Projektarbeit

Instant Messaging System

Pa1 Sna01

Fabio Soldati, IT3a
Martin Seelhofer, IT3a

Project Summary

Aufgabe unserer Projektarbeit war es, ein Instant Messaging System zu entwickeln, welches in der Lage ist, mit mobilen Teilnehmern (via SMS) zusammenzuarbeiten. Dabei lag das Hauptinteresse unserer Partnerfirma, der Enterprise Communications AG aus Winterthur, im Lösen des sogenannten Reply-Problems. Denn sendet ein Benutzer, der kein Handy besitzt, eine Nachricht aus einem LAN über einen Gateway (z.B. über ein Short Message Service Center oder ein GSM-Modem) an einen mobilen Teilnehmer, so fehlt letzterem eine eindeutige Nummer, an welche er eine mögliche Antwort senden könnte. Vom Instant Messenger ICQ war aber bekannt, dass es einen Weg geben muss, das Reply-Problem zu lösen.

Beim Auseinandersetzen mit der angesprochenen Problematik haben wir vier mögliche Lösungen gefunden und genauer analysiert. Diese lauten „Empfänger in der Nachricht kodieren“, „Unbenutzte Protokollfelder missbrauchen“, „Routing mit einem GSM-Modem-Array“ und „Verwenden eines speziellen SMSC, welches das Anhängen von eigenen Nummern an die Service-Center Nummer erlaubt“. Die eleganteste Methode, zumindest was das Beantworten von eintreffenden Nachrichten betrifft, ist sicher die zuletzt erwähnte. Sie wird auch von ICQ verwendet.

Während der Projektarbeit haben wir festgestellt, dass MTN (Mobile Telephone Networks) aus Südafrika im Moment der einzige Anbieter von Telekommunikations-Diensten zu sein scheint, welcher das Anfügen von eigenen Nummern an die SMSC-Nummer erlaubt. Diese Lösung des Reply-Problems wäre also nur in Frage gekommen, wenn MTN dazu bereit gewesen wäre, mit uns zusammenzuarbeiten. Trotz diversen An- und Nachfragen konnten wir aber weder eine klare Ab- noch eine definitive Zusage erwirken, so dass wir uns für eine andere Lösung entscheiden mussten. Aus Kostengründen fiel die Lösung mit dem GSM-Modem-Array weg. Die Lösung mit den unbenutzten Protokollfeldern mussten wir nach einer langen Testserie ebenfalls begraben, so dass als einziges noch das manuelle Eingeben des Empfängers im Nachrichtentext übrigblieb.

Das Instant Messaging System haben wir so modular wie möglich aufgebaut, damit ein Technologie- oder Formatwechsel so einfach wie möglich vollzogen werden kann. Auf Front-End-Seite haben wir ein kleines HTML-Formular entworfen, welches dem Versand von Nachrichten und dem Verwalten von Verteilerlisten dient. Als Datenbank-Back-End haben wir die Active Directory Services von Windows 2000 und Exchange verwendet. Dazwischen arbeiten einige Dienste und COM-Objekte beim Abfragen von Informationen und dem Empfangen und Versenden von Nachrichten zusammen. Für das Initiieren eines Nachrichtenaustauschs auf der Seite des mobilen Teilnehmers haben wir ein spezielles Format definiert, welches wir mit regulären Ausdrücken beschrieben haben.

Zum Projektablauf kann gesagt werden, dass wir abgesehen von der nicht zustande gekommenen Zusammenarbeit mit MTN keine grösseren Schwierigkeiten zu bewältigen hatten. Dies kann einerseits darauf zurückgeführt werden, dass wir beide schon seit längerem im Bereich Software-Erstellung tätig sind und deshalb sicher schon einen gewissen Erfahrungsschatz besitzen, andererseits aber auch auf die gewählten Technologien. So hat sich unser loses System-Design in diversen Belangen als äusserst ideal erwiesen. Die Integrationsphase beanspruchte gerademal einen halben Tag.

Inhaltsverzeichnis

Vorwort

Die vorliegende Dokumentation ist das Resultat unserer Projektarbeit (PA) zum Thema ‚Instant Messaging mit SMS'. Die PA ist aufgrund einer Anfrage der Enterprise Communications AG, Winterthur entstanden, welche vor allem an einer Analyse und möglichen Lösungen für Probleme im Zusammenhang mit der Kommunikation mittels Kurznachrichten (SMS) zwischen Computer- und Handy-Benutzern, interessiert war.

Aufbau

Das Dokument ist im Wesentlichen in die folgenden vier Teile gegliedert:

- Projektbeschrieb und Pflichtenheft
- Analyse des SMS-Reply-Problems
- Softwaredesign vom Instant Messaging Service
- Grundlagen der eingesetzen Technologien

Da die Enterprise Communications AG in erster Linie an der Analyse des SMS-Reply-Problems interessiert war, haben wir diese sehr detailliert dokumentiert, in der Hoffnung, dass diese Informationen weiterverwendet können.

Auf das Ausdrucken des ganzen SourceCodes haben wir absichtlich verzichtet. Vielmehr haben wir den Code mit einem Dokumentations-Tool namens DoxyGen (ähnlich wie Doc++ unter Java) zu einheitlichen HTML-Seiten zusammengefügt, die alle auf der Begleit-CD abgelegt sind. Die grundlegenden Technologien, die wir eingesetzt haben und besonders interessante Stellen im SourceCode sind zusätzlich in den Grundlagen-Kapiteln beschrieben.

Dank

An dieser Stelle möchten wir uns für die angenehme Zusammenarbeit mit Herrn Dr. Andreas Steffen von der ZHW und Herrn Claudio Grazioli von der Enterprise Communications AG (ECAG) bedanken. Ein spezieller Dank geht auch an Patrik Dreyer von der ECAG für seine Unterstützung mit den regulären Ausdrücken, seiner XMiddleware und dem DoxyGen Dokumentations-Tool.

Fabio Soldati , IT3a

Martin Seelhofer, IT3a

Kapitel 1 Das Projekt

1.1 Projektbeschreibung

Es soll ein Instant Messaging System entwickelt werden, bei dem neben Arbeitsstationen auch Mobiltelefone über SMS eingebunden werden können.

Das System soll aus folgenden Komponenten bestehen:

- **Server Applikation** unter Windows 2000 mit primitiver Benutzerverwaltung (z.Bsp. Microsoft Exchange), sowie einem GSM Modem als Windows 2000 System Service.
- **Client Applikation** unter Windows, die sich beim Server anmeldet und mit der Meldungen gesendet und empfangen werden können.
- **Mobiltelefone**, welche die SMS Nachrichten empfangen und auch so beantworten können, dass die SMS-Antwort als Meldung in der Client Applikation erscheint.

Die Anwendung soll ähnlich zur Instant Messaging/SMS Funktionalität von ICQ ausgestaltet werden. Wird eine SMS-Nachricht von einem Mitarbeiter A aus der Client Applikation heraus an einen Mobilteilnehmer B gesendet, so soll die SMS-Antwort ihren Weg zurück zu A finden, obwohl das GSM-Modem der Server Applikation nur eine einzige GSM-Nummer für besitzt und den Verkehr für alle Clients Applikationen behandelt.

1.1.1 Aufgaben:

- Einarbeiten in die SMS-Thematik. Installation der Infrastruktur.
- Definition eines Pflichtenhefts auf der Basis der Kundenbedürfnisse.
- Erstellen einer SW-Spezifikation auf der Basis des Pflichtenhefts.
- Definition der Schnittstellen und Module.
- Einbindung von Third-Party Modulen und Applikationen.
- Objektorientiertes Klassendesign.
- Implementation und Austesten der Anwendung.
- Erstellen einer kurzen Installations- und Bedienungsanleitung.
- Dokumentation der einzelen SW-Module und der Gesamtapplikation.

1.2 Vorstudie

Die Vorstudie nahm etwa die ersten zwei Wochen der Projektarbeit in Anspruch. Dabei haben wir eine Liste von möglichen Ansätzen und Lösungen erstellt, welche wir hier präsentieren möchten. Einige Punkte sind auch erst im späteren Lauf der Projektarbeit dazugekommen.

1. SMS-Reply-Problem lösen:
    a. Large Account bei einem Mobile-Provider (vgl. ICQ: Nr. hinter spez. Service-Nummer). Abklären mit Schweizer Providern (Swisscom, Diax, Orange)
    b. Freies SMS-Protokoll-Feld missbrauchen.
    - Was ist UCP?
    - Wie ist UCP aufgebaut?
    - SMS-Message-Reference-Feld zur Erkennung der Verbindung benutzen.
    c. Lösung mit mehreren Modems, bzw. Moblienummern prüfen. Gewisse Handys unterstützen bis zu sechs verschiedene Nummern.
    d. Empfänger jeweils als erstes in die SMS eintragen.

2. Benutzerverwaltung:
    a. Exchange (mit Exchange Benutzerverwaltung)
    b. Eigenes DB-Format (Performance nebensächlich, aber Konsistenz?)
    c. Active Directory von Windows 2000

3. GSM-Modem:
    a. Installation
    b. Modem-Treiber ‚erkunden'
    c. Middleware (evtl. sogar Treiber) selbst schreiben?
    d. SIM-Karte auftreiben

4. Client-Applikation:
    a. Outlook-COM-AddIn (eigene Maske)
    b. Web-Client (ASP)
    c. normales Dialogfenster (mit VC++ und MFC)

5. Zugriff auf die Benutzerverwaltung:
    a. LDAP
    b. SQL
    c. ADSI

6. Format der Nachrichtendatei definieren:
    a. einfaches Textfile
    b. ASCII-File
    c. XML-File

7. Optional: Konfiguration des Server-Dienstes über SMS (z.B. neue Gruppe einrichten)

1.3 Pflichtenheft

Ausgehend von der in der Vorstudie erstellten Liste haben wir ein Pflichtenheft verfasst, welches die Aufgaben des zu erstellenden Instant Messaging Systems anwendungsorientierter wiedergibt. Dieses Pflichtenheft ist ausschliesslich aus unseren eigenen Ideen entstanden. Natürlich müsste beim Verkaufen einer solchen Lösung mindestens noch die Partnerfirma beigezogen werden, um die Richtigkeit und Vollständigkeit des Pflichtenhefts zu überprüfen. Aufgrund der kurzen uns zur Verfügung stehenden Zeit haben wir aber auf dieses Vorgehen verzichtet.

Das zu erstellende Instant Messaging System soll folgende Anforderungen erfüllen:

1. Zugriff auf Active Directory:
    a. Authentifizierung (Benutzername, Passwort).
    b. Daten von einem Kontakt auslesen (E-Mail Adresse, Mobilenummer).
    c. Über E-Mail-Adresse oder Mobilenummer den Benutzernamen auslesen.
    d. neue Verteilerliste erstellen.
    e. Benutzer einer Verteilerlisten hinzufügen.
    f. Aus Sicherheitsgründen ist das erstellen von neuen Benutzern nicht erlaubt.

2. Erstellen eines Jobs (IMSManager):
    a. Testen ob Sender und Empfänger für diesen Dienst berechtigt sind.
    b. Auflösen von Benutzernamen über E-Mail-Adressen bzw. Mobilenummer.
    c. Erstellen eines Jobs für die Outbox.

3. Web-Client:
    a. Login
    b. Neue Meldung erstellen:
    - Subject- und Text-Feld.
    - Liste aller Gruppen, bzw. alle Benutzer.
    - Gruppen und Benutzer können in Empfängerfeld gefügt werden.
    - Auswahl zwischen E-Mail- und SMS-Meldung.
    - Anzeige verbleibender Anzahl Buchstaben (Grenze für SMS: 160 Zeichen).

c. Verteilerlistenverwaltung:
   - neue Verteilerlisten erstellen.
   - Benutzer hinzufügen, bzw. löschen.

4. SMS-Empfänger: (SMSReceiver)
   a. Ansteuerung des M20 Modems.
   b. Auslesen neuer SMS-Meldungen vom Modem.
   c. Meldungstyp und Empfängernamen aus dem Meldungstext lesen.
   d. Job an IMSManager weiterleiten.

5. Outbox überwachen (Outboxlistener)
   a. Prüfen ob neue Jobs in der Outbox liegen.
   b. Job Verarbeiten:
      - als E-Mail über der Exchangeserver versenden
      - oder als SMS über die EasySMS.dll versenden

1.4 Zeitlicher Ablauf

Auf der nächsten Seite ist der geplante Ablauf der Projektarbeit dem effektiven gegenübergestellt.

1.5 Projektstand

Aufgrund des reibungslosen Ablaufs unserer Projektarbeit waren wir in der Lage, alle im Pflichtenheft erwähnten Punkte wenigstens ansatzweise in unsere Instant Messaging-Lösung einzubauen. Wir möchten aber trotzdem nicht darauf verzichten, im nächsten Kapitel noch einige Verbesserungsvorschläge zu erwähnen.

Instant Messaging System
Projektarbeit PA1 2001

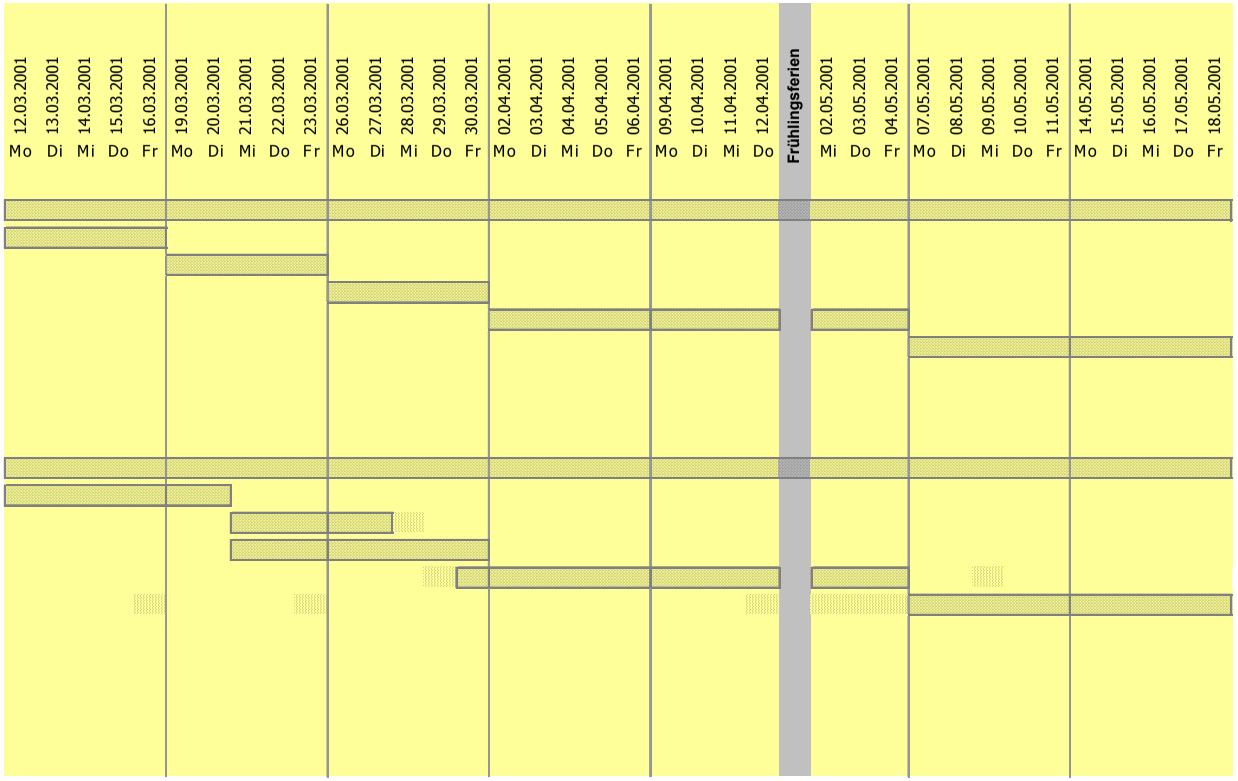
Fabio Soldati, IT3a
Martin Seelhofer, IT3a
Zürcher Hochschule Winterthur

Geplanter Ablauf

Projektarbeit 1 / Sna 01	8 Wochen
Einlesen ins Thema der PA / Installation	1 Woche
Installation / Konfiguration Server / Analyse	1 Woche
Moduldesign / Erste Implementationen	1 Woche
Implementation / Tests	2.5 Wochen
Dokumentation / Bug-Fixes / Zus. Features	2 Wochen

Effektiver Ablauf

Projektarbeit 1 / Sna 01	8 Wochen
Einlesen ins Thema der PA / Installation	1.5 Wochen
Installation / Konfiguration Server / Analyse	1 Woche
Moduldesign / Erste Implementationen	1.5 Wochen
Implementation / Tests	3 Wochen
Dokumentation / Bug-Fixes / Zus. Features	2 Wochen



1.6 Verbesserungsvorschläge – ToDo

In diesem Abschnitt werden einige Vorschläge gemacht, wie das entstandene Instant Messaging System verbessert werden könnte. Obwohl darin z.T. konkrete Implementationsdetails referenziert werden, zu welchen die Grundlagen erst in späteren Kapiteln nachgereicht werden, gehört dieses Kapitel zum Bereich Projektmanagement. Aus diesem Grund haben wir es auch an dieser Stelle platziert und nicht an den Schluss der Dokumentation geschoben.

1.6.1 Logfile

Um die Administration des IMS zu vereinfachen und die Betriebssicherheit zu erhöhen ist noch eine Log-Datei nötig. Beim aktuellen Projektstand werden nur Fehlermeldungen an das Windows-Ereignisprotokoll weitergeleitet. Informationsmeldungen werden zur Zeit nur im Debug-Modus auf der Konsole angezeigt. Diese Meldungen beinhalten Informationen über neue Jobs, unbekannte Sender oder Empfänger, Status vom Modem, usw. Weil aber das Ereignisprotokoll nicht für grosse Mengen von Einträgen gedacht ist, sollten die Informationsmeldungen in eine separate Log-Datei geschrieben werden, die bei Bedarf vom Systemadministrator eingesehen werden kann.

1.6.2 Fehlermeldung an Benutzer

Der Benutzer sollte benachrichtigt werden, wenn ein Fehler aufgetreten ist. Wenn zum Beispiel ein Handybenutzer ein ungültiges Format in seine SMS angibt oder der gewünschte Benutzer nicht existiert, sollte er darüber benachrichtigt werden. Dies könnte einerseits über ein SMS (was allerdings Mehrkosten mit sich bringt) oder eine E-Mail geschehen. Anderseits könnte eine benutzerspezifische Log-Auflistung über das Web-Interface angeboten werden, die vom jeweiligen Anwender eingesehen werden kann.

1.6.3 Meldungsart

Der Empfänger sollte bestimmen können wie er eine Nachricht empfangen will, das heisst als E-Mail oder als SMS. Dies kann ähnlich wie bei ICQ über ein kleines Dialogfenster geschehen. Man könnte auch über Active Directory feststellen, ob ein Benutzer sich eingeloggt hat. Falls dies der Fall ist, geht man davon aus, dass der Empfänger an seinem Arbeitsplatz ist und eine Meldung als E-Mail empfangen kann.

1.6.4 Addin für Outlook

Zusätzlicher Client als Addin für Outlook erstellen. Dieses Addin muss über die gleichen Funktionen wie der bereits bestehende Web-Client aufweisen.

1.7 Ausblick

Da der Telekommunikationsmarkt sich immer noch in einem sehr starken Wandel befindet, wird die Zukunft mit Sicherheit neue Möglichkeiten eröffnen, die für unsere Projektarbeit von Bedeutung sein können. Vor allem für das SMS-Reply-Problem wird es vielleicht elegantere Lösungen geben:

Zum einen hat MTN South Africa durchblicken lassen, dass sie in Zukunft ihr SMSCenter auch für andere Kunden freilegen werden. Dann müsste man einfach das Prinzip von ICQ übernehmen. Dies könnte für kleine Unternehmen eine interessante Lösung sein.

Eine andere interessante Alternative könnte das programmieren der Handy-SIM-Karte sein. Diese Möglichkeit haben wir schon während der Projektarbeit kurz ins Auge gefasst, dann aber fallengelassen, weil das Einarbeiten zu umfangreich gewesen wäre. Aktuelle SIM-Karten können anscheinend mit Java programmiert werden. Für das Reply-Problem müsste man die Empfangeseinheit des Handy dazu bringen, dass sie dem SMS eine Zusatzinformation hinzufügt, die dann zum ‚Zurück-Routen' wieder benutzt werden kann.

Kapitel 2 Das SMS-Reply Problem

Der Anwendungsfall für die Reply-Funktion sieht folgendermassen aus:

Ein Anwender sendet von seinem Computer direkt über das SMSC (Short Message Service Center) eine SM (Short Message) an eine MS (Mobile Station). Der Empfänger kann nun über die Reply-Funktion seiner MS eine Antwort senden, die als E-Mail beim ursprünglichen Sender wieder ankommt.

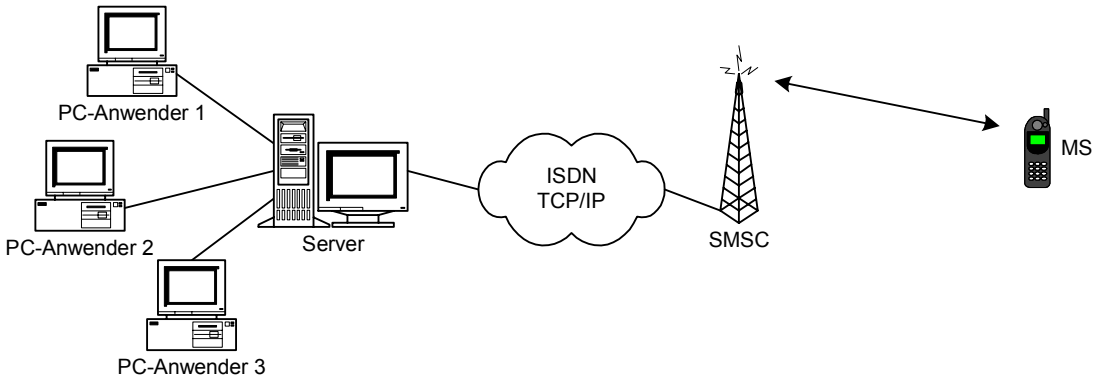


Abbildung 1: SMS Reply

Das Problem ist nun, dass die SM von der MS wieder den Weg zum richtigen PC-Anwender zurück findet. Wenn die MS eine Reply-Meldung an den Server zurück sendet, muss dieser wissen, welcher Anwender die SM ursprünglich abgesendet hat. Um dies zu ermöglichen muss eine zusätzliche Routing-Information in der SM mitgegeben werden. Von dem Instant Messenger ICQ war uns bekannt, dass eine brauchbare Möglichkeit für das Routing existiert. Bei der Analyse dieses Problems haben wir vier Möglichkeiten gefunden, die in den folgenden Abschnitten aufgezeigt werden.

2.1 Lösungsansätze

2.1.1 Methode 1 – Ziel im SMS angeben

Beschreibung

Dies ist die wohl uneleganteste aller Methoden. Um den Weg zum Sender zurück zu finden, muss der Handybesitzer den Empfänger im SMS angeben. Um nicht allzu lange Empfängername zu erhalten, kann man für den Empfänger einen Kurznamen nehmen. Zum Beispiel könnte man hier an der Fachhochschule die Zeichenfolge vor dem ‚@zhwin.ch' Email-Namen nehmen. Ein SMS an johndoe@zhwin.ch würde zum Beispiel wie folgt beginnen:

```
johndoe Hallo John, wie geht's?
```

Fazit

Vorteile: Man kann den Empfänger auch erreichen wenn kein SMS von ihm empfangen wurde. Als einzige aller Methoden die wir getestet haben, kann man auf diese Weise auch ein PC-Benutzer erreichen ohne dass man bereits eine SM von ihm erhalten hat. Dies kommt natürlich daher, dass diese Methode gar keine wirkliche Reply-Antwort erstellt.

Nachteile: Man muss den Kurznamen des Empfängers kennen.

Mühsames eingeben des Kurznamens über die Handytastatur. Bei einigen Herstellern wie Nokia, kann dieses Problem mit sogenannten Textbausteinen entschärft werden. Diese Textbausteine können vordefinierte Texte enthalten. In unserem Fall also der Name des Empfängers.

2.1.2 Methode 2 – Unbenutzte Bytes im SMS-Protokoll missbrauchen

Beschreibung

In vielen Protokollen werden meist einige Reserven-Bytes für zukünftige Anwendungen spezifiziert. Ein solch unbenutztes Byte würde vollständig ausreichen um den Server mit genügend Routing-Infomationen zu versorgen. Wie unsere Analyse des SMS-Protokolls gezeigt hat, gingen die Designer dieses Protokolls sehr sparsam mit solchen Reserven-Bytes um. Dies liegt natürlich daran, dass SMS ursprünglich nur als Benachrichtigung für neue Nachrichten in der Voicebox gedacht war.

Das SMS-Protokoll

Für SMS Übertragungen stehen sechs verschiedene PDUs (Protocol Data Units) zur Verfügung:

- **SMS-DELIVER:** überträgt eine SM vom SMSC zur MS.
- SMS-DELIVER-REPORT: überträgt eine Bestätigung (falls notwendig).
- **SMS-SUBMIT:** überträgt eine SM von der MS zum SMSC.
- SMS-SUBMIT-REPORT: überträgt eine Bestätigung (falls notwendig).
- SMS-STATUS-REPORT: überträgt eine Status Meldung.
- SMS-COMMAND: überträgt ein Kommando von der MS zum SMSC.

Für unsere Betrachtungen sind nur SMS-DELIVER und SMS-SUBMIT von Bedeutung. Der PDU-Header ist bei beiden fast identisch:

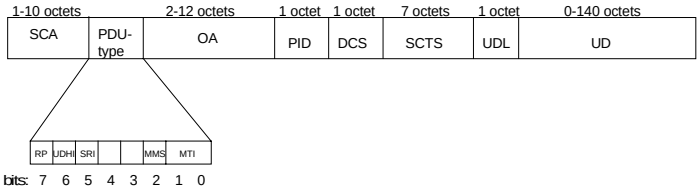


Abbildung 2: PDU-Header SMS-DELIVER

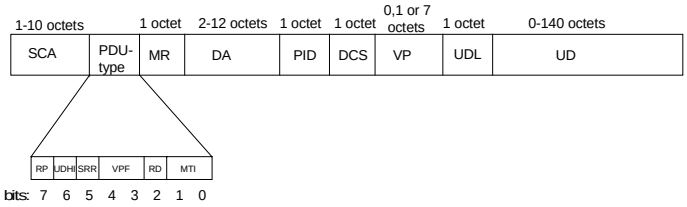


Abbildung 3: PDU-Header SMS-SUBMIT

Abkürzung	Bezeichnung	Beschreibung
SCA	Service Center Adress	Telefonnummer vom Service Center (optional)
PDU Type	Protocol Data Unit Type	
- RP	Reply Path	Gesetzt wenn der Reply Path benutzt werden soll.
- UDHI	User Data Header Indicator	Gesetzt wenn das UD ein Header enthält.
- SRI	Status Report Indication	Gesetzt wenn das SME einen Status Report will
- SRR	Status Report Request	Gesetzt wenn die MS einen Status Report will
- VPF	Validity Period Format	Gesetzt wenn die Validity Period angegeben ist
- MMS	More Messages to Send	Gesetzt wenn noch Meldungen folgen
- RD	Reject Duplicate	Verweigert Duplikate
- MTI	Message Type Indicator	Beschreibt den Meldungstyp: 00: SMS-DELIVER 01: SMS-SUBMIT
MR	Message Reference	Aufsteigende Nummer (0..255) von SMS-SUBMIT Meldungen. Wird vom SMSC gesetzt.
OA	Originator Adress	Adresse vom ursprünglichen SME
DA	Destination Adress	Ziel-Adresse
PID	Protocol Identifier	Setzt den Typ der SM (FAX, Voice etc)
DCS	Data Coding Scheme	Bestimmt den Typ der UserData
SCTS	Service Center Time Stamp	TimeStamp der vom SMSC gesetzt wird
VP	Validity Period	Gibt die Gültigkeitsdauer der SM an
UDL	User Data Length	Länge der Daten (UD)
UD	User Data	Daten von der SM

Erläuterungen zu speziellen Parametern

Im folgenden Abschnitt werden nur die Parameter beschrieben, bei denen man zusätzliche Informationen anfügen könnte oder die sonst für das Reply-Problem von Bedeutung sind.

Format von SCA, OA, DA

Diese Telefonnummernfelder haben alle das gleiche Format. Wie aus Abbildung 4 hervorgeht, sind jeweils zwei Nummern in einem Byte untergebracht und dies in umgekehrter Reihenfolge. Falls eine ungerade Anzahl Nummern vorliegt, muss das zweitletzte Nibble den Wert Fh enhalten.

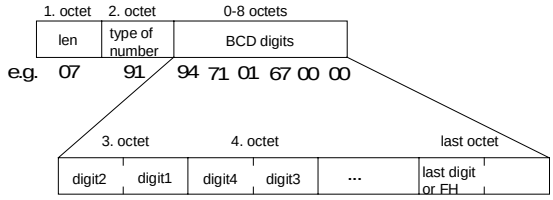


Abbildung 4: Format der Telefonnummern

Type of Number:

81H: die folgende Nummer ist International

91H: die folgende Nummer ist National

SCA – Service Center Adress

Mit diesem Parameter kann angeben werden über welches SMSC die SM gesendet werden soll. Dieser Parameter ist optional. Wenn nichts angeben wird, benutzt die MS den Default-Wert, der auf der SIM-Karte gespeichert ist.

RP – Reply Path

Wenn der Reply Path gesetzt ist, sendet die MS die Antwort über das im SCA angegebene SMSC senden.

PID – Protocol Identifier

Der PID wird gemäss unseren Informationen von den meisten SMSC nicht unterstützt. Der Wert ist in der Regel 00H.

DCS – Data Coding Sheme

Im DCS wird angegeben wie die User Data interpretiert werden soll. Es kann zwischen 7 und 8Bit Daten gewählt werden. Das DCS Feld wir auch häufig benutzt um dem Handy mitzuteilen, dass herstellerspezifische Daten folgen. So signalisieren zum Beispiel Nokia Handys über das DCS, dass Bilder- oder Klingeltönedaten im Datenfeld stehen, die dann von der MS ausgewertet werden müssen.

Beispiele für SM im PDU Format

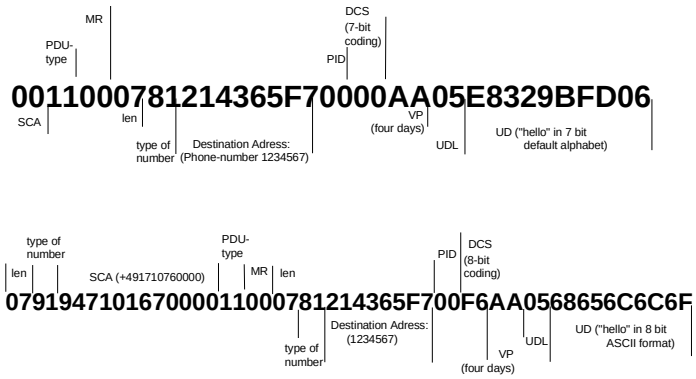


Abbildung 5: Zwei Beispiele über PDU

Möglichkeiten, Tests und Ergebnisse

Telefonnummer erweitern

Beschreibung:

Beim Telefon kann man an die normale Nummer weitere Ziffern anhängen, die von der Zentrale an den Empfänger weitergeleitet werden. Dasselbe haben wir über die MS getestet.

Resultat:

Unsere Tests haben gezeigt, dass SM mit zulangen Nummern von den SMSCs erst gar nicht weitergeleitet werden.

Service Center Adress erweitern

Beschreibung:

Über Service Center Adress kann ein anderes SMSC gewählt werden. So kann man zum Beispiel festlegen, dass eine Relpy-Meldung über das SMSC von Swisscom zurück kommen muss. Dazu ist zusätzlich das Reply-Path-Flag auf 1 zu setzen.

Resultat:

Unsere Tests haben gezeigt, dass die meisten SMSC die zu langen Nummern nicht akzeptieren. Ein Ausnahme scheint hier www.mtn.co.za zu sein (Siehe Kapitel 2.1.4), die für ICQ einen speziellen Dienst eingerichtet haben.

Message Referenz Feld überschreiben

Beschreibung:

Das Message Referenz Feld wird vom SMSC benutzt um die Nachrichtenübertragung zu überwachen.

Resultat:

Gemäss unseren Tests wird das MR-Feld von der SMSC mit einem eigenen Wert überschrieben.

Data Coding Sheme

Beschreibung:

Im DCS wird angegeben wie die User Data interpretiert werden soll.

Resultat:

Für diesen Parameter haben wir herausgefunden, dass die Hersteller dieses Feld für Handy-spezifische Übertragungen benutzen. Bei Nokia werden z.B. Klingelmelodien und Bilder über diesen Parameter spezifiziert. Für eine Hersteller-unabhängige Lösung kann dieser Parameter also auch nicht genutzt werden.

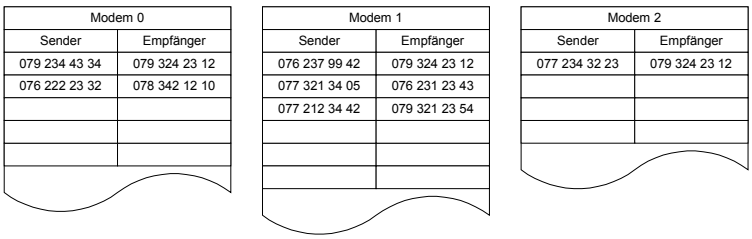
Fazit

Unbenutzte Bytes im SMS-Protokoll, die man für das Routing missbrauchen kann, scheint es nicht zu geben. Einerseits scheiterte es an den SMSCs, die zulange Telefonnummern nicht unterstützen. Anderseits haben viele Handyhersteller eigene Funktionen in die SMS-Übertragung implementiert. Dies macht es sehr schwierig eine brauchbare Lösung für alle Handytypen anzubieten. Auch unsere Nachforschungen auf dem Internet haben gezeigt, dass mehrere andere Projekte diesen Lösungsansatz fallengelassen haben.

2.1.3 Methode 3 – Routing mit mehreren Telefonnummern

Beschreibung

Am SMS-Routing-Server sind mehrere Modems angehängt mit verschiedenen Telefonnummern. Im Normalfall sollten etwa 10 Modems ausreichen. Dazu muss der Server für jedes Modem eine Routing-Liste mit den Empfängern und Sendern verwalten.



Modem 0	
Sender	Empfänger
079 234 43 34	079 324 23 12
076 222 23 32	078 342 12 10

Modem 1	
Sender	Empfänger
076 237 99 42	079 324 23 12
077 321 34 05	076 231 23 43
077 212 34 42	079 321 23 54

Modem 2	
Sender	Empfänger
077 234 32 23	079 324 23 12

Abbildung 6: Routing Listen für Modems

Im folgenden Flussdiagramm wird der Ablauf aufgezeichnet:

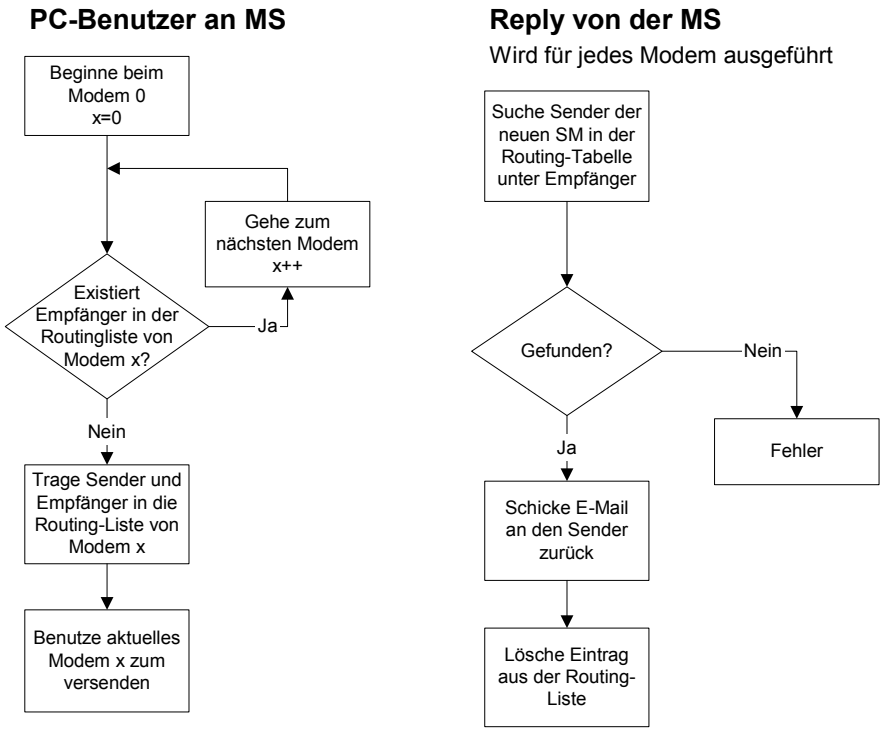


Abbildung 7: Ablauf für mehrere Modems

In dem Flussdiagramm wird davon ausgegangen, dass es genügend freie Modems hat und das jeder Empfänger eine Antwort schickt. In der Praxis wird dies wohl kaum der Fall sein. Um trotzdem einen reibungslosen Ablauf zu gewährleisten muss in die Routingtabelle ein zusätzlicher Timestamp eingefügt werden. Wenn nun kein freies Modem mehr gefunden werden kann, wird nach dem ältesten Eintrag gesucht und dieser durch den neuen Wert ersetzt.

Fazit

Vorteile: Diese Methode funktioniert ohne zusätzlichen Kurznamen im SMS-Text. Wie www.ecall.ch beweist, wird diese Methode auch in der Praxis eingesetzt.

Nachteile: Man braucht mehrere Modems. Für KMUs ist dies kaum eine empfehlenswerte Lösung. Die Anschaffung der Modems und die Monatsgebühren für die SIM-Karten würden den Nutzen für die meisten kleineren Betriebe wohl bei Weitem überschreiten.

2.1.4 Methode 4 – Über spezielles SMSC

Beschreibung

Bei der Analyse vom SMS-Protokoll (Kapitel 2.1.2) haben wir auch die SM, die über ICQ versendet werden untersucht. Und zwar haben wir über ICQ eine SM vom User 29593694 an unser GSM-Modem gesendet und dann den empfangenen Text untersucht.

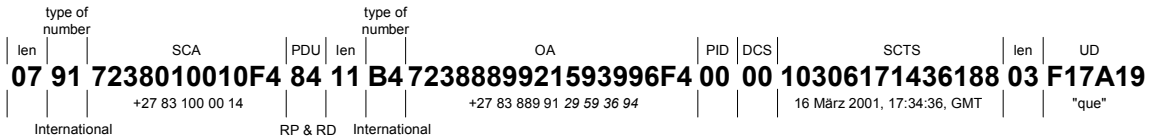


Abbildung 8: PDU von ICQ-SM

Abbildung 8 zeigt den empfangenen String und die daraus resultierenden Werte. Weil das Reply-Path Flag gesetzt wurde, muss eine Reply-Meldung über das angegebene SMSC zurückgehen. Die Nummer +27 83 100 00 14 von dem angegebenen SMSC führte uns nach Südafrika (+27 ..), genauer gesagt zu MTN South Africa (www.mtn.co.za). Wenn man die Originator Adress ‚27 83 889 91 29 59 36 94' entschlüsselt, ergibt sich daraus die MSN ‚+27 83 889 91' und unsere ICQ Nummer ‚29593694'. Daraus folgt, dass ICQ mit MTN in Zusammenarbeit die SMSC-Nummer immer so erweitert haben, dass auch längere Telefonnummern akzeptiert und weitergeleitet werden.

Unsere Anfragen bei MTN bestätigten dies. Leider konnten wir trotz mehrmaligem Nachfragen keinen solchen Zugang für unsere Projektarbeit bekommen. Andere SMSCs, die diese Erweiterung unterstützen scheint es nicht zu geben.

Fazit

Vorteile: Dies ist die wohl eleganteste aller Methoden. Sie ist international anwendbar, weil SMSCs normalerweise Roaming-Verträge mit vielen anderen Providern haben. Auch das Routing wird auf diese Art denkbar einfach. Es muss lediglich für jeden Benutzer eine PIN erstellt werden, der dann an die Originator Adress angehängt werden muss. Falls MTN diesen Zugang auch für kleinere Unternehmen (ICQ gehört AOL) öffnet, könnte dies auch eine sehr kostengünstige Variante sein.

Nachteile: Schwierig (für uns war es unmöglich) einen Vertrag mit MTN zu erhalten.

2.2 Zusammenfassung

Die Methode 2 mit dem Ändern von Bytes im SMS-Protokoll hat keine Lösungsmöglichkeit ergeben. Die Lösung mit mehreren Modems ist zwar ein cleverer Weg, kommt für die Anwendungen und Kunden der Enterprise Communications AG nicht in Frage, weil diese Lösung zu teuer ist. Für die beste Methode über ein spezielles SMSC haben wir leider zu wenig Unterstützung von MTN South Africa bekommen. Aus diesen Gründen mussten wir uns auf die Methode 1 mit dem Kurznamen im SMS-Text festlegen – Die uneleganteste.

Kapitel 3 Unser Instant Messaging System

3.1 Projektdesign

Nachdem wir uns eingehend mit der Problematik der ‚SMS-Reply-Funktion' auseinandergesetzt und verschiedene Lösungsansätze ausgearbeitet hatten, begannen wir, das Konzept für das zu erstellende Instant Messaging System aufzustellen. Um Erweiterungen oder Technologie-Wechsel so einfach wie möglich einbauen bzw. vollziehen zu können, haben wir uns speziell darauf konzentriert, eine möglichst ‚lose' Architektur zu entwerfen. Unsere Software-Module sind deshalb so konzipiert, dass das Auswechseln oder Ergänzen des Einen keine oder nur minimale Konsequenzen für ein anderes Modul nach sich zieht. Das Ergebnis dieser Bemühungen präsentiert sich wie folgt:

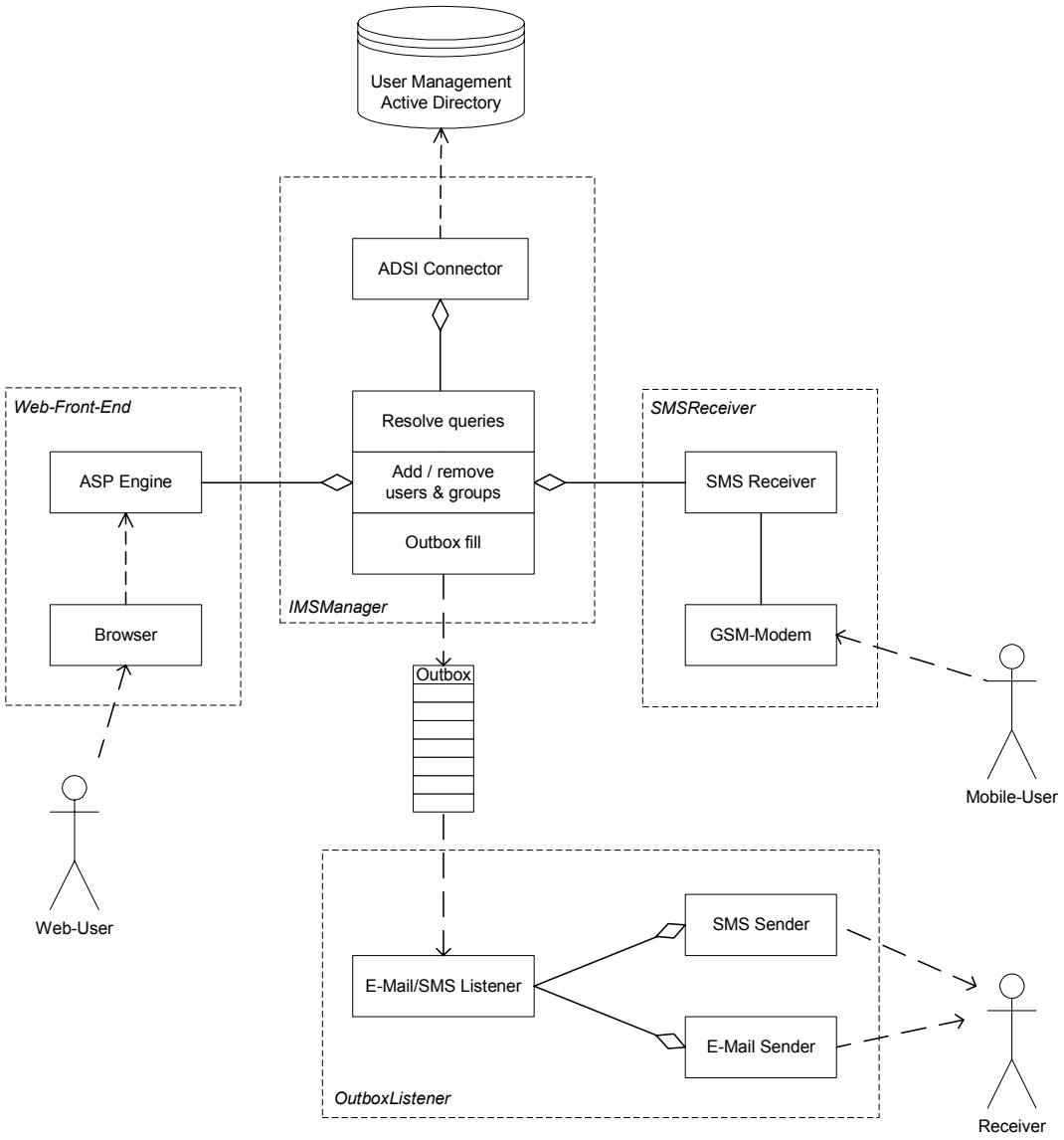


Abbildung 9: Projektstruktur ‚Instant Messaging System'

Die vier wesentlichen Bestandteile des Instant Messaging Systems, namentlich das ‚Web-Front-End', der ‚IMSManager', der ‚OutboxListener' und der ‚SMSReceiver' werden in den folgenden Kapiteln beschrieben. Dabei steht nicht die technische Realisierung, sondern die konzeptionelle und die aufgabenorientierte Sicht im Vordergrund. Details zur Implementation können der beigelegten CD, den Kapiteln über die eingesetzten Technologien und dem Anhang entnommen werden.

3.1.1 Der IMSManager

Das Aufgabengebiet der Kernkomponenten war eigentlich von Anfang an klar: Diese müssen einerseits eine Userverwaltung bieten und andererseits das Aufbereiten von Nachrichten an einen oder mehrere Empfänger erlauben. Wir mussten uns also für ein Datenbank-Back-End entscheiden und uns Gedanken über das Versenden von Nachrichten machen.

Windows 2000 bietet mit dem Active Directory leistungsfähige und voll ins System integrierte Verzeichnisdienste an. Im Active Directory können sowohl Computer, wie auch Benutzer und Exchange-Postfächer verwaltet werden. Deshalb haben wir den Entschluss gefasst, unsere Benutzerverwaltung vollständig mit den Active Directory Services zu realisieren.

Obwohl das Modul ‚IMSManager' auch mit einer Klasse hätte realisiert werden können und deshalb eigentlich auch nur ein Binary (in Form einer Windows-DLL) hätte resultieren müssen, haben wir dessen Funktionalität in 3 COM-Klassen aufgespalten. Die erste Klasse dient der Zugriffskontrolle (Benutzername und Passwort), die zweite dem Abfragen von Informationen aus dem Datenbank-Back-End und die dritte der Koordination und Zusammenfassung dieser Funktionen, sowie der zusätzlichen Methode ‚SendMessage'. Wie aus der Projektstruktur ersichtlich haben wir uns dafür entschieden, das Senden von Nachrichten nicht direkt, sondern über die ‚Outbox' erfolgen zu lassen. Dies bot uns die Möglichkeit, eine weitere Entkopplung von zwei an sich unterschiedlichen Funktionalitäten (Aufbereitung der Nachrichten, Abfragen von Mobile-Nummern, Email-Adressen, etc. vs. Versenden der Nachrichten) zu erreichen. Die Outbox wird im nächsten Kapitel noch ausführlich beschrieben.

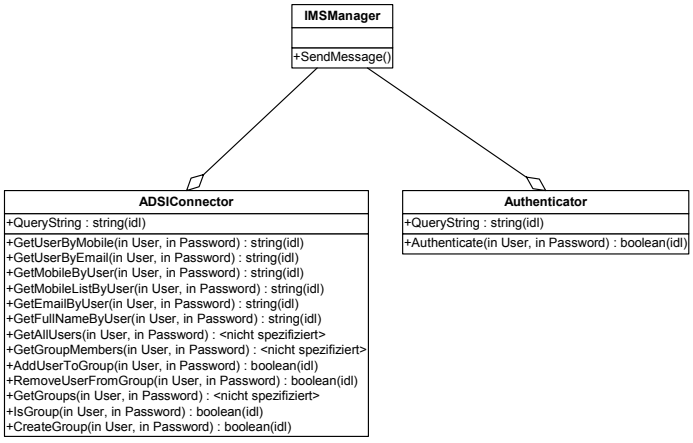


Abbildung 10: Klassendiagramm Modul ‚IMSManager'

3.1.2 Die Outbox und das XML-Nachrichtenformat

Wie im letzten Kapitel erwähnt, haben wir das Aufbereiten von Nachrichten und das Versenden derselben voneinander getrennt. Genau genommen werden die vom Modul ‚IMSManager' erzeugten Nachrichten in einem speziell dafür eingerichteten Verzeichnis (in der vorliegenden Projektarbeit war dies ‚C:\outbox') abgelegt, eine Notifikation ausgelöst und die neuen Nachrichten versendet. Es war deshalb nötig, ein spezielles Format für diese Nachrichten einzuführen, welches von beiden betroffenen Modulen, dem ‚IMSManager' und dem ‚OutboxListener' geschrieben, gelesen und interpretiert werden kann.

Bei der Bestimmung des Nachrichtenformats sind wir dem aktuellen XML-Hype gefolgt und haben uns für den Einsatz der Extensible Markup Language (XML) entschieden. Die Struktur wurde dabei bewusst einfach gehalten, so dass Sie problemlos interpretierbar ist:

Listing 5.3.1.1: Beispiel einer Nachricht (XML)

```
 1  <XML Id="MessageJobDocument">
 2     <Message Id="1445">
 3           <MessageType>SMS</MessageType>
 4           <SendingAttempts>0</SendingAttempts>
 5           <TimeStamp>
 6                 <Date>
 7                       <Year>2001</Year>
 8                       <Month>5</Month>
 9                       <Day>11</Day>
10                 </Date>
11                 <Time>
12                       <Hour>12</Hour>
13                       <Minute>8</Minute>
14                       <Seconds>39</Seconds>
15                 </Time>
16           </TimeStamp>
17           <From>
18                 <Name>johnsmith</Name>
19                 <EMail>johnsmith@zhwin.ch</EMail>
20                 <MobileNumber>+41 79 000 0000</MobileNumber>
21           </From>
22           <Body>
23                 <Subject>Test</Subject>
24                 <Text>Yolla</Text>
25           </Body>
26           <To>
27                 <EMail>mikemeyers@zhwin.ch</EMail>
28                 <MobileNumber>+41 76 000 0001</MobileNumber>
29                 <Name>mikemeyers</Name>
30           </To>
31     </Message>
32  </XML>
```

3.1.3 Das Web-Front-End

Die Anforderungen an das Front-End bestanden darin, den einfachen Zugang zur Funktionalität unseres Instant Messaging Systems zu erlauben. Zu diesem Zweck haben wir ein Web-Front-End entworfen, welches ein Formular zum Versenden von Nachrichten anbietet. Über dieses Front-End ist es auch möglich, neue Verteilerlisten (Gruppen) zu generieren und diesen verschiedene Benutzer zuzuordnen. Um die Handhabung des Front-Ends möglichst einfach zu gestalten, haben wir diverse Elemente mit Javascript aufgepeppt, so wird zum Beispiel beim Schreiben einer (SMS-)Nachricht laufend die noch zur Verfügung stehende Anzahl Buchstaben angezeigt.

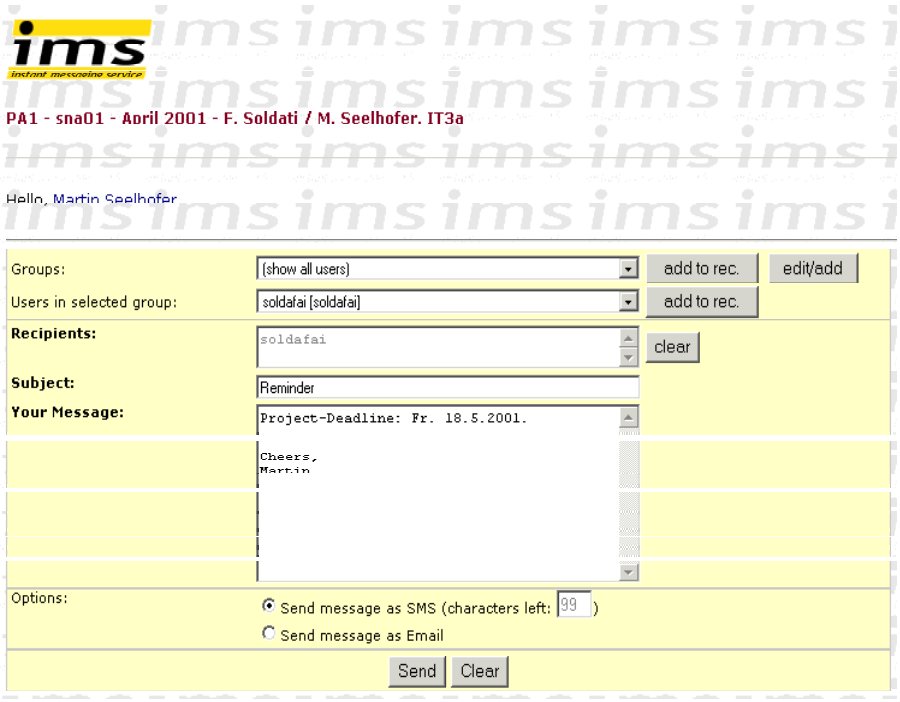


Abbildung 11: Web-Front-End

3.1.4 Das Modul SMSReceiver

Diesen Modul ist grundsätzlich für die Ansteuerung des GSM-Modems zuständig. Da dies alles im Hintergrund vonstatten geht, ist es als Dienst (Service) zu implementieren. Beim Starten dieses Dienstes muss zuerst das Modem initialisiert werden. Dazu muss der aktuelle Betriebszustand getestet werden. Wenn das Modem neu gestartet wurde, ist die Eingabe des PINs nötig, ansonsten kann die Initialisierung übersprungen werden.

Danach sollte geprüft werden, ob schon SMs auf der SIM-Karte gespeichert sind. Ist dies der Fall, müssen diese Meldungen gleich verarbeitet werden. Danach kann der SMSReceiver-Dienst in einen Sleepmode gesetzt werden, der erst durch neue Meldungen am COM Port wieder unterbrochen wird.

Die Klasse CModemCtrl übernimmt dabei die allgemeine Ansteuerung des COM-Ports. Diese Klasse ist demzufolge auch für andere Projekte einsetzbar. Die davon abgeleitete Klasse CModemM20 ist nur für das Siemens-Modem M20 gültig. Sie implementiert die Modem-spezifischen Methoden. Mittels der Klasse CProcessMessage kann der Messagestring extrahiert und an den IMSManager weitergegeben werden. Diese Klasse muss eine Liste mit allen Empfängern aus dem SM-Text generieren und den Meldungstyp E-Mail bzw. SMS festlegen. Wie das Nachrichtenformat sein muss, ist dem Kapitel 3.3 (Nachrichtenformat der Short Message) zu entnehmen. Die Kommunikation mit dem Modem und das Format des Messagestrings ist in Kapitel 4.4 (GSM Modem Steuerung) genauer beschrieben.

Nachdem eine Nachricht verarbeitet wurde, ist diese von der SIM-Karte wieder zu entfernen, weil der Speicherplatz einer SIM-Karte auf ca. 10 Meldungen beschränkt ist.

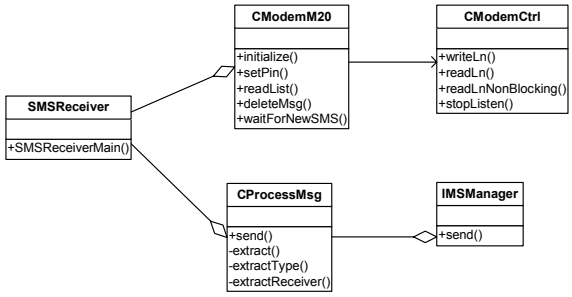


Abbildung 12: Klassendiagramm SMSReceiver

3.1.5 Das Modul OutboxListener

Für dieses Modul gilt gleiches wie für den SMSReceiver. Da es im Hintergrund laufen kann, ist dieses Modul ebenfalls als Dienst implementierbar.

Die Aufgabe ist im Grunde genommen sehr einfach. Beim Starten müssen allfällige Jobs von der Outbox verarbeitet werden. Dann geht der OutboxListener solange in den Wartezustand, bis ein neuer Job in der Outbox anliegt. Wenn ein neuer Job vorhanden ist, wird zuerst dem Dateinamen des Jobfiles ein ‚#' Charakter vorangestellt, um zu signalisieren das der Job in Bearbeitung ist. Dann muss der Dienst das Jobfile auslesen und entscheiden ob die Meldung an die EasySMS.dll oder den Exchange-Server weiterzuleiten ist. Dies übernimmt die Klasse CSendJob.

Wenn der Job fertig verarbeitet wurde, muss er wieder von der Outbox entfernt werden.

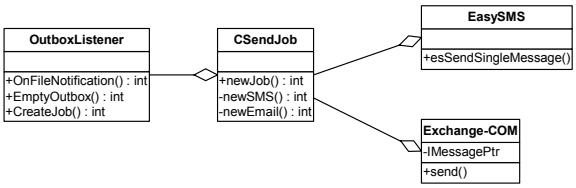


Abbildung 13: Klassendiagramm OutboxListener

3.2 Die User-Verwaltung mit dem Active Directory

Bei der User-Verwaltung haben wir uns entschlossen, auf die Active Directory Services zurückzugreifen. Dazu haben wir einen Container ‚SMSUsers' eingerichtet, in welchem alle IMS-relevanten Benutzer (mit Win2000-Account), Kontakte (kein Win2000-Account, nur Name, Email, Mobile) und Verteilerlisten (Gruppen) abgelegt werden. Zur Verwaltung dient ein Snap-In der Microsoft Management Console mit dem Titel ‚Active Directory Benutzer und Computer':

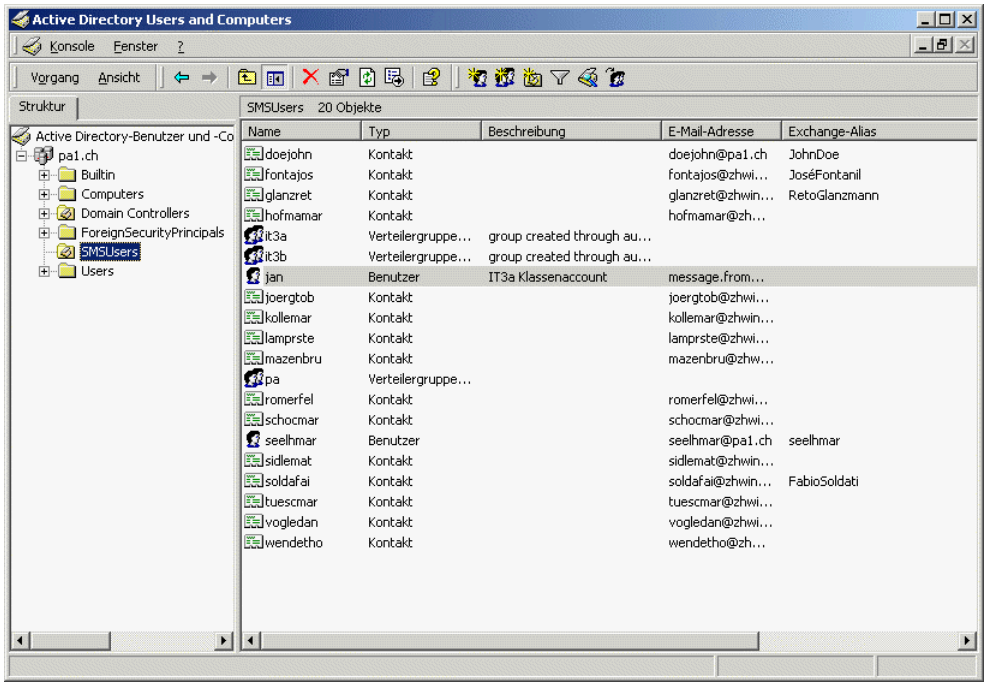


Abbildung 14: Der Container ‚SMSUsers' im Active Directory

3.3 Nachrichtenformat der Short Message

Für die folgenden Betrachtungen wird angenommen, dass ein mobiler Benutzer eine Nachricht an einen im Active Directory eingetragenen Benutzer senden möchte:

Damit der IMSManager entscheiden kann an wen und in welcher Form – E-Mail oder SMS – eine SM weitergeleitet werden kann, muss der Handybenutzer diese Informationen im SM-Text angeben.

Weil die Eingabe über die Handytastatur schon von sich aus sehr mühsam ist, sollte diese Routing-Information mit möglichst wenig Zeichen auskommen. Das System sollte auch fehlertolerant sein. Gewisse Handys beherrschen zum Beispiel die Umschaltung zwischen Gross- und Kleinschreibung nicht. Oder ein Benutzer kann sich vielleicht nicht genau an das Eingabeformat erinnern.

Aus diesen Gründen haben wir uns auf das folgende Format geeinigt (als regulärer Ausdruck formuliert):

```
[S:|SMS:|E:|EMAIL:]?[Empfänger][,Empfänger]* [Text]
```

Anmerkungen:

- Der Meldungstyp und der/die Empfänger sind *nicht* case-sensitive.
- Der Meldungstyp muss mit einem `,:'` abgeschlossen werden.
- `S:` oder `SMS:` ergibt den Meldungstyp SMS.
- `E:` oder `EMAIL:` ergibt den Meldungstyp EMAIL.
- Wenn kein Typ angegeben ist, wird der Standardtyp EMAIL gesetzt.
- Der Name des Empfängers ist der Benutzername, der im Active Directory festgelegt wurde.
- Mehrere Empfänger werden durch Kommas getrennt.
- Der Text beginnt nach dem *ersten* Leerzeichen.

Kapitel 4 Hilfsmittel und Technologien

4.1 Das Component Object Model (COM)

COM ist eine ‚offene' Architektur für eine ‚plattformunabhängige' Entwicklung von Client-/ Server-Applikationen (so der Original-Text im Glossar der MSDN Library vom Oktober 2000). Sie basiert auf objekt-orientierter Technologie und wurde von Microsoft in Zusammenarbeit mit Digital Equipment Corporation im Jahre 1994 der breiten Öffentlichkeit zugänglich gemacht. Ursprünglich wurde das Component Object Model unter dem Namen OLE 2.0 eingeführt, es bildete also den Nachfolgemechanismus zu OLE (Object Linking and Embedding). Die letztgenannte Technologie erlaubte es, dass jede Anwendung seine speziellen Funktionen und Fähigkeiten anderen Applikationen zur Verfügung stellen konnte. Auf diesem Weg war es denn auch möglich, Dokumente mit bunt durchmischtem Inhalt aufzubauen, deren Teile aus verschiedenen Anwendungen stammen konnten. OLE bildete also – wie der Name andeutet – die Grundlage für das Verknüpfen und Einbetten von Dokumentteilen (Objekten) in fremde Dokumente (Container).

Mit COM hat Microsoft das Konzept des Anbietens von Funktionalität ausgebaut, vereinheitlicht und stärker mit der Programmiersprache C++ in Beziehung gebracht. Die Anordnung von COM-Objekten im Speicher entspricht denn auch demjenigen von C++-Objekten. Dies hat einige Konsequenzen, so müssen z.B. alle Compiler, welche COM-Objekte erzeugen können müssen, den Aufbau von C++-Objekten beherrschen. Andererseits erlaubt es diese Vorschrift auch, dass z.B. in mit Delphi (ein objektorientierter Pascal-Dialekt) geschriebene Objekte problemlos mit in C++ geschriebenen Komponenten zusammenarbeiten können.

Den Ausschlag für die Wahl von COM als Objektmodell für die Module im vorliegenden Projekt gab aber eigentlich eine andere Stärke dieser Technologie: Bei der üblichen Softwareentwicklung können die nach Aufgaben getrennten Module nur mit grossem Aufwand unabhängig voneinander getestet, kompiliert und verteilt bzw. integriert werden. Beim Einsatz von COM werden diese Arbeiten geradezu zum Kinderspiel. Das Resultat des Entwicklungsvorgangs für eine COM-Klasse besteht nämlich normalerweise aus einer Bibliothek (Windows-DLL), welche auf einfache Weise im System registriert und wieder entfernt werden kann. Die COM-Klasse kann dabei nach erfolgter Registrierung durch simple Angabe des Namens (z.B. „Word.Application" für das COM-Objekt Microsoft Word) verwendet und programmiert bzw. automatisiert werden.

Durch die Erweiterung von COM in Richtung DCOM (Distributed COM) wurde das Component Object Model auch in die Lage versetzt, über Rechnergrenzen hinweg zu funktionieren, was bislang nur einigen wenigen Technologien, wie z.B. CORBA (Common Object Request Broker Architecture) oder RPC (Remote Procedure Calls), vorbehalten war. Dabei können bereits vorhandene, ausprogrammierte COM-Objekte durch wenig Mehraufwand (je nach den eingesetzten Tools nur ein simples Neu-Kompilieren) auch für den verteilten Einsatz fit gemacht werden.

4.1.1 ClassID und ProgID einer COM-Klasse

COM verwendet zur eindeutigen Referenzierung von Schnittstellen / Klassen / Objekten 128-Bit-Werte, die sogenannten ‚globally unique identifiers' (GUIDs), welche aufgrund ihres Zustandekommens weltweit eindeutig sind. Normalerweise wird eine GUID aus der MAC-Adresse der Netzwerkkarte, der aktuellen Uhrzeit und sonst noch ein paar Parametern berechnet. Ist keine Netzwerkkarte vorhanden, so werden stattdessen einige Angaben aus dem BIOS verwendet. Es sei übrigens schlimmer, eine GUID mehrmals zu gebrauchen, als die Zahnbürste seines Nachbarn zu benutzen...

GUIDs werden in einem speziellen Format dargestellt, die 128-Bit-Werte werden in hexadezimaler Darstellung wie folgt gruppiert:

```
{[8 Ziffern]-[4 Ziffern]-[4 Ziffern]-[16 Ziffern]}
```

Die GUID für Microsoft Word 2000 (9.0) lautet z.B.:

```
{000209FF-0000-0000-C000-000000000046}
```

Eine GUID, welche eine COM-Klasse eindeutig identifiziert, wird CLSID genannt. GUIDs, welche Schnittstellen identifizieren heissen IIDs. Eine andere Bezeichnung für GUID lautet UUID (universally unique identifier).

Man kann sich leicht vorstellen, dass diese Art der eindeutigen Referenzierung nicht für alle Anwendungen geeignet ist, deshalb existiert auch eine sprachliche Ersatzdarstellung für die GUIDs, die sogenannte ProgID. Für die Benennung von COM-Klassen hat sich folgende Konvention durchgesetzt:

```
<Hersteller>.<Klasse>.<Version>
```

Die ProgID für Microsoft Word 2000 (9.0) lautet somit:

```
Word.Application.9
```

oder in versionsunabhängiger Form:

```
Word.Application
```

4.1.2 Fehlercodes

Da im COM-Umfeld eine riesige Menge an Fehlerquellen existiert, wurde zur Fehlerbehandlung ein 32-Bit Typ mit speziellem Aufbau definiert und diesem der Name HRESULT gegeben.

Dieser Typ besteht aus den folgenden Teilen:

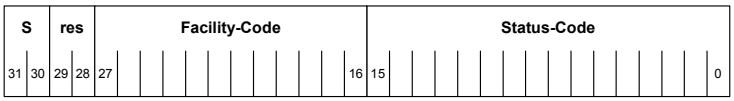


Abbildung 15: Datentyp `HRESULT`

S (Severity Code)	00 = Erfolg 01 = Zur Information 10 = Warnung 11 = Fehler
Facility-Code	Umfeld bzw. Kategorie. Beispiele: 1 = Fehler im RPC-Umfeld (Remote Procedure Calls) 4 = benutzerdefinierter Fehler 7 = Win32-Fehler
Status-Code	Bezeichnung (Nummer) des konkret aufgetretenen Fehlers

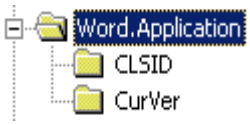
Der Fehlercode `0x80010007` steht beispielsweise für `RPC_E_SERVER_DIED`, also für die Warnung, dass in der Kategorie RPC der Fehler `SERVER_DIED` aufgetreten ist.

4.1.3 Die Windows-Registry

Das Verzeichnis aller durch GUIDs referenzierten Objekte (COM-Klassen, -Schnittstellen, etc.) befindet sich in der Windows Registrierungsdatenbank, kurz ‚Registry'. Eigentlich unter dem Schlüssel ‚HKEY_LOCAL_MACHINE\SOFTWARE\Classes' gespeichert, findet man es (Registry Editor verwenden: regedit.exe) auch unter dem Schlüssel ‚HKEY_CLASSES_ROOT'.

Für COM-Objekte existiert an der beschriebenen Stelle ein Unterschlüssel, welcher den Namen der ProgID des Objektes trägt. An dieser Stelle sind ein Querverweis auf die zugehörige CLSID (Schlüssel ‚CLSID') und bei versionsunabhängigen ProgIDs noch die Angabe der aktuellen Version (Schlüssel ‚CurVer') gespeichert.

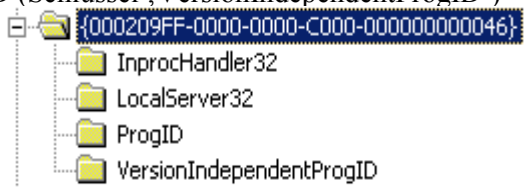
Beispiel für ‚Word.Application':



Schlüssel	Element	Typ	Inhalt
Word.Application	(Standard)	REG_SZ (Zeichenfolge)	Microsoft Word-Anwendung
CLSID	(Standard)	REG_SZ	{000209FF-0000-0000-C000-000000000046}
CurVer	(Standard)	REG_SZ	Word.Application.9

Die eigentlichen Informationen liegen aber in einem anderen Schlüssel, welcher als Bezeichnung die CLSID der COM-Klasse trägt. Hier steht z.B. der Pfad zur Datei mit dem ausführbaren Code (LocalServer32). Zusätzlich findet sich hier ein Verweis zurück auf die ProgID (Schlüssel ‚ProgID') bzw. die versionsunabhängige ProgID (Schlüssel ‚VersionIndependentProgID')

Beispiel für Microsoft Word:



Schlüssel	Element	Typ	Inhalt
{...}	(Standard)	REG_SZ (Zeichenfolge)	Microsoft Word-Anwendung
LocalServer32	(Standard)	REG_SZ	C:\Office\WINWORD.EXE /Automation
ProgID	(Standard)	REG_SZ	Word.Application.9

COM-Server (DLLs oder EXE-Dateien) können so aufgebaut werden, dass diese sich (beinahe) selbständig registrieren und aus dem System wieder entfernen können. Dazu müssen z.B. für eine DLL einfach die Funktionen `DllRegisterServer` und `DllUnregisterServer` implementiert und exportiert (also von aussen zugänglich gemacht) werden. Anschliessend kann die DLL über die Kommandozeile (z.B. ‚Start'-‚Ausführen') auf einfache Weise registriert (‚regsvr32.exe comserver.dll') und wieder entfernt (‚regsvr32.exe –u comserver.dll') werden.

Beispiel für das Registrieren einer COM-Server-DLL:

```
Regsvr32.exe ADSIConnector.dll
```

Beispiel für das Entfernen einer COM-Server-DLL:

```
Regsvr32.exe –u ADSIConnector.dll
```

4.1.4 Datentypen

COM unterstützt alle C++-Datentypen. Wegen der zunehmenden Internationalisierung im Bereich Software-Entwicklung wurden allerdings die meisten Funktionen des COM-Subsystems, welche Text als Parameter erwarten oder zurückgeben, nur als Unicode-Varianten entworfen. Der entsprechende C++-Typ ist eigentlich ‚unsigned short *' (statt ‚char *' oder ‚unsigned char *'), allerdings ist diese Bezeichnung zugegebenermassen nicht sehr aussagekräftig. Deshalb wurden folgende Aliases (per #define) zu ‚unsigned short' erstellt: OLECHAR und wchar_t. Zusätzlich definiert die Entwicklungsumgebung Microsoft Visual Studio auch noch den Typ _TEXT, welcher je nach Projekteinstellungen 8- oder 16-Bit Zeichenfolgen erzeugt (gelöst durch Makros: `#ifdef UNICODE ...`).

Um den Datenaustausch zwischen verschiedenen Programmiersprachen zu vereinfachen wurde zudem ein zusätzlicher, sehr umfangreicher Variablen-Typ geschaffen, der Typ ‚VARIANT'. Dieser kapselt in einer C++-union diverse Objekte, so z.B. 8-, 16- und 32-Bit-Zahlenwerte, Pointer auf 16-Bit Zeichenfolgen, Pointer auf Variablen vom Typ ‚VARIANT' etc. Die Details dieses Typs können in der MSDN-Library unter dem Stichwort ‚VARIANT and VARIANTARG' im Kapitel Automation nachgeschlagen werden.

4.1.5 Schnittstellen

Unter einer COM-Schnittstelle versteht man die Zusammenfassung einer oder mehrerer Funktionen bzw. Methoden zu einer logischen Gruppe. Im Bereich der Programmiersprache C++ wird eine solche Schnittstelle als rein abstrakte Basisklasse deklariert.

Während beim Einbinden von Klassen (zur Erzeugung von Objekten dieser Klasse) im Source-Code (z.B. C++-Klassen) alle Details einer Implementierung bekannt sind, trifft dies beim Binden eines Objektes zur Laufzeit keineswegs zu. Natürlich muss ein Programm nicht wissen, *wie* eine bestimmte Funktion (bzw. Methode) in einem Objekt *implementiert* ist, es muss aber sehr wohl wissen, *ob* und *wie* diese *verwendet* werden kann. Zu diesem Zweck setzt COM die erwähnten Schnittstellen (Interfaces) ein, welche in einer eigenen Sprache, der Microsoft Interface Definition Language (MIDL, eine Erweiterung der IDL für den Bereich ‚Remote Procedure Calls') beschrieben werden. Schnittstellen geben dabei vor, wie Funktionsaufrufe erfolgen müssen, definieren also die Art des Aufrufs, implementieren ihrerseits aber nichts. Soll eine COM-Klasse eine bestimmte Schnittstelle unterstützen, so müssen deren Funktionen (in der richtigen Reihenfolge) z.B. in einer C++-Klasse ausprogrammiert werden.

Die Definition einer einfachen Schnittstelle in der von COM verwendeten IDL könnte für eine einfache COM-Klasse Robot z.B. etwa so aussehen:

Listing 4.1.1: Definition einer einfachen Schnittstelle in MIDL:

```
1  [object, uuid(AC3D78B4-AD53-4680-A6F2-F3A0F558BD62)]
2  interface IRobot : IUnknown
3  {
4     HRESULT Step([in] int length);
5     HRESULT Rotate([in] int angle);
6  };
```

Dieses Beispiel definiert einen Roboter, welcher neben den Methoden der Schnittstelle IUnknown die beiden Methoden ‚Step' (Schritt einer bestimmten Länge vorwärts) und ‚Rotate' (Drehung um einen bestimmten Winkel) zur Verfügung stellt.

Bemerkungen:

Zeile 1 Diese Zeile gibt die IID dieser Schnittstelle an

Zeile 2 COM-Schnittstellen können (wie C++-Klassen) abgeleitet werden. Die Basis-Schnittstelle aller (gültigen) COM-Objekte ist IUnknown.

Zeile 4 Durch die Angabe von Parameter-Attributen (z.B. `[in]`) kann der Typ bzw. die Richtung der Funktions-Parameter festgelegt werden.

4.1.6 COM-Objekte und -Klassen

Während Schnittstellen Informationen über die von einem COM-Server angebotenen Funktionen liefern, werden diese durch eine COM-Klasse implementiert. Instanzen (reale Ausprägungen im Speicher) einer solchen Klasse bezeichnet man analog zu C++ mit ‚COM-Objekt'.

Alle COM-Klassen müssen (per Definition) das grundlegende Interface `IUnknown` implementieren. Deshalb wird als nächstes kurz auf dieses eingegangen.

Die Schnittstelle IUnknown

In COM existiert eine ganze Reihe vordefinierter Schnittstellen. Die Grundlegendste ist dabei `IUnknown`, alle selbstdefinierten Interfaces müssen von dieser Schnittstelle abgeleitet werden. Dies bedeutet, dass alle COM-Objekte die Funktionen bzw. Methoden dieser Schnittstelle implementieren müssen. Die von IUnknown verlangten Funktionen / Methoden sind:

```
UINT AddRef();
UINT Release();
HRESULT QueryInterface(REFIID riid, void **ppv);
```

Da COM-Objekte bei Gebrauch dynamisch in den Speicher geladen werden, braucht es einen Mechanismus, um den benötigten Speicher wieder freizugeben. Dazu besitzen COM-Objekte Referenz-Zähler, mit welchen Sie entscheiden, wann Sie sich selber aus dem Speicher entfernen können. Per Aufruf von `AddRef` wird der Referenz-Zähler um eins erhöht, mit dem Aufruf von `Release` wird dieser verringert. Erreicht der Zähler beim Aufruf von `Release` die Zahl 0, so darf bzw. muss sich das Objekt aus dem Speicher entfernen, da damit angezeigt wird, dass niemand mehr einen Verweis (Zeiger) auf eine Schnittstelle des Objektes besitzt.

Die Funktion `QueryInterface` dient dem Abfragen einer bestimmten Schnittstelle. Will ein Objekt einen Zeiger auf eine Schnittstelle abfragen, so muss die konkrete Implementierung der Funktion `QueryInterface` per Aufruf von `AddRef` den Referenzzähler selbständig erhöhen. Der Client muss dann nach dem Gebrauch des COM-Objektes per `Release` den Referenzzähler wieder erniedrigen.

Da die Funktionen der Schnittstelle `IUnknown` von zentraler Bedeutung für das korrekte Allozieren und Freigeben von Speicher für COM-Objekte sind, wird von vielen Entwicklungsumgebungen (u.a. auch dem von uns verwendeten Visual Studio von Microsoft) eine Standard-Implementierung dieser Funktionen angeboten. Bei der Verwendung von Microsoft Visual Studio können beispielsweise die Klassen der ActiveX Template Library (ATL) verwendet und abgeleitet werden. Im Gegensatz zu den Microsoft Foundation Classes (MFC) bleiben kompilierte Projekte unter Verwendung der ATL erstaunlich schlank.

COM-Objekte erzeugen

An dieser Stelle soll kurz vorgeführt werden, wie in VBScript und C++ ein COM-Objekt erzeugt wird.

Listing 4.1.2: Ein Beispiel in VBScript:

```
 1  dim obj
 2
 3  set obj = CreateObject("ADSIConnector.Main")
 4
 5  if obj.IsGroup("pa1") then
 6    MessageBox "pa1 ist eine Gruppe"
 7  else
 8    MessageBox "pa1 ist KEINE Gruppe"
 9  end if
10
11  set obj = nothing
```

Bemerkungen:

Zeile 1 Hier wird eine Variable deklariert. Da in VBScript keine Typisierung existiert, bzw. alle Variablen vom Typ ‚VARIANT' sind, steht hier keine Typangabe wie z.B. bei C++ oder Pascal.

Zeile 3 Eine Instanz der COM-Klasse ‚ADSIConnector.Main' wird erzeugt und ein Verweis (intern ein Pointer, gekapselt in die VARIANT-Variable ‚obj') darauf gespeichert.

Zeile 5 Mit der Funktion ‚IsGroup' wird geprüft, ob im Active Directory eine Gruppe mit dem angegebenen Namen („pa1")eingetragen ist.

Zeile 11 Der Verweis auf das COM-Objekt wird freigegeben. Der Referenzzähler geht auf 0 zurück und das COM-Objekt ‚ADSIConnector.Main' wird entladen.

Listing 4.1.3: Ein Beispiel in C++:

```
 1  IUnknown      *pUnk;
 2  HRESULT hr;
 3  CLSID   clsid = ...;
 4
 5  hr = CoInitialize(NULL);
 6  if (SUCCEEDED(hr))
 7  {
 8    hr = CoCreateInstance(clsid,NULL,CLSCTX_SERVER,
 9                          IID_IUnknown,(void **) &pUnk);
10    if (SUCCEEDED(hr))
11    {
12      …
```

Bemerkungen:

Zeile 1 Für die grundlegendste Schnittstelle `IUnknown` existiert der benutzerdefinierte Typ `IUnknown`. `pUnk` ist also ein Pointer auf die Schnittstelle `IUnknown`.

Zeile 2 Siehe Kapitel ‚Fehlercodes' für Details zum Typ HRESULT.

Zeile 3 Anstelle der Pünktchen müsste hier die ClassID des gewünschten COM-Objektes eingetragen werden.

Zeile 5 Um COM-Funktionen und –Objekte verwenden zu können, muss sich eine COM-Client-Anwendung zuerst beim Laufzeitsystem anmelden. Dies geschieht mit der Funktion `CoInitialize`. Mit `CoUninitialize` wird die Client-Anwendung wieder vom Laufzeitsystem abgemeldet.

Zeile 6 SUCCEEDED ist ein Makro, welches wie folgt definiert ist:

```
#define SUCCEEDED(Status) ((HRESULT)(Status) >= 0)
```

Es wird also die übergebene Zahl auf >=0 getestet und entsprechend TRUE oder FALSE ‚zurückgegeben'.

Zeile 8 Mit `CoCreateInstance` wird ein COM-Objekt erzeugt, wobei dieser Funktion die Klassen-ID (GUID der Klasse), die Interface-ID (GUID des Interfaces) und ein Pointer auf den Schnittstellenzeiger übergeben werden muss. Nach erfolgreicher Ausführung der Funktion enthält der Zeiger `pUnk` also einen gültigen Verweis auf die Schnittstelle `IUnknown` des COM-Objektes.

Zeile 12 Die Freigabe des Objektes erfolgt nach Gebrauch durch den Aufruf der Funktion `Release()`.

Da der Umgang mit CLSIDs etwas mühsam ist, gibt es die API-Funktion `ProgIDFromCLSID`. Dank dieser Funktion kann statt einer CLSID auch eine ProgID dazu verwendet werden, ein COM-Objekt zu erzeugen, indem die ProgID zur Laufzeit in die zugehörige CLSID umgewandelt wird. Dazu könnte das obige Beispiel um folgende Zeilen ergänzt werden:

```
7a  hr = CLSIDFromProgID(L"ADSIConnector.Main",&clsid);
```

Das ‘L’ vor der Zeichenfolge sorgt dafür, dass ein Unicode-String erzeugt wird, der statt 8 16 Bit breit ist.

Anhand der obigen Beispiele könnte der Eindruck entstehen, dass VBScript und C++ in der Art und Weise wie COM-Objekte erzeugt werden stark voneinander abweichen und dass VBScript dazu weniger Aufwand betreiben muss. Würde man die durch den VBScript-Code ausgeführten Anweisungen jedoch in C++ audrücken, so wäre der entstandene Code sogar um einiges umfangreicher als das obige C++-Beispiel. Dies liegt daran, dass VBScript, wie der Name schon sagt, eine Skript-Sprache ist. Der Code wird also von einem Interpreter während der Ausführung laufend übersetzt. Bei näherem Hinschauen scheint deshalb eine Anweisung wie diejenige in Zeile 5 des Beispiels – der Teil ‚obj.IsGroup(...)' – gar nicht in einen Aufruf der Methode ‚IsGroup' übersetzt werden zu können, da kompilierte C++-Methoden ja keine Namen in Textform tragen, sondern normalerweise über Zeiger auf eine bestimmte Speicherstelle aufgerufen werden.

COM umgeht das Problem mit der Zuordnung von Methoden-Bezeichnungen zu deren Adressen im Speicher mit der Standardschnittstelle ‚IDispatch'. Implementiert eine COM-Klasse diese Schnittstelle, so kann durch Aufruf der IDispatch-Methode GetIDsOfNames eine ‚ID' der per Zeichenfolge angegebenen Methode abgefragt werden. Danach kann die gewünschte Methode mit Invoke (gehört ebenfalls zur Schnittstelle IDispatch) durch Angabe der erhaltenen ID aufgerufen werden. Details zum Interface IDispatch sind z.B. in der MSDN-Library unter dem Stichwort ‚IDispatch Interface' im Kapitel ‚Automation', oder in [B5] zu finden.

Programmiert man COM-Objekte direkt in C++ sollte der Einsatz des doppelten Umwegs über GetIDsOfNames und Invoke aus Performance-Gründen auf ein Minimum beschränkt und der Weg über das Einbinden (#include bzw. #import) von Header- und Interface-Definitions-Dateien gewählt werden.

Abfragen von Schnittstellen und Aufrufen von Methoden

COM-Klassen können diverse Schnittstellen implementieren. Es braucht deshalb einen Mechanismus, um Interfaces zur Laufzeit abzufragen. Wie im letzten Kapitel gesehen existiert zu diesem Zweck die Methode QueryInterface. Üblicherweise fragt also eine Client-Anwendung (COM-Client) ein COM-Objekt (COM-Server) per QueryInterface an, ob dieses eine bestimmte Schnittstelle unterstützt und erhält bei erfolgreicher Ausführung einen gültigen Zeiger auf das gewünschte Interface zurück. Im Wesentlichen zeigt dieser Pointer auf eine Funktionstabelle (vgl. vtable bei C++-Klassen).

Der Abruf einer Schnittstelle beginnt immer bei IUnknown, da diese Schnittstelle von allen COM-Objekten unterstützt werden muss. Ein Beispiel für das Abrufen der weiter oben beschriebenen Schnittstelle IRobot könnte etwa wie folgt aussehen:

Listing 4.1.4: Beispiel für das Abrufen einer COM-Schnittstelle in C++

```
 1 IUnknown *pUnk;
 2 IRobot   *pRobot;
 3 HRESULT  hr;
 4
 5 hr = CoCreateInstance(clsid,NULL,CLSCTX_SERVER,
 6                       IID_IUnknown,(void **) &pUnk);
 7 if (SUCCEEDED(hr))
 8 {
 9   hr = pUnk->QueryInterface(IID_IDispatch, (void **) & pRobot);
10   if (SUCCEEDED(hr))
11   {
12     pRobot->Step(10);           // Einen Schritt vorwärts machen
13     pRobot->Rotate(45);         // Um 45 Grad rotieren (per Def.
rechts)
14     pRobot->Release();          // Referenzzähler erniedrigen
15   }
16 }
17 pUnk->Release();
```

Der Referenz-Zähler für das Robot-Objekt verhält sich dabei so:

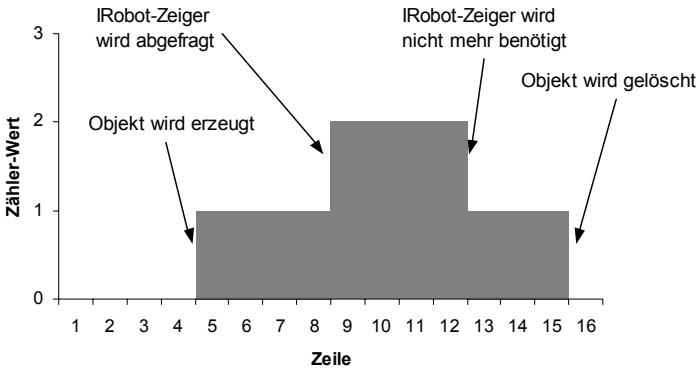


Abbildung 16: Referenz-Zähler für Robot-Klasse

Bemerkungen:

Zeile 1 Für die grundlegendste Schnittstelle `IUnknown` existiert der benutzerdefinierte Typ `IUnknown`. `pUnk` ist also ein Pointer auf die Schnittstelle `IUnknown`.

Zeile 2 Um später einen Pointer auf das `IRobot`-Interface zu erhalten, wird hier der Speicherplatz dazu reserviert.

Zeile 3 Siehe Kapitel ‚Fehlercodes' für Details zum Typ HRESULT.

Zeile 5 Mit `CoCreateInstance` wird ein COM-Objekt erzeugt, wobei dieser Funktion die Klassen-ID (GUID der Klasse), die Interface-ID (GUID des Interfaces) und ein Pointer auf den Schnittstellenzeiger übergeben werden muss. Nach erfolgreicher Ausführung der Funktion enthält der Zeiger `pRobot` also einen gültigen Verweis auf die Schnittstelle `IRobot` des COM-Objektes.

Zeile 7 SUCCEEDED ist ein Makro, welches wie folgt definiert ist:

```
#define SUCCEEDED(Status) ((HRESULT)(Status) >= 0)
```

Es wird also die übergebene Zahl auf >=0 getestet und entsprechend TRUE oder FALSE ‚zurückgegeben'.

Zeile 9 Hier findet die Abfrage eines Zeigers auf die Schnittstelle `IRobot` statt. (Siehe Kapitel ‚Die Schnittstelle IUnknown' für Details).

Zeile 12 An dieser Stelle kann das COM-Objekt Robot über den Schnittstellenzeiger `pRobot` gesteuert werden. Im Beispiel erfolgen zwei Aufrufe von Methoden.

Zeile 14 Wird die Schnittstelle `pRobot` nicht mehr gebraucht, so muss diese mit `Release` freigegeben werden. (Siehe Kapitel ‚Die Schnittstelle IUnknown' für Details).

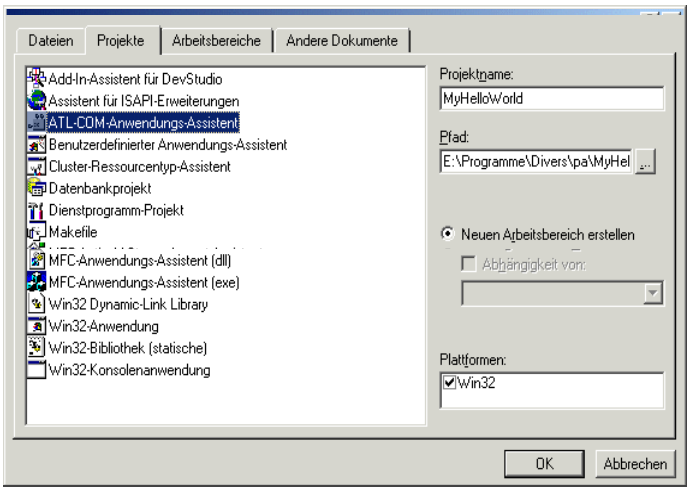
Zeile 17 Damit der verwendete Speicher für das COM-Objekt Robot wieder freigegeben werden kann, muss auch noch der Zeiger auf die `IUnknown`-Schnittstelle freigegeben werden. (Siehe Kapitel ‚Die Schnittstelle IUnknown' für Details).

4.1.7 Implementieren von eigenen COM-Klassen

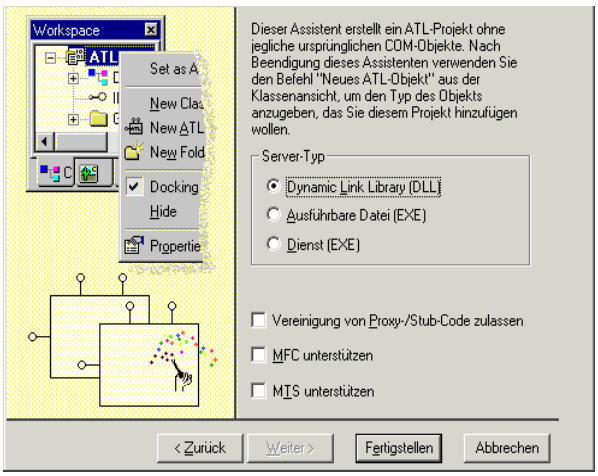
Zur Implementierung von eigenen COM-Klassen bietet Microsoft Visual Studio die ActiveX Template Library (ATL) an. Dies ist eine Sammlung von C++-Klassentemplates, mit welchen sich durch einfaches Ableiten mit sehr wenig Aufwand vollständige COM-Klassen realisieren lassen.

Anhand eines einfachen Beispiels soll dies demonstriert werden. Das Beispiel ist :

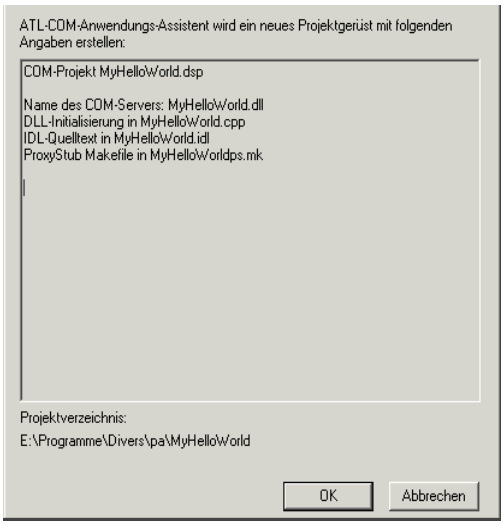
1. Erzeugen eines neuen ATL-Projektes:



2. Auswählen des Kompilierungsresultats:



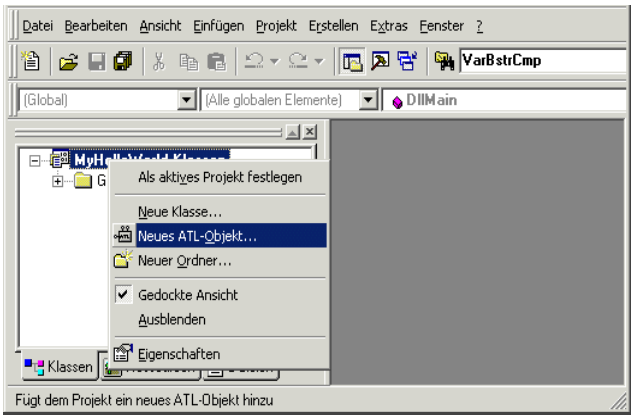
3. ‚Fertigstellen'. Danach zeigt Visual Studio die automatisch erzeugten Dateien an:



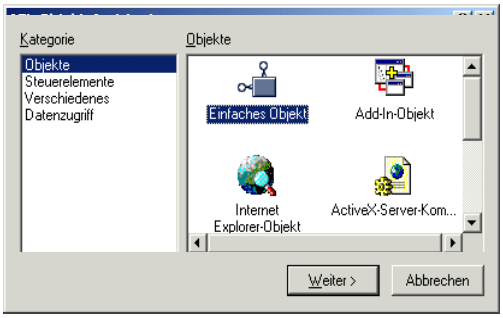
Visual Studio erzeugt u.a. folgendes automatisch:
- GUID der entstehenden COM-Klasse
- Code zur Registrierung (Registry-Einträge)
- Rahmen der DLL-Datei

Es kann also direkt mit der Entwicklung der COM-Klasse(n) begonnen werden.

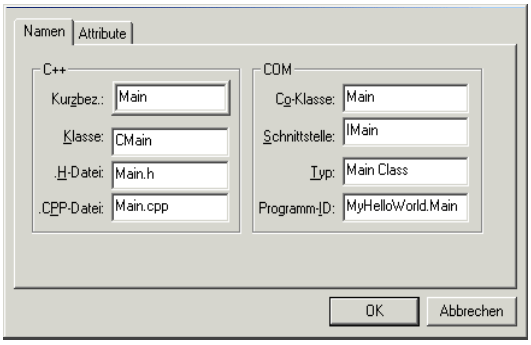
4. Hinzufügen eines neuen ATL-Objektes (Rechte Maustaste verwenden):



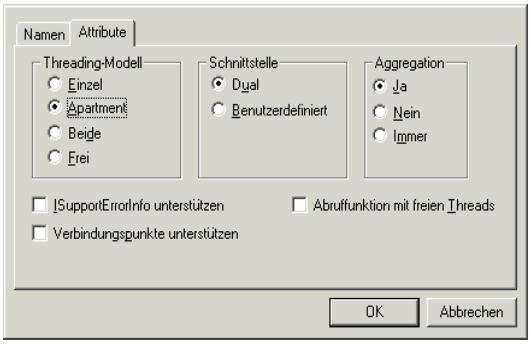
a) Auswählen des Objekt-Typs:



b) Angeben des Namens:



c) Festlegen der Attribute:

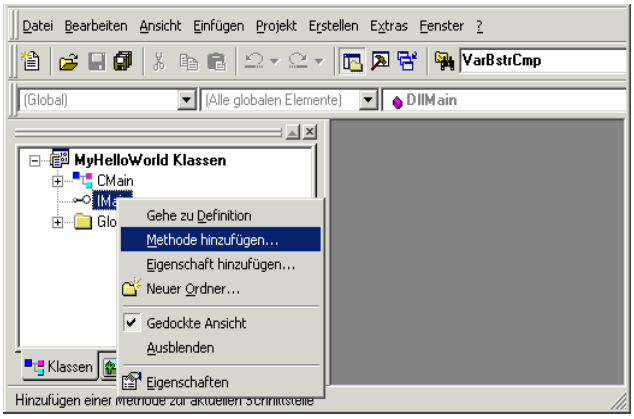


Hinweis:
Damit per ATL-COM-Assistent erzeugte COM-Objekte per Skript gesteuert werden können, muss in diesem Dialogfenster unbedingt darauf geachtet werden, dass bei der Optionsgruppe ‚Schnittstelle' die Auswahl ‚Dual' getroffen wird (vgl. IDispatch-Schnittstelle am Schluss des Abschnitts ‚COM-Objekte erzeugen')!

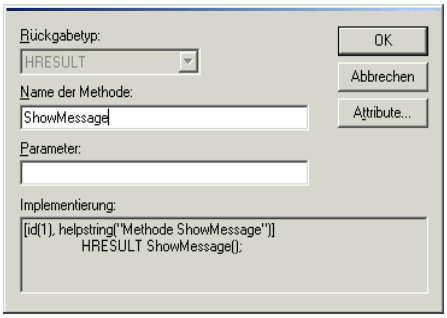
Visual Studio erzeugt einige neue Dateien und ändert einige bestehende Dateien ab. Dies wird im Fenster ‚Arbeitsbereich' (links) dargestellt:

‚CMain' wird in der Quell-Datei ‚Main.h' bzw. ‚Main.cpp' deklariert bzw. definiert und ‚IMain' wird in der Quell-Datei ‚MyHelloWorld.idl' eingefügt.

5. Eine neue Methode hinzufügen (Rechte Maustaste auf dem Interface ‚IMain'):



In unserem Beispiel soll eine Methode ‚ShowMessage' wie folgt hinzugefügt werden:



Visual Studio fügt folgendes automatisch hinzu:
- GUID des entstehenden Interfaces
- Interface-Definition (in ‚MyHelloWorld.idl')
- Deklaration (‚Main.h') und Implementations-Gerüst der Methode in der Klasse ‚CMain'

Die neue Methode ist im Fenster ‚Arbeitsbereich' sichtbar (Alle Zweige eingeblendet):

6. Code der Methode ‚ShowMessage' (im Source-File ‚Main.cpp') schreiben:

Listing 4.1.5: Code der Methode ShowMessage:

```
1  STDMETHODIMP CMain::ShowMessage()
2  {
3    MessageBox(NULL,"Hello world","ATL-COM-Example",MB_OK);
4
5    return S_OK;
6  }
```

Bemerkungen:

Zeile 3 Die Parameter der API-Funktion ‚MessageBox' sind:
- Fenster-Handle des Parent-Fensters (oder NULL für keinen Besitzer)
- Nachrichtentext
- Titel des Fensters
- Aussehen des Fensters (angezeigte Buttons, Symbole, etc.)

7. Visual Studio registriert die COM-Klasse nach erfolgreicher Kompilierung automatisch im System, so dass das Objekt sofort verwendet werden kann.

Das Resultat des Kompilierungsvorgangs ist eine selbständige Softwarekomponente, welche verteilt und auch in anderen Systemen registriert werden kann. COM bietet somit v.a. bei der sogenannt ‚inkrementellen' Softwareentwicklung (Softwarekomponenten werden schon früh zur Verwendung freigegeben und dann laufend korrigiert → häufige Updates, Bugfixes, Patches) immense Vorteile, da nur die betroffenen Module (DLLs) neu ausgeliefert und registriert werden müssen, was ein sehr einfacher Vorgang ist. Dem Problem mit den verschiedenen Version wird damit begegnet, dass im Normalfall eine Versionsabfrage in COM-Klassen eingebaut wird, so dass ein Client notfalls zur Laufzeit noch entscheiden kann, ob er eine bestimmte Schnittstelle verwenden will.

Um die eben erstellte Klasse zu verwenden, um also Objekte (Instanzen) der COM-Klasse ‚MyHelloWorld.Main' erzeugen zu können braucht es eine Client-Anwendung. Dies kann z.B. ein einfaches Skript sein, welches wie folgt aussehen könnte:

Listing 4.1.6: Beispiel-COM-Klasse ‚MyHelloWorld.Main' verwenden:

```
1  Dim obj ' Deklaration
2  Set obj = CreateObject("MyHelloWorld.Main")' Erzeugen des Objekts
3  obj.ShowMessage     ' Aufrufen der Methode
4  Set obj = Nothing   ' Freigeben des Objektes
```

Die erscheinende Meldung präsentiert sich so:

4.2 Dienste

4.2.1 Überblick

Dienste bzw. Services existieren bei Microsoft nur für Windows NT und Windows 2000. Dieses Kapitel gilt also nicht für die Windows 9x Familie. Die Beispiele sind alle in C++ geschrieben.

Entgegen den normalen ausführbaren Dateien laufen Dienste in der Regel nicht als Prozesse unter dem Konto des angemeldeten Benutzers. Stattdessen läuft ein Dienst gewöhnlich in einem speziellen Konto, das man als *lokales System* bezeichnet, oder in einem Konto, das speziell für den Dienst konfiguriert ist. Mit Diensten steuert man normalerweise die Hardware, die an den Computer angeschlossen ist, oder stellt Ressourcen bereit, die jederzeit verfügbar sein müssen.

Direkt über das GUI können Dienste mit Benutzern nicht kommunizieren – statt dessen arbeiten sie mit dem *Dienststeuerungs-Manager* von Windows zusammen. Der Dienststeuerungs-Manager speichert Informationen über jeden Dienst, der auf dem Computer installiert ist, und verfolgt den Status des Dienstes beim Starten, Ausführen und Beenden.

4.2.2 Dienste und das Ereignisprotokoll

Im Gegensatz zu den meisten Programmen kann ein Dienst nicht davon ausgehen, dass immer ein Benutzer angemeldet ist. Der Dienst kann damit auch nicht einfach ein Dialogfeld oder eine andere Nachricht für den Benutzer anzeigen. Aus diesem Grund verwenden Dienste das *Ereignisprotokoll* von Windows als Speicherort für Nachrichten, die für einen Administrator oder Benutzer von Interesse sein können.

Um die Ereignisse im Ereignisprotokoll etwas zu ordnen, existierten standardmässig das *Anwendungs*-, *Sicherheits*- und *Systemprotokoll*. Für grössere Services besteht die Möglichkeit eigene Ereignisprotokolle in die Ereignisanzeige zu implementieren. So existiert zum Beispiel für den DNS-Dienst ein eigenes DNS-Protokoll. Für normale Anwendungen reicht aber meist das Anwendungsprotokoll.

4.2.3 Dienste und Systemsicherheit

Wie bereits erwähnt, ist eine Dienst gewöhnlich mit einem Computer, statt mit einem bestimmten Benutzer verbunden. Benutzer kommen und gehen, aber von einem Dienst erwartet man, dass er weiterhin läuft, auch wenn kein Benutzer auf einem Computer angemeldet ist. Windows erlaubt es den Diensten, Benutzeranmeldungen und –abmeldungen zu überdauern, indem die Dienste ihre Sicherheitsinformationen in der Datenbank des Dienststeuerungs-Managers hinterlegen können.

Wenn man einen Dienst erstellt und ihn mit dem Dienststeuerungs-Manager registriert, muss man dem Dienststeuerungs-Manager mitteilen, unter welchem Sicherheitskontext der Dienst laufen soll. Der Dienststeuerungsmanager verwendet die Kontoinformationen aus der Datenbank, wenn er den Dienst startet.

Windows Dienste verwenden drei verschiedene Kontenarten:

- Das lokale Systemkonto
- Das Konto des momentan interaktiven Benutzers
- Ein spezifiziertes Konto im System

Das lokale Systemkonto ist das einfachste aller drei und reicht für unser Projekt vollständig aus. Für dieses Konto hält sich der Verwaltungsaufwand in Grenzen. Man muss keine Kontoinformationen verwalten, in der Standardeinstellung kann man auf die komplette Steuerung des Computers zugreifen und ein Dienst wird nicht unterbrochen, wenn interaktive Benutzer Sitzungen auf dem Computer einrichten oder beenden. Der einzige Nachteil des lokalen Sicherheitskontos besteht darin, dass es in einer Domäne nur wenig Einfluss hat. Aber dies war für unsere Projekt nicht von Bedeutung.

4.2.4 Dienste programmieren

Wie bereits erwähnt, kommuniziert der Dienst während seiner Lebensdauer mit dem Dienststeuerungs-Manager. Der Dienststeuerungs-Manager startet den Dienst, und wenn sich dessen Zustand ändert, muss er den Dienststeuerungs-Manager darüber informieren.

Zum Beispiel ist der Dienststeuerungsmanager zu informieren, wenn der Dienst

- gerade gestoppt oder gestartet wird,
- seinen laufenden Zustand erreicht hat,
- pausiert
- oder stoppt.

Dienststeuerungs-Manager mit dem Dienststatus aktualisieren

Der Dienststeuerungs-Manager ist ständig über den Status des Dienstes zu informieren. Dies geschieht über die API-Funktion:

```
SetServiceStatus(hServiceStatus hStatus,
                 lpServiceStatus &ssStatus)
```

- `hServiceStatus`: Eine `SERVICE_STATUS_HANDLE`-Struktur, die ein erfolgreicher Aufruf der Funktion `RegisterServiceCtrlHandler` im vorneherein zurückgibt.
- `lpServiceStatus`: Ein Zeiger auf eine `SERVICE-STATUS`-Stuktur. Diese Struktur enthält mehrere Member-Variablen von denen für uns nur `dwServiceType` und `dwCurrentState` von Bedeutung waren. Über `dwServiceType` gibt man an ob der Dienst in einem eigenen Prozessraum läuft oder ob sich mehrere Dienste einen Prozess teilen. Und mit `dwCurrentState` teilt man dem Dienststeuerungsmanager den aktuellen Zustand mit. Hier gibt es mehrere Werte wie z. B. `SERVICE_START_PENDING, SERVICE_RUNNING ...`

Der Lebenszyklus eines Windows Dienstes

Der Lebenszyklus eines Windows Dienstes lässt sich in drei Hauptphasen gliedern:

- Dienst starten
- Dienst anhalten
- Dienst beenden

In den folgenden zwei Abschnitten wird nur das Starten und Stoppen gezeigt. Das Anhalten war für unsere zwei Dienste nicht von Bedeutung.

Einen Windows Dienst starten

Die Startsequenz für einen Windows Dienst ist in Abbildung 17 wiedergegeben.

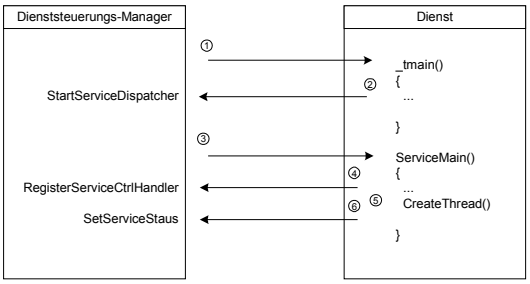


Abbildung 17: Die Reihenfolge beim Starten eines Dienstes

1. Der Dienststeuerungs-Manager ruft den Haupteinsprungspunkt für die registrierte ausführbare Datei auf, die den Dienst enthält.
2. Der Dienst führt einen Rückruf an die Funktion `StartServiceCtrl-Dispatcher` des Betriebssystems aus.
3. Der Dienststeuerungs-Manager ruft die Funktion `ServiceMain` des Dienstes auf.
4. Der Dienst registriert seine Steuerungsroutine mit der Funktion `Register-ServiceCtrlHandler`.
5. Der Dienst startet typischerweise einen Thread, der die Aufgaben des Dienstes ausführt.
6. Der Dienst benachrichtigt den Dienststeuerungs-Manager, dass der Dienst gestartet hat.

Der erste Schritt in Abbildung 17 zeigt, dass der Dienststeuerungs-Manager den Einsprungspunkt aufruft. Dies ist normalerweise `main`, `wmain` oder `WinMain` – wenn man das Makro `_tmain` benutzt entscheidet der Compiler zwischen dem normalen `main` und dem UNICODE `wmain`.

Listing 4.2.1: Minimalversion einer main-Funktion in Windows

```
 1  int cdecl _tmain()
 2  {
 3    SERVICE_TABLE_ENTRY ste[] =
 4    {
 5      _T("MyService"), ServiceMain,
 6      NULL, NULL
 7    };
 8    StartServiceCtrlDispatcher(ste);
 9    return 0;
10  }
```

In der Struktur `SERVICE_TABLE_ENTRY` ist vor allem die Einsprungsadresse der Servicehauptfunktion `ServiceMain` von Bedeutung.

Nun folgt das Listing für die ServiceMain Funktion, das eigentliche Programm, welches letztendlich arbeiten soll.

Listing 4.2.2: Beispiel einer ServiceMain-Funktion

```
 1  VOID WINAPI ServiceMain(DWORD argc, TCHAR* argv[])
 2  {
 3    SERVICE_STATUS_HANDLE ssh;
 4    ssh = RegisterServiceCtrlHandler(_T("MyService"), MyCtrlHandler);
 5
 6    // Dienststeuerungs-Manager über den Start benachrichtigen.
 7    TellScmThatServiceIsStarting();
 8
 9    // Dienststeuerungs-Manager benachrichtigen, dass Dienst läuft.
10    TellScmThatServiceIsRunning();
11
12    // Dienst ausführen
13    RunService();
14
15    // Dienststeuerungs-Manager benachrichtigen, dass Dienst stoppt.
16    TellScmThatServiceHasStopped();
17  }
```

Das Listing 4.2.2 ist in sehr vereinfachter Form gehalten. Trotzdem zeigt es die grundsätzlichen Schritte, die jede `ServiceMain`-Funktion einhalten muss:

1. Die Steuerungsroutine des Dienstes ist über die Funktion `RegisterService-CtrlHandler` zu registrieren.
2. Den Dienststeuerungs-Manager über den aktuellen Zustand benachrichtigen.
3. Der Dienst sollte alle erforderlichen Initialisierungen ausführen. Falls dies fehlschlägt muss der Dienststeuerungs-Manager unbedingt benachrichtigt werden, dass der Dienst stoppt. Ansonsten ist es schwierig den Dienst wieder aus dem Dienststeuerungs-Manager zu entfernen.
4. Nach Abschluss der Initialisierung kann der Dienst mit seiner eigentlichen Arbeit beginnen. Meist wird an dieser Stelle ein Workerthread gestartet. Dies vereinfacht die Verwaltung – z. B. kann der Thread über Events wieder gestoppt werden, falls dies vom Dienststeuerungs-Manager verlangt wird.

Nachdem die Steuerungsroutine beim Dienststeuerungs-Manager registriert wurde, wird diese bei Bedarf vom Dienststeuerungs-Manager aufgerufen. Wenn der Service zum Beispiel gestoppt werden soll, wird die Steuerungsroutine aufgerufen. Das Listing 4.2.2 zeigt eine vereinfachte Implementierung.

Listing 4.2.3: Steuerungsroutine

```
 1  VOID WINAPI MyCtrlHandler(DWORD dwControl)
 2  {
 3    switch(dwControl)
 4    {
 5      case SERVICE_CONTROL_PAUSE:
 6        PauseTheService();
 7        break;
 8
 9      case SERVICE_CONTROL_CONTINUE
10        ResumeTheService();
11        break;
12
13      case SERVICE_CONTROL_STOP
14        StopTheService();
15        break;
16
17        // andere Fälle
18    }
19  }
```

Einen Windows Dienst stoppen

Die Startsequenz für einen Windows Dienst wird in Abbildung 18 aufgezeigt.

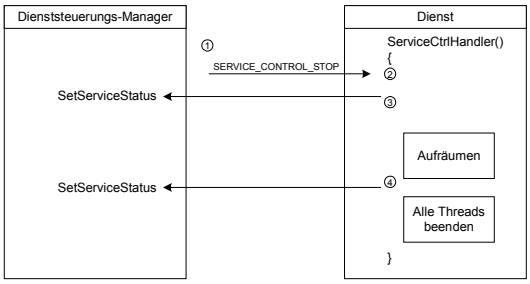


Abbildung 18: Der Ablauf beim Herunterfahren eines Dienstes

1. Der Dienststeuerungs-Manager sendet eine `SERVICE_CONTROL_STOP`-Nachricht an die Steuerungsroutine des Dienstes.
2. Der Dienst teilt wiederum dem Dienststeuerungs-Manager mit `SERVICE_STOP-PENDING` mit, dass er am beenden ist.
3. Alle Ressourcen freigeben.
4. Mit `SERVICE_STOPPED` das Beenden bestätigen.

Modifizierte Version für einfacheres Debuggen

Weil ein Dienst keine direkte Ausgaben an den Benutzer machen kann, sondern nur an die Ereignisanzeige, kann das Programmieren von Diensten etwas mühsam sein. Eine Abhilfe davon kann mit folgendem Code geschaffen werden. Zuerst wird getestet wer den Dienst gestartet hat. Falls er vom Dienststeuerungs-Manager gestartet wurde, gibt er mögliche Meldungen direkt an die Ereignisanzeige. Wird er jedoch von der Kommandozeilen, bzw. Entwicklungsumgebung gestartet, gehen die Meldungen zurück ans Kommandofenster.

Listing 6.4.2.4: Umschalten zwischen Kommandozeilenausgabe und Ereignisanzeige

```
 1  bool g_RunningAsService;
 2  int cdecl _tmain()
 3  {
 4    SERVICE_TABLE_ENTRY ste[] =
 5    {
 6      _T("MyService"), ServiceMain,
 7      NULL, NULL
 8    };
 9
10    g_RunningAsService = true;
11    if (StartServiceCtrlDispatcher(ste) == false)
12    {
13      // Keine Verbindung zum Dienststeuerungs-Manager, weil Start von
14      // der Befehlszeile
```

```
15      g_RunningAsService = false;
16      ConsoleMain();
17    }
18    return 0;
19  }
20
21  // ServiceMain für diesen Dienst
22  VOID WINAPI ServiceMain(DWORD argc, TCHAR* argv[])
23  {
24    ...
25    return 0;
26  }
27
28  // ConsoleMain für diesen Dienst
29  int ConsoleMain()
30  {
31    ...
32    return 0;
33  }
```

Daten in das Ereignisprotokoll schreiben

Diese scheinbar einfache Aufgabe hat sich doch noch als grössere Hürde erwiesen. Um mehrsprachige Applikationen zu unterstützen gibt es unter Windows Nachrichtenressourcendateien, die für jedes Projekt in den gewünschten Sprachen erstellt und dann mit dem MessageCompiler erzeugt werden können. Für unser Projekt wäre dies aber viel zu aufwendig gewesen. Wenn nun ohne diese Dateien gearbeitet wird, kann die Ereignisanzeige die Meldungen nicht richtige auflösen und die Ausgabe sieht dann wie in Abbildung 19 aus.

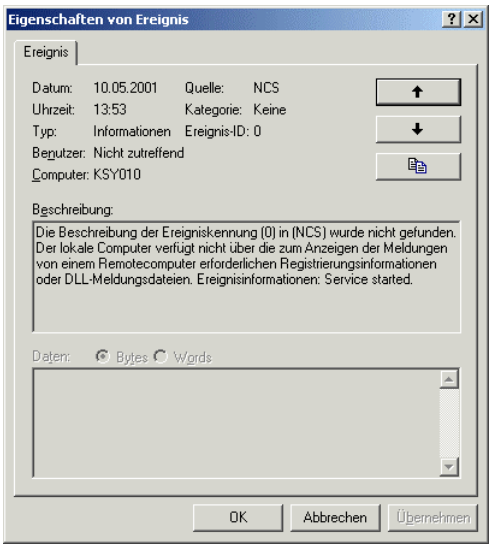


Abbildung 19: Anzeige einer Nachricht die nicht richtig formatiert ist

Um dies zu verhindern muss trotzdem eine minimale Nachrichtenressourcendatei geschrieben werden und dann mit dem MessageCompiler übersetzt werden. Weil dies sich als etwas trickreich herausgestellt hat und wir nirgends eindeutige Beschreibungen gefunden haben, sind im folgenden nochmals alle Schritte aufgelistet.

Listing 4.2.5: Nachrichtenresoucendatei ServiceEventLogMsg.mc

```
 1  MessageIdTypeDef=DWORD
 2
 3  SeverityNames=(Informational  = 0x1: STATUS_SEVERITY_INFORMATIONAL
 4                 Error          = 0x3: STATUS_SEVERITY_ERROR
 5                 )
 6
 7  LanguageNames=(Neutral=0x0000:MSG00000}
 8
 9  MessageId=0x0 SymbolicName=MSG_INFO_1
10  Severity=Informational
11  Facility=Application
12  Language=Neutral
13  %1
14  .
15
16  MessageId=0x1 SymbolicName=MSG_ERROR_1
17  Severity=Error
18  Facility=Application
19  Language=Neutral
20  %1
21  .
```

Diese Datei muss dann mit dem MessageCompiler übersetzt werden. Dieser erzeugt dann die beiden Dateien ServiceEventLogMsg.h und ServiceEventLogMsg.rc die in Visual C++ in den Workspace eingebunden werden können. Unter Project-Settings kann der Aufruf des MessageCompiler gemäss Abbildung 20 automatisiert werden.

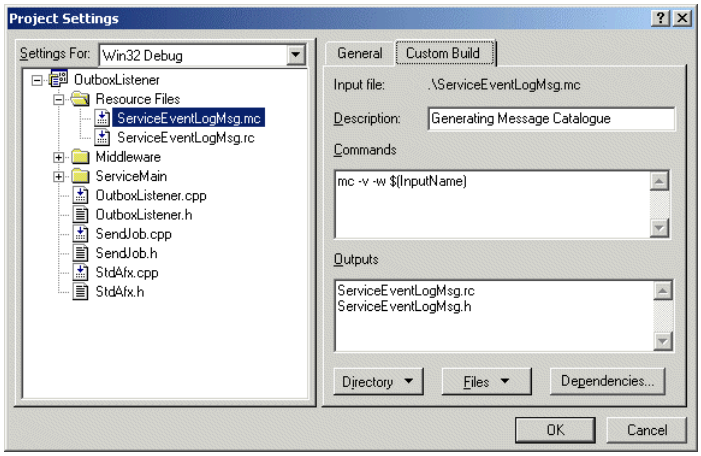


Abbildung 20: Project Settings für den MessageCompiler

Als letzter Schritt muss der Dienst in der Windows Registry eingetragen werden und zwar an folgenden Stellen:

- Beim Schlüssel (wobei myService der Name des Dienstes ist):
  `[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet\Services\Eventlog\Application\myService]`
  Müssen die folgenden zwei Werte gesetzt werden:
  `EventMessageFile = [Pfad]\[Dienst-Dateiname]`
  `TypesSupported = [alle unterstützen Typen]`

- Beim Schlüssel:
  `[HKEY_LOCAL_MACHINE\SYSTEM\ControlSet\Services\Eventlog\Application]`
  Muss der Name (nicht der Dateiname!) dem Wert `Sources` hinzugefügt werden.
  Achtung dieser Wert ist vom Typ REG_MULTI_SZ, d. H ein WideChar. Also nicht vergessen den Namen richtig zu formatieren.

Nach diesen Schritten ist die Ausgabe an das Ereignisprotokoll trivial:

Listing 4.2.6: Ausgabe einer Meldung an das Ereignisprotokoll

```
 1  VOID WriteToEventLog(WORD Type, LPCTSTR pMsg)
 2  {
 3    DWORD EventID;
 4    HANDLE hEventSrc;
 5    switch (Type) {
 6      case EVENTLOG_INFORMATION_TYPE:
 7        EventID = MSG_INFO_1;
 8        break;
 9      case EVENTLOG_ERROR_TYPE:
10        EventID = MSG_ERROR_1;
11        break;
12    }
13
14    hEventSrc = RegisterEventSource(NULL, g_szService);
15
16    // Ausgabe der Meldung
17    ReportEvent(hEventSrc, Type, 0, EventID, NULL, 1, 0, pMsg, NULL);
18
19    DeregisterEventSource(hEventSrc);
20  }
```

Registrierung eines Dienstes

Bei der Implementierung unserer Dienste haben wir die Registrierung relativ einfach gehalten. Die beiden Dienste können mit den Befehlen

`myService.exe /register`

`myService.exe /unregister`

registriert, bzw. wieder entfernt werden. Wie dies gemacht werden kann ist dem Source-Code zu entnehmen. Eine schönere Lösung wäre ein Applet für die Systemsteuerung oder ein PlugIn für die Microsoft Management Konsole unter Windows 2000, aber dafür fehlte uns leider die Zeit.

4.3 Das Active Directory

Microsoft hat als Basis der verteilten Netzwerkstruktur von Windows 2000 einen neuen Verzeichnisdienst eingeführt, das ‚Active Directory'. Dieses bietet „sicheres" und strukturiertes bzw. hierarchisches Ablegen von Information über interessante Objekte eines Unternehmens-Netzwerkes. So können z.B. Benutzer, Computer, Services und vieles mehr im Active Directory verwaltet werden. Dabei bietet dieses reichhaltige Unterstützung beim Auffinden von Information und dem Arbeiten damit.

Das Werkzeug zur Verwaltung des Active Directory's ist ein Snap-In der Microsoft Management Console (MMC). Im deutschen ‚Windows 2000 Advanced Server' heisst es ‚Active Directory-Benutzer und – Computer' und ist über ‚Start' – ‚Programme' – ‚Verwaltung' erreichbar. Allerdings muss der Server als Domänencontroller konfiguriert sein, damit das Active Directory darauf als Server-Dienst eingesetzt werden kann.

In unserer Projektarbeit haben wir die Fähigkeiten der Active Directory Services für die Userverwaltung (Datenbank, Back-End) verwendet. Dazu haben wir Gebrauch vom ‚Active Directory Service Interface' – kurz ADSI – gemacht, welches das Abfragen, Ändern und Ergänzen des Verzeichnisinhaltes unterstützt. Zu diesem Zweck stellt es eine riesige Menge an COM-Schnittstellen zur Verfügung, von denen wir in unseren Modulen rege Gebrauch gemacht haben. Da die Implementationen dieser COM-Schnittstellen praktisch ausnahmslos die IDispatch-Schnittstelle (siehe Kapitel ‚Das Component Object Model') implementieren, sind diese auch per Skript steuerbar:

Listing 4.3.1: Ein Beispiel für das Abfragen der Mobile-Nr eines Benutzers (VBScript):

```
 1  Dim obj
 2  Dim user, userobj
 3  Dim mobile
 4
 5  user = InputBox("Please specify the user:","Get mobile by user")
 6
 7  set obj = GetObject("LDAP://ksy010/OU=SMSUsers,DC=pa1,DC=ch")
 8  obj.filter = Array("User")
 9  for each userobj in obj
10    if userobj.cn = user then
11          mobile = userobj.Get("Mobile")
12          exit for
13    end if
14  next
15  set obj = Nothing
16
17  if mobile="" then
18    MsgBox "User not found or no mobile nr in the AD for user " & user
19  else
20    MsgBox "The mobile nr for the user '" & user & "' is " & mobile
21  end if
```

Bemerkungen:

Zeile 1-3 Deklarationen der gebrauchten Variablen (wäre in VBScript nicht nötig, verbessert aber die Übersicht).

Zeile 7 Erzeugen eines Active Directory-Zugriffsobjektes. Hier per LDAP-Zeichenfolge realisiert. Das Objekt referenziert die angegebene Organisationseinheit (OU = ‚Organizational Unit') ‚SMSUsers'.

Zeile 8 Filtern der Unterobjekte des Objektes ‚SMSUsers' nur nach Usern.

Zeile 9 Schleife über alle User-Objekte der Organisationseinheit ‚SMSUsers'.

Zeile 11 Falls der gesuchte User gefunden wurde, wird hier dessen Mobile-Nr (Eigenschaft ‚Mobile') abgerufen.

Das ADSI ist übrigens nicht nur auf das Active Directory als Verzeichnisdienst beschränkt, sondern erlaubt auch den Zugriff auf andere Directory Services, wie z.B. NDS von Novell, Netscape's Directory Service oder LDAP 2.0-kompatible Verzeichnisdienste.

4.4 GSM Modem Steuerung

4.4.1 Allgemeines

Für unsere Projektarbeit stand uns das GSM-Modem M20 von Siemens zur Verfügung. Das M20 unterstützt im wesentlichen alle Funktionen, die man von Handys her kennt. Für den Betrieb benötigt man eine normale SIM-Karte. Wie beim Handy werden die Benutzerdaten wie Adressbuch, SMS, Ruflisten usw. auf der SIM-Karte abgespeichert.

Das Modem kann über eine RS232 Schnittstelle angesteuert werden. Die Standardeinstellung ist *19200 Bits pro Sekunde, 8 Daten Bits , keine Parität* und *1 Stopbit*.

M20 Hochfahren und Einbuchen ins Netz

1. Das M20 Terminal wird durch drücken des Zündschalters am Netzteil gestartet. Dann sollte die LED am Modem mit Blinken beginnen.
2. Nun muss der PIN eingegeben werden. Nach etwa 5 Sekunden leuchtet die LED kontinuierlich. Nun ist das M20 bereits im normalen Betriebszustand.

4.4.2 Beispiel einer Terminalsession

Im folgenden Listing beschränken wir uns nur auf die projekt-relevanten Schritte. Die empfangenen Zeichen sind kursiv angeben.

Listing 4.4.1: Einloggen, initialisieren und empfangen einer Message

```
 1  OK
 2  AT+CPIN?            // Abfrage was für ein PIN benötigt wird
 3  +CPIN: SIM PIN
 4
 5  OK
 6  AT+CPIN="1234"      // Eingabe des Pins
 7  OK
 8
 9  AT+CMGF=1           // von PDU in Textmodus wechseln
10  OK
11
12  +CMTI: "SM", 1      // Dies wird signalisiert wenn ein neues
13                      // SMS empfangen wurde. Hier Nr. 1
14
15  AT+CMGL="ALL"       // Auflisten aller SMS Nachrichten
16  +CMGL: 1,"REC UNREAD","+41763392464",,"01/05/11,09:35:06+80"
17  Doejohn hi john
18
19  OK
20  AT+CMGD=1           // Nachricht Nr. 1 löschen
21  OK
```

Wie Listing 4.4.1 zeigt, ist die ganze Sache ziemlich einfach. Mit der Abfrage `AT+CPIN?` in Zeile 2 kann man prüfen, ob der PIN schon eingegeben wurde. Dies ist von Vorteil, wenn zum Beispiel nur der Dienst neu gestartet wird und das Modem aber immer noch im Betriebszustand läuft. Falls der PIN eingegeben werden muss, geschieht dies über `AT+CPIN="xxxx"`.

Glücklicherweise unterstützt das M20 neben dem Bytecodierten PDU-Mode, auch einen normalen Textmode. Dies wird in Zeile 9 mit `AT+CMGF=1` gesetzt. Nun ist das M20 im gewünschten Betriebsmodus.

Wenn nun ein neues SMS empfangen wird, sendet das M20 die folgende Zeichenfolge (wie Zeile 12 zeigt): `+CMTI: „SM", 1`

Wenn diese Meldung empfanden wird, kann die SIM Karte mit dem Befehl `AT+CMGL="ALL"` ausgelesen werden. Nun folgt eine Liste mit allen SM im Speicher. Weil unser Dienst die neuen Meldungen fortlaufend verarbeitet und dann wieder löscht, wird im Betriebszustand immer nur eine Meldung erscheinen. Wenn aber der Dienst neu gestartet wird, können hier auch mehrere Meldungen erscheinen. Wie wir die MessageId, den Absender und die Meldung aus dieser Liste extrahiert haben, zeigt das Kapitel 0.

Wenn die SM erfolgreich verarbeitet wurde, kann sie mit `AT+CMGD=1` wieder gelöscht werden. Wobei die Zahl am Ende dieses Befehls, der MessageId entsprechen muss.

Listing 4.4.2: Liste aller auf der SIM-Karte gespeicherten Meldungen

```
+CMGL: 1,"REC READ","232",,"01/04/18,13:32:08+00"
This is the first message.
+CMGL: 2,"REC READ","0041763333333",,"01/04/18,13:42:46+80"
This is the second message.
OK
```

4.4.3 Auslesen eine Meldung mit Regular Expressions

Um das Auslesen der Meldung möglichst einfach und fehlertolerant zu machen, haben wir dies mit regulären Ausdrücken realisiert. Dazu haben wir die XRegExp Klasse von Patrik Dreyer von der Enterprise Communcations AG benutzt. Die Syntax entspricht dem allgemeinen Standard für Regular Expressions und wird hier nicht näher erklärt. Speziell ist nur die Zusatzfunktion `replace` für das ‚Auseinandernehmen' der einzelnen Tokens. Mit `replace` kann ein Ausdruck der in Klammern steht extrahiert werden, wobei `replace(„\\1")` dem Ausdruck in der ersten Klammer entspricht.

Listing 4.4.3: Auslesen der ersten Meldung

```
1  RegExp.set("(.*)(\+CMGL:.?)([0-9]+)(,"REC.*",")([+0-9]*)(",.*,".*")
2              (.*)(\+CMGL.)*(OK)");
3  if (RegExp.find(Buffer) != -1)
4  {
5    MsgId     = (const char*)RegExp.replace("\3");
6    Sender    = (const char*)RegExp.replace("\5");
7    Message   = (const char*)RegExp.replace("\7");
8  }
```

Der reguläre Ausdruck in Listing 4.4.3 sieht auf den ersten Blick etwas kryptisch aus. Anderseits zeigt es auch, dass mit nur einigen Zeilen Code ein mehr oder weniger komplizierter String, der mit unterschiedlichen Token getrennt ist, auf sehr effiziente Art aufgeteilt werden kann. Letztendlich kopiert dieser Programmabschnitt die MeldungsId, den Absender und die Meldung der ersten SM in drei Stringvariablen.

4.4.4 Ansteuerung des COM-Ports unter Win32

Der COM Port wird unter Windows wie eine normale Datei behandelt. Geöffnet wird er also über die Funktion CreateFile und gelesen über ReadFile, bzw. geschrieben über WriteFile. Der COM Port kann nach dem Öffnen über die Funktion SetCommState mit den gewünschten Werten gesetzt werden.

Für langsame I/O-Operation müssen Daten im asynchronen Modus (OVERLAPPED) betrieben werden. Für rechenintensive Programme hat dies den Vorteil, dass man während den Wartezeiten andere Operationen ausführen kann.

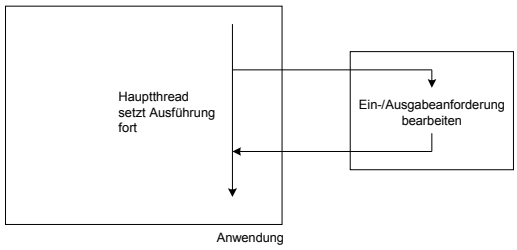


Abbildung 21: Asynchrone Dateieingabe/-ausgabe

Dieser Ablauf muss folgendermassen implementiert werden:

1. Zuerst muss in der Struktur `OVERLAPPED` unter `hHandle` ein Ereignisobjekt erstellt werden (siehe **Listing 4.2.2**)
2. Lese-/Schreiboperation mit `ReadFile/WriteFile` in Auftrag geben.
3. Mit den Synchronisationsfunktionen `WaitForSingleObject`, bzw. `WaitFor-MultipleObjects` warten bis die IO-Operation beendet hat.
4. Mit `GetOverlappedResult` erhält man das Resultat.

Weil unsere Dienste keine rechenintensiven Schritte ausführen müssen, macht die asynchrone Dateieinegabe/-ausgabe für unsere Anwendung wenig Sinn. Deshalb warten unsere Schreib- und Lesefunktionen bis die Übertragung beendet ist (Siehe Abbildung 22).

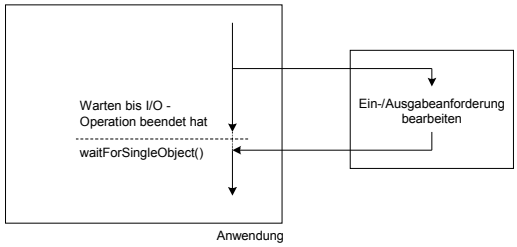


Abbildung 22: Asynchrone Dateieingabe/-ausgabe

Die folgenden Listings sollen die Kommunikation mit dem GSM-Modem verdeutlichen.

4.4.5 Beispiel-Programme

Listing 4.4.4: Öffnen der Com-Schnittstelle und initialisieren des EvtChar-Ereignisses:

```
 1  void OpenCom(WCHAR *pPort)
 2  {
 3    HANDLE   hCom;
 4    DCB      dcb;
 5    OVERLAPPED Overlapped;
 6
 7    hCom = CreateFile(pPort,
 8               GENERIC_READ | GENERIC_WRITE,
 9               0,         //com-devices must have exclusive-access
10               NULL,      //no security attributes
11               OPEN_EXISTING,
12               FILE_FLAG_OVERLAPPED,    // overlapped I/O
13               NULL       //hTemplate must be NULL for com-devices
14               );
15
16    GetCommState(hCom, &dcb);
17    dcb.BaudRate = 19200;
18    dcb.ByteSize = 8;
19    dcb.Parity   = NO_PARITY;
20    dcb.StopBits = 1;
21    dcb.EvtChar  = 10;  // the event-char is the ASCII Code 10 for NL.
22    SetCommState(hCom, &dcb);
23
24    // Create an event object for the overlapped structure.
25    Overlapped.hEvent = CreateEvent(NULL,   // no security attributes
26                                    FALSE,  // auto reset event
27                                    FALSE,  // not signaled
28                                    NULL    // no name
29                                    );
30    // Starts the event listener
31    SetCommMask(hCom, EV_RXFLAG);
32  }
```

Listing 4.4.5: Leseoperation vom Com-Gerät

```
 1  void readCom(CString &Buffer)
 2  {
 3    COMSTAT Comstat;
 4    DWORD Readed;
 5    DWORD Error;
 6
 7    // gets as many bytes as there are in the queue.
 8    ClearCommError(hCom, &Error, &Comstat);
 9
10    ReadFile(hCom,
11             Buffer.getBuffer(Comstat.cbInQue + 1),
12             Comstat.cbInQue,
13             NULL,
14             &Overlapped);
15
16    // Waits until the read-operation has finished.
17    WaitForSingleObject(m_Overlapped.hEvent, INFINITE);
18
19    GetOverlappedResult(hCom, &Overlapped, &Readed, TRUE);
20
21    Buffer.releaseBuffer(Readed);
22  }
```

Listing 6.4.4.6: Schreiboperation auf das Com-Gerät

```
 1  void writeLn(CString Buffer)
 2  {
 3    DWORD Written;
 4
 5
 6    WriteFile(hCom,
 7              (const char*)Buffer,
 8              Buffer.getLength(),
 9              NULL,
10              &Overlapped);
11
12    // Waits until the write-operation has finished.
13    WaitForSingleObject(Overlapped.hEvent, INFINITE);
14    GetOverlappedResult(hCom, &Overlapped, &Written, TRUE);
15  }
```

4.4.6 Auf Ereignisse am COM-Port warten

Um SMs aus dem Modem zu lesen, könnte man mit Polling von Zeit zu Zeit den Speicher auslesen und testen ob neue SMs anliegen. Weil aber das M20 eine Meldung ausgibt, wenn eine neue SM anliegt, ist dies viel eleganter über Ereignisse zu lösen. Dazu kann man in der DCB Struktur unter `EvtChar` ein gewüschtes Zeichen angeben, für das ein Ereignis ausgelöst werden soll. Wenn nun zum Beispiel `EvtChar='a'` gesetzt ist, wird sobald ein ‚a' in der Empfangs-Queue anliegt, ein Ereignis ausgelöst. Weil die AT-Kommandos immer mit CR+LF beendet werden, haben wir den `EvtChar` auf LF, bzw. den ASCII-Char 10 gesetzt. Das Ereignis wird mit `WaitCommEvent` abgefangen. Weil die Com-Schnittstelle im asynchronen Modus betrieben werden muss, überspringt das Programm die `WaitCommEvent`-Funktion. Deshalb muss auch hier mit `WaitForSingleObject` auf das Ereignis gewartet werden:

Listing 6.4.4.7: Auf ein Zeichen am Com-Gerät warten:

```
 1  bool waitForEvent()
 2  {
 3    DWORD EvtMask;
 4    DWORD Error;
 5    /**
 6     * Here it waits until the ascii char 10 appears in the incomming
 7     * data queue. Because we're running the com-device in overlapped
 8     * mode, the program waits first on WaitForSingleObject
 9     */
10    if (!WaitCommEvent(hCom, &EvtMask, &Overlapped))
11    {
12      // Checks if the com-device is pending.
13      if (GetLastError() == ERROR_IO_PENDING)
14      {
15        WaitForSingleObject(Overlapped.hEvent, INFINITE);
16        if (EvtMask & EV_RXFLAG)
17        {
18          return true;
19        }
20      }
21    }
22    return false;
23  }
```

4.5 Webseiten dynamisch erzeugen mit ASP

Während Webseiten häufig in statischer Form auf den Webservern liegen, also als reine HTML-Dokumente gespeichert sind, braucht es bei Seiten mit sich dauernd änderndem Inhalt die Möglichkeit, diese dynamisch zu erzeugen, um immer die aktuellsten Informationen anzeigen zu können. Dies ist insbesondere dann nötig, wenn Datenbanken mit einem Front-End basierend auf Web-Technologie ergänzt werden.

Zur Erzeugung des Front-Ends unseres Instant Messengers haben wir auf der Server-Seite eine weitere Microsoft-Technologie eingesetzt: die ‚Active Server Pages', kurz ‚ASP'. Das Einbetten von server-seitigem Skript-Code erfolgt dabei in beinahe derselben Art und Weise, wie dies bei den meisten anderen vergleichbaren Technologien erfolgt (z.B. PHP oder auch Java mit den Java Server Pages, kurz JSPs). HTML und Skript-Code können wie üblich bunt durchmischt werden, wobei ein Skript-Block durch die Zeichenfolgen ‚<%' (Einleiten eines Skript-Blocks) und ‚%>' (Abschliessen eines Skript-Blocks) eingerahmt werden muss. Als Skript-Sprache kommt dabei VBScript zum Einsatz, welches z.B. vom IIS (Internet Information Sever), welcher u.a. auf Windows 2000-Systemen eingesetzt werden kann, interpretiert wird. Auf Client-Seite kommt reines HTML an, welches von den Browsern wie üblich dargestellt werden kann.

Listing 4.5.1: Beispiel einer einfachen ASP-Seite

```
 1  <html>
 2
 3  <head>
 4  <title>ASP-Beispiel</title>
 5  </head>
 6
 7  <body>
 8  <p>Diese Webseite liegt auf dem Server
 9  <%
10    Response.Write Request.ServerVariables("SERVER_NAME")
11  %>
12  und wurde von der Adresse
13  <%
14    = Request.ServerVariables("REMOTE_HOST")
15  %>
16  abgerufen</p>
17  </body>
18
19  </html>
```

Bemerkungen

Zeile 1-8 / 16-19	HTML-Rahmen der Webseite
Zeile 9 / 13	Einleitung des Skript-Blocks
Zeile 10 / 14	Ausgabe von Server-Informationen. Der IIS stellt einige Objekte standardmässig zur Verfügung, so z.B. das Request-Objekt, welches u.a. auch dazu verwendet wird, den Inhalt von Formularfeldern abzufragen)
Zeile 11 / 15	Abschluss des Skript-Blocks

Unter anderem diente uns ASP in der vorliegenden Projektarbeit auch als Rahmen für das Sammeln von Erfahrungen im COM-Umfeld. So haben wir zu Beginn diverse Test-Seiten erstellt, mit welchen wir z.B. den Zugriff auf das Active Directory mit ADSI-Objekten erlernt haben.

Listing 4.5.2: Beispiel, welches via ADSI auf die Active Directory Services zugreift:

```
 1  <html>
 2
 3  <head>
 4  <title>ADSI-Testzugriff</title>
 5  </head>
 6
 7  <body>
 8  <p>In 'SMSUsers' befinden sich folgende Benutzer:<br>
 9  <%
10    call Main
11  %>
12  </p>
13  </body>
14
15  </html>
16
17  <%
18
19  ' Hauptroutine
20  sub Main
21    Dim user
22    Dim ad
23    Dim cnt
24
25    set ad = GetObject("LDAP://ksy010/OU=SMSUsers,DC=pa1,DC=ch")
26    ad.filter = Array("User")
27    cnt = 0
28    for each user in ad
29          cnt = cnt+1
30          Response.Write "- " & user.cn & "<br>"
31    next
32    Response.Write "Es wurde(n) " & cnt & " User gefunden"
33    set ad=nothing
34  end sub
35
36  %>
37
```

Kapitel 5 Installation / Benutzeranleitung

5.1 Installation

Die grösste Arbeit für die Installation ist das aufsetzen des Windows 2000 Servers mit Exchange, einer eigenen Domain und Active Directory Unterstützung. Wie dies funktioniert wird aber in diesem Dokument nicht verraten, denn einerseits haben wir uns bereits im KSy-Praktikum genügend damit herumgeschlagen und anderseits gibt es genügend Literatur zu diesem Thema.

Der Rest ist dann ein Kinderspiel:

1. Im Active Directory muss eine neue *Organisationseinheit* mit dem Namen „SMSUsers“ erstellt werden. In diese Einheit können dann alle Benutzer, die für den IMS berechtigt sein soll, vom User-Verzeichnis verschoben werden. Es können auch neue Kontakte für andere Benutzer erstellt werden.
2. Alle COM-Objekte und Dienste im Verzeichnis „Binaries“ registrieren. Die DLLs vom Verzeichnis „Binaries\system32“ ins Systemverzeichnis kopieren. Einige Einträge für die EasySMS.dll in die Registry schreiben, usw.

   Um dies zu vereinfachen haben wir im Verzeichnis „Binaries“ auf der CD-Rom die Batch-Datei „*register.bat“*, bzw. „*unregister.bat“* erstellt, die all dies übernimmt.
3. Jetzt muss man nur noch das Modem am Com-Port 1 anhängen und die beiden Dienste *OutboxListener* und *SMSReceiver* im Dienst-Manager starten. Ob beides geklappt hat, kann noch in der Ereignisanzeige unter Anwendungen nachgeprüft werden.

5.2 Bedienung übers Handy

Um den Instant Messenger Service zu benutzen muss die Telefonnummer des GSM Modems gespeichert werden. Über diese Nummer erfolgt dann jegliche Kommunikation.

Damit die Meldung den richtigen Weg findet, muss man im SMS-Text den gewünschten Empfänger angeben. Wenn die Meldung an mehrere Empfänger gesendet werden soll, werden diese durch Kommas getrennt. Natürlich kann man auch eine Meldung an Gruppen senden. Die eigentliche Meldung muss mit einem Leerzeichen von der Empfängerliste getrennt werden.

Beim Instant Messenger Service kann im SMS zusätzlich angegeben werden, wie die Meldung den Empfänger erreichen soll – über E-Mail oder SMS. Dazu gibt man vor der Benutzerliste

- `S:`, `s:`, `SMS:`, oder `Sms:` an, wenn ein SMS weiter gesendet werden soll und
- `E:`, `e:`, `EMAIL:` oder `EMail:` an, wenn ein EMAIL gesendet werden soll.

Wenn nichts angeben wird, bekommt der Empfänger ein E-Mail.

Abbildung 23 zeigt einige Beispiele:

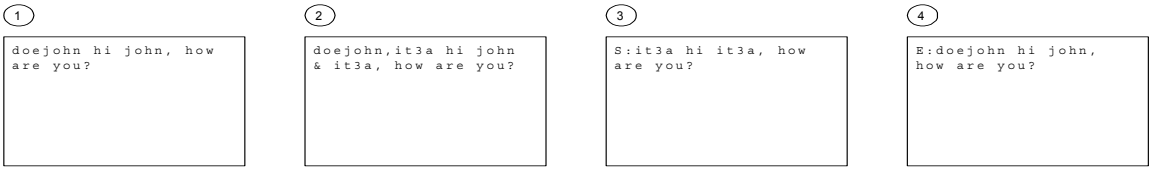


Abbildung 23 Eingabebeispiele für Bedienung übers Handy

1. Eine E-Mail Meldung an John Doe.
2. Eine E-Mail Meldung an John Doe und die Klasse IT3a.
3. Eine SMS Meldung an alle Studenten der Klasse IT3a.
4. Eine E-Mail Meldung an John Doe.

5.3 Bedienung übers Web-Interface

Die Bedienung über den Browser ist unserer Meinung nach sehr intuitiv. Wir haben uns an gängige Regeln gehalten und das Interface gleich aufgebaut, wie es von den meisten Web-basierten E-Mail-Clients her bekannt ist. Aus diesem Grund, verzichten wir an dieser Stelle auf eine detaillierte Bedienungsanleitung.

Anhang A Quellenverzeichnis

Bücher

[B1] „Das UML-Benutzerhandbuch“, G. Booch u.a.
Addison-Wesley 1999, ISBN 3-8273-1486-0

[B2] „GSM and personal communication handbook“, S. M. Redl u.a.
Artech House Publishers 1998, ISBN 0-89006-957-3

[B3] „Windows 2000 developer’s guide“, N. Williams / C. Gross
Markt & Technik 2000, ISBN 3-8272-5702-6

[B4] „Go To COM“, P. Loos
Addison-Wesley 2001, ISBN 3-8273-1678-2

[B5] „Programmieren mit COM und CORBA“, J. Hofmann u.a.
Carl Hanser Verlag 2001, ISBN 3-446-21479-8

Webseiten

Die wichtigsten Links (weitere auf der Begleit-CD):

[URL1] www.icq.com/icq123/replysmsmess.html
Anleitung für ICQ Reply-Funktion

[URL2] www.mtn.co.za
MTN South Africa – SMSC von ICQ

[URL3] www.ecall.ch
Schweizer SMS-Provider (benutzt Methode mit mehreren Modems)

[URL4] http://www.nobbi.com/sms_pdu.htm
Sehr informative Seite über den PDU-Header. Bietet auch eine Software für das decodieren des PDU-Formats an.

[URL5] http://telefoner.siemens.se//fragorsvar/downloads/a1_sms.pdf
Spezifikationen vom PDU-Format

Sonstiges

[S1] „MSDN Library – October 2000“, Diverse
Microsoft Developer Network 2000

Stichwortverzeichnis

O

P

R

S

T

U

V

W

X

Z