter überschaubar als komplexe, unübersichtliche Anwendungen
- Änderungen an der internen Struktur einer Komponente oder Erweiterungen dieser wirken sich nicht auf das Gesamtsystem aus
- Durch Wiederverwendung der vorhandenen Komponenten verringert sich der Entwicklungsaufwand und neue Dienste lassen sich schneller erstellen

Damit die Erzeugung von neuen Diensten und die Anpassung von bereits realisierten Diensten möglichst einfach und ohne großen Zeitaufwand erfolgen kann, sollten also Teilfunktionen, die häufig verwendet werden, extrahiert und in generischer Form zur Verfügung gestellt werden. Jede der zur Verfügung gestellten, generischen Basiskomponenten realisiert dann eine abgeschlossene Funktionalität und kann in jeder beliebigen Applikation eingesetzt werden.

Zunächst wird mit Hilfe der bereits realisierten Dienste bestimmt, welche Teilfunktionen als Basiskomponenten zur Verfügung gestellt werden sollen. In den darauf folgenden Abschnitten werden die einzelnen Komponenten und ihre Funktionalitäten genauer beschrieben.

5.1.1 Bestimmung der sich wiederholenden Teilfunktionen

Um festzulegen, welche Komponenten für eine Wiederverwendung geeignet sind und somit nicht neu entwickelt werden müssen, werden die bereits realisierten Applikationen auf sich wiederholende Teilfunktionen untersucht.

In jeder Applikation wird - oft auch mehrmals - geprüft, ob der Text der Kurznachricht ein bestimmtes Wort oder eine bestimmte Zeichenfolge enthält. Schon allein die Häufigkeit spricht für eine Auslagerung dieser Funktion in eine generische Komponente. Eine weitere Funktion, die des öfteren vorkommt, ist die Zerlegung des Kurznachrichtentextes in einzelne Wörter bzw. Zeichenfolgen. Für die Zerlegung können beliebige Zeichen, z.B. Leerzeichen, Punkte, Kommas oder eine ihrer Kombinationen, als Separatoren fungieren. Mit der Zerlegung eng verknüpft ist das Herausfiltern einer im Text enthaltenen E-Mail-Adresse. Dazu wird geprüft, ob die Struktur eines Textteils dem Aufbau einer E-Mail-Adresse entspricht, und dieser Teil anschließend zurückgegeben.

Neben den eben genannten Funktionen, die vorwiegend die Textbearbeitung betreffen, ist es häufig nötig, einem Teilnehmer eine entsprechende Antwort oder Information zukommen zu lassen. Dies kann entweder durch das Senden einer Kurznachricht oder einer E-Mail erfolgen. Beim E-Mail-Versand wurden verschiedene Formen verwendet: nur Text, Text mit Anhang und als HTML-Seite. Hier bietet es sich ebenfalls an, diese Teilfunktionen in generische Komponenten auszulagern.

Ein weiterer Funktionsbereich, der in vielen Applikationen benötigt wird, ist der Zugriff auf eine Tabelle in der Datenbank. Dabei kommen verschiedene Datenbankzugriffe zum Einsatz: Stellen von Anfragen, Ändern, Einfügen und Löschen. Damit man sich nicht jedes Mal um eine Datenbank-Verbindung kümmern und die Syntax der einzelnen Operationen nachschlagen muss, ist es sinnvoll, alle Datenbank-Operationen in allgemeiner Form zur Verfügung zu stellen.

Der letzte Funktionsbereich, der unabhängig von einer Applikation bereitgestellt werden sollte, umfasst eine Reihe von Zeit- bzw. Datumsrückgaben. Die darin enthaltenen Funktionen kommen zwar in den bisher realisierten Applikationen nicht so oft vor, werden aber im Hinblick auf Push-Dienste, die auf einem Zeitereignis beruhen, sicher von Nutzen sein.

5.1.2 Das Textbearbeitungs-Modul

In diesem Modul sind alle Operationen, die die Textbearbeitung betreffen, zusammengefasst. Da zur Ausführung einer Applikation immer die gerade eingegangene Kurznachricht übergeben wird, wird pro Applikationsdurchlauf genau eine Kurznachricht verarbeitet. Daher ist dem Modul für die Textbearbeitung auch immer der Text der gerade zu verarbeitenden Kurznachricht zugeordnet. Repräsentiert wird der Text durch das Attribut *message*.

Die funktionalen Komponenten, die dieses Modul bietet, werden im Folgenden vorgestellt:

`isInMessage()`
Mit dieser Funktion wird geprüft, ob im Text der Kurznachricht die übergebene Zeichenkette enthalten

ist. Dazu wird das Attribut *message* nach der übergebenen Zeichenkette durchsucht. Der Suchbereich kann auch auf einen Teil des *message*-Attributs beschränkt werden, wenn beispielsweise nur von Interesse ist, ob die angegebene Zeichenkette in den ersten zehn Zeichen vorkommt.

`partsOfMessage()`
Diese funktionale Komponente zerlegt den Text der Kurznachricht, also das *message*-Attribut, in einzelne Wörter bzw. Zeichenfolgen und liefert diese zurück. Als Trennpunkt kann prinzipiell jedes beliebige Zeichen fungieren, in den meisten Fällen aber wird nach Leerzeichen, Punkten, Kommas bzw. einer Kombination dieser Zeichen separiert. Nach der Separation wird eine Liste der einzelnen Textteile zurückgegeben.

`getEmailAddress()`
Diese Funktion dient dazu, aus einer Menge von Zeichenfolgen diejenige zurückzuliefern, die eine E-Mail-Adresse darstellt. Dazu muss die Struktur jeder übergebenen Zeichenfolge mit der Syntax einer E-Mail-Adresse verglichen werden. Fällt bei einer der Zeichenfolgen dieser Vergleich positiv aus, wird die Zeichenfolge ausgegeben. Dabei ist zu beachten, dass die zurückgelieferte Zeichenfolge nur dem Aufbau nach einer E-Mail-Adresse entspricht, ob die E-Mail-Adresse tatsächlich existiert, wird nicht überprüft.

5.1.3 Das Versand-Modul

Das Versand-Modul beinhaltet alle funktionalen Komponenten, die für das Versenden von Informationen, Antworten oder sonstigen Inhalten verantwortlich sind. Das Senden kann auf zwei verschiedene Arten erfolgen: per SMS oder per E-Mail.

`sendSMS()`
Diese Komponente ist für das Senden einer Kurznachricht zuständig. Dabei können alle variierenden Angaben - Empfänger, Absender und Kurznachrichtentext - vor jedem Senden neu gesetzt werden. Für den Empfänger ist eine Telefonnummer, egal ob internationales Format oder nicht, anzugeben, für den Absender ein Name oder eine Kurzwahlnummer und der Kurznachrichtentext als eine Folge von höchstens 160 Zeichen.

`sendEmail()`
Mit Hilfe dieser funktionalen Komponente ist der Versand von E-Mails möglich. Die Komponente unterstützt praktisch jede Art von E-Mail: E-Mail mit reinem Text oder als HTML-Seite und mit oder ohne Anhang. Je nachdem, ob die E-Mail nur Text oder eine HTML-Seite anzeigen soll, wird entweder der Text oder der HTML-Code übergeben. Für den Anhang wird eine Liste der Dateien, die angehängt werden sollen, übergeben. Falls es sich um eine E-Mail ohne Anhang handelt, kann diese Liste auch leer sein. Des weiteren sind die E-Mail-Adresse des Empfängers, die - möglicherweise fiktive - E-Mail-Adresse des Absenders, und der Betreff der E-Mail anzugeben.

5.1.4 Das Datenbankzugriffs-Modul

Das in diesem Abschnitt beschriebene Modul vereinigt alle funktionalen Komponenten, die einen Zugriff auf die MySQL-Datenbank ermöglichen. Bevor auf die Datenbank zugegriffen werden kann, muss allerdings eine Verbindung zu ihr aufgebaut werden. Pro Applikationsdurchlauf sollte dies genau einmal erfolgen. Das Attribut *conn* verweist dann auf die bestehende Datenbankverbindung.

Anfragen an die und Operationen auf der relationalen Datenbank sind in der Datenbanksprache MySQL zu formulieren. Die einzelnen dafür zuständigen funktionalen Komponenten werden nun nacheinander vorgestellt:

`select()`
Diese Komponente führt beliebige Datenbankanfragen durch. Die Anfragen sind typischerweise aus einer SELECT-Klausel, einer FROM-Klausel und einer optionalen WHERE-Klausel zusammengesetzt. Für eine Anfrage sind folgende Angaben zu machen: Das in die SELECT-Klausel einzusetzende Attribut, die in die FROM-Klausel einzusetzende Tabelle und die gesamte WHERE-Klausel. So kann jedes Mal das gewünschte Attribut, die Tabelle, auf die zugegriffen werden soll, und die Bedingung, die für die Selektion erfüllt sein muss, spezifiziert werden. Zurückgegeben werden alle Attributwerte, die in der Ergebnismenge der Anfrage enthalten sind.

`selectDistinct()`
Diese Komponente leistet dasselbe wie die eben beschriebene Komponente mit dem Unterschied, dass alle Duplikate vor der Rückgabe der Ergebnismenge automatisch entfernt werden.

`insert()`
Diese Komponente fügt einen neuen Datensatz in eine Tabelle der Datenbank ein. Jedes Einfügen besteht aus einem INSERT-Teil und einem VALUES-Teil. Daher ist zum einen die Tabelle, in die eingefügt werden soll, und zum anderen eine Liste der Attributwerte, die in die Tabelle eingefügt werden sollen, zu übergeben. Die Reihenfolge der Attributwerte in der Liste sollte der Attributreihenfolge in der Tabelle entsprechen, da die einzelnen Attributwerte der Reihe nach in die Spalten der entsprechenden Tabelle geschrieben werden.

`update()`
Mit dieser Komponente werden Änderungen in einer Tabelle der Datenbank durchgeführt. Eine Änderung setzt sich aus drei Teilen zusammen: UPDATE-, SET- und WHERE-Klausel. Deshalb sind bei einer Änderung die Tabelle, die geändert werden soll, die gesamte SET-Klausel und die gesamte WHERE-Klausel anzugeben. In der SET-Klausel werden alle Attribute mit den Werten, die ihnen neu zugewiesen werden sollen, aufgeführt und die WHERE-Klausel gibt, wie gehabt, die zu erfüllende Bedingung an.

`delete()`
Diese letzte Komponente ist für das Löschen von Datensätzen aus einer Tabelle der Datenbank zuständig. Beim Löschen ist anzugeben, aus welcher Tabelle gelöscht werden soll (DELETE-Teil), und welche Bedingung auf den zu löschenden Datensatz bzw. die zu löschenden Datensätze zutreffen soll (WHERE-Teil). Für den Fall, dass alle Datensätze aus der Tabelle gelöscht werden sollen, kann der WHERE-Teil einfach weggelassen werden.

5.1.5 Das Zeit-Modul

Das Zeit-Modul stellt alle nötigen Zeitfunktionen zur Verfügung. Dafür ist ein Objekt, das die aktuellen Zeit- und Datumsangaben bereitstellt, erforderlich. Das Attribut calendar verweist auf eben dieses Objekt.

Die einzelnen funktionalen Komponenten dieses Moduls sind:

`getMinute/Hour/Day/Month()`

Diese Komponenten liefern die aktuelle Minute, die aktuelle Stunde, den aktuellen Tag im Monat und den aktuellen Monat als ganze Zahl zurück.

`getDayOfWeek()`
Mit Hilfe dieser Komponente wird der aktuelle Tag der Woche in eine Wochentagsbezeichnung umgewandelt. Zurückgeliefert wird der aktuelle Tag als Mo, Di, Mi, Do, Fr, Sa oder So.

5.2 Komposition - Modellierung komplexer Dienste

Ein wesentlicher Aspekt bei der Modularisierung ist die Wiederverwendung der verschiedenen Komponenten für die Erzeugung neuer Dienste. In diesem Abschnitt geht es darum, komplexe Dienste aus bereits existierenden Komponenten zusammenzusetzen. Als Grundlage dafür dienen die im letzten Abschnitt beschriebenen Komponenten. Dazu werden zwei komplexe Dienste modelliert, die sich aus einer Kombination dieser Komponenten ergeben. Beiden Diensten müssen durch das Senden einer entsprechenden Kurznachricht angestoßen werden.

5.2.1 Dienst 1: Versand von angeforderten PDF-Dateien

Der Dienst, der in diesem Abschnitt modelliert wird, ist für den Versand einer E-Mail mit zwei PDF-Dateien im Anhang verantwortlich. Damit ein mobiler Teilnehmer diesen Dienst nutzen kann, muss er eine Kurznachricht mit einem vorgegebenen Keyword und seiner E-Mail-Adresse an die SMS-Plattform senden. Die Kurznachricht wird dann dem Keyword entsprechend verarbeitet, d.h. der Dispatcher holt die dem Keyword zugeordnete Applikation und stößt die Applikation mit Übergabe der Kurznachricht an (siehe Kapitel 3). Die Applikation realisiert dann den eigentlichen Dienst.

Der Dienst sieht vor, dass jedem Dienstnutzer nach Eingang seiner Kurznachricht eine E-Mail mit den beiden PDF-Dateien gesendet wird. Das ist natürlich nur möglich, wenn in der empfangenen Kurznachricht eine E-Mail-Adresse enthalten ist. Ist dies der Fall, wird die E-Mail-Adresse extrahiert und eine E-Mail mit einem kurzen Text und den PDF-Dateien im Anhang an diese Adresse gesendet. Wird keine E-Mail-Adresse gefunden, kann auch keine E-Mail versendet werden und der Dienst ist zu Ende.

Der Dienstablauf sieht wie folgt aus:

1. Der Text der gesendeten Kurznachricht wird in einzelne Wörter oder auch Zeichenfolgen zerlegt. Da die Kurznachricht eigentlich nur das Keyword und die E-Mail-Adresse enthalten sollte und diese im Normalfall durch ein Leerzeichen oder Komma getrennt angegeben werden, ist die Zerlegung anhand von Leerzeichen und Kommas durchzuführen. Punkte kommen hier nicht in Frage, da sonst die E-Mail-Adresse zerlegt werden würde.

2. Die bei der Zerlegung entstandenen Textteile werden der Reihe nach auf Übereinstimmung mit der Syntax einer E-Mail-Adresse geprüft. Stimmt einer der Textteile mit der Syntax einer E-Mail-Adresse überein, wird dieser zurückgeliefert.

3. Falls eine E-Mail-Adresse gefunden wurde, wird eine E-Mail mit einem vorgegebenen Text als Inhalt, den PDF-Dateien im Anhang und einem ebenfalls vorgegebenen Betreff erzeugt und an diese Adresse gesendet.

Für jeden dieser Schritte kann eine der Komponenten, die in Abschnitt 5.1 definiert wurden, eingesetzt werden:

1. Zerlegung des Textes ≫ `partsOfMessage()` aus dem Textbearbeitungs-Modul
2. Extrahieren der E-Mail-Adresse ≫ `getEmailAddress()` aus dem Textbearbeitungs-Modul
3. Erzeugen und Senden der E-Mail ≫ `sendEmail()` aus dem Versand-Modul

Abbildung 5.1 zeigt ein Modell des Dienstablaufs mit den einzelnen Komponenten.

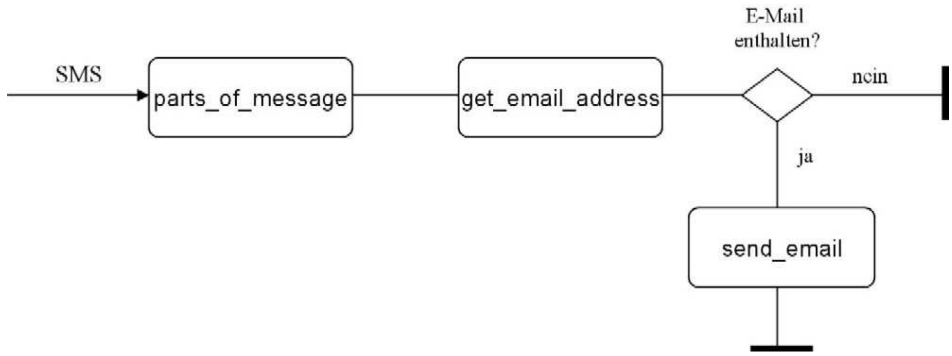


Abbildung 5.1: Komponenten des 1. Dienstes

5.2.2 Dienst 2: Abonnement von Horoskop-Sprüchen

Der hier modellierte Dienst bietet mobilen Teilnehmern die Möglichkeit, sich für den Erhalt eines täglichen Horoskop-Spruchs zu registrieren. Dazu muss der mobile Teilnehmer eine Kurznachricht mit einem vorgegebenen Keyword und seinem Sternzeichen an die SMS-Plattform senden. Die Kurznachricht wird, wie immer, dem Keyword entsprechend verarbeitet und der Applikation, die diesen Dienst realisiert, übergeben (siehe Kapitel 3).

Der Dienst sieht vor, dass jeder noch nicht registrierte Dienstnutzer nach Eingang seiner Kurznachricht in der Datenbanktabelle *HoroskopAbonnenten* registriert wird und anschließend eine Kurznachricht mit seinem heutigen Horoskop an ihn gesendet wird. Durch die Registrierung erhält der Dienstnutzer eine Woche lang täglich sein Horoskop. Ist der Dienstnutzer bereits registriert, so wird sein Abonnement um eine Woche verlängert und er erhält eine Kurznachricht, die ihn darüber informiert. Kann in der Kurznachricht kein Sternzeichen gefunden werden, so ist weder eine Registrierung noch das Senden eines Horoskop-Spruchs möglich. Auch in diesem Fall ist der Dienstnutzer per SMS darüber zu informieren.

Für diesen Dienst müssen in der Datenbank die drei Tabellen *HoroskopAbonnenten*, *Horoskope* und *Wochenhoroskop* angelegt werden. In der *HoroskopAbonnenten*-Tabelle werden die Abonnenten des Horoskopdienstes verwaltet. Dazu können folgende Informationen über den Abonnenten gespeichert werden: Seine Telefonnummer, sein Sternzeichen, die Anzahl der Sprüche, die er noch bekommt und die Nummer seiner Kurznachricht in der *InboundSMS*-Tabelle (siehe Kapitel 3). Die *Horoskope*-Tabelle enthält alle Horoskop-Sprüche, die automatisch durchnummeriert werden. Dadurch ist jedem Spruch eine eindeutige Nummer zugeordnet. In der *Wochenhoroskop*-Tabelle werden die Nummern der Sprüche, die jedes Sternzeichen in der aktuellen Woche erhält, gehalten. Über die Nummern können die entsprechenden Horoskop-Sprüche aus der *Horoskope*-Tabelle geholt werden. In der Datenbank haben diese Relationen folgende Struktur:

- **`HoroskopAbonnenten:`**
  ```
  id int primary key,
  abonnent varchar(20),
  sternzeichen varchar(20),
  credits int,
  inboundSMSid long references InboundSMS,
  status int
  ```
- **`Horoskope:`**
  ```
  id int primary key,
  spruch varchar(160)
  ```
- **`Wochenhoroskop:`**
  ```
  wochentag varchar(8) primary key,
  steinbock int,
  wassermann int,
  fisch int,
  widder int,
  stier int,
  zwilling int,
  krebs int,
  löwe int,
  jungfrau int,
  waage int,
  skorpion int,
  schütze int
  ```

Die einzelnen Schritte der Diensterbringung sind:

1. Im Text der Kurznachricht wird nach dem enthaltenen Sternzeichen gesucht.
2. Falls kein Sternzeichen gefunden wird, wird an den Dienstnutzer eine Kurznachricht gesendet mit dem Hinweis darauf, dass das Sternzeichen nicht lesbar war.
3. Ist ein Sternzeichen in der Nachricht enthalten, so wird überprüft, ob der Sender der Nachricht bereits in der *HoroskopAbonnenten*-Tabelle registriert ist oder nicht.
4. Ist der Sender der Kurznachricht schon als Abonnent registriert, so wird ihm eine Kurznachricht mit dem Inhalt, dass sein Horoskop-Abonnement um eine Woche verlängert wird, zugesendet und sein Guthaben um 7 erhöht.
5. Falls der Sender noch nicht registriert ist, wird ein neuer Datensatz mit seinen Daten in die *HoroskopAbonnenten*-Tabelle eingefügt, d.h. seine Telefonnummer, sein Sternzeichen, ein Guthaben von 7 Sprüchen, die Nummer seiner Kurznachricht in der *InboundSMS*-Tabelle und der Status 0.
6. Anschließend wird der zum Sternzeichen des Abonnenten passende Horoskop-Spruch des aktuellen Tages mit Hilfe der *Horoskope*- und *Wochenhoroskop*-Tabelle geholt und an den Abonnenten per SMS gesendet. Nach dem Senden wird sein Guthaben um 1 herabgesetzt.

Auch bei diesem Dienst lassen sich die einzelnen Schritte mit Hilfe Komponenten aus Abschnitt 5.1 realisieren:

1. Suche nach enthaltenem Sternzeichen ≫ `isInMessage()` aus dem Textbearbeitungs-Modul
2. Senden der Nachricht "Sternzeichen fehlt" ≫ `sendSMS()` aus dem Versand-Modul
3. Prüfen, ob Sender registriert ist ≫ `select()` aus dem Datenbankzugriffs-Modul
4. Senden der Nachricht "Abo verlängert"≫ `sendSMS()` aus dem Versand-Modul; Erhöhen des Guthabens ≫ `update()` aus dem Datenbankzugriffs-Modul
5. Einfügen des Abonnenten ≫ `insert()` aus dem Datenbankzugriffs-Modul
6. Senden des Horoskop-Spruchs ≫ `sendSMS()` aus dem Versand-Modul; Reduzieren des Guthabens ≫ `update()` aus dem Datenbankzugriffs-Modul

In Abbildung 5.2 wird der Ablauf dieses Dienstes mit den beteiligten Komponenten modellhaft dargestellt.

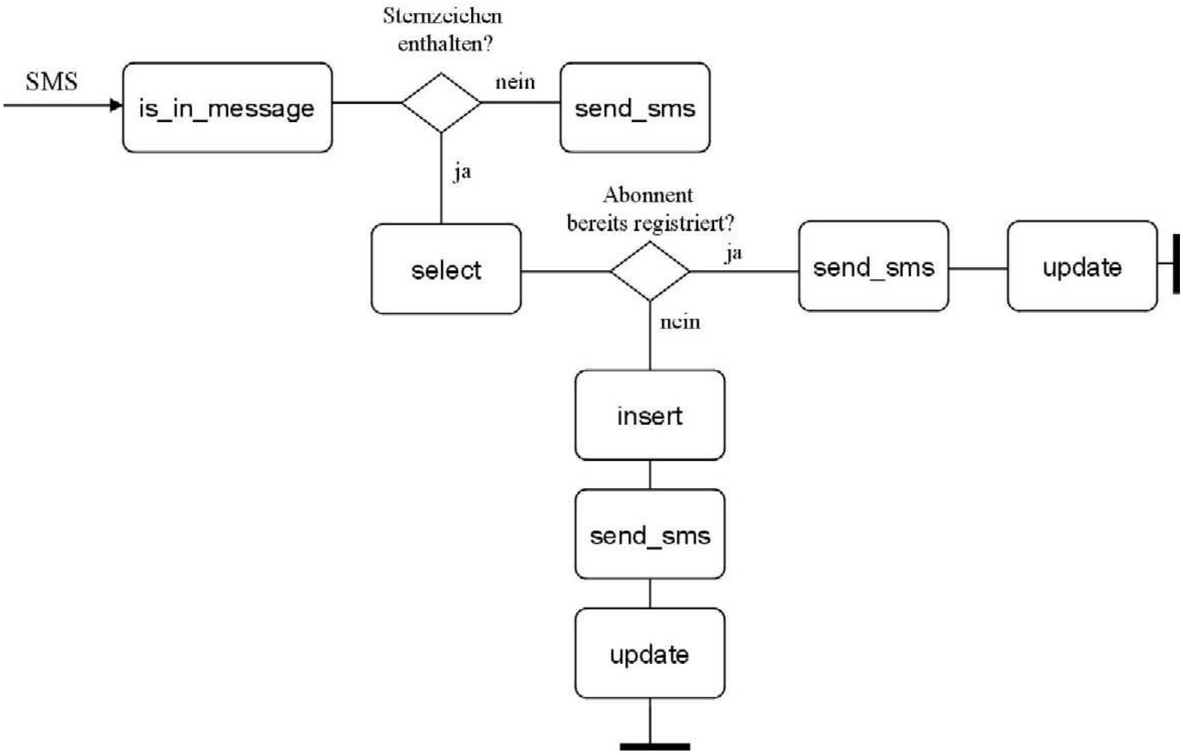


Abbildung 5.2: Komponenten des 2. Dienstes

5.3 Entwurf der Regel-Engine

In Abschnitt 3.3 wurde als ein großer Nachteil der SMS-Plattform aufgeführt, dass kein Mechanismus existiert, der ein automatisches Senden von Kurznachrichten, ausgelöst durch den Eintritt eines bestimmten Ereignisses, ermöglicht. Im weiteren wird ein Konzept vorgestellt, das die Realisierung von Push-Diensten, insbesondere das ereignisgesteuerte Senden von Kurznachrichten, unterstützt.

Grundlage dafür ist eine Datenbasis, die Informationen über die Interessen und Vorlieben von mobilen Teilnehmern und damit potentiellen Empfängern einer Push-Nachricht bereitstellt. In dem hier vorgestellten Konzept werden die Informationen über die mobilen Teilnehmer automatisch, also ohne deren Zutun gewonnen. Die gewonnenen Informationen werden als Interessenprofile in der Datenbank abgespeichert. Im ersten Abschnitt dieses Unterkapitels wird detaillierter auf das Sammeln der Benutzerinformationen, die Erstellung der Interessenprofile und ihre Verwaltung eingegangen.

Der nächste Schritt ist das Definieren von Regeln. Die Regeln dienen dazu, festzulegen, bei welchem Ereignis welche Aktion ausgelöst wird. Mit einer Regel kann also spezifiziert werden, welche Bedingung erfüllt sein muss bzw. Bedingungen erfüllt sein müssen, damit das Senden einer Push-Nachricht ausgelöst wird. Die Definition des Ereignisses ist praktisch nichts anderes als die Angabe einer oder mehrerer Bedingungen, die erfüllt sein müssen, bevor die Aktion ausgeführt wird. Bei der Aktion handelt es sich in den meisten Fällen nicht um eine einzige Aktion, sondern um eine Reihe von Aktionen, die nacheinander ausgeführt werden. Deshalb wird jede Aktion durch ein eigenes Objekt realisiert. In Abschnitt 5.3.2 wird genauer beschrieben, wie die Regeln in diesem Konzept definiert werden und auf welche Weise sie in der Datenbank gespeichert werden.

Für den Zugriff und die Verwaltung der Regeln ist die *Rulebase* zuständig. Die *Rulebase* interagiert mit der Datenbank und ermöglicht somit den Zugriff auf die Regeln bzw. auf die Ereignisse und Aktionen. Eine Spezifikation der *Rulebase* findet man in Abschnitt 5.3.3 .

Die Definition von Regeln allein reicht nicht aus, um Push-Dienste zu realisieren, da die Regeldefinition an sich noch keinen Automatismus bietet, der das Senden einer Push-Nachricht bei Eintritt eines Ereignisses veranlasst. Dazu ist ein eigenständiger Prozess nötig, der in periodischen Abständen überprüft, ob eines der Ereignisse eingetroffen ist. Für jedes Ereignis, bei dem die Prüfung positiv ausgefallen ist, muss die zugeordnete Aktion ausgeführt werden. Für die periodische Überprüfung der Ereignisse und das Anstoßen der zugeordneten Aktionen ist der *Eventhandler* verantwortlich. Dieser wird im vierten Abschnitt dieses Unterkapitels vorgestellt.

Schließlich wird im letzten Abschnitt das daraus resultierende Implementierungsmodell der Regel-Engine dargestellt.

5.3.1 Die Profil-Tabelle

Ein grundlegender Aspekt bei der Realisierung von Push-Diensten ist das Sammeln von Benutzerinformationen und die anschließende Aufbereitung der gesammelten Informationen in der Form, dass daraus ein Mehrwert für den Dienstnutzer abgeleitet werden kann. In Bezug auf SMS-basierte Dienste stehen vor allem die Interessen und Vorlieben der mobilen Teilnehmer im Vordergrund. Nur wenn man weiß, welche Dienste oder Aktionen für einen Teilnehmer sonst von Interesse sind, kann man auch abschätzen, welche Angebote oder Informationen für ihn von Nutzen sein könnten und welche nicht. Durch das Sammeln und sinnvolle Aufbereiten von Benutzerinformationen ist es also möglich, festzulegen, welche Push-Nachricht für einen Teilnehmer einen offensichtlichen Mehrwert schafft und welche eher das Gegenteil bewirkt.

Im Grunde gibt es zwei Möglichkeiten, um an Informationen über einen Benutzer heranzukommen. Entweder er gibt sie freiwillig an oder man findet sie heraus. Freiwillige Angaben haben den Vorteil, dass der Benutzer selbst Daten über sich bereitstellt. Der Nachteil bei freiwilligen Angaben ist allerdings, dass man die Echtheit der bereitgestellten Daten nur bedingt kontrollieren kann, was ihren Wert deutlich mindert. Außerdem ergibt sich hier sofort das Problem des Datenschutzes. Je persönlicher die gespeicherten Daten sind, desto restriktiver schränken die Datenschutzbestimmungen den Handlungsfreiraum ein.

Im Gegensatz dazu hat man bei automatisch gewonnenen Daten in der Regel keine Probleme mit dem Datenschutz, da die Daten meist anonym sind, d.h. an keine konkrete Person gekoppelt sind. An Informationen über die Interessen eines Teilnehmers kann man beispielsweise gelangen, indem man die Kommunikationsaktivitäten dieses Teilnehmers aufnimmt und in geeigneter Form speichert. Grundsätzlich muss sich hier die Frage gestellt werden, welche Daten dem Dienstanbieter wirklich etwas bringen und Aufschluss über die Interessen des Teilnehmer geben.

Für die SMS-Plattform ist die zweite Möglichkeit besser geeignet. Es besteht weder die Notwendigkeit, die gespeicherten Daten an eine konkrete Person zu koppeln, noch rechnet sich der Aufwand, die Teilnehmer direkt zu kontaktieren, um Informationen über ihre Interessen zu erfragen. Bei der SMS-Plattform bietet es sich nahezu an, die Interaktionen zwischen den mobilen Teilnehmern und der Plattform mitzuloggen und daraus Interessenprofile zu erstellen. Für die Realisierung von SMS-basierten Push-Diensten reicht es vollkommen aus, die Handynummer der Teilnehmer mitzuführen, die wahre Identität der Teilnehmer ist dabei eher nebensächlich.

Jede Interaktion eines Teilnehmers mit der SMS-Plattform wird in der *InboundSMS*-Tabelle der Datenbank festgehalten. Das bedeutet, für jede Interaktion wird die Telefonnummer des Absenders, die gewählte Kurzwahlnummer, das Datum des Eintreffens beim SMSC, der Netzbetreiber und die Nachricht an sich gespeichert. Mit Hilfe dieser Daten können Profile für die Teilnehmer erstellt werden. Beispielsweise lassen sich daraus Informationen über die Dienste, die der Teilnehmer genutzt hat, über den Anbieter, der den Dienst bereitgestellt hat, über die Häufigkeit der Dienstnutzung sowie über den Zeitpunkt, wann ein Dienst das letzte Mal genutzt worden ist, extrahieren.

Die automatisch gewonnenen Profile werden wieder in einer Tabelle der Datenbank abgespeichert. Diese Profil-Tabelle wird in der Datenbank durch folgende Relation repräsentiert:

```
Profile:
Teilnehmer varchar(20),
Applikation varchar(20),
Anbieter varchar(20),
Anzahl int,
Datum datetime,
Status int
```

Unter dem Attribut *Teilnehmer* wird die Handynummer des Teilnehmers in internationalem Format geführt. Das Attribut *Applikation* gibt an, welchen Dienst der Teilnehmer genutzt hat. Da die Dienste in der Regel durch Applikationen realisiert werden, ist der Name der Applikation abzuspeichern. Unter dem Attribut *Anbieter* wird angegeben, wer den Dienst angeboten hat. Wie oft der Dienst genutzt worden ist, ist dem Attribut *Anzahl* zugeordnet. Das Attribut *Datum* zeigt an, wann der Dienst das letzte Mal genutzt worden ist, und das *Status*-Attribut dient dazu, anzugeben, ob bereits eine Push-Nachricht gesendet worden ist oder nicht.

Da zu jedem Teilnehmer gespeichert wird, welchen Dienst bzw. welche Dienste er genutzt hat, kann abgeleitet werden, welche Aktionen für ihn von Interesse sind. Nimmt beispielsweise ein Teilnehmer sehr häufig an Preisrätseln, so kann er darauf aufmerksam gemacht werden, dass ein neues Rätselheft mit vielen Preisrätseln auf den Markt kommt. Oder für einen Teilnehmer, der regelmäßig Aktienkurse abfragt, ist es sicher interessant, über den plötzlichen Anstieg einer Aktie unmittelbar informiert zu werden.

Die Angabe des Anbieters mag zwar nicht unbedingt erforderlich sein, da die Zuordnung zwischen Dienst und Anbieter meist eindeutig ist, wurde aber eingefügt, um bei Anfragen den Zugriff auf weitere Tabellen zu vermeiden. Auch aus den Anbietern, deren Dienste ein Teilnehmer genutzt hat, lassen sich Vorlieben und Interessen des Teilnehmers ableiten. Nimmt beispielsweise ein Teilnehmer regelmäßig an Aktionen ein bestimmten Zeitung teil, so hat er vielleicht Interesse an dem Erscheinen einer neuen Zeitung in ähnlichem Stil.

Die Anzahl und das Datum dienen dazu, festzustellen, wie oft ein Teilnehmer einen bestimmten Dienst genutzt hat und wie lange die Nutzung zurückliegt. Falls ein Dienst einen Benutzer nur zu einer einmaligen Teilnahme bewegt hat, hat ihn der Dienst höchst wahrscheinlich nicht sehr überzeugt. Und wenn der Nutzungszeitpunkt schon lange zurückliegt, kann man davon ausgehen, dass der Teilnehmer an den Diensten des Anbieters kein Interesse mehr hat.

5.3.2 Regeldefinition

Auf der Profil-Tabelle aufbauend können Regeln erstellt werden, die das Senden von Push-Nachrichten steuern. Mit Hilfe der Regeln kann man definieren, bei welchem Ereignis welche Aktion ausgeführt wird. Eine Regeldefinition besteht also immer aus zwei Teilen, der Ereignisdefinition und der Zuweisung einer Aktion, die bei Eintritt des Ereignisses ausgeführt werden soll. Da eine Aktion meist mehrere Schritte beinhaltet, wird die Definition der Aktionen in gesonderte Objekte ausgelagert.

Ereignisdefinition

Eine Ereignisdefinition ist im Grunde nichts anderes als die Spezifikation eines booleschen Ausdrucks. Ein Ereignis kann das Erreichen eines bestimmten Zeitpunkts, das Eintreffen einer neuen Information oder das Eintreten einer bestimmten Situation sein. Beispielsweise könnte allen Teilnehmern, die regelmäßig eine bestimmte Zeitschrift lesen, zwei Tage vor Erscheinen einer Zeitschrift mit ähnlichem Leserkreis eine Kurznachricht gesendet werden, die sie über das Herauskommen informiert. In diesem Fall wäre als Ereignis zu definieren, dass das aktuelle Datum zwei Tage vor dem Erscheinungsdatum ist:

```
aktuellerMonat==erscheinungsMonat && aktuellerTag==erscheinungsTag-2
```

Um eine weitere Ereignisdefinition zu demonstrieren, wird das Beispiel mit den Börsenkursen wieder aufgenommen. Ein Teilnehmer, der täglich Börsenkurse abfragt, könnte per SMS darüber informiert werden, wenn eine Aktie einen bestimmten Grenzwert überschreitet oder unterschreitet. Die Ereignisdefinition dazu sähe wie folgt aus:

```
aktuellerIndex>obererGrenzwert || aktuellerIndex<untererGrenzwert
```

Ein Ereignis ist eingetreten, falls die Ereignisdefinition oder besser der boolesche Ausdruck den Wahrheitswert TRUE zurückliefert.

Aktionen

Da die Definition einer Aktion weitaus komplexer als die Definition eines Ereignisses ist, wird jede Aktion ein in einem *Action*-Objekt implementiert, das bei Bedarf angestoßen werden kann und die einzelnen Schritte der Aktion ausführt.

Die Aktion, die dem Ereignis aus dem ersten Beispiel im Abschnitt Ereignisdefinition zugeordnet ist, setzt sich beispielsweise aus folgenden Schritten zusammen: Zuerst müssen alle Teilnehmer, die eine ähnliche Zeitschrift lesen, aus der Profil-Tabelle geholt werden. Dann muss jedem dieser Teilnehmer die entsprechende Kurznachricht gesendet werden und schließlich muss noch der Status in der Profil-Tabelle geändert werden, damit er diese Nachricht nicht noch einmal erhält.

Angestoßen wird eine Aktion über die `execute()`-Methode des zugehörigen *Action*-Objekts. In der `execute()`-Methode ist die eigentliche Aktion implementiert. Die Ausführung der `execute()`-Methode kommt also der Ausführung der Aktion gleich.

Damit gewährleistet ist, dass eine Aktion angestoßen werden kann, muss jedes *Action*-Objekt die Methode `execute()` bereitstellen. Deshalb müssen alle *Action*-Objekte das *Action*-Interface implementieren, das nichts weiter als den `execute()`-Methoden-Rumpf enthält.

Zuweisung Ereignis ≫ Aktion

Falls ein Ereignis eintritt, muss genau festgelegt sein, welche Aktion bzw. Aktionen dann ausgeführt werden müssen. Auf diese Zuweisung muss jederzeit zugegriffen werden können und sie muss persistent abgespeichert sein. Deshalb wurde sich für eine Tabelle in der Datenbank als Ablage für die Regeldefinitionen entschieden.

In der Datenbank weist diese Tabelle folgende Struktur auf:

```
RegelTabelle:
Nummer int primary key,
Ereignis varchar(50),
Aktion varchar(20)
```

Das erste Attribut gibt die Nummer der Regel an und dient zur eindeutigen Identifikation. Unter dem Attribut Ereignis wird die Ereignisdefinition wie vorher beschrieben abgespeichert und unter dem Attribut Aktion der Name des entsprechenden *Action*-Objekts, dessen `execute()`-Methode bei Ereigniseintritt aufgerufen werden soll.

5.3.3 Rulebase

Die *Rulebase* bietet die Basis für das Verwalten der Regeldefinitionen. Die *Rulebase* holt sich bei der Initialisierung die aktuellen Regeldefinitionen aus der Datenbank und hält sie in einem lokalen Speicher. Sie ermöglicht den Zugriff auf die Ereignisdefinitionen und die zugeordneten Aktionen. Außerdem ist sie für das Löschen von Regeln aus der Regel-Tabelle verantwortlich. Da die Regeln normalerweise direkt in die Datenbank geschrieben werden und die *Rulebase* bei ihrer Initialisierung alle Regeln aus der Regel-Tabelle einliest, ist das Einfügen von neuen Regeln in die Regel-Tabelle nicht vorgesehen.

Da in der Tabelle *RegelTabelle* der Datenbank nur der Name des *Action*-Objekts und nicht das Objekt selbst gespeichert werden kann, ist ein *Mapper* nötig. Dieser *Mapper* hat die Aufgabe zu dem Namen eines *Action*-Objekts das *Action*-Objekt selbst zurückzuliefern. Um ein *Action*-Objekt zurückliefern zu können, braucht die *Rulebase* also einen *Mapper*. Auf den *Mappers* wird am Ende dieses Abschnitts näher eingegangen.

Die *Rulebase* hat folgende Attribute:

- *eventList* bezeichnet eine Liste mit allen Ereignissen, die in der Regel-Tabelle gespeichert sind.
- *actionList* bezeichnet eine Liste mit allen *Action*-Namen, die in der Regel-Tabelle gespeichert sind, und ist analog zu *eventList* aufgebaut.
- *dbo* verweist auf das Datenbankzugriff-Modul aus 5.1 und dient zum Zugriff auf die Regel-Tabelle.
- *ereignisWerte* beinhaltet die Wahrheitswerte aller Ereignisdefinitionen zum Zeitpunkt der letzten Überprüfung.

Die *Rulebase* bietet zur Regelverwaltung eine Reihe von Methoden an:

`getActionList()`
Diese Methode gibt eine Liste aller *Action*-Objekt-Namen zurück, also die Liste *actionList*.

`getEventList()`
Diese Methode gibt eine Liste aller Ereignisdefinitionen zurück, also die Liste *eventList*.

`getAction()`
Diese Methode liefert ein *Action*-Objekt zurück. Übergeben wird der Name des *Action*-Objekts. Für diese Methode wird der *Mapper* benötigt, da die *Rulebase* nur eine Liste der Namen der *Action*-Objekte führt und nicht auf die *Action*-Objekte an sich zugreift.

`addRule()`
Diese Methode fügt den Listen *eventList* und *actionList* das übergebene Ereignis und die übergebene Aktion hinzu. Außerdem wird die Methode `addAction()` des *Mappers* mit dem Aktionsnamen als Übergabeparameter aufgerufen. Diese Methode wird beim Einlesen der Regeldaten aus der Regel-Tabelle eingesetzt, um die Regeln lokal zu speichern.

`removeRule()`
Diese Methode entfernt eine Regel aus der Regel-Tabelle. Damit verbunden ist das Löschen des entsprechenden Ereignisses aus der Liste *eventList* sowie das Löschen der entsprechenden Aktion aus der *actionList*. Abschließend wird der Aktionsname und das zugeordnete *Action*-Objekt aus der Tabelle *actionMapping* des *Mappers* gelöscht.

`updateEreignis()`
Diese Methode gibt eine Liste mit den Wahrheitswerten, die die einzelnen Ereignisdefinitionen liefern, zurück. Dabei werden für die Parameter, die in den Ereignisdefinitionen vorkommen, ihre aktuellen Werte eingesetzt. Geliefert werden die aktuellen Werte der Parameter vom *Eventhandler*, der im nächsten Abschnitt genauer beschrieben wird.

Mapper

Der *Mapper* verwaltet eine Tabelle mit den Namen der *Action*-Objekte und den *Action*-Objekten selbst. Über die Namen kann somit auf die eigentlichen Objekte zugegriffen werden. Das einzige Attribut des *Mappers* ist *actionMapping*. Dieses Attribut beinhaltet die aktuelle Tabelle mit den Namen aller *Action*-Objekte und ihren zugeordneten *Action*-Objekten.

Die Methoden des *Mappers* sind im einzelnen:

`addAction()`
Diese Methode fügt der Mapping-Tabelle *actionMapping* einen neuen Namen und ein neues *Action*-Object hinzu. Übergeben wird der Name des *Action*-Objekts.

`removeAction()`
Diese Methode löscht aus der Mapping-Tabelle *actionMapping* einen Namen und das dazugehörige *Action*-Object. Übergeben wird der Name des zu löschenden *Action*-Objekts.

`getAction()`
Diese Methode holt aus der Mapping-Tabelle *actionMapping* ein *Action*-Object und liefert es zurück. Übergeben wird der Name des gewünschten *Action*-Objekts.

5.3.4 Eventhandler

Der *Eventhandler* stellt die zentrale Komponente bei der Realisierung von SMS-basierten Push-Diensten dar. Er übernimmt die dazu notwendige Triggerfunktion. Der *Eventhandler* hat verschiedene Aufgaben. Er ist für das Erzeugen bzw. Einholen der aktuellen Werte derjenigen Parameter zuständig, die bei den Ereignisdefinitionen eine Rolle spielen. Er hat die Aufgabe in regelmäßigen Abständen zu prüfen, ob die Ereignisdefinitionen mit den aktuellen Werten auf wahr abgebildet werden. Und letztendlich ist er dafür verantwortlich, für jede wahr gewordene Ereignisdefinition die dem Ereignis zugeordnete Aktion anzustoßen.

Im Folgenden wird eine detailliertere Beschreibung der Aufgaben des *Eventhandlers* gegeben:

Für das Einholen der aktuellen Werte muss der *Eventhandler* auf die entsprechenden Ressourcen zugreifen können. Der Zugriff auf die aktuellen Werte erfolgt typischerweise über get()-Methoden. Deshalb ist für jeden Parameter eine get()-Methode zu implementieren, über die den aktuellen Wert des Parameters zugegriffen werden kann. Damit können alle Parameter regelmäßig neu gesetzt werden.

Nach Einholen der aktuellen Werte müssen die Ereignisdefinitionen neu überprüft werden. Damit der *Eventhandler* einen Zugriff auf die in der Datenbank gespeicherten Regeln bekommt, braucht er eine *Rulebase*. Deshalb wird bei der Initialisierung des *Eventhandlers* eine *Rulebase* erzeugt und ihm zugeordnet. Nun kann er mit Hilfe der `updateEreignis()`-Methode der *Rulebase* überprüfen, welche Wahrheitswerte die Ereignisdefinitionen mit den aktuellen Werten liefern.

Für jede Ereignisdefinition, die auf wahr abgebildet worden ist, muss das zugehörige *Action*-Objekt geholt werden. Dazu muss sich der *Eventhandler* zunächst über die `getActionList()`-Methode der *Rulebase* den Namen der zugeordneten Aktion beschaffen. Da ihm der Name allein nicht viel bringt, ist ein weiterer Zugriff auf eine Methode der *Rulebase* nötig. Mit Hilfe der `getAction()`-Methode erhält der *Eventhandler* das *Action*-Objekt, das dem Namen der Aktion zugewiesen ist.

Sobald der *Eventhandler* das *Action*-Objekt erhalten hat, ruft er die `execute()`-Methode des *Action*-Objekts auf und stößt somit die Ausführung der eigentlichen Aktion an.

Das Sequenzdiagramm in Abbildung 5.3 gibt einen Überblick über den Gesamtablauf.

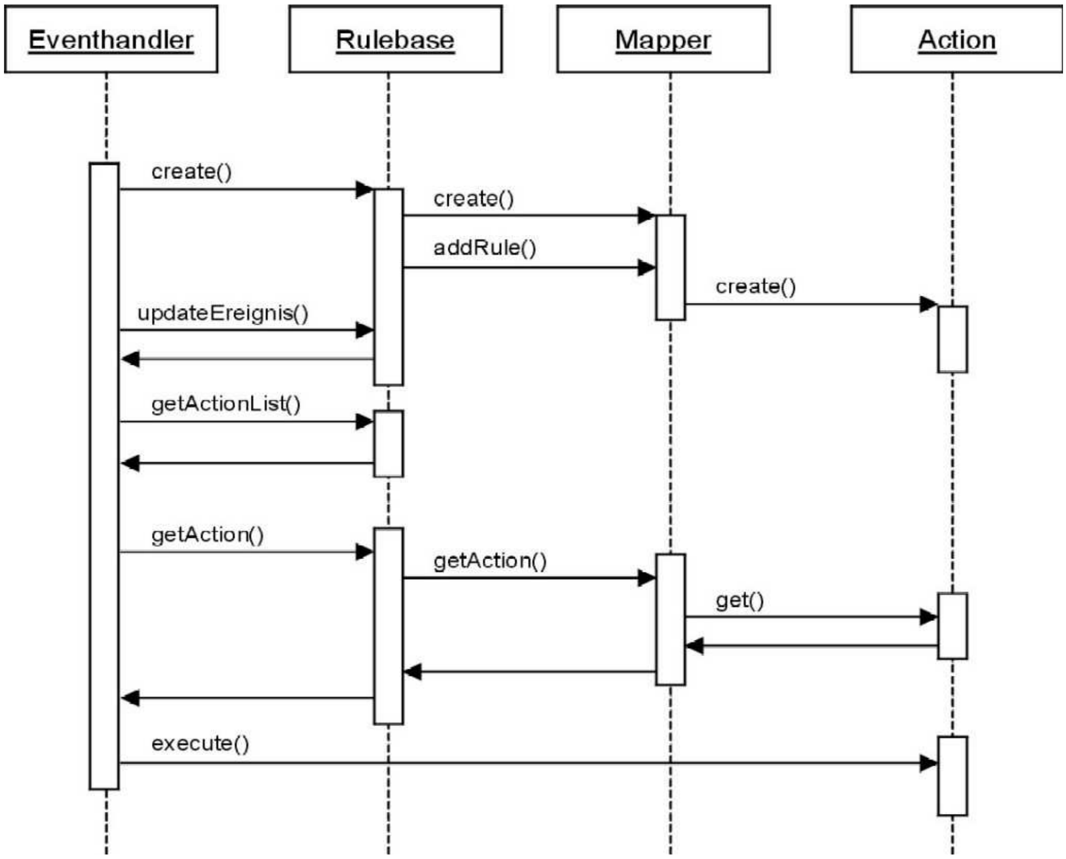


Abbildung 5.3: Sequenzdiagramm für die Abläufe im *Eventhandler*

Damit die eben beschriebenen Aufgaben in regelmäßigen Abstanden durchgeführt werden, ist sicherzustellen, dass der *Eventhandler* periodisch angestoßen wird. Dazu muss der *Eventhandler* das *Reminder*-Interface implementieren und den gesamten Ablauf unter der `remind()`-Methode realisieren. Über ein *Timer*-Objekt, dem der *Eventhandler* als *Reminder*-Objekt zugeordnet ist, wird in vorgegebenen Zeitabständen die `remind()`-Methode des *Eventhandlers* aufgerufen.

5.3.5 Implementierungsmodell

In diesem Abschnitt sollen nun die in den vorigen Abschnitten vorgestellten Komponenten, ihre Beziehungen zueinander und der Gesamtzusammenhang bildlich dargestellt werden.

Abbildung 5.4 zeigt das der Implementierung zugrunde liegende Objektmodell.

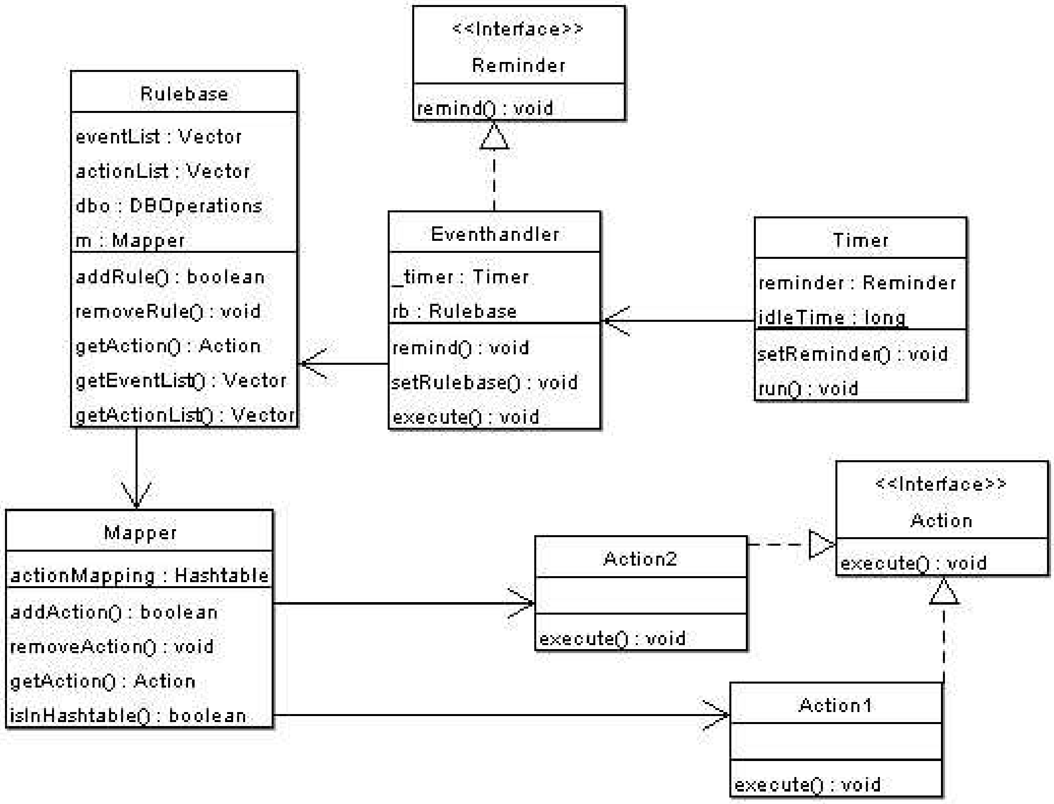


Abbildung 5.4: Implementierungsmodell der Regel-Engine

Kapitel 6

Implementierung

In diesem Kapitel erfolgt nun die prototypische Implementierung des im letzten Kapitel vorgestellten Konzepts. Zuerst wird die um die um die implementierten Komponenten erweiterte Architektur dargestellt. Dann wird die Umsetzung der Module beschrieben. Darauf aufbauend erfolgt die Implementierung der Dienste, die im letzten Kapitel modelliert worden sind. Abschließend wird auf die Realisierung der Regel-Engine näher eingegangen. Für die gesamte Implementierung wurde die Programmiersprache Java verwendet.

6.1 Erweiterte Architektur der SMS-Plattform

Nach der Erweiterung der SMS-Plattform um die Module und die Regel-Engine sieht die Architektur der SMS-Plattform wie in Abbildung 6.1 dargestellt aus.

Die erweiterte Architektur hat die Regel-Engine und die Module integriert und stellt den Applikationen, den Aktionen und der *Rulebase* die Basiskomponenten der Module zur Verfügung.

6.2 Implementierung der Module

Für die Module wurde das Package `basics` angelegt. Jedes Modul ist durch eine Klasse realisiert, also enthält das Package folgende vier Klassen: *MessageOperations*, *DBOperations*, *SendOperations* und *TimeOperations*. Die funktionalen Komponenten der Module sind durch Methoden der Klassen realisiert.

6.2.1 Die Klasse *MessageOperations*

Die Klasse *MessageOperations* realisiert das Textbearbeitungs-Modul. Der Kopf dieser Klasse sieht folgendermaßen aus:

```java
import java.util.regex.Matcher;
import java.util.regex.Pattern;
import sms.inbound.dispatcher.InboundSMS;
```

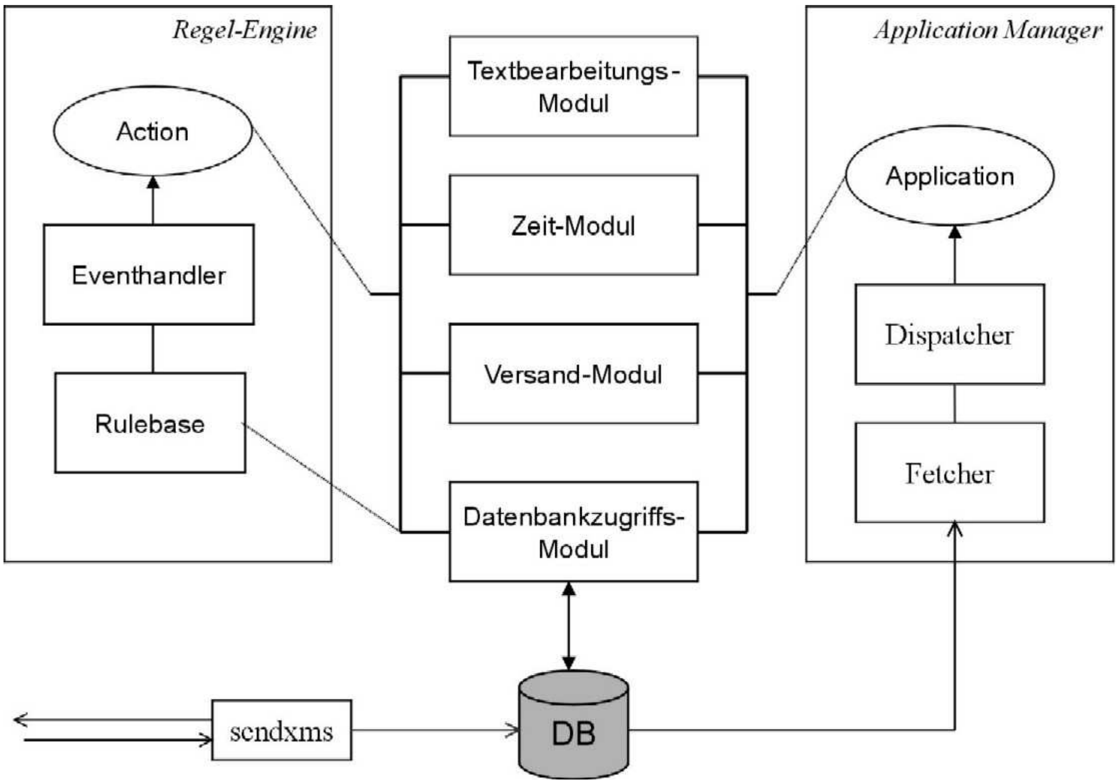


Abbildung 6.1: Erweiterte Architektur der SMS-Plattform

```
public class MessageOperations {
```

Die ersten beiden Methoden werden zur Suche nach der Syntax einer E-Mail-Adresse benötigt, eine Instanz die letzten importierten Klasse wird im Konstruktor übergeben.

```
private String message = "";

public MessageOperations(InboundSMS sms) {
            message = sms.getMessage();
}
```

Das Attribut *message* ist vom Typ String und ihm wird im Konstruktor der Text der Kurznachricht, die durch die übergebene *InboundSMS*-Instanz repräsentiert wird, zugewiesen.

Die in Kapitel 5.1 definierten funktionalen Komponenten des Textbearbeitungs-Moduls werden durch die im weiteren aufgeführten Methoden implementiert.

Diese beiden folgenden Methoden prüfen, ob der übergebene String in dem Kurznachrichtentext enthalten ist. Bei der zweiten Methode wird der Suchbereich auf einen Teilstring. Die Indizes *start* und *end* geben dabei an, bei dem wievielten Zeichen der Teilstring beginnt und wo er endet.

```
public boolean isInMessage(String string) {
      String str = string.toUpperCase();
      if(message.toUpperCase().indexOf(str) != -1) return true;
```

```
        else return false;
}

public boolean isInMessage(String string, int start, int end) {
        String str = string.toUpperCase();
        String messagePart = message.substring(start, end);
        if(messagePart.toUpperCase().indexOf(str) != -1) return true;
        else return false;
}
```

Die Methode, die den Text der Nachricht in einzelne Zeichenfolgen zerlegt, sieht so aus:

```
public String[] partsOfMessage (String separators) {
        String[] messageParts;
        messageParts = Pattern.compile(separators).split(message);
        return messageParts;
}
```

Die Zeichen, die mit separators übergeben werden, bestimmen die Trennpunkte bei der Zerlegung. Diese Methode wird oft in Kombination mit der nächsten Methode eingesetzt, die aus einem Array von Strings denjenigen zurückliefert, der dem Aufbau nach eine E-Mail-Adresse darstellt. Der String pattern zeigt die Syntax einer E-Mail-Adresse.

```
public String getEmailAddress(String[] messageParts) {
        String pattern =
        "((^\\w+)|(^\\w+.\\w+.\\w+)|(^\\w+-\\w+.\\w+)|(^\\w+.\\w+)|" +
        "(^\\w+-\\w+)|(^\\w+-\\w+-\\w+))(@)[a-z0-9-_]+[\\w.-_]+[\\w.-_]+";
        String[] parts = new String [messageParts.length];
        String emailAddress = "";
        for(int i=0; i<messageParts.length; i++) {
              parts[i] = messageParts[i].toLowerCase();
              Pattern p = Pattern.compile(pattern);
              Matcher m = p.matcher(parts[i].toString());
              boolean found = m.matches();
              if(found) {
                    emailAddress = parts[i];
              }
        }
        return emailAddress;
}
```

6.2.2 Die Klasse *SendOperations*

Die Klasse *SendOperations* realisiert das Versand-Modul. Das in dieser Klasse der Versand von E-Mails und Kurznachrichten angeboten wird, sieht der Import-Teil etwas umfangreicher aus:

```
import java.io.*;
import java.net.*;
import java.util.Properties;
import javax.activation.*;
import javax.mail.*;
```

```
import org.apache.log4j.Logger;
import sms.inbound.dispatcher.TelephoneNumber;
import sms.inbound.helper.LogManager;

public class SendOperations {
```

Im Konstruktor wird nur der Logger gesetzt, der Fehlermeldungen oder andere Informationen auf der Konsole ausgibt.

```
private Logger logger;

public SendOperations() {
      logger = LogManager.getLogger(this.getClass().getName());;
}
```

Im Folgenden werden die Methoden dieser Klasse beschrieben. Begonnen wird mit der `sendSMS()`-Methode, die das Senden einer Kurznachricht mit den übergebenen String-Parametern erledigt. Für das Senden ist ein Servlet zuständig, an das einfach die übergebenen Parameter weitergeleitet werden.

```
public void sendSMS(String receiver, String sender, String smsText) {
      try {
            URL servletURL = new URL("http://62.245.222.18/cgi-bin/servlet.pl?"
            + "adC=" + URLEncoder.encode(receiver)
            + "&oAdC=" + sender + "&sms="
            + URLEncoder.encode(smsText));
            BufferedReader inrdr = new BufferedReader
                  (new InputStreamReader(servletURL.openStream()));
            inrdr.close();
            logger.info("SMS␣versendet␣an␣" + receiver);
      }
      catch (Exception err) {
      logger.error(err.toString());
      }
}
```

Anstelle des Strings *receiver* kann auch eine TelephoneNumber-Instanz übergeben werden. Diese wird dann in einen String umgewandelt, der Rest bleibt gleich:

```
public void sendSMS(TelephoneNumber receiverTN, String sender,
                                    String smsText) {
      String receiver = receiverTN.getInternationalFormat();
      // wie oben ..
}
```

Die dritte Methode in dieser Klasse ist für das Senden von E-Mails verantwortlich. Da diese Methode ziemlich komplex ist, werden nur die Teile gezeigt, in denen einer der übergebenen Parameter vorkommt.

```
public void sendEmail(String sender, String receiver, String subject,
            String emailContent, String mimeType, String[] attachmentFiles) {
      String host = "127.0.0.1";
      String mail_from = sender;
```

```
        String mail_to = receiver;
        try {
                // Setzen verschiedener properties..
                message.setFrom(new InternetAddress(mail_from));
                message.addRecipient(Message.RecipientType.TO,
                              new InternetAddress(mail_to));
                message.setSubject(subject);
                // Erzeugen der E-Mail ..
                messageBodyPart.setContent(emailContent, mimeType);
                if(attachmentFiles.length > 0) {
                        for(int i=0; i<attachmentFiles.length; i++) {
                               String filename = attachmentFiles[i];
                              // Anhängen der Dateien an die E-Mail ..
                    }
                }
                Transport.send(message);
                logger.info("Email␣versendet␣an␣" + mail_to);
         }
         // catch-Methoden ..
}
```

Die letzte Methode in dieser Klasse dient zur Umwandlung von Text- oder HTML-Dateien in einen String. Dies ist nötig, da die `sendEmail()`-Methode für den Inhalt der E-Mail ein String-Objekt erwartet.

```
public String fileToString(String filename) {
        String data = "";
        try {
               //Create a URL for the desired page
                URL url = new URL(filename);
                // Read all the text returned by the server
                BufferedReader in = new BufferedReader
                              (new InputStreamReader(url.openStream()));
                String str;
                while ((str = in.readLine()) != null) {
                        if(filename.endsWith("txt")) {
                              str = str + "\n";
                        }
                        data = data + str;
                }
                in.close();
         }
        // catch-Methoden ..
        return data;
}
```

6.2.3 Die Klasse *DBOperations*

Die Klasse *DBOperations* realisiert das Datenbankzugriffs-Modul. Deshalb wird das SQL-Package von Java im Kopf der Klasse importiert.

```
import java.sql.*;
import java.util.Vector;
import org.apache.log4j.Logger;
import sms.inbound.helper.LogManager;

public class DBOperations {
```

Hier ist der Konstruktor etwas umfangreicher, da bei Initialisierung dieser Klasse eine Verbindung zur Datenbank erstellt wird. Zum Aufbau einer Verbindung zur Datenbank sind Informationen über den Treiber, den Datenbanknamen, sowie Name und Passwort des Nutzers nötig. Zum Schließen der Verbindung bietet diese Klasse die Methode `close()`.

```
private Connection conn = null;
private Logger logger;

public DBOperations (){
        logger = LogManager.getLogger(this.getClass().getName());
        try {
            Class.forName("org.gjt.mm.mysql.Driver");
            conn =
            DriverManager.getConnection("jdbc:mysql://localhost:3306/Inbound?",
                                                   "smsuser", "smsuser");
        }
        catch (ClassNotFoundException e) {
            logger.error(e.getMessage());
        }
        catch (Exception e) {
            logger.error(e.getMessage());
        }
}
```

Für Zugriffe auf die Datenbank wurden die funktionalen Komponenten in einzelne Methoden umgesetzt. Eine Datenbankanfrage kann mit folgenden Methoden durchgeführt werden. Beide Methoden führen Datenbankanfragen mit den übergebenen Strings durch. Typisch für Datenbankanfragen in Java ist, dass die Anfrage als Statement der Verbindung übergeben wird und durch `executeQuery()` die Ergebnismenge erzeugt wird. Die Ergebnismenge rs enthält dann alle Ergebnisse der Anfrage hintereinander gereiht.

```
public Vector select(String attribute, String table, String whereClause) {
        Vector results = new Vector();
        String selectStatement = "SELECT␣" + attribute + "␣FROM␣" + table +
                                                 "␣" + whereClause;
        try {
              PreparedStatement ps = conn.prepareStatement(selectStatement);
              ResultSet rs = ps.executeQuery();
              while(rs.next()) {
                    results.add(rs.getString(attribute));
              }
              ps.close();
        }
        catch (SQLException e) {
              logger.error(e.toString());
```

```
        }
        return results;
}

public Vector selectDistinct(String attribute, String table,
                                    String whereClause) {
        Vector results = new Vector();
        String selectStatement = "SELECT␣DISTINCT" + attribute + "␣FROM␣" +
                                            table + "␣" + whereClause;
        // wie oben ..
}
```

Der Unterschied zwischen den beiden Methoden besteht darin, dass letztere automatisch Duplikate aus der Ergebnismenge entfernt. Ganz ähnlich zu den eben beschriebenen Methoden gestalten sich auch die nächsten Methoden. Bei der `insert()`-Methode ist allerdings das Erstellen des Statements etwas komplexer. Die übergebenen Werte müssen der Reihe nach an das Statement angehängt werden. Das wird in der for-Schleife realisiert. Der boolesche Wert *firstId* dient zur Angabe, ob das erste Attribut eine Nummer ist, die automatisch erhöht wird. Falls dies zutrifft, wird diese Nummer automatisch erzeugt und muss nicht übergeben werden.

```
public void insert(String table, boolean firstId, Vector values) {
        String valuesString;
        String insertStatement;
        if(firstId) valuesString = "last_insert_id(id),␣'";
        else valuesString = "'";
        for(int i=0; i<values.size(); i++) {
                if(i == values.size()-1) valuesString = valuesString +
                                            values.get(i) + "'";
                else valuesString = valuesString + values.get(i) + "',␣'";
        }
        insertStatement = "INSERT␣INTO␣" + table + "␣VALUES(" +
                                            valuesString + ")";
        try {
                PreparedStatement ps = conn.prepareStatement(insertStatement);
                ps.executeQuery();
                ps.close();
        }
        catch (SQLException e) {
                logger.error(e.toString());
        }
}
```

Die `update()`-Methode ist relativ einfach, da sowohl die komplette SET- als auch die komplette WHERE-Klausel angegeben werden muss. Somit kann das Statement ohne großen Aufwand zusammengesetzt werden.

```
public void update(String table, String setClause, String whereClause) {
        String updateStatement = "UPDATE␣" + table + "␣" + setClause +
                                            "␣" + whereClause;

        try {
                PreparedStatement ps = conn.prepareStatement(updateStatement);
```

```
            ps.executeUpdate();
            ps.close();
      }
      catch (SQLException e) {
            logger.error(e.toString());
      }
}
```

Die nächste Methode löscht Datensätze aus einer Tabelle der Datenbank. Falls die ganze Tabelle gelöscht werden soll, ist eine leere WHERE-Klausel zu übergeben.

```
public void delete(String table, String whereClause) {
      String deleteStatement;
      if(whereClause.length()==0) deleteStatement = "DELETE␣FROM␣" + table;
      else deleteStatement = "DELETE␣FROM␣" + table + "␣" + whereClause;

      // üblicher try-catch-Teil ..
}
```

6.2.4 Die Klasse *TimeOperations*

Die Klasse *TimeOperations* realisiert das Zeit-Modul mit seinen Komponenten. Für die Methoden dieser Klasse ist eine Instanz der *Calendar*-Klasse nötig. Deswegen wird diese importiert:

```
import java.util.Calendar;

public class TimeOperations {
```

Im Konstruktor wird dem *calendar*-Atttribut eine Instanz der *Calendar*-Klasse zugewiesen:

```
private Calendar c;

public TimeOperations() {
      calendar = Calendar.getInstance();
}
```

Die Umsetzung der Komponenten in Methoden sieht wie folgt aus:

```
public int getDay() {
      return calendar.get(Calendar.DAY_OF_MONTH);
}
public int getMonth() {
      return calendar.get(Calendar.MONTH)+1;
}
public int getHour() {
      return calendar.get(Calendar.HOUR_OF_DAY);
}
public int getMinute() {
      return calendar.get(Calendar.MINUTE);
}
```

Diese vier Methoden liefern den jeweils eine Integer-Zahl zurück und zwar für den aktuellen Tag, den aktuellen Monat, die aktuelle Stunde und die aktuelle Minute. Da die Zahlen für den Monat bei 0 beginnen, wird die Monatszahl automatisch um 1 erhöht.

Die letzte Methode in der *TimeOperations*-Klasse, `getDayOfWeek()`, liefert den aktuellen Tag der Woche zurück, aber nicht als Zahl, sondern als String in Form von ”Mo” , ”Di” , ”Mi” , ”Do” , ”Fr” , ”Sa” oder ”So” .

6.3 Implementierung der kombinierten Dienste

Für die Entwicklung der beiden Dienste, die im letzten Kapitel mit Hilfe der funktionalen Komponenten der Module modelliert wurden, wurde das Package `testaplications` angelegt. Da beiden Dienste das Senden einer Kurznachricht vorausgeht und sie durch den Dispatcher angestoßen werden müssen (siehe Kapitel 3), werden sie durch Applikationen realisiert, die das *SMSApplication*-Interface implementieren, und es ist jeweils ein der Applikation zugeordnetes Keyword einzurichten. Um die Methoden der im vorigen Abschnitt beschriebenen Klassen nutzen zu können, müssen beide Applikationen das Package `basics` importieren.

6.3.1 Die Klasse *Application1*

Die Klasse *Application1* stellt eine Implementierung des pdf-Versand-Dienstes aus Kapitel 5.2 dar. Sie importiert unter anderem das Package `basics` und implementiert das *SMSApplication*-Interface, wodurch eine Implementierung der `execute()`-Methode erzwungen wird.

```
import java.util.Stack;
import org.apache.log4j.Logger;
import com.burdadigital.smsapplication.basics.*;
import sms.inbound.dispatcher.*;
import sms.inbound.helper.LogManager;

public class Application1 implements SMSApplication{
```

Die Applikation weist folgenden Rumpf auf: Im Konstruktor wird der Logger für Konsolen ausgaben gesetzt. Anschließend wird ein Thread gestartet, damit die Applikation nicht die gesamte SMS-Plattform blockiert, falls sie für die Ausführung etwas länger braucht. Solange der erzeugte Stack *_inboundStack* leer ist, passiert allerdings nichts. Sobald eine Kurznachricht in die *InboundSMS*-Tabelle eingegangen ist, dessen Keyword dieser Applikation zugeordnet ist, ruft der Dispatcher die `execute()`-Methode auf und übergibt die *InboundSMS*-Instanz. Die `execute()`-Methode schiebt die *InboundSMS* in den Stack und alles, was innerhalb der if-Schleife steht wird abgearbeitet.

```
private Logger _logger = null;
private Stack _inboundStack = new Stack();

public Application1() {
      _logger = LogManager.getLogger(this.getClass().getName());
      _logger.info(this.getClass().getName() + " initialized");

      new Thread() {
```

```
            public void run() {
                  while(true) {
                        if(!_inboundStack.empty()) {
                              // siehe unten ..
                        }
                  } // end while
            } //end run
      } //end Thread
      .start();
}

public synchronized void execute(InboundSMS inboundSMS) {
      _inboundStack.push(inboundSMS);
}
```

Die eigentliche Dienstausführung erfolgt also innerhalb der if-Schleife. Die einzelnen Code-Zeilen werden im weiteren erklärt:

```
InboundSMS sms = (InboundSMS) _inboundStack.pop();
String[] partsOfSMS;
String emailAddress;
String file = "file:/raid10/framework/V3_0/apps/txt/Test.txt";
String[] attachment = {"/raid10/framework/V3_0/apps/pdf/ernährung.pdf",
                 "/raid10/framework/V3_0/apps/pdf/hautprobleme.pdf"};
```

Die *InboundSMS*-Instanz wird vom Stack geholt und die kompletten Pfadangaben für die Dateien, die der E-Mail beigefügt werden sollen, Parametern zugewiesen.

```
MessageOperations mo = new MessageOperations(sms);
partsOfSMS = mo.partsOfMessage("\\s|[,]");
emailAddress = mo.getEmailAddress(partsOfSMS);
_logger.info("Email:␣" + emailAddress);
```

Ein *MessageOperations*-Objekt wird erzeugt und mit Hilfe der Methoden `partsOfMessage()` und `getEmailAddress()` wird die Kurznachricht in einzelne Textteile zerlegt und das Teil mit der E-Mail-Adresse ausgegeben.

```
SendOperations so = new SendOperations();
if(emailAddress != "") {
      String sender = "pdf@zeitschrift.de";
      String emailContent = so.fileToString(file);
      if(emailContent == "") _logger.info("Fehler␣beim␣Umwandeln␣der␣Datei");
      else {
            String subject = "pdf␣von␣'Zeitschrift'";
            so.sendEmail(sender, emailAddress, subject, emailContent,
                              "text/plain", attachment);
      }
}
else _logger.info("Keine␣Email-Adresse␣gefunden");
```

Zunächst wird ein *SendOperations*-Objekt erzeugt. Falls eine E-Mail-Adresse gefunden werden konnte, werden die entsprechenden Parameter gesetzt und die Methode `sendEmail()` mit diesen Parametern aufgerufen.

6.3.2 Die Klasse *Application2*

Die Klasse *Application2* realisiert den in Abschnitt 5.2 modellierten Horoskop-Dienst. Das Grundgerüst dieser Klasse ist ganz analog zu dem der Klasse *Application1*. Auch hier wird das `basics`-Package importiert und das Interface *SMSApplications* implementiert:

```
import java.util.*;
import org.apache.log4j.Logger;
import com.burdadigital.smsapplication.basics.*;
import sms.inbound.dispatcher.*;
import sms.inbound.helper.LogManager;

public class Application2 implements SMSApplication{
```

Der Rumpf der Applikation entspricht bis auf der Bezeichnung des Konstruktors exakt dem Rumpf der Klasse *Application1*. Deswegen wird er hier nicht noch einmal aufgeführt. Worin sich die beiden Klassen unterscheiden, ist der Anweisungsteil innerhalb der if-Schleife. Dieser wird im Folgenden Schritt für Schritt durchgegangen. Zunächst einmal sollten die verwendeten Attribute aufgeführt werden:

```
private Stack _inboundStack = new Stack();
private Logger _logger = null;
private String Aboverlängerung = "Zu␣sendender␣SMS-Text␣..";
private String Fehlermeldung = "Zu␣sendender␣SMS-Text␣..";

private String SternzListe[] =
{
      "STEINBOCK",
      "WASSERMANN",
      "FISCH",
      "WIDDER",
      "STIER",
      "ZWILLING",
      "KREBS",
      "LÖWE",
      "JUNGFRAU",
      "WAAGE",
      "SKORPION",
      "SCHÜTZE"
};
```

Nun kommen die sich innerhalb der if-Schleife befindenden Code-Zeilen:

```
InboundSMS sms = (InboundSMS) _inboundStack.pop();
String sender = sms.getOriginating_address().getInternationalFormat();
int guthaben = 7;
MessageOperations mo = new MessageOperations(sms);
SendOperations so = new SendOperations();
TimeOperations to = new TimeOperations();
String sternzeichen = "";
boolean keinSternzeichen = true;
for(int i=0; i<12; i++) {
```

```
    if(mo.isInMessage(SternzListe[i])) {
        keinSternzeichen = false;
        sternzeichen = SternzListe[i];
    }
}
```

Auch hier wird die *InboundSMS* vom Stack geholt. Anschließend wird die Absendernummer dem Attribut sender zugewiesen und Objekte der Modul-Klassen erzeugt. Dann wird mit Hilfe der `isInMessage()`-Methode geprüft, ob im Text ein Sternzeichen enthalten ist. Wird TRUE zurückgeliefert, wird das Sternzeichen dem Attribut *sternzeichen* zugewiesen.

```
if(keinSternzeichen) so.sendSMS(sender, "YLHoroskop", Fehlermeldung);
else {
DBOperations dbo = new DBOperations();
String where1 = "WHERE␣abonnent␣=␣'" + sender + "'";
Vector result1 = dbo.select("abonnent", "HoroskopAbonnenten", where1);
```

Falls kein Sternzeichen gefunden wurde, wird dem Absender eine entsprechende Kurznachricht gesendet. Falls das Sternzeichen extrahiert werden konnte, wird ein *DBOperations*-Objekt erzeugt und in der *HoroskopAbonnenten*-Tabelle nachgeschaut, ob der Absender nicht schon registriert ist.

```
if(result1.isEmpty()) {
Vector InsertWerte = new Vector();
InsertWerte.add(sender);
InsertWerte.add(sternzeichen);
InsertWerte.add(new Integer(guthaben));
InsertWerte.add(new Long (sms.getId()));
InsertWerte.add("0");
dbo.insert("HoroskopAbonnenten", true, InsertWerte);
```

Falls der Absender noch kein Abonnent ist, wird ein `insert()` mit seinen Werten durchgeführt. Für die Übertragung der einzufügenden Werte werden die Werte der Reihe nach in einen Vektor eingefügt.

```
String where2 = "WHERE␣Wochentag␣='" + to.getDayOfWeek() + "'";
Vector result2 = dbo.select(sternzeichen, "Wochenhoroskop", where2);
if(!result2.isEmpty()) {
    String spruchNummer = result2.firstElement().toString();
    result2 = dbo.select("spruch", "Horoskope", "WHERE␣id␣='" +
                        spruchNummer + "'");
    if(!result2.isEmpty()) {
        String spruch = result2.firstElement().toString();
        _logger.info("Spruch:␣" + spruch);
        so.sendSMS(sender, "YLHoroskop", spruch);
        String where3 = "WHERE␣abonnent␣=␣'" + sender + "'";
        dbo.update("HoroskopAbonnenten", "SET␣credits=credits-1",
                                where3);
    }
}
```

Mit Hilfe der `select()`-Methode wird nun zuerst auf die Tabelle *Wochenhoroskop* zugegriffen, um die Spruchnummer zu holen, und anschließend auf die *Horoskope*-Tabelle, um anhand der Nummer den Horoskop-Spruch zu holen. Waren beide Datenbankanfrage erfolgreich, so enthält die zurückgelieferte Ergebnismenge *result2* jedes Mal genau ein Objekt. Der Horoskop-Spruch wird mittels `sendSMS()` an den Absender bzw. Abonnenten gesendet. Anschließend wird das Guthaben des Abonnenten um 1 herabgesetzt, indem die `update()`-Methode mit der entsprechenden SET-Klausel aufgerufen wird.

```
}
else {
      so.sendSMS(sender, "YLHoroskop", Aboverlängerung);
      dbo.update("HoroskopAbonnenten", "SET␣credits=credits+7",
                        "WHERE␣abonnent␣=␣'" + sender + "'");
}
dbo.close();
```

Für den Fall, dass der Absender bereits registriert ist, wird eine Kurznachricht mittels der `sendSMS()`-Methode an den Absender gesendet, die ihn darüber informiert, dass sein Abonnement verlängert würde. Anschließend ist das *credits*-Attribut in der *HoroskopAbonnenten*-Tabelle um 7 zu erhöhen durch Aufrufen der `update()`-Methode mit der entsprechenden SET-Klausel. Zum Schluss wird die Verbindung zur Datenbank abgebaut.

6.4 Implementierung der Regel-Engine

Für die Umsetzung des Regel-Engine-Modells aus Kapitel 5.3 wurde das Package `pushaplications` angelegt. In diesem Package befinden sich alle notwendigen Klassen. Die Klassen realisieren die in Kapitel 5.3 definierten Komponenten *Rulebase*, *Mapper* und *Eventhandler*. Ferner sind das Interface *Reminder* und die Klasse *Timer* enthalten, die den *Eventhandler* in regelmäßigen Zeitabständen anstoßen. Schließlich ist noch das Interface *Action*, das jede Klasse, die eine Aktion realisiert, implementieren muss. Das zugrunde liegende Klassendiagramm ist in Abschnitt 5.3.4 dargestellt.

6.4.1 Die Klasse *Mapper*

Die Klasse *Mapper* bietet die Möglichkeit, über den Klassennamen einer Aktion auf eine Instanz der Aktion zuzugreifen. Dazu wird die Hashtable *ActionMapping* erzeugt:

```
import java.util.Hashtable;

public class Mapper {
      private Hashtable actionMapping = new Hashtable();
```

Auf die anfangs erzeugte Hashtable greifen nun alle Methoden der *Mapper*-Klasse zu. Die Methode `addAction()` instantiiert unter dem übergebenen Klassennamen ein neues Objekt einer *Action*-Klasse und fügt der Hashtable sowohl den Klassennamen der Aktion als auch das instantiierte *Action*-Objekt mit `ActionMapping.put()` hinzu.

```
public boolean addAction(String actionClassname) {
      try {
            Class cl= Class.forName(
```

```
                    "com.burdadigital.smsapplication.testapplications."
                                                + actionClassname);
            Action action= (Action) cl.newInstance();
            actionMapping.put(actionClassname.toLowerCase(), action);

            return true;
        }
        catch (Exception e) {
            e.printStackTrace();
            return false;
        }
}
```

Folgende Methode entfernt aus der Hashtable den angegebenen Klassennamen der Aktion und die zughörige Instanz der Klasse.

```
public void removeAction(String action) {
        actionMapping.remove(action.toLowerCase());
}
```

Mit der Methode `getAction()` wird eine Referenz auf die Instanz der Klasse, die zu dem übergebenen Klassennamen gehört, gelegt und zurückgegeben.

```
public Action getAction(String classname) {
return (Action) actionMapping.get(classname.toLowerCase());
}
```

Die letzte Methode in der *Mapper*-Klasse liefert den Wert TRUE, falls für den übergebenen Klassennamen ein Eintrag in der Hashtable exisitert.

```
public boolean isInHashtable(String action){
        return actionMapping.containsKey(action.toLowerCase());
}
```

6.4.2 Die Klasse *Rulebase*

Die Klasse *Rulebase* holt sich die Regeldefinitionen aus der Datenbank und stellt Methoden zum Zugriff darauf zur Verfügung. Deshalb importiert sie die Klasse *DBOperations* aus dem `basics`-Package.

Nach Initialisierung der Vektoren für die Ereignisdefinitionen und zugeordneten Aktionen, wird ein *Mapper*-Objekt für den Zugriff auf *Action*-Objekte. Im Konstruktor wird dann ein Objekt der *DBOperations*-Klasse instantiiert für den Datenbankzugriff und durch zweimaliges Ausführen der `select()`-Methode die Ereignis- und die Aktion-Spalte aus der Regel-Tabelle eingelesen. Anschließen wird der `addRule()`-Methode jedes Ereignis zusammen mit dem Klassennamen der zugeordneten Aktion übergeben. Die `addRule()`-Methode wird gleich im Anschluss vorgestellt.

```
import java.util.Vector;
import com.burdadigital.smsapplication.basics.DBOperations;

public class Rulebase {
```

```java
        boolean[] ereignisWerte = new boolean[50];
        private Vector eventList = new Vector();
        private Vector actionList = new Vector();
        private DBOperations dbo;

        private Mapper m = new Mapper();

        public Rulebase() {
              dbo = new DBOperations();
              Vector v1 = dbo.select("Ereignis", "RegelTabelle");
              Vector v2 = dbo.select("Aktion", "RegelTabelle");

              for(int i=0; i <v1.size(); i++) {
                    addRule(v1.get(i).toString(), v2.get(i).toString());
              }
        }
```

Die folgende Methode fügt den Vektoren *eventList* und *actionList* die übergebenen Strings hinzu. Außerdem ruft sie die `addAction()`-Methode des *Mappers* auf, damit dieser eine entsprechende Instanz der Aktion initiiert und sie dem Klassennamen der Aktion zuweist. So wird gewährleistet, dass die Vektoren und die Hashtable parallel zueinander aufgebaut sind.

```java
public boolean addRule(String event, String action) {
      try {
            //in Vektoren einfügen
            eventList.add(event);
            actionList.add(action);
            //in Hashtable einfügen
            m.addAction(action);
            return m.isInHashtable(action);
      }
      catch (Exception e){
            return false;
      }

}
```

Die nächsten beiden Methoden liefern dieVektoren, die die Regeldefinitionen halten, zurück.

```java
public Vector getActionList() {
      return actionList;
}

public Vector getEventList() {
      return eventList;
}
```

Die Methode `getAction()` gibt eine Referenz auf ein *Action*-Objekt zurück. Dazu wird der *Mapper* eingesetzt, da er die Referenzen auf die Klassen-Instanzen verwaltet.

```java
public Action getAction(String action) {
      return m.getAction(action);
}
```

Mit folgender Methode können die Ereignisdefinitionen mit den aktuellen Parametern, die der *Eventhandler* liefert, neu überprüft werden. Jede Ereignisdefinition wird auf TRUE oder FALSE abgebildet. Die Wahrheitswerte werden der Reihe nach in ein boolesches Array geschrieben.

```
public boolean[] updateEreignis(Eventhandler e) {
      // Setzen der Ereigniswerte mit den akt. Parametern ..
      return ereignisWerte;
}
```

Schließlich gibt es noch die Methode `removeRule()`, mit der eine Regel komplett gelöscht wird, d.h. das Ereignis und die Aktion sind aus den Vektoren, aus der Hashtable des *Mappers* und aus der Datenbank zu löschen. Damit die Nummerierung in der Datenbanktabelle konsistent bleibt, wird ein Update der Nummern durchgeführt.

```
public void removeRule(int i) {
      String action = actionList.get(i-1).toString();
      eventList.remove(i-1);
      actionList.remove(i-1);
      m.removeAction(action);
      dbo.delete("RegelTabelle", "WHERE␣Nummer␣=" +i);
      dbo.update("RegelTabelle", "SET␣Nummer=Nummer-1", "WHERE␣Nummer>" +i);
}
```

6.4.3 Die Klasse *Eventhandler*

Die Klasse *Eventhandler* ist für die Steuerung und den Mechanismus der Regel-Engine zuständig. Sie implementiert das SMSAplication-Interface, damit sie überhaupt beim Start des Frameworks initialisiert wird, und das *Reminder*-Interface, damit sie über die `remind()`-Methode periodisch angestoßen werden kann.

```
import java.util.Vector;
import org.apache.log4j.Logger;
import sms.inbound.dispatcher.*;
import sms.inbound.helper.LogManager;
import com.burdadigital.smsapplication.basics.*;

public class Eventhandler implements SMSApplication, Reminder{
```

Im Konstruktor wird eine Instanz der Klasse *Rulebase* erzeugt und dem *Eventhandler* zugeordnet. Dadurch hat der *Eventhandler* Zugriff auf die Regeldaten. Anschließend wird ein *Timer*-Objekt instantiiert, der *Reminder* gesetzt und der *Timer* gestartet. Der *Timer* stößt dann die `remind()`-Methode des *Eventhandlers* in den angegebenen Zeitabständen an. Da der *Eventhandler* das *SMSApplication*-Interface implementiert, muss er auch die `execute()`-Methode bereitstellen.

```
private Logger _logger = null;

private Rulebase rb;
private Timer _timer = null;

public Eventhandler() {
```

```
        _logger = LogManager.getLogger(this.getClass().getName());
        _logger.info(this.getClass().getName() + "␣initialized");
        Rulebase r = new Rulebase();
        setRulebase(r);
        new Thread() {
              public void run() {
                    while (true) {
                    }
              }
        }
        .start();
        _timer = new Timer();
        _timer.setReminder(this);
        _timer.start();
}

public synchronized void execute(InboundSMS inboundSMS) {}
```

Die Zuweisung einer Rulebae-Instanz erfolgt mit Hilfe folgender Methode:

```
public void setRulebase(Rulebase rb) {
      this.rb = rb;
}
```

Die `remind()`-Methode ist das Kernstück der *Eventhandlers*. Von hier aus werden praktisch alle Funktionen gesteuert und der Mechanismus der Regel-Engine realisiert. Die `remind()`-Methode wird periodisch in bestimmten Zeitabständen vom *Timer*-Objekt aufgerufen. Bei jedem Aufruf werden die aktuellen Werte der Parameter geholt. Dazu müssen die Parameter als Attribute des *Eventhandlers* eingeführt werden und für jeden Parameter eine `get()`-Methode implementiert werden, über die auf den Parameter zugegriffen werden kann. Nach Setzen der neuen Parameterwerte wird die `updateEreignis()`-Methode der *Rulebase* aufgerufen. Durch Übergabe des *Eventhandler*-Objekts erhält die *Rulebase* einen Verweis auf den *Eventhandler* und kann somit auf die aktuellen Parameterwerte zugreifen. Das zurückgelieferte boolesche Array *eventValues* wird nun durchgegangen. Für jedes Ereignis, das auf wahr abgebildet worden ist, wird das zugewiesene *Action*-Objekt geholt und dessen `execute()`-Methode aufgerufen.

```
public void remind() {
LogManager.getLogger(this.getClass().getName()).
                  debug("Push-Reminder␣was␣called.");

DBOperations dbo = new DBOperations();
      TimeOperations to = new TimeOperations();
      boolean[] eventValues;
      //relevante Parameter neu setzen ..
eventValues = rb.updateEreignis(this);

      for(int i=0; i<eventValues.length; i++) {
            boolean eventIsTrue = eventValues[i];
            if(eventIsTrue){
            _logger.info("Ereignis␣" + rb.getEventList().get(i) +
                                          "␣eingetroffen");
            String action = rb.getActionList().get(i).toString();
```

```
            Action a = rb.getAction(action);
            a.execute();
            _logger.info("execute()-Methode␣von␣" + action +
                                              "␣aufgerufen");
            }
      }
}
```

6.4.4 Die Klasse *Timer*

In der *Timer*-Klasse kann die Zeitspanne, die zwischen einem Aufruf der `remind()`-Methode und dem nächsten vergehen soll, gesetzt werden. In diesem Fall sind es 5 Minuten. Außerdem wird hier ein Verweis auf das Objekt, das die `remind()`-Methode implementiert, gelegt.

```
import sms.inbound.helper.LogManager;

public class Timer extends Thread {
      private Reminder reminder = null;
// Reminder alle 5 Minuten
private final static long idleTime = 5 * 60 * 1000;

public void setReminder(Reminder reminder) {
      LogManager.getLogger(this.getClass().getName()).
                              debug("Push-Reminder␣set.");
            this.reminder = reminder;
}
```

Schließlich wird noch ein Thread gestartet, der für *idleTime* schläft, also für 5 Minuten, und dann die `remind()`-Methode des zugewiesenen Objekts aufgerufen.

```
public void run() {
      while (true) {
            try {
                  Thread.sleep(idleTime);
            } catch (InterruptedException iex) {
                  iex.printStackTrace();
            }
            this.reminder.remind();
      }
}
```

6.4.5 Das Interface *Reminder*

Das Interface *Reminder* stellt nichts weiter als den Rumpf der Methode `remind()` bereit:

```
public interface Reminder {
      public void remind();
}
```

6.4.6 Das Interface *Action*

Das Interface *Action* stellt nichts weiter als den Rumpf der Methode `execute()` bereit:

```
public interface Action {
      public void execute();
}
```

Kapitel 7

Test und Evaluierung

In diesem Kapitel wird die Funktionsfähigkeit der zuvor beschriebenen Implementierung anhand eines Testdurchlaufs nachgewiesen. Nach einer kurzen Beschreibung der Testumgebung werden die notwendigen Tabellen in der MySQL-Datenbank erzeugt. Anschließend werden die implementierten Dienste und die Regel-Engine mit Hilfe von Testdaten getestet. Abschließend erfolgt eine Bewertung der eigenen Architektur. Dazu wird der Anforderungskatalog aus Kapitel 4 herangezogen.

7.1 Testumgebung

Für den Test wurde ein CVS-Testsystem eingerichtet, das auf dasselbe CVS-Repository wie das Live-System zugreift. Das Testsystem weist dieselbe Package-Struktur wie das Life-System auf, arbeitet aber nicht mit derselben Datenbank wie das Live-System. Damit die für das Live-System entwickelten Klassen auch auf dem Testsystem laufen, wurde die Namensgebung der Live-Datenbank übernommen. Das Testsystem stellt, genau wie das Live-System, für die *InboundSMS*-Tabelle die Datenbank *Inbound* zur Verfügung. In dieser Datenbank sind auch die Tabellen für den Test anzulegen. Sowohl Live-System als auch Testsystem laufen auf einem Linux-Rechner.

Vor dem Test werden in der Konfigurationsdatei, auf die der *KeywordApplicationMapper* zugreift, die Keywords 'TEST1' und 'TEST2' eingerichtet und den Applikationen *Application1* und *Application2* zugeordnet.

7.1.1 Erzeugen der Tabellen

Die in Kapitel 5 beschriebenen Tabellen

- *HoroskopAbonnenten*,
- *Horoskope*,
- *Wochenhoroskop*,
- *Profile* und
- *RegelTabelle*

werden mit folgendem MySQL-Kommando in der Datenbank eingerichtet:

```
CREATE TABLE <Tabellenname> (
<Atributname> <Typ> {<Option>}
...;
);
```

Für die *HoroskopAbonnenten*-Tabelle sieht das wie folgt aus:

```
CREATE TABLE HoroskopAbonnenten(
id int auto_increment,
abonnent varchar(20),
sternzeichen varchar(20),
credits int,
inboundSMSId long,
status int,
primary key (id));
```

Die restlichen Tabellen werden analog eingerichtet.

7.1.2 Einfügen der Testdaten

Für die Horoskop-Applikation müssen in die *Horoskope*-Tabelle Sprüche und in die *Wochenhoroskop*-Tabelle Nummern, die auf die Sprüche verweisen, eingefügt werden. Um das Einfügen so einfach wie möglich zu gestalten, werden keine echten Sprüche, sondern nur Spruch1, .., Spruch10 verwendet. Das Kommando dazu sieht so aus:

```
INSERT INTO Horoskope VALUES (1,'Spruch1');
```

In der *Wochenhoroskop*-Tabelle werden die Nummern von 1 bis 10 auf die einzelnen Spalten verteilt. Um auch diesen Schritt abzukürzen, werden nur bis zum aktuellen Tag, also Donnerstag, Nummern eingefügt. In die *Profile*-Tabelle wird folgender Datensatz geschrieben (meine Handynummer, der Dienst, der Anbieter, die Anzahl der Dienstnutzung, das Datum der letzten Dienstnutzung, der aktuelle Status):

```
1, '+491798221065', 'YLHoroskop', 'YL', 3, '26.08.2003', 0
```

Die Regel-Tabelle habe ich mit zwei Definitionen gefüllt, auf die im nächsten Abschnitt genauer eingegangen wird:

```
'h==14&&min<20&&min>15', 'Action1'
'day==28&&month==8', 'Action2'
```

7.1.3 Implementierung der Test-Aktionen

Damit die Funktionsfähigkeit der Regel-Engine getestet werden kann, wurden zwei Regeln definiert und die entsprechenden Aktionen implementiert. Die Aktionen befinden sich im Package `testapplications`.

Die erste Regel legt fest, dass täglich jedem Horoskop-Abonnenten ein Spruch automatisch zugesendet wird und zwar dann, wenn ein bestimmter Zeitpunkt eingetreten ist. Beim Zeitpunkt wurde sich nach der aktuellen Zeit gerichtet. Die Spanne von 5 Minuten kommt dadurch zustande, dass der *Eventhandler* alle 5 Minuten angestoßen wird, und somit genau einmal am Tag in dieses Zeitfenster kommt. Die auszuführende Aktion wurde in der Klasse *Action1* implementiert. Zuerst müssen alle Abonnenten, die noch mindestens einen Spruch guthaben und deren Status gleich 0 ist, aus der Abonnenten-Tabelle geholt werden. Dann muss für jeden Abonnenten anhand seines Sternzeichens der Horoskop-Spruch des Tages selektiert und an ihn gesendet werden. Abschließend ist sein Guthaben um 1 zu erniedrigen.

Die zweite Regel legt fest, dass Leserinnen der Zeitschrift YoungLisa über das Erscheinen der neuen Zeitschrift Sugar informiert werden und zwar zwei Tage vor Erscheinen. Für den Test wurde natürlich der aktuelle Tag angegeben. Die auszuführende Aktion wurde in der Klasse *Action2* implementiert. Für diese Regel müssen aus der Tabelle *Profile* alle Teilnehmer extrahiert werden, bei denen unter dem Attribut *Anbieter* 'YL' und unter dem Attribut *Anzahl* mindestens eine 3 zu finden ist. Außerdem muss der Status gleich 0 sein, ein Indiz dafür, dass dem Teilnehmer noch keine Kurznachricht geschickt worden ist. Jedem der extrahierten Teilnehmer ist nun eine Kurznachricht, die über das Erscheinen der Sugar informiert, zu senden. Anschließend ist der Status auf 1 zu setzen, damit der Teilnehmer die Nachricht nicht noch einmal erhält.

7.1.4 Anpassen des Eventhandlers und der Rulebase

Im *Eventhandler* sind die alle Parameter, die in den Regeldefinitionen eingesetzt werden, als Attribute aufzuführen:

```
private int h;
private int min;
private int day;
private int month;
```

Sobald die `remind()`-Methode des *Eventhandlers* aufgerufen wird, müssen die Parameter neu gesetzt werden. In diesem Testbeispiel erfolgt dies über eine Instanz der *TimeOperations*-Klasse:

```
TimeOperations to = new TimeOperations();
h = to.getHour();
min = to.getMinute();
day = to.getDay();
month = to.getMonth();
```

Außerdem sind für den Zugriff auf die Parameter folgende `get()`-Methoden bereitzustellen:

```
public int getHour() {return this.h;}
public int getDay() {return this.day;}
public int getMinute() {return this.min;}
public int getMonth() {return this.month;}
```

In der *Rulebase* ist nur die Methode, die auf die aktuellen Parameter zugreift, entsprechend anzupassen. Hier sind die Ereignisse noch einmal als boolesche Ausdrücke mit den `get()`-Methoden des

Eventhandlers anstelle der Parameter anzugeben. Ein Aufruf der Methode bewirkt, dass bei allen Ereignissen die aktuellen Parameterwerte des *Eventhandlers* eingesetzt werden und durch TRUE oder FALSE anzeigen, ob sie eingetreten sind oder nicht.

```
public boolean[] updateEreignis(Eventhandler e) {
      ereignisWerte[0]= e.getHour()==14&&e.getMinute()<20&&e.getMinute()>=15;
      ereignisWerte[1]= e.getDay()==28&&e.getMonth()==8;

      return ereignisWerte;
}
```

7.1.5 Testdurchlauf

Zunächst müssen die Änderungen auf den Klassen sowie die neuen Klassen mit

```
cvs commit
```

in das CVS-Repository geschrieben werden, damit das CVS-Testsystem darauf zugreifen kann. Anschließend ist das Testsystem mit

```
./buildall.sh
```

zu starten. Dieser Befehl bewirkt, dass das Testsystem seine eigene Kopie auf den selben Stand wie das Repository bringt, den Code compiliert und das Framework startet.

Durch den Start des Frameworks werden alle Klassen, die das Interface SMSApplications implementieren initialisiert, d.h. der *Eventhandler* und die beiden Applikationen *Application1* und *Application2* sind nun aktiv.

Zum Testen der Applikationen wird je eine Kurznachricht mit einem der folgenden Textinhalte an die Kurzwahlnummer 82280 gesendet:

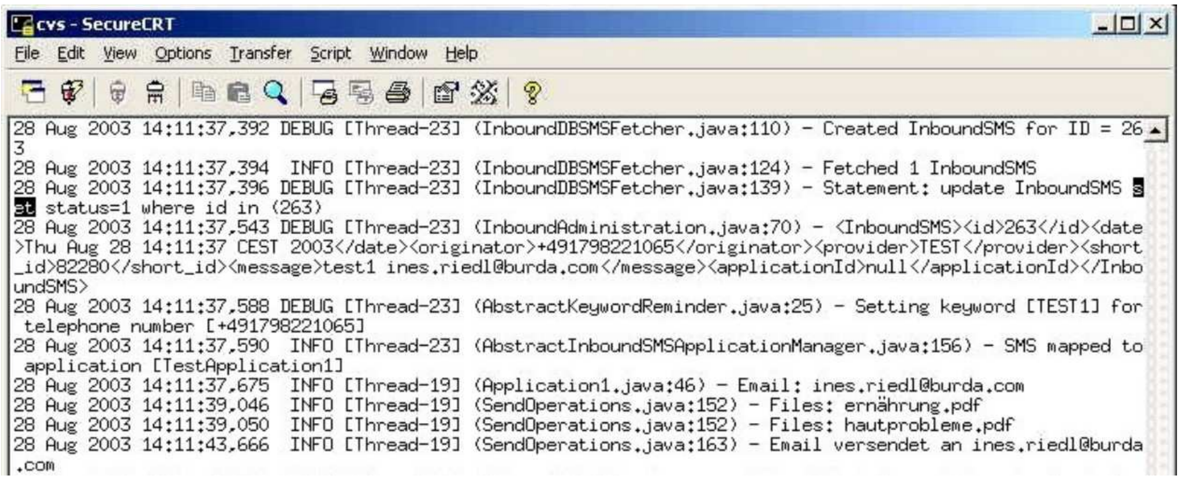
```
Test1 ines.riedl@burda.com
```

```
Test2 steinbock
```

Die erste Kurznachricht erzeugt eine Reihe von Konsolenausgaben, die in Abbildung 7.1 zu sehen sind:

Die ersten vier Zeilen im Konsolenfenster zeigen an, dass der *Inbound DBSMSFetcher* die Kurznachricht aus der *InboundSMS*-Tabelle eingelesen hat, und in den nächsten drei Zeilen sind die Attribute zu sehen, die der *InboundSMS*-Instanz bei der Erzeugung zugewiesen werden. Dann wird angegeben, dass das Keyword 'TEST1' gefunden wurde und die Kurznachricht der Applikation *Application1* zugeordnet wurde. Die letzten fünf Zeilen beziehen sich auf Schritte, die während der Applikationsausführung durchgeführt wurden.

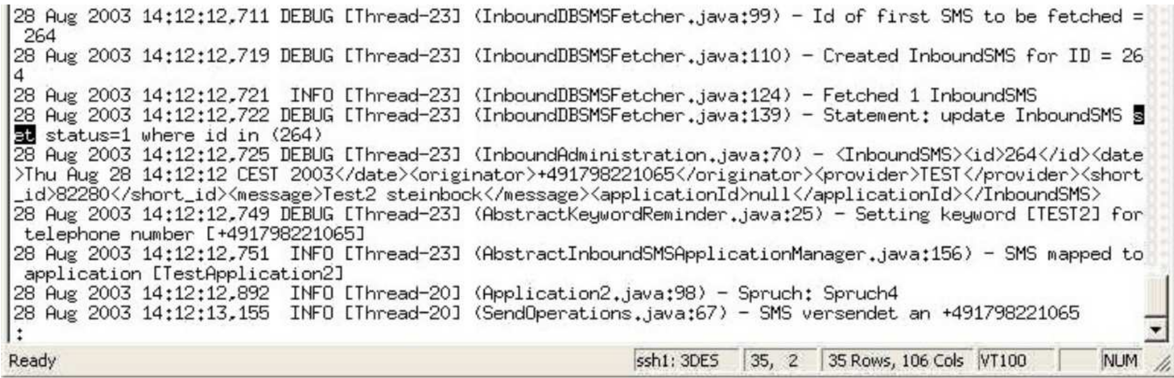
In *Application1* wird nach dem Extrahieren der E-Mail-Adresse die Log-Information 'Email: ines.riedl@burda.com' generiert. Für das Erzeugen und Versenden der E-Mail ist das Modul *SendOperations* zuständig. In der Klasse *SendOperations* wird für jede Datei, die angehängt wird ('ernährung.pdf' und 'hautprobleme.pdf'), und nach Senden der E-Mail ('Email versendet an ines.riedl@burda.com') ein Log-Eintrag erstellt. Einige Minuten später hatte ich die E-Mail in meinem Posteingang.



```
cvs - SecureCRT
File Edit View Options Transfer Script Window Help
28 Aug 2003 14:11:37,392 DEBUG [Thread-23] (InboundDBSMSFetcher.java:110) - Created InboundSMS for ID = 26
3
28 Aug 2003 14:11:37,394  INFO [Thread-23] (InboundDBSMSFetcher.java:124) - Fetched 1 InboundSMS
28 Aug 2003 14:11:37,396 DEBUG [Thread-23] (InboundDBSMSFetcher.java:139) - Statement: update InboundSMS s
et status=1 where id in (263)
28 Aug 2003 14:11:37,543 DEBUG [Thread-23] (InboundAdministration.java:70) - <InboundSMS><id>263</id><date
>Thu Aug 28 14:11:37 CEST 2003</date><originator>+491798221065</originator><provider>TEST</provider><short
_id>82280</short_id><message>test1 ines.riedl@burda.com</message><applicationId>null</applicationId></Inbo
undSMS>
28 Aug 2003 14:11:37,588 DEBUG [Thread-23] (AbstractKeywordReminder.java:25) - Setting keyword [TEST1] for
 telephone number [+491798221065]
28 Aug 2003 14:11:37,590  INFO [Thread-23] (AbstractInboundSMSApplicationManager.java:156) - SMS mapped to
 application [TestApplication1]
28 Aug 2003 14:11:37,675  INFO [Thread-19] (Application1.java:46) - Email: ines.riedl@burda.com
28 Aug 2003 14:11:39,046  INFO [Thread-19] (SendOperations.java:152) - Files: ernährung.pdf
28 Aug 2003 14:11:39,050  INFO [Thread-19] (SendOperations.java:152) - Files: hautprobleme.pdf
28 Aug 2003 14:11:43,666  INFO [Thread-19] (SendOperations.java:163) - Email versendet an ines.riedl@burda
.com
```

Abbildung 7.1: Konsolenausgaben für erste Kurznachricht

Die zweite Kurznachricht hat die in Abbildung 7.2 aufgeführten Konsolenausgaben zur Folge:

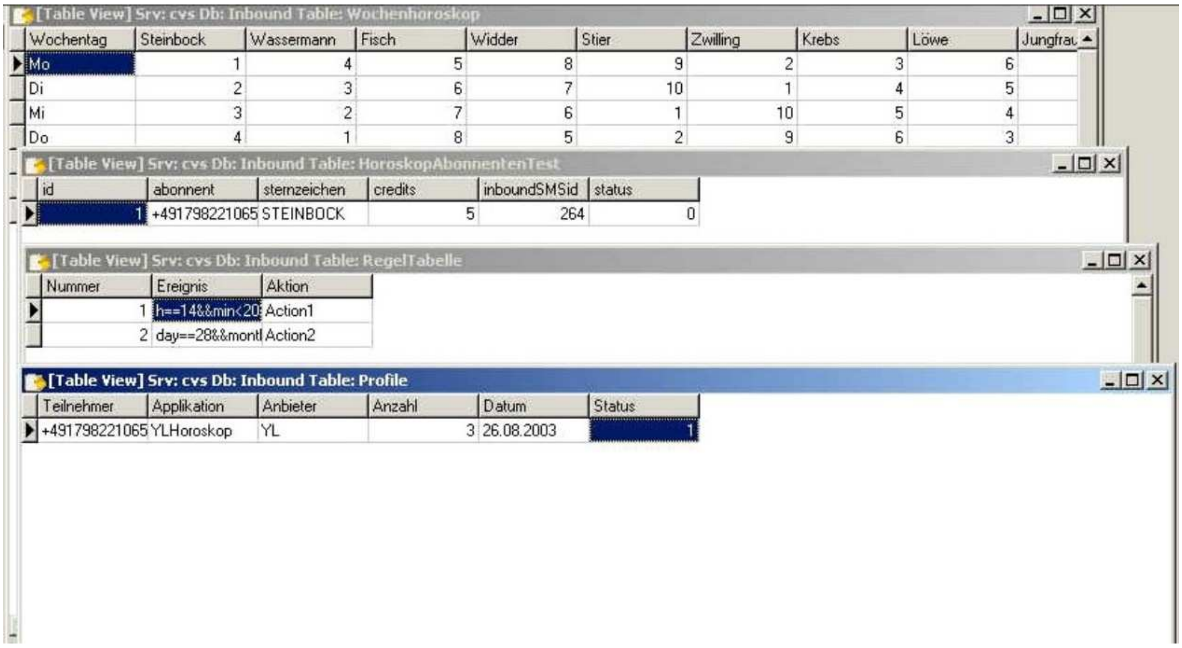


```
28 Aug 2003 14:12:12,711 DEBUG [Thread-23] (InboundDBSMSFetcher.java:99) - Id of first SMS to be fetched =
 264
28 Aug 2003 14:12:12,719 DEBUG [Thread-23] (InboundDBSMSFetcher.java:110) - Created InboundSMS for ID = 26
4
28 Aug 2003 14:12:12,721  INFO [Thread-23] (InboundDBSMSFetcher.java:124) - Fetched 1 InboundSMS
28 Aug 2003 14:12:12,722 DEBUG [Thread-23] (InboundDBSMSFetcher.java:139) - Statement: update InboundSMS s
et status=1 where id in (264)
28 Aug 2003 14:12:12,725 DEBUG [Thread-23] (InboundAdministration.java:70) - <InboundSMS><id>264</id><date
>Thu Aug 28 14:12:12 CEST 2003</date><originator>+491798221065</originator><provider>TEST</provider><short
_id>82280</short_id><message>Test2 steinbock</message><applicationId>null</applicationId></InboundSMS>
28 Aug 2003 14:12:12,749 DEBUG [Thread-23] (AbstractKeywordReminder.java:25) - Setting keyword [TEST2] for
 telephone number [+491798221065]
28 Aug 2003 14:12:12,751  INFO [Thread-23] (AbstractInboundSMSApplicationManager.java:156) - SMS mapped to
 application [TestApplication2]
28 Aug 2003 14:12:12,892  INFO [Thread-20] (Application2.java:98) - Spruch: Spruch4
28 Aug 2003 14:12:13,155  INFO [Thread-20] (SendOperations.java:67) - SMS versendet an +491798221065
:
Ready    ssh1: 3DES    35, 2    35 Rows, 106 Cols    VT100    NUM
```

Abbildung 7.2: Konsolenausgaben für zweite Kurznachricht

Das erste Stück des Konsolenfensters bezieht sich, wie vorher auch, auf die Verarbeitung der zweiten Kurznachricht. Auch hier wird das Einlesen durch den *InboundDBSMSFetcher*, die Attribute, die aus der *InboundSMS*-Tabelle extrahiert wurden, und die Zuordnung der Kurznachricht zur Applikation *Application2* anhand des Keywords 'TEST2' angezeigt. Die für den Test relevanten Log-Ausgaben befinden sich in den letzten drei Zeilen. In *Application2* wird für den aus der *Horoskope*-Tabelle selektierten Spruch ein Log-Eintrag erzeugt ('Spruch: Spruch4'). Abbildung 7.3 gibt einen Überblick über die Tabelleneinträge in der Datenbank nach dem Testlauf. Die oberste Tabelle (*Wochenhoroskop*) zeigt in der Steinbock-Spalte unter Donnerstag die Nummer 4 und nach Abschnitt 7.1.2 ist Nummer 4 Spruch4 in der Tabelle *Horoskope* zugewiesen. Dass ich als Abonnent eingefügt wurde, ist an der *HoroskopAbonnenten*-Tabelle gleich darunter zu sehen. Für das Senden des Spruchs ist das Modul *SendOperations* zuständig. In *SendOperations* wird für jede gesendete Kurznachricht eine Log-Ausgabe erzeugt ('SMS versendet an +491798221065'). Einen kurzen Augenblick später hatte ich die Kurznachricht auf meinem Handy.

Da die Regel-Engine automatisch aktiv werden soll, ist für den Test der Regel-Engine keine Aktion meinerseits nötig. Es ist abzuwarten, bis der Timer, der ja alle fünf Minuten anspringt, aktiv wird


[Table View] Srv: cvs Db: Inbound Table: Wochenhoroskop

Wochentag	Steinbock	Wassermann	Fisch	Widder	Stier	Zwilling	Krebs	Löwe	Jungfrau
Mo	1	4	5	8	9	2	3	6	
Di	2	3	6	7	10	1	4	5	
Mi	3	2	7	6	1	10	5	4	
Do	4	1	8	5	2	9	6	3	

[Table View] Srv: cvs Db: Inbound Table: HoroskopAbonnentenTest

id	abonnent	sternzeichen	credits	inboundSMSid	status
1	+491798221065	STEINBOCK	5	264	0

[Table View] Srv: cvs Db: Inbound Table: RegelTabelle

Nummer	Ereignis	Aktion
1	h==14&&min<20	Action1
2	day==28&&montl	Action2

[Table View] Srv: cvs Db: Inbound Table: Profile

Teilnehmer	Applikation	Anbieter	Anzahl	Datum	Status
+491798221065	YLHoroskop	YL	3	26.08.2003	1

Abbildung 7.3: Tabellenüberblick

und den *Eventhandler* anstößt. Das Anstoßen des *Eventhandlers* wird durch den Log-Eintrag 'Push-Reminder was called' angezeigt. Dies ist erfolgt, wie in Abbildung 7.4 an der ersten Zeile zu sehen ist. Der *Eventhandler* gibt alle eingetroffenen Ereignisse und den Aufruf der `execute()`-Methode einer Aktion in der Konsole aus. Dazwischen befinden sich die Log-Einträge, die während der Ausführung einer Aktion entstehen.

Abbildung 7.4 zeigt, dass das Ereignis der ersten Regel eingetroffen ist und die zugehörige Aktion *Action1* gestartet wurde. Die Tabellenübersicht weist in der Regel-Tabelle die beiden Ereignisdefinitionen auf. In der Konsole sind nun die Log-Ausgaben von *Action1* zu sehen. Da ich der einzige Abonnent in der Tabelle bin, erhalte nur ich einen Horoskop-Spruch passend zu meinem Sternzeichen. Im Normalfall sollte es allerdings so sein, dass ein Abonnent nur einen Spruch pro Tag bekommt. Zu diesem Zweck wurde das Status-Attribut in der Tabelle *HoroskopAbonnenten* eingeführt, das anzeigt, ob der Abonnent an diesem Tag schon einen Spruch bekommen hat. Der *Eventhandler* gibt die Meldung, dass er die `execute()`-Methode einer Aktion aufgerufen hat, immer erst aus, wenn die Aktion beendet ist.

Auch das Ereignis der zweiten Regel konnte auf TRUE abgebildet werden ('Ereignis day==28&&month==8 eingetroffen'). Wie in der Konsole zu sehen ist, wird die zugeordnete Aktion *Action2* gestartet. Da ich mich mit den passenden Daten in die Tabelle eingetragen habe, wird mir eine Kurznachricht mit dem Hinweis auf das Erscheinen der neuen Sugar zugesendet und mein Status in der *Profile*-Tabelle auf 1 gesetzt, damit ich diese Nachricht nicht ein weiteres Mal erhalte. Dass dies auch tatsächlich erfolgt ist, zeigt der Tabellenüberblick in Abbildung 7.3 . Nach fünf Minuten wird der *Eventhandler* erneut angestoßen. An den Log-Ausgaben in Abbildung 7.5 ist zu erkennen, dass das erste Ereignis diesmal nicht eingetroffen ist und dementsprechend *Action1* nicht aufgerufen wird. Das zweite Ereignis trifft zwar ein und *Action2* wird auch gestartet, aber da *Action2* keine passenden Teilnehmer in der Tabelle *Profile* findet, wird diesmal keine Kurznachricht versendet.

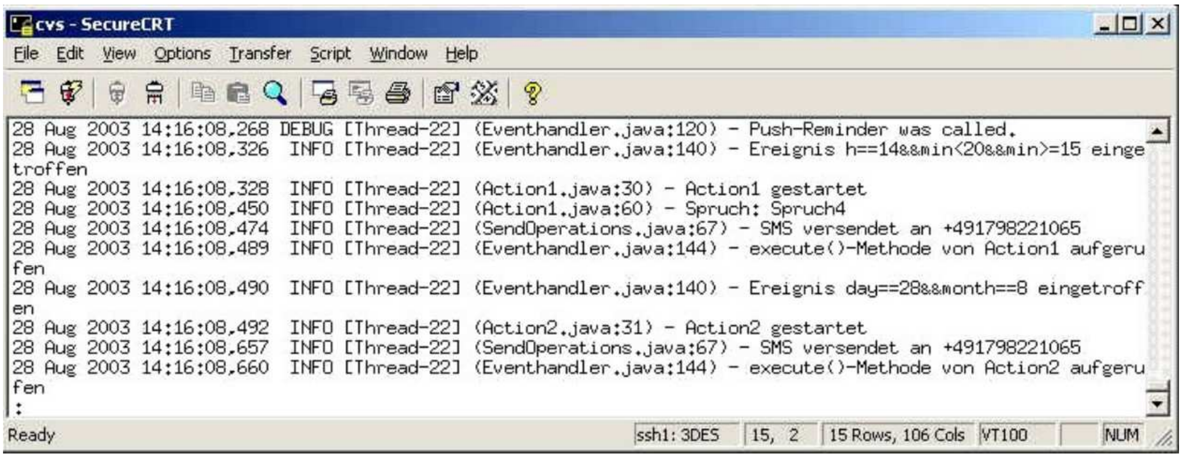


Abbildung 7.4: Konsolenausgaben für ersten Push

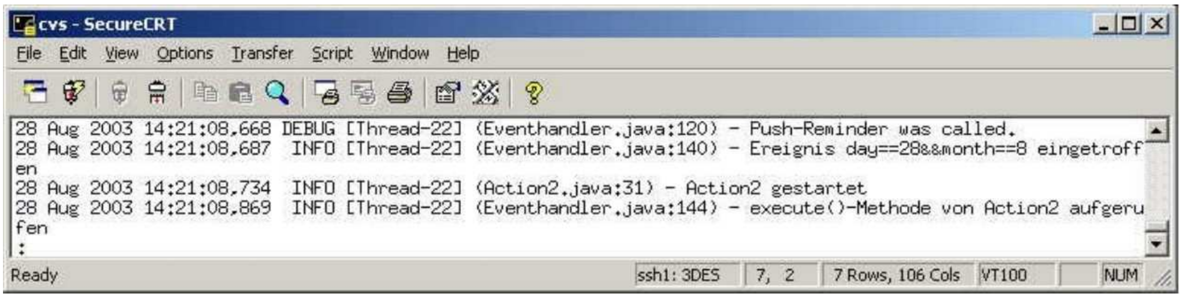


Abbildung 7.5: Konsolenausgaben für zweiten Push

7.2 Evaluierung

In diesem Abschnitt wird die erstellte Architektur anhand der Kriterien, die in dem Anforderungskatalog in Kapitel 4 aufgeführt sind, bewertet.

7.2.1 SMS-Unterstützung

Durch die Anbindung der Architektur an die SMSCs aller Netzbetreiber über das Tool SendXMS, wird das Versenden und Empfangen von Kurznachrichten unterstützt. Da alle Netzbetreiber angebunden sind und das Toll SendXMS jedes SMSC-Protokoll bereitstellt, ist die Kommunikation mit jedem SMS-fähigen Gerät möglich.

7.2.2 Verfügbarkeit

Bei Erwartung von hohem Nachrichtenaufkommen können zu den Netzbetreibern weitere Verbindungen geöffnet werden, damit die SMSCs ihre empfangenen Kurznachrichten sofort an die SMS-Plattform weiterleiten können. Dies ist nötig, da die Speicherkapazitäten der SMSCs beschränkt sind und Kurznachrichten gelöscht werden müssten, falls ihre Weiterleitung nicht möglicht ist. In der Regel läuft das System 24 Stunden 7 Tage die Woche, allerdings führt manchmal ein Fehler in einer Applikation zum Stillstand des gesamten Systems. Für diesen Fall wäre ein Alarm-Mechanismus oder ähnliches von Vorteil, der die zuständige Person darüber informiert, die dann den Fehler behebt und das System neu startet. So kann die Zeitspanne des Stillstandes erheblich verkürzt werden.

7.2.3 Einfache Dienstgenerierung

Die Module mit den Basiskomponenten machen die Entwicklung von neuen Diensten schnell und einfach. Man muss nicht jedes Mal die Syntax für den Datenbankzugriff oder das Zusammenstellen einer E-Mail nachschlagen bzw. den Code dafür aus einer anderen Applikation kopieren. Das mit dem Kopieren verbundene, mühsame Anpassen des kopierten Code-Teils fällt damit auch weg. Durch die Verwendung der Module und ihrer Basiskomponenten wird der Entwicklungsaufwand und die verwendete Menge an Code deutlich reduziert. Und wenn sich neue Teilfunktionen ergeben, können diese einfach in eines der Module integriert werden bzw. ein neues Modul dafür angelegt werden.

7.2.4 Einfache Dienstanpassung

Die Verwendung der Komponenten macht die entwickelten Applikationen leichter lesbar und überschaubar. Für eine einfache Änderung müssen nun nicht mehr ein paar Seiten Code nach der Zeile, die zu ändern ist, durchsucht werden. Und dass nach einer Änderung noch weitere Änderungen an anderen Stellen im Code durchzuführen sind, ist auch nicht mehr gegeben. Bei Erweiterungen ist es nicht viel anders. Soll einer Applikation eine weitere Funktion hinzugefügt werden, so ist nur der Aufruf der entsprechenden Komponente in den Code einzufügen. Falls für die Funktion noch keine Komponente existiert, muss diese natürlich erst implementiert und dann entsprechend integriert werden.

7.2.5 Ereignisgesteuertes Senden

Die entwickelte Regel-Engine ermöglicht die Realisierung von Push-Diensten. Die Definition und persistente Speicherung von Regeln, die das Verhalten eines Push-Dienstes bestimmen, wird unterstützt. Es wird regelmäßig überprüft, ob eines der definierten Ereignisse eingetroffen ist, und die zugeordneten Aktionen werden bei Ereigniseintritt auch tatsächlich ausgeführt. Allerdings ist die Ereignisdefinition nicht ganz ausgereift. Da nur boolesche Ausdrücke erlaubt sind und in einer MySQL-Datenbank keine booleschen Ausdrücke gespeichert werden können, sind die Ereignisdefinitionen in der Datenbank als Zeichenfolge gespeichert. Das hat zur Folge, dass die booleschen Ausdrücke noch einmal extra per Hand in die Klasse *Rulebase* eingefügt werden müssen. Und das ist sicher nicht die beste Lösung. Da zum Zeitpunkt der Implementierung die einzig anwendbaren Ereignisse Zeitereignisse waren, konnte die Regel-Engine auch nur damit getestet werden. Wie sie mit anderen Ereignissen funktioniert, muss sich erst noch zeigen.

7.2.6 Verwaltung von Benutzerinformationen

Die Tabelle *Profile* bietet die Möglichkeit, Informationen über die mobilen Teilnehmer zu speichern und sie für Push-Dienste zu verwenden. Alle Informationen können direkt aus den eingegangenen

Kurznachrichten extrahiert werden. Anhand der Daten kann man durchaus ableiten, welcher Teilnehmer für eine Push-Nachricht als Empfänger in Frage kommt und welcher nicht. In dieser Beziehung sollte man jedoch sehr vorsichtig sein, damit sich kein Teilnehmer belästigt fühlt. Aber bei Push-Diensten ist diese Abschätzung generell sehr schwierig. Stellt sich heraus, dass für einen Push-Dienst noch weitere Informationen über den Teilnehmer gebraucht werden, kann der Tabelle das entsprechende Attribut einfach beigefügt werden.

Kapitel 8

Zusammenfassung und Ausblick

Diese Arbeit wurde in Zusammenarbeit mit BURDAWIRELESS, ein New Media Kompetenzzentrum des Hubert Burda Media Verlags, angefertigt. Im Rahmen dieser Arbeit wurde aufbauend auf der rudimentären SMS-Plattform von BURDAWIRELESS eine Architektur für SMS-basierte Dienste konzipiert und prototypisch umgesetzt.

Einführung und Grundlagen

In der Einführung wurde die grundlegende Problemstellung zusammengefasst und die sich daraus ergebenden Anforderungen kurz dargestellt.

Um eine Wissensgrundlage im Hinblick auf die Vorgänge beim Short Message Service zu schaffen, wurden folgende Bereiche der Mobilkommunikation erläutert:

- Im Bereich der Basistechnologien wurden Eigenschaften der genutzten Frequenzbereiche und der Signalausbreitung beschrieben. Ferner wurden der Begriff der Modulation sowie Multiplex- und Zugriffsverfahren für die gemeinsame Nutzung des Mediums Luft und das Prinzip von zellularen Systemen erklärt.
- Als bekanntestes zellulares System wurde GSM vorgestellt. GSM ist das am weitesten verbreitete Mobilfunksystem und war das erste System, das den Short Message Service bereitgestellt hat. Nach einer Einführung in die Architektur und Beschreibung der Luftschnittstelle des GSM-Systems, wurde die technische Realisierung des SMS aufgezeigt
- Schließlich wurde noch ein Ausblick auf UMTS, das Mobilfunksystem der 3. Generation, gegeben. UMTS ist um einiges schneller und leitungsfähiger als das GSM-System und bietet vor allem in Bezug auf Datendienste weitaus mehr Möglichkeiten.

Entwicklungsumgebung bei Burda

Da die SMS-Plattform die Basis für die Entwicklung war, wurde sich mit der Architektur und den Problembereichen der Plattform eingehender befasst. Die genauen Abläufe und Vorgänge in der Plattform wurden in Form eines typischen Szenarios verdeutlicht. Der Empfang der Kurznachrichten und ihre anschließende Verarbeitung ist durch das Application Manager Framework gut gelöst. Als Problembereiche stellten sich die Erzeugung von neuen Diensten, die Anpassung von bereits realisierten Diensten und die Realisierung von Push-Diensten heraus.

Anforderungen und bestehende Architekturen

Bei der Anforderungsanalyse spielten die Problembereiche der Plattform eine wesentliche Rolle. Insgesamt wurden folgende Anforderungen, die eine Architektur für SMS-basierte Dienste erfüllen sollte, identifiziert.

1. Inhärenterweise muss eine Architektur, die SMS-basierte Dienste zur Verfügung stellt, den Empfang und Versand von Kurznachrichten unterstützen.
2. Eine hohe Verfügbarkeit sollte in jedem Fall gegeben sein, mobile Teilnehmer müssen immer und überall mit der Plattform kommunizieren können.
3. In Bezug auf die Diensterzeugung und die Dienstanpassung stehen Aspekte wie Einfachheit und geringer Aufwand im Vordergrund. Weder das Erzeugen noch das Anpassen von Diensten sollte zeitintensiv und aufwändig sein.
4. In vielen Fällen wäre der Einsatz von Push- anstatt von Pull-Diensten sinnvoll. Das Senden von Push-Nachrichten sollte automatisch bei Eintritt eines vordefinierten Ereignisses erfolgen.
5. Die Basis für Push-Dienste sind Informationen, die etwas über die Interessen der Teilnehmer aussagen. Dafür ist das Sammeln und Speichern von Benutzerinformationen in Profilen nötig.

Diese Anforderungen wurden verwendet, um bestehende Architekturen und Ansätze zu bewerten. Fazit der Bewertung war, dass keine der bewerteten Architekturen alle geforderten Kriterien unterstützt.

Implementierungskonzept

Basierend auf den Gegebenheiten der SMS-Plattform wurde ein Konzept entwickelt, das die Anforderungen umsetzen soll. Um die Erzeugung neuer Dienste und die Anpassung bestehender Dienste einfacher zu gestalten, wurden Module spezifiziert, die wiederverwendbare Komponenten mit genau definierten Funktionalitäten zur Verfügung stellen. Die benötigten Funktionalitäten wurden aus bereits realisierten Diensten abgeleitet. Die Komposition der Komponenten wurde anhand von zwei konkreten Diensten modelliert.

Zur Realisierung der Push-Dienste wurde eine Regel-Engine konzipiert, die das automatische Senden von Nachrichten abhängig von Ereignissen unterstützt. Dafür wurde die Regeldefinition festgelegt und ein Triggermechanismus entwickelt.

Implementierung

Ausgehend von dem entworfenen Konzept wurden Basiskomponenten, die allgemein nutzbar und wiederverwendbar sind, implementiert. Anschließend wurden dann die zwei modellierten Dienste entwickelt. Bei der Entwicklung wurden die bereits implementierten Komponenten zu Hilfe genommen. Dadurch war eine Reduzierung des Entwicklungsaufwands und vor allem der Code-Menge möglich.

Weiterhin wurde der Entwurf der Regel-Engine prototypisch umgesetzt. Der Triggermechanismus wurde so umgesetzt, dass in regelmäßigen Zeitabständen die Ereignisse überprüft werden und bei Eintreffen der Ereignisse die in den Regeln definierten Aktionen zur Ausführung gebracht werden.

Test und Evaluierung

Abschließend wurden die implementierten Dienste und die Regel-Engine auf ihre Funktionsfähigkeit getestet. Dazu wurden geeignete Testdaten in die Datenbank geschrieben. Sowohl der Test der Dienste als auch der Test der Regel-Engine verliefen positiv. Für den Test der Dienste war das Senden einer Kurznachricht nötig. Beide leisteten die gewünschte Funktionalität. Die Regel-Engine hat sich, wie erwartet, im vorgegebenen 5-Minuten-Takt angestoßen und bei Ereigniseintritt die für den Test implementierten Aktionen ausgeführt.

Die Evaluierung der entwickelten Architektur ergab, dass nun jede der aufgeführten Anforderungen Unterstützung findet. Die entwickelten Komponenten bieten eine gute Basis zur Erzeugung neuer Dienste. Diese Basismenge wird sicher nach und nach um weitere Komponenten erweitert. Die Regel-Engine ermöglicht die Realisierung von Push-Diensten, allerdings ist die Abfrage der Ereignisdefinitionen noch nicht automatisiert. Dieser Teil lässt sich sicher noch optimieren.

Ausblick

Wegen der geringen Nutzung des MMS (Multimedia Message Service) beschränken sich die Dienste der Architektur auf das Format SMS. Die vorgesehene Einführung von UMTS Ende dieses Jahres könnte daran sehr schnell etwas ändern. Ob MMS-basierte Dienste einen ähnlichen Zuspruch finden wie SMS-basierte Dienste ist noch nicht abzusehen, aber durchaus denkbar. Eine Erweiterung der Plattform in Richtung MMS-basierte Dienste ist in absehbarer Zukunft zu erwarten.

Abkürzungsverzeichnis

AMPS Advanced Mobile Phone System

ATM Asynchronous Transfer Mode

BSC Base Station Contoller

BSS Base Station System

BTS Base Transceiver Station

CCH Control Channel

CDM Code Division Multiplexing

CDMA Code Division Multiple Access

CM Call Management

DCS Digital Cellular System

DECT Digital Enhanced Cordless Telecommunications

ETSI European Telecommunications Standards Institute

FDM Frequency Division Multiplexing

FDMA Frequency Division Multiple Access

GMSC Gateway MSC

GPRS General Packet Radio Service

GSM Global System for Mobile Communications

HLR Home Location Register

HSCSD High Speed Circuit Switched Data

IMT International Mobile Telecommunications

IP Internet Protocol

ISDN Integrated Services Digital Network

ITU International Telecommunication Union

IWF Interworking Function

IWMSC Interworking MSC

LA Location Area

MM Mobility Management

MMS Multimedia Message Service

MS Mobile Station

MSC Mobile Switching Center

NSS Network and Switching Subsystem

OSS Operation Subsystem

PLMN Public Land Mobile Network

PSTN Public Switched Telephone Network

RR Radio Resource

RSS Radio Subsystem

SACCH Slow Associated Dedicated CCH

SDCCH Stand-alone Dedicated CCH

SDM Space Division Multiplexing

SDMA Space Division Multiple Access

SM MO Short Message Mobile Originated

SM MT Short Message Mobile Terminated

SM-AL Short Application Relay Layer

SM-RL Short Message Relay Layer

SM-TL Short Message Transfer Layer

SME Short Message Entity

SMS Short Message Service

SMSC Short Message Service Center

SS Supplementary Service

SS7 Signaling System No. 7

TCH Traffic Chanel

TDM Time Division Multiplexing

TDMA Time Division Multiple Access

TE Terminal

TPDU Transfer Protocol Data Unit

UMTS Universal Mobile Telecommunications System

VLR Visitor Location Register

WAP Wireless Application Protocol

Literaturverzeichnis

[AM 02] AMBERG M., FIGGE S. UND WEHRMANN J.: *Compass - Ein Kooperationsmodell für situationsabhängige mobile Dienste*. Technischer Bericht, Friedrich-Alexander Universität Nürnberg, 2002.

[BR 99] BHASKARAN R., KATZ R. UND ANTHONY J.: *Personal Mobility in the ICEBERG Integrated Communication Network*. Technischer Bericht, University of California, Berkeley, 1999.

[CK 01] CURRAN K., CRAIG R.: *A Short Message Service Application for Delivering Urgent Information to Stundents*. Technischer Bericht, University of Ulster, Nothern Ireland, 2001.

[ETSI] European Telecommunications Standard Institute, http://www.etsi.org .

[ETSIa] *Point-to-Point Short Message Service (SMS), V7.1.0.* ETSI Rec. GSM 03.40, 1998.

[ETSIb] *Technical Realization of the SMS Point-to-Point, V7.5.0.* ETSI Rec. GSM 03.40, 1998.

[FM 98] FRANKLIN M., ZDONIK S.: *'Data in Youf Face': Push Technology in Perspective*. International Conference on the Management of Data, 1998.

[HM 02] HOFMANN M., RÖSSLER C.: *Gestaltung und Entwicklung proaktiver Dienste*. Technischer Bericht, Friedrich-Alexander Universität Nürnberg, 2002.

[HS 03] HÜBENTHAL S., KAHNT T. UND RYLL R.: *Mobile Datennetze -GSM, CEDT, UMTS*. Technischer Bericht, TFH Berlin, 2003.

[ITU] *International Mobile Communiations, Set of Recommendations*. International Telecommunications Union, 2000, http://www.itu.int/imt .

[Java] *Java-Site*, http://java.sun.com .

[Kern 95] KERNER, H.: *Rechnernetze nach OSI.* Addison-Wesley, 1995.

[KM 02] KEIDL M., SELTZSAM S., STOCKER K. UND KEMPER A.: *Web Services*, Kapitel 10. dpunkt-Verlag, 2002.

[Krat 95] KRATZ, R.: *Adaptation and Mobility in Wireless Information Systems*. Technischer Bericht, University of California, Berkeley, 1995.

[Krus 94] KRUSCH, W.: *Neue Dienste im intelligenten Telefonnetz.* R. v. Decker, 1994.

[ME 03] MARQUARDT E., THEOBALD K. UND ZURTH M.: *Mobile Datennetze - Nahbereichstechniken*. Technischer Bericht, TFH Berlin, 2003.

[MySQL] *MySQL-Site*, http://www.mysql.de .

[OPC] OPC FOUNDATION: *OPC Data Access Automation Interface Standard, V.2.02*, 1999.

[OS 02] OKUJAVA S., WIENER M.: *Architektur und Modllierung einer Kommunikationsschnittstelle in XML*. Technischer Bericht, Friedrich-Alexander Universität Nürnberg, 2002.

[Papa 02] PAPADOPOULOS, V. KAPSALIS S. KOUBIAS G.: *OPC-SMS : A wireless Gateway to OPC-based Data Sources*. Elsevier Science B. V., Seiten 437–451, 2002.

[RB 02] RAMAN B., AGARWAL S.: *The SAHARA Model for Service Composition Across Multiple Providers*. Technischer Bericht, University of California, Berkeley, 2002.

[RE 02] RAHM E., VOSSEN G.: *Web & Datenbanken*. dpunk-Verlag, 2002.

[RH 01] RAO H., CHANG D., CHEN Y. UND CHEN M.: *iMobile: A Proxy-Based Platform for Mobile Services*. Technischer Bericht, University of California, 2001.

[Roth 02] ROTH, J.: *Mobile Computing - Grundlagen, Technik, Konzepte*. dpunkt-Verlag, 2002.

[Schi 00] SCHILLER, J.: *Mobilkommunikation*. Addison-Wesley, 2000.

[SendXMS] *SendXMS-Site*, http://www.sendxms.de .

[Sieg 01] SIEGMUND, G.: *Intelligente Netze - Technik, Dienste, Vermarktung*. Hüthig, 2001.

[Walk 98a] WALKE, B.: *Mobilfunknetze und ihre Protokolle - Band1*. Teubner Stuttgart, 1998.

[Walk 98b] WALKE, B.: *Mobilfunknetze und ihre Protokolle - Band2*. Teubner Stuttgart, 1998.

[Wehr 01] WEHRMANN, J.: *Situationsabhängige Dienste - Grundlagen ihrer Entwicklung*. Technischer Bericht, FAU Nürnberg, 2001.

[Zobe 01] ZOBEL, J.: *Mobile Business und M-Commerce*. Hanser-Verlag, 2001.