# SyncStar

## Datensynchronisation zwischen Microsoft Outlook und einem Mobiltelefon

**Studienarbeit August 2003**

**HAW Hamburg / Fachbereich TI**

**Betreut durch Prof. Dr. Kai von Luck**

Verfasser:

Heiko Juretzka

Matr.-Nr. 1383425

Inhaltsverzeichnis

# Einleitung

Der Computer gehört in der heutigen Zeit zur Standardausstattung eines jeden Haushaltes. Seine Einsatzmöglichkeiten sind nahezu unbegrenzt. In der Regel wird er aber nur für ein paar wenige Anwendungen benutzt. Hierzu gehören in erster Linie die Nutzung als Schreibmaschine, das E-Mailen, Surfen, Chatten sowie Homebanking und das Verwalten der persönlichen Adressen und Termine.

Es stellt sich die Frage, warum sein Einsatzgebiet im privaten Bereich so eingeschränkt ist. Eine Ursache ist sicherlich, dass in den 90iger Jahren das Hauptaugenmerk der Forschung auf der Entwicklung leistungsfähigere Systeme lag. Es wurde weniger darüber nachgedacht, welche Anforderungen eine Privatperson an einen Computer stellt. Das Resultat ist, dass viele Haushalte über einen leistungsstarken Rechner verfügen, aber seine Möglichkeiten nicht annähernd ausschöpfen.

Der Boom der letzten Jahre im Bereich der PDAs zeigt deutlich den Trend, den viele Anwender fordern. Hierbei handelt es sich um Mobilität, welche der Standardcomputer aufgrund seiner Bauweise nicht bieten kann. Zwar lässt sich einwenden, dass es möglich ist, sich ein Laptop zu kaufen. Die hohen Anschaffungskosten stehen aber meistens in keinem Verhältnis zum Nutzen. Auch die Industrie hat dies erkannt, und stellt nun viele kleinere Geräte her, die nur einige wenige Aufgaben erfüllen, aber dafür Preiswerter sind und in eine Jackentasche passen.

Mobile Computing ist das Schlagwort der heutigen Computerindustrie. Die Visionen reichen von intelligenter Bekleidung bis hin zum sprechenden Telefon. Zwei Geräte haben sich aber schon durchgesetzt. Das sind zum einen das Mobiltelefon und zum anderen der PDA. Einen weiteren Bereich der Mobilen Computer stellen tragbare CD-Spieler und MP-3 Player dar.

Diese Lösungen stillen zwar das Verlangen nach Mobilität, werfen aber auch gleichzeitig ein neues Problem auf. Sind die Anwender bereit, mehrere Geräte gleichzeitig mit sich herumzutragen? Die neusten Generationen der Mobiltelefone und PDAs beantwortet diese Frage mit einem NEIN. Das Mobiltelefon Siemens ME45 beinhaltet z.B. einen Organizer und ein Adressbuch. Somit wird für die meisten Anwender der PDA überflüssig. Selbst ein Mobiltelefon mit eingebautem MP3-Player existiert schon auf dem Markt. Auf der anderen Seite gibt es auch PDAs, die ein Telefon beinhalten. Das Mobiltelefon und der PDA wachsen also zusammen.

Ein weiteres Kriterium, welches zurzeit den Einsatz mehrere Geräte unpraktisch erscheinen lässt, ist das Entstehen redundanter Datenmengen. So haben der PC, der PDA und das Mobiltelefon jeweils ein eigenes Adressbuch. Hier bietet Bluetooth einen Ansatz, dieses Problem komfortabel zu lösen. Vorraussetzung ist aber, dass die Hersteller einen Standard definieren, wie z.B. ein Adressbucheintrag oder ein Termin gesendet wird. Da jeder Hersteller aber eine starke Markposition anstrebt, werden nur wenige Kooperationen zustande kommen.

Davon ausgehend, dass jeder Anwender ein Gerät besitzt, welches seinen mobilen Anforderungen entspricht, bleibt nach dem heutigen Stand der Technik noch ein Problem. Das Interface zwischen dem Anwender und dem Gerät. Ein in Taschengröße gebautes Gerät bietet nur sehr eingeschränkte Möglichkeiten Eingaben zu tätigen. Hier ist der PC mit seiner Tastatur das zurzeit effektivste Gerät auf dem Markt. Eine komfortable Eingabemöglichkeit und gleichzeitige Mobilität erfordert zurzeit

noch zwei Geräte. Dies hat zur Folge, dass wir zusätzlich eine Möglichkeit benötigen, die am PC eingegebenen Daten in das mobile Gerät zu übertragen und umgekehrt.

In der vorliegenden Arbeit beinhaltet das erste Kapitel die Identifizierung der Anwendungsfälle aus der Sicht des Benutzers. Im zweiten Kapitel analysieren wir die im vorherigen Kapitel erstellten Anwendungsfalldiagramme und leiten hieraus die Teilsysteme und deren Aufgaben ab. In den folgenden vier Kapiteln (3-6) setzen wir die Aufgabe jedes einzelnen Teilsystems in ein Programmdesign um. Im siebten Kapitel setzen wir das Programmdesign in einen Pseudocode um. Eine zusammenfassende Bewertung und einen Ausblick auf mögliche Erweiterungen in Kapitel 8 beschließt die Arbeit.

# 1. Anwendungsfälle

## 1.1. Manueller Datenabgleich

Aufgrund der fehlenden Synchronisationssoftware ist jeder Anwender dazu gehalten, den Datenabgleich manuell auszuführen. Schnell sind die Grenzen dieser Methode erreicht. Unternimmt der Anwender mehrere Änderungen in einem System, so muss er sich all diese merken, um zu gegebener Zeit auch im anderen System dieselben Änderungen vorzunehmen.

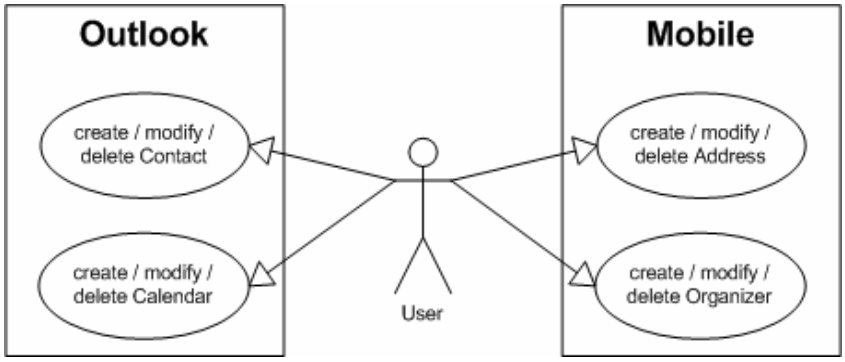



**Abbildung 1**

## 1.2. Automatischer Datenabgleich

Bei einer automatisierten Synchronisation bekommt der Anwender eine Software zur Seite gestellt, die die Aufgaben des Datenabgleichs übernehmen kann. Es ist nicht mehr notwendig, dass der Anwender das Mobiltelefon oder Outlook bedient. Er muss lediglich bekannt geben, welches Mobiltelefon an welcher Schnittstelle angeschlossen ist und danach die Synchronisation starten.

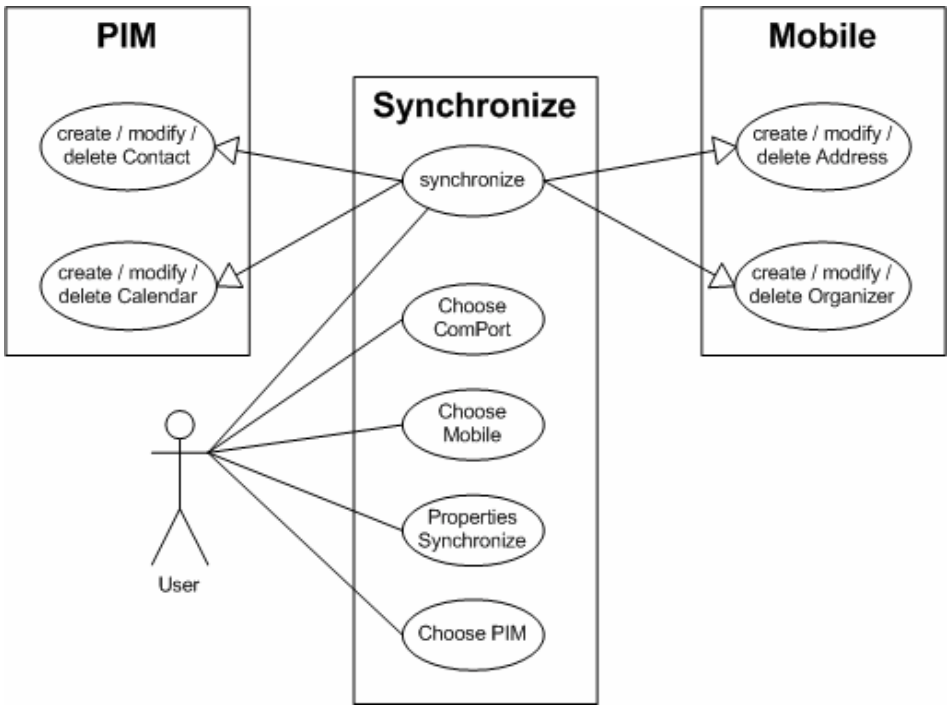



**Abbildung 2**

# 2. Teilsysteme

Für eine strukturierte Lösung ist es notwendig, die gesamte Problematik in mehrere kleinere Teilsysteme zu zerlegen. Das Anwendungsfalldiagramm (Abbildung 2) aus dem vorherigen Abschnitt zeigt die ersten Teilsysteme.

## 2.1. Automatisierung

Bevor wir jedoch die einzelnen Teilsysteme definieren, legen wir den Mechanismus fest, mit dessen Hilfe die einzelnen Teilsysteme miteinander kommunizieren sollen. Hierfür benutzen wir die Automatisierung von Microsoft.

Automatisierung bedeutet, dass ein Programm einen Teil seiner Funktionalität auf dem Weg über das Betriebssystem anderen Programmen zur Verfügung stellt. Dieses Modell ist Client-Server basiert. Das Buch „Visual C++ .NET“ von Dr. Susanne Wigard [1] beschreibt diesen Mechanismus detailliert.

## 2.2. PIM (Personal Information Manager)

Es gibt zahlreiche PIMs auf dem Softwaremarkt. Wir möchten uns auf den am weitesten verbreiteten konzentrierten. Hierbei handelt es sich um Microsoft Outlook. In Microsoft Outlook sind die Adressen und Termine gespeichert, die jeweils mit den Daten aus einem Mobiltelefon abgeglichen werden sollen. Die Eingabe und Verwaltung der Kontakte sowie der Termine in Outlook, ist kein Schwerpunkt dieser Studienarbeit. Hierfür verweise ich auf das Buch „Microsoft Outlook 2002 von Rita Wendel [2].

Für eine automatisierte Synchronisation ist es notwendig auf die Persistenzschicht von Microsoft Outlook zuzugreifen. Dies ist direkt über die Mechanismen der Automatisierung möglich.

## 2.3. Mobiltelefon

Auch das Mobiltelefon gehört zum Gesamtsystem der Datensynchronisation. Je nach Typ des Telefons speichert es Telefonbucheinträge, Adressbucheinträge und Termine. Die Handhabung des jeweiligen Mobiltelefons entnehmen sie bitte der dazugehörigen Betriebsanleitung des Herstellers.

Auch das Mobiltelefon bietet eine Möglichkeit auf dessen interne Struktur zuzugreifen. Die Persistenzschicht wird mit Hilfe des GSM-Standards realisiert. Hierbei handelt es sich um die in der Spezifikation ETSI GSM 7.07 und 7.05 definierten „AT Commands“.

Zusätzlich möchten die Hersteller ihre Kunden durch immer aufwendigere Funktionen an das hauseigene Produkt binden. Somit ist der GSM-Standard nicht mehr ausreichend und jeder Hersteller definiert zusätzlich eine eigene firmenspezifische Erweiterung des AT-Befehlssatzes. Die Erweiterungen werden von den Herstellern veröffentlicht.

## 2.4. Synchronisation

Das dritte erkennbare Teilsystem ist die Software, die den Datenabgleich zwischen dem Mobiltelefon und MS Outlook (PIM) durchführt. Die Aufgabe besteht darin, herauszufinden, welche Kontakte und Termine in Outlook bzw. im Mobiltelefon geändert, neu angelegt oder gelöscht worden sind. Anschließend muss der Abgleich der Daten durchgeführt werden.

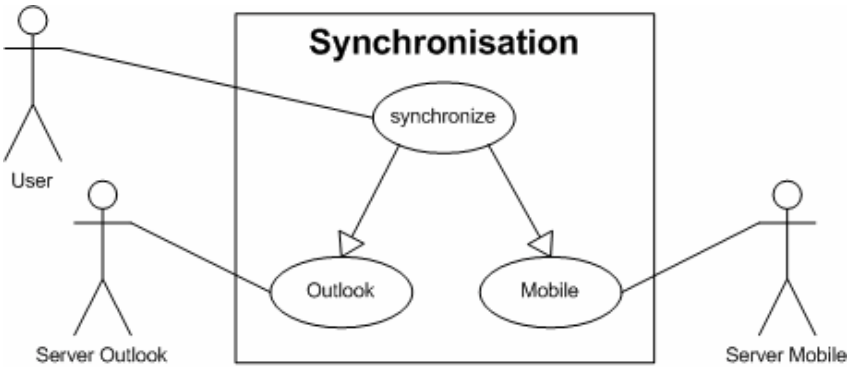



**Abbildung 3**

Für den Datenabgleich ist es notwendig, dass dieses Teilsystem auf die gespeicherten Daten in MS Outlook bzw. dem Mobiltelefon zugreift und in eine gemeinsame Sprache konvertiert. Dies ermöglicht erst das Synchronisieren der Daten.

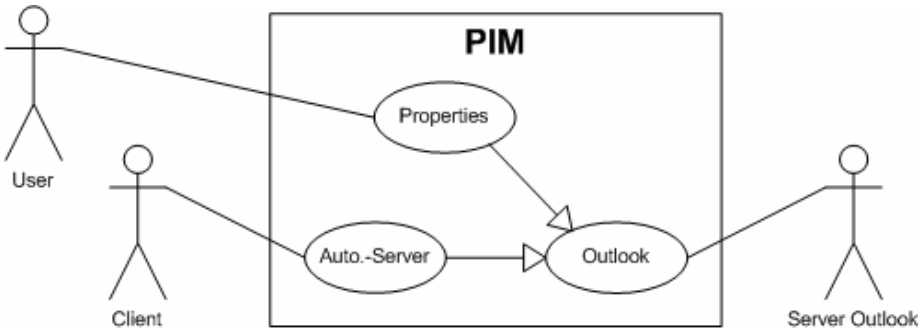
Bei MS Outlook können wir mit Hilfe der Automatisierung auf die interne Struktur zugreifen. Zur Umsetzung benötigen wir eine Übersicht, des in Outlook integrierten Automatisierungsservers. Hier empfehlen wir das Buch Microsoft Outlook / Visual Basic Schritt für Schritt [3].

Der Zugriff auf die Daten des Mobiltelefons ist nicht so komfortabel wie bei Microsoft Outlook. Kein Hersteller bietet einen Automatisierungsserver für seine Geräte an. Hinzu kommt noch, dass jedes Telefon seine eigene Sprache mitbringt und an verschiedenen externen Schnittstellen mit dem PC verbunden werden kann.

Um eine gut strukturierte Software zu erlangen, ist es notwendig, die Synchronisation in mehrere Teilsysteme zu zerlegen.

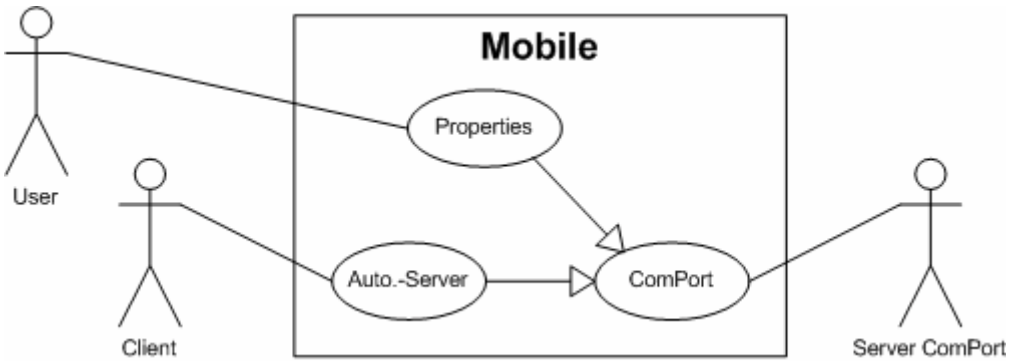
### 2.4.1. PIM - Outlook

Das erste Teilsystem steuert das Objektmodell von Microsoft Outlook. Es ist in der Lage Kontakte und Termine auszulesen und in die gemeinsame Synchronisationssprache umzurechnen. Ebenso kann es umgekehrt Informationen aus der Synchronisation in Microsoft Outlook speichern.
Da Microsoft Outlook ermöglicht mehrere Adressbücher anzulegen, kann der Anwender hier wählen, welches Telefonbuch synchronisiert werden soll.



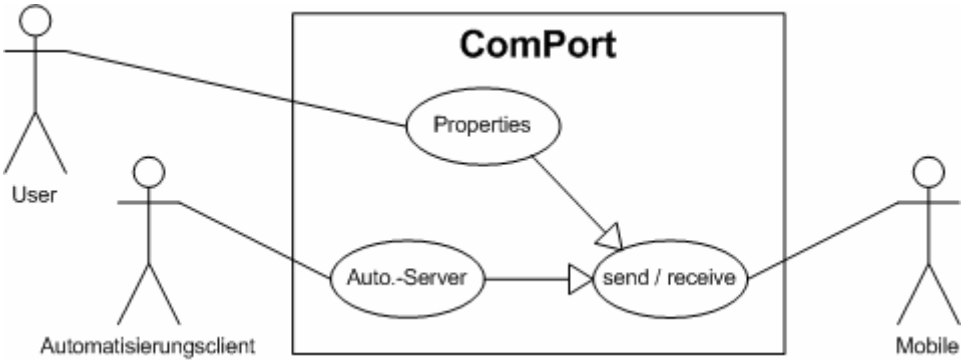

### 2.4.2. Mobile

Je nachdem welches Mobiltelefon angeschlossen ist, müssen dem Mobiltelefon entsprechende Routinen angewendet werden. Hierzu gehört ein Interpreter, der die Informationen in eine allgemeingültige Sprache übersetzt. Auch die Befehlssätze unterscheiden sich oft nach Hersteller und Modellserien. Alle vom Mobiltelefon abhängigen Informationen und Mechanismen werden in diesem Teilsystem implementiert. Es ist die Aufgabe des Anwenders dem System über die „Einstellungen“ den Mobiltelefontyp mitzuteilen.
Der Mechanismus der Automatisierung ermöglicht die Kommunikation mit der Außenwelt.



### 2.4.3. ComPort

Im laufe der letzten Jahren haben sich die serielle-, USB- und Infrarotschnittstelle am PC durchgesetzt. Seit kurzem gibt es auch noch die Bluetooth Technologie. Um unsere zu erstellende Software in Bezug auf die Schnittstellen so flexible wie möglich zu halten, erstellen wir ein eigenes Teilsystem für die bidirektionale Kommunikation über die oben erwähnten Schnittstellen. Ein Automatisierungsserver ist auch hier die Schnittstelle für andere Teilsysteme. Der Anwender muss hier über den Anwendungsfall „Properties“ einstellen, an welcher Schnittstelle das Mobiltelefon angeschlossen ist.



# 3. Synchronisation

## 3.1. Aufgabenbeschreibung

Dieses Teilsystem ist das Herzstück unserer Synchronisationssoftware. Hier sollen die Entscheidungen getroffen werden, ob ein Datensatz verändert wurde und ob er in das jeweils andere Teilsystem übertragen werden muss.

## 3.2. Problemidentifikation

### 3.2.1. Zwischencode zu Backend

Werden Daten vom Mobiltelefon oder von Outlook entgegengenommen so benötigen wir ein Backend um diese Daten (Zwischencode) weiterverarbeiten zu können. Der Zwischencode wird von den Teilsystem Mobile und MS Outlook generiert.

### 3.2.2. Primärschlüssel

Um einen Datensatz eindeutig zu identifizieren wird ein Primärschlüssel verwendet. In der Regel ist dies eine künstlich generierte Zahl. Diese grundlegende Funktionalität wird weder von MS Outlook noch von den Mobiltelefonen unterstützt. Doch wie können wir feststellen, welche Tupel zusammengehören?

### 3.2.3. Datum-Zeit-Stempel

Der Datum-Zeit-Stempel basiert auf der Systemzeit und ist eine wichtige Information bei unserer Synchronisation. Er speichert den Zeitpunkt der letzten Änderung eines Tupels. Wir müssen aber davon ausgehen, dass die Systemzeit des Computers nicht mit der Uhrzeit des Mobiltelefons übereinstimmt. Gibt es Grenzen, die einen Datum-Zeit-Stempel ungültig werden lassen?
Ist die Datum-Zeit-Differenz zweier geänderter Datensätze kleiner als die Differenz der beiden Systemuhren, so ist nicht mehr festzustellen welcher der beiden Tupel der aktuellere ist. Um dies zu vermeiden, ist es notwendig regelmäßig eine Zeitsynchronisation durchzuführen.

### 3.2.4. Zeitsynchronisation

Es ist wichtig, dass alle an der Synchronisation beteiligten Geräte dieselbe Systemzeit als Grundlage für den Datum-Zeit-Stempel haben. Das Buch Administering NDS von Nancy Cadjan und Jeffrey L. Harris [4] behandelt dieses Thema ausführlich.
Wir wählen die Methode „forced time“ die die Strategie verfolgt, dass sich alle Geräte nach einem Zeitserver richten. Diese Aufgabe übernimmt bei uns der Computer, da er in der Lage ist über das Internet mit Hilfe eines NTP Servers eine sehr genaue Uhrzeit zu bekommen. Mobiltelefone hingegen unterstützen solche Funktionalitäten nicht.
Die Zeitsynchronisation kann nur durchgeführt werden, während das Mobiltelefon und der Computer verbunden sind. Ansonsten sorgen die RTCs (Real Time Clock) für einen konstant weiterlaufenden Zahlenwert, der als Uhrzeit interpretiert werden kann.

### 3.2.5. Modifiziert

Beide Datenbanken speichern den Zeitpunkt der letzten Änderung eines Tupels. Es ist nicht möglich festzustellen, welche Änderung im Einzelnen an einem Tupel durchgeführt wurde. Erschwerend kommt hinzu, dass weder Outlook noch die Mobiltelefone den Zeitpunkt der letzten Synchronisation speichern. Wir sind also nur in der Lage festzustellen, welcher Datensatz als letztes geändert worden ist und müssen diesen somit als den aktuellen Tupel übernehmen.

### 3.2.6. Datenbanken - View

Jeder Hersteller eines Mobiltelefons legt andere Schwerpunkte in seinen Adressbüchern und spart somit teuren Speicherplatz. Microsoft Outlook hingegen hat zum Ziel, dass alle erdenklichen Informationen in Bezug auf eine Person im Adressbuch gespeichert werden. Somit ist das Adressbuch eines Mobiltelefons ein View (Teilmenge) des Adressbuches aus Outlook.

### 3.2.7. Stringvergleich mit unterschiedlichen Zeichensätze

Ein Mobiltelefon unterstützt den Zeichensatz aus dem GSM-Protokoll. Microsoft Outlook dagegen den erweiterten Latin Zeichensatz. Beide Zeichensätze besitzen eine Schnittmenge. Es kann also passieren, dass innerhalb eines Strings zwei Zeichen zwar ungleich sind, aber als gleich gewertet werden müssen.

### 3.2.8. Maximale Stringlänge

Aufgrund der kleinen Bauweise hat ein Mobiltelefon nur sehr begrenzten Speicherplatz. Hieraus resultiert, dass z.B. der Name eines Adressbucheintrages eine maximale Zeichenlänge nicht überschreiten darf. Microsoft Outlook hat diese Einschränkung nicht.
Vergleichen wir zwei Strings, kann diese Beschränkung dazu führen, dass diese Strings als ungleich angesehen werden, obwohl sie eigentlich zueinander gehören.

### 3.2.9. Frontend zu Zwischencode

Sollen die Daten zum Mobiltelefon oder zu Microsoft Outlook übertragen werden, so benötigen wir ein jeweils ein Frontend, das die Daten in einen Zwischencode übersetzt.

### 3.2.10. Fehlererkennung

Jede der einzelnen Funktionen dieses Teilsystem muss sich selbst überwachen. Sollte ein Fehler auftreten, so sind geeignete Fehlerroutinen zu starten.

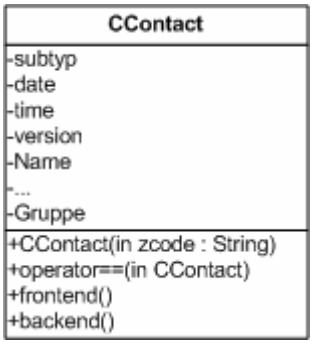
## 3.3. Klassenmodelle

### 3.3.1. Synchronisation


Startet ein Anwender die Synchronisation, so wird als erstes ein Objekt der Klasse CSync generiert. Anschließend wird die Kommunikation zum Mobiltelefon und zu Microsoft Outlook hergestellt. Ist dies erfolgreich geschehen, so kann nun ein Datensatz nach dem anderen mit den Datensätzen aus dem jeweiligen anderen System synchronisiert werden.

Es gibt viele verschiedene Möglichkeiten, zwei Datenbanken miteinander abzugleichen. Da unsere Datenmengen nicht indiziert sind und auch keinen eindeutigen Primary Key besitzen, bleibt uns nur die Methode jeden Tuple einer Datenbank mit allen anderen der anderen Seite zu vergleichen. Wird auf der gegenüberliegenden Seite kein passendes Element gefunden, so kann es bedeuten, dass ein neues Eelement angelegt worden ist, oder ein altes gelöscht wurde. Werder Outlook noch die Mobiltelefone bieten einen Mechanismus um diese Situaton eindeutig zu lösen. Die Firma Palm hat eine Lösung für dieses Problem entwickelt. Sie markiert zu löschende Daten. Beim nächsten Update kann man nun feststellen, ob der Tuple neu oder alt ist.

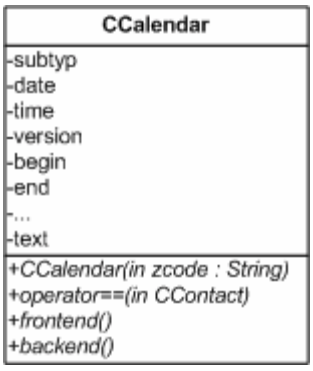
### 3.3.2. Kontakte




Die Klasse CContact speichert alle Informationen eines Adressbucheintrages. Gleichzeitig ist jedes Objekt dieser Klasse in der Lage einen Zwischencode zu scannen oder die vorhandenen Informationen in den Zwischencode umzurechnen. Dies wird durch das integrierte Front- bzw. Backend ermöglicht.

Mit Hilfe der Operatorüberladung führen wir die Synchronisation zweier Kontakte durch.

### 3.3.3. Kalender




Die Klasse CCalendar gleicht von der Struktur der Klasse CContact. Der Einzige Unterschied besteht darin, dass nicht Adressinformationen, sondern Termine synchronisiert werden.

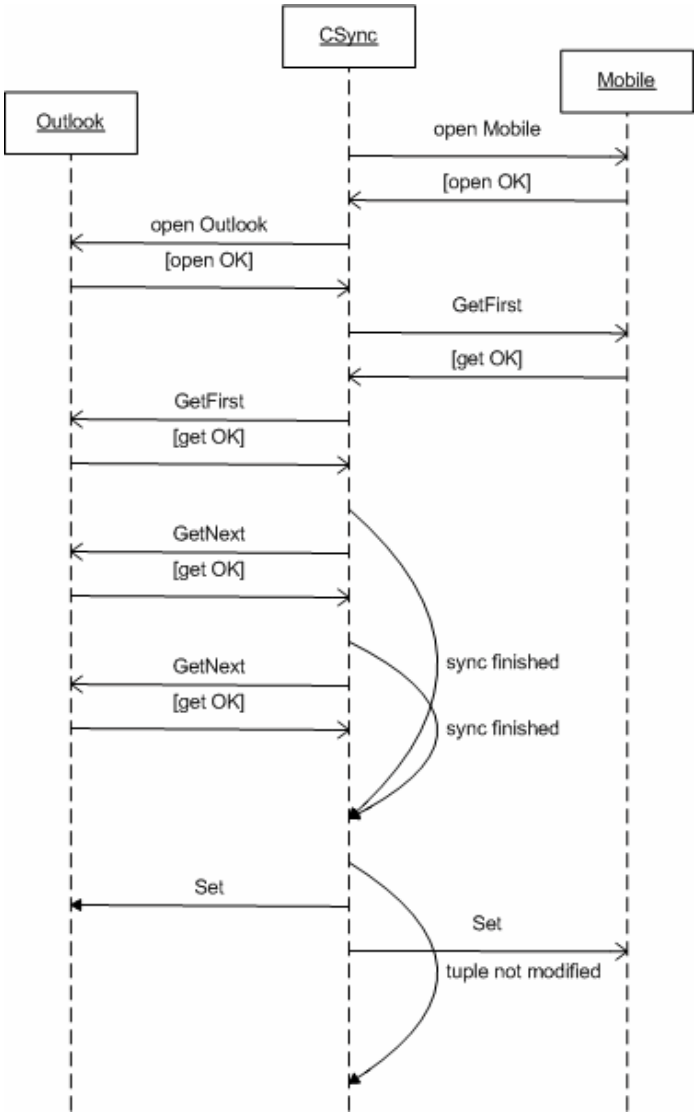
## 3.4. Reihenfolge-Diagramme

Aufgrund der sehr eingegrenzten Möglichkeiten, auf die Tuple zuzugreifen verfolgen wir folgende Strategie bei der Synchronisation. Zuerst wird eine Verbindung zum Mobiltelefon und auch zu Outlook geöffnet. Ist dies erfolgreich geschehen, so wird jedes Element der ersten Liste mit jedem Element der zweite Liste. Hieraus resultieren dann drei Fälle:

**a.** Ein passender Datensatz wurde in der anderen Liste gefunden und konnte synchronisiert werden. Dies hat zur Folge, dass die synchronisierten Tuple in die Teilsysteme geschrieben werden.

**b.** Ein passender Datensatz wurde gefunden, aber es konnte nicht synchronisiert werden. Hier gibt es eine Fehlermeldung und die Daten werden so nicht verändert.

**c.** Es gibt keinen passenden Datensatz in der gegenüberliegenden Liste. Dies kann zustande kommen, weil ein Datensatz gelöscht wurde, oder weil eun Datensatz neu angelegt wurde. Die Entscheidung ob gelöscht oder neu angelegt wird trägt der Anwender mit Hilfe eines Dialogfeldes.



# 4. PIM

## 4.1. Aufgabenbereich

PIM ist die Abkürzung für Personal Information Manager. Wir entwickeln diese Software für Microsoft Outlook. Dieses Teilsystem ermöglicht den Zugriff auf die in Outlook gespeicherten Informationen und konvertiert diese in den Zwischencode. Außerdem Hilft es uns, auf Änderungen an der Objektstruktur von Outlook, flexibel reagieren zu können. Des Weiteren halten wir uns die Option offen ohne größere Umstellungen die Software auch für andere PIMs zu erweitern.

## 4.2. Problemidentifikation

### 4.2.1. Wer suchet der findet

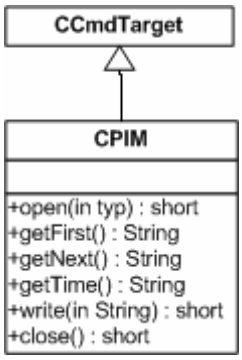
Es gibt in Outlook nicht die Möglichkeit, einen Kontakt direkt anzusprechen. Es ist z.B. nicht möglich den fünften Kontakt direkt anzusprechen. Möchten wir dieses erreichen, so müssen wir zum ersten Kontakt gehen und dann uns viermal den nächsten Kontakt geben lassen.

### 4.2.2. Welches Adressbuch

Microsoft Outlook bietet dem Anwender die Möglichkeit mehrere Adressbücher zu erstellen. Es ist auch möglich, einen Kontakteintrag in mehrere dieser Adressbücher zu speichern. Hierdurch könnten redundante auch im Mobiltelefon entstehen.

## 4.3. Klassenmodell

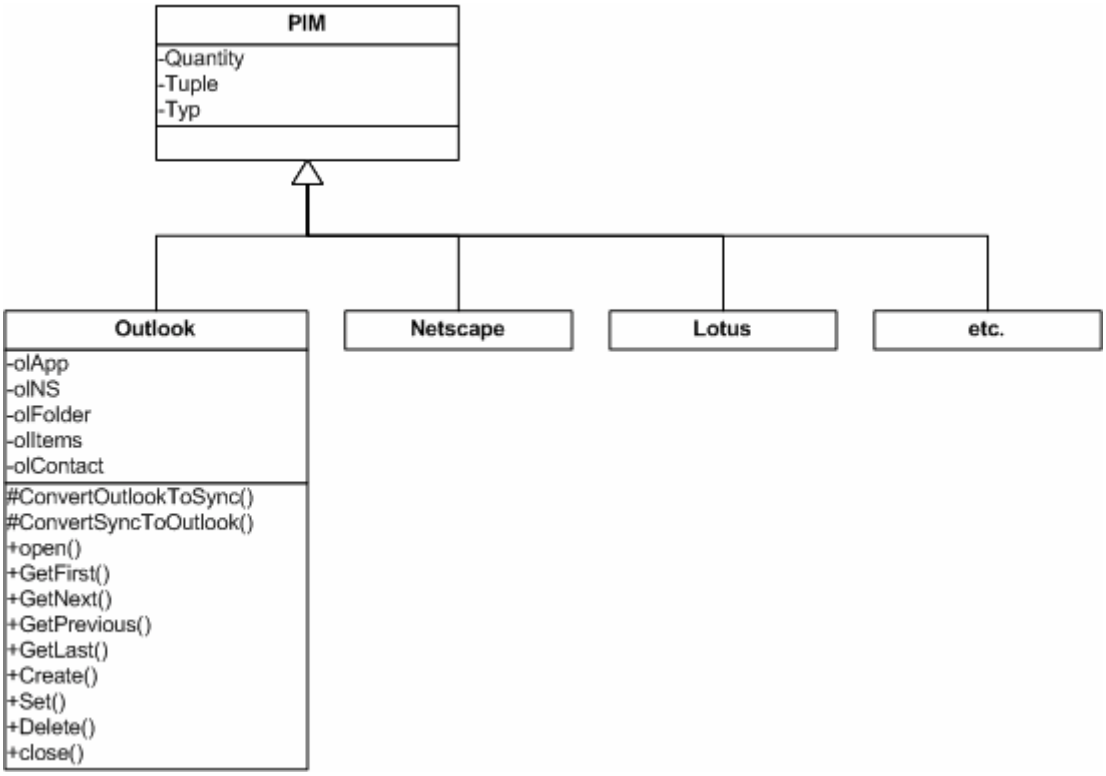
### 4.3.1. Automatisierungsserver




Auch das PIM Teilsystem stellt für die Synchronisation einen Automatisierungsserver zur Verfügung. Dieser bietet die Möglichkeit über „open“ das Adressbuch oder den Kalender von Microsoft Outlook zu öffnen. Mit den Funktionen „getFirst“, „getNext“ und „write“ können wir Einträge wählen und nach dem synchronisieren wieder speichern.

### 4.3.2. PIM

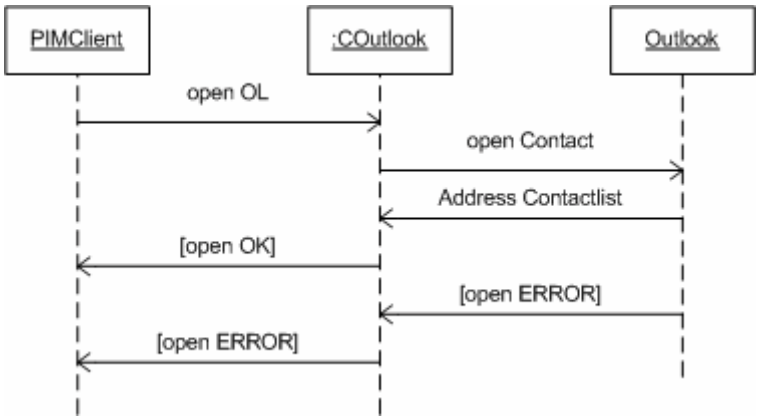
Ein Kriterium moderner Software ist die Flexibilität. Die erste Version dieser Software funktioniert ausschließlich mit Microsoft Outlook. Das Design der Klassen ist aber so angelegt, dass wir ohne größeren Programmieraufwand auch auf andere PIMs zugreifen können.



## 4.4. Reihenfolge-Diagramme

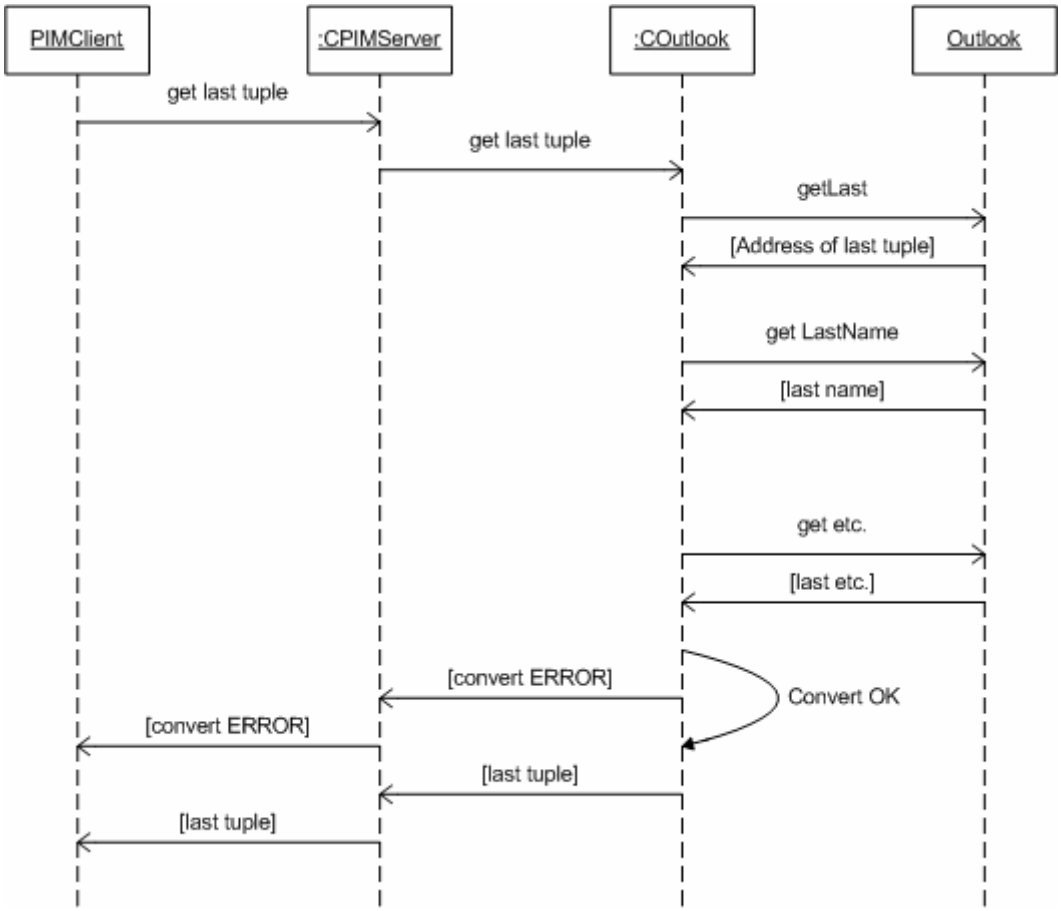
Durch die Vorgabe des Automatisierungsserver für Outlook durch Microsoft sind wir gezwungen deren Datenbankverwaltung zu übernehmen. Diese gibt einen nicht die Möglichkeit direkt auf den fünften Datensatz zuzugreifen. Es ist nur möglich, sich den ersten Datensatz geben zu lassen und sich anschließend vier Datensätze weiterzuhangeln. Dies wird durch die Funktionen „GetFirst“, „GetNext“, „GetPrevious“ und „GetLast“ realisiert. Möchten wir einen Tuple ändern, so werden über die unzähligen put-Funktionen immer die Daten des aktuellen Tuple verändert. Des Weiteren gibt es eine Möglichkeit einen neuen Tuple anzulegen oder einen alten zu löschen.
Die Haupt

### 4.4.1. Outlook öffnen



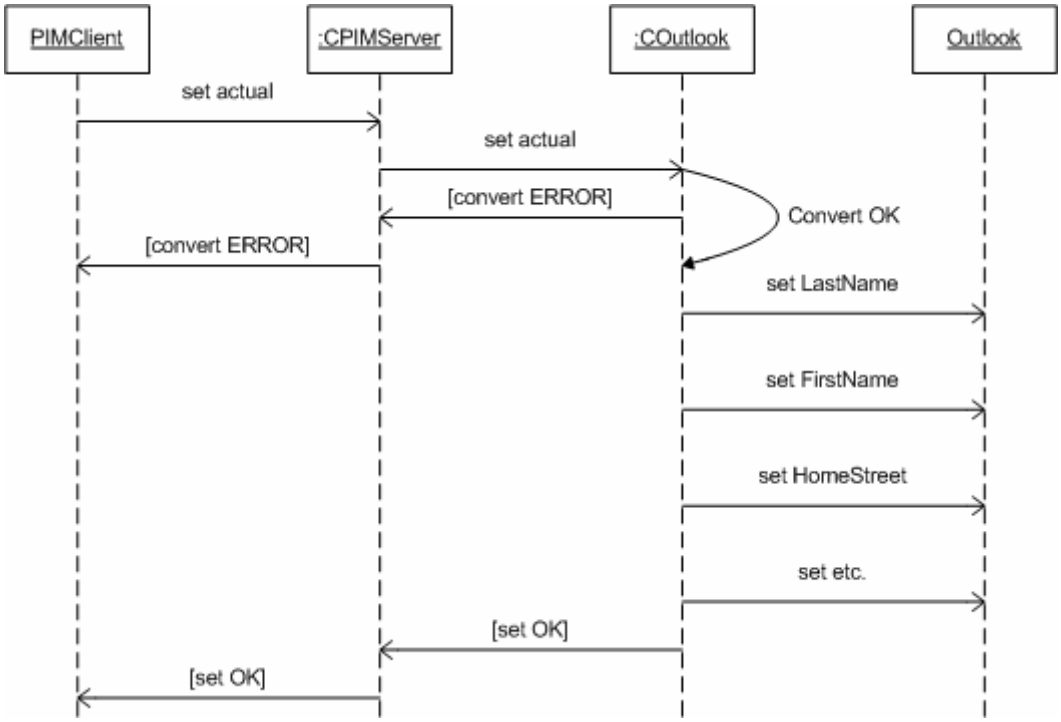
### 4.4.2. Lesezugriff

Outlook stellt über den Automatisierungsserver vier Funktionen zu Verfügung, mit dem der Zugriff auf die interne Datenbank realisiert wird. Hierbei handelt es sich um GetFirst, GetNext, GetPrevious und GetLast. Alle vier Funktionen beziehen sich auf einen Tuple. Diese Funktionen haben als Rückgabewert die Adresse des gewünschten Tupleobjektes. Mit dessen Hilfe können wir die Daten einzeln auslesen und in den Zwischencode konvertieren. Es ist nicht möglich, direkt auf den Datensatz „Meier, Hans“ zuzugreifen. Stellvertretend für alle vier Funktionen haben wir ein Reihenfolge-Diagramm dargestellt.



### 4.4.3. Schreibzugriff

Auch hier ist es nicht möglich dem Automatisierungsserver von Outlook einen kompletten Tuple zu übergeben.



# 5. Mobile

## 5.1. Aufgabenbeschreibung

Dieses Teilsystems besteht zum einen aus zwei Interpretern. Diese konvertieren die zu übertragenen Informationen so, dass das jeweils gegenüberliegende Teilsystem diese verarbeiten kann. Um das Teilsystem Synchronisation von der Sprache der Mobiltelefone zu entkoppeln, benutzen wir die Methode der Zwischencodeerzeugung.

Zum Zweiten werden hier alle Tuple aus dem Mobiltelefon zwischengespeichert. Dies hat den entscheidenden Vorteil, dass wir die Tuple schneller durchsuchen können und wenn nötig auch sortieren könnten. Würden wir dagegen die Daten im Mobiltelefon belassen, so müssten wir einen Zugriff auf einen Tuple mit ca. einer Sekunde veranschlagen.

Die letzte Aufgabe besteht in darin, der Außenwelt eine einfache aber Effektive Schnittstelle zur Verfügung zu stellen. Hierbei handelt es sich wieder um einen Automatisierungsserver.

## 5.2. Problemidentifikation

### 5.2.1. Befehlssatz

Jeder Hersteller hat die standardisierten AT-Commands firmenspezifisch erweitert. Es ist also notwendig neben den Standardbefehlen spezielle Befehle zu integrieren.

### 5.2.2. Bezeichnung

Jedes Mobiltelefon hat seine eigene Semantik. Es ist nicht zwingend erforderlich, dass z.B. der Vorname immer mit VN bezeichnet wird.

### 5.2.3. Kodierung

Es ist auch möglich, dass ein Hersteller seine Daten in einem speziellen Format überträgt. Das Mobiltelefon Siemens ME45 verlangt z.B., dass die zu übertragenen Daten in hexadezimale Werte konvertiert werden.

### 5.2.4. Frontend - Zwischencode - Backend

Werden Informationen vom Mobiltelefon empfangen und zum Teilsystem Synchronisation weitergerecht, so verhält es sich dieses Teilsystem als Frontend.

Sollen hingegen Daten der Synchronisation zum Mobiltelefon übertragen werden, so empfangen wir die Daten in der Sprache des Zwischencodes und rechnen diese dann in den Zielcode um.

### 5.2.5. Pakete

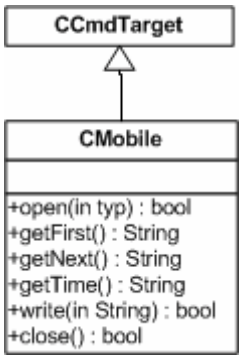
Oft müssen große Datenpakete bei der Übertragung in mehrere kleinere Pakete zerteilt werden. Beim Senden werden diese hier zerteilt und beim Empfangen wieder zusammengesetzt.

### 5.2.6. Identifizierung

Damit das Ergebnis dieses Teilsystems auch verwertbar ist, muss sichergestellt werden, dass auch wirklich der richtige Mobiltelefontyp angeschlossen ist.

## 5.3. Klassenmodelle

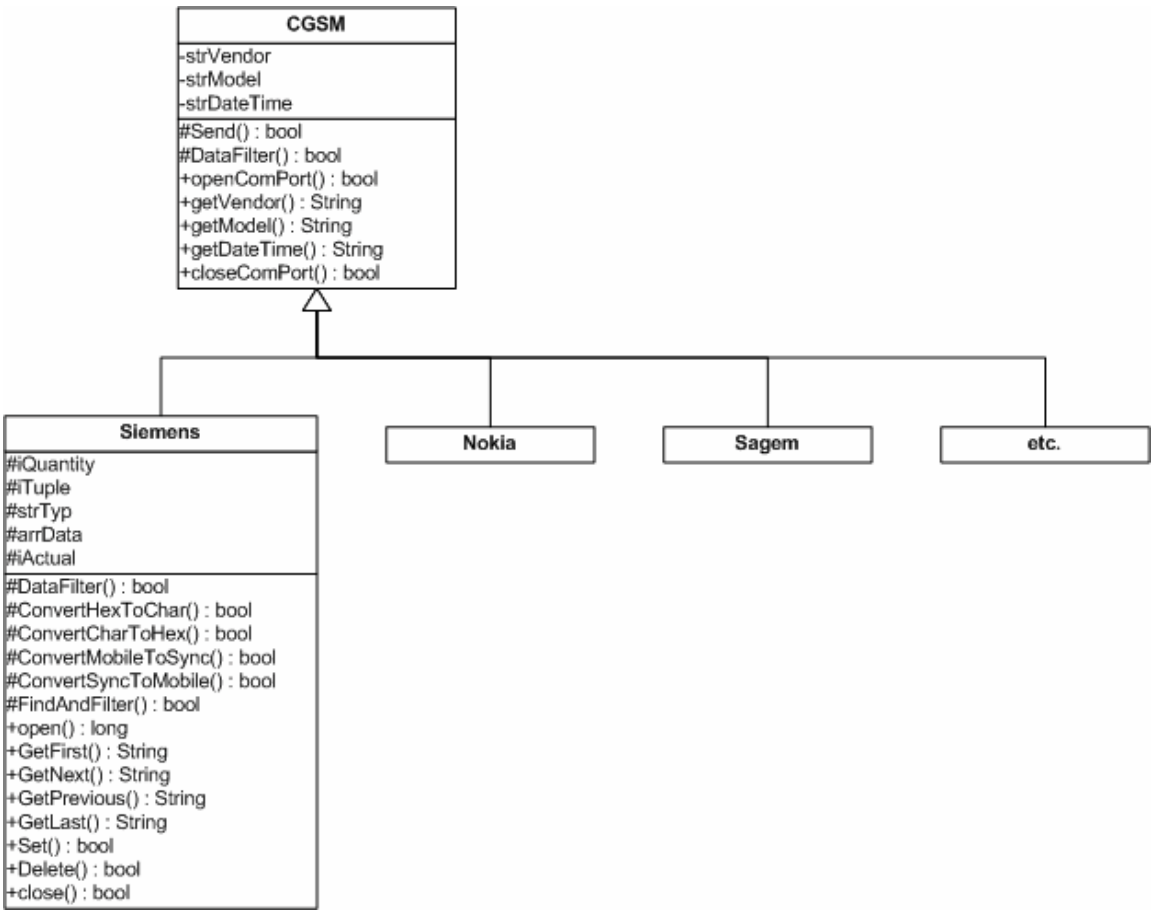
### 5.3.1. Automatisierungsserver




Der Automatisierungsserver „CMobile“ ist die Schnittstelle zum übergeordneten Teilsystem Synchronisation. Er bietet alle notwendigen Funktionalitäten für den Datentransfer mit dem speziellen Mobiltelefon. Nachdem ein Objekt dieser Klasse konstruiert wurde, wird ein Objekt der Klasse GSM erstellt.

### 5.3.2. Mobiltelefone

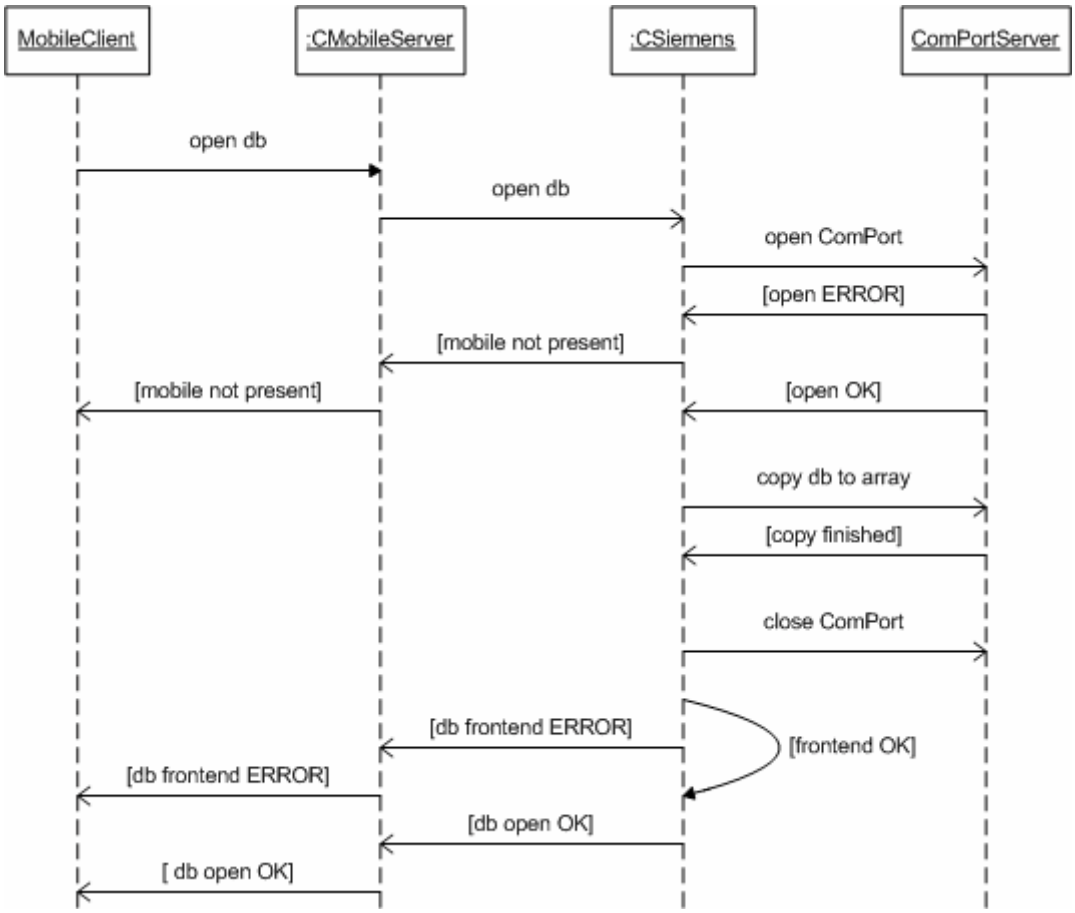
Viele Mobiltelefone der preiswerteren Modelle unterstützen nur den GSM-Standard. Somit ist eine Basisklasse sinnvoll, die den AT-Befehlssatz implementiert hat. Bei spezielleren Typen müssen wir durch die Technik der Ableitung die Funktionalität der Basisklasse GSM erweitern. Auch hier beschränken wir uns in der ersten Version auf ein spezielles Mobiltelefon.



## 5.4. Reihenfolge-Diagramme (Sequence)

### 5.4.1. Verbindung zum Mobiltelefon herstellen

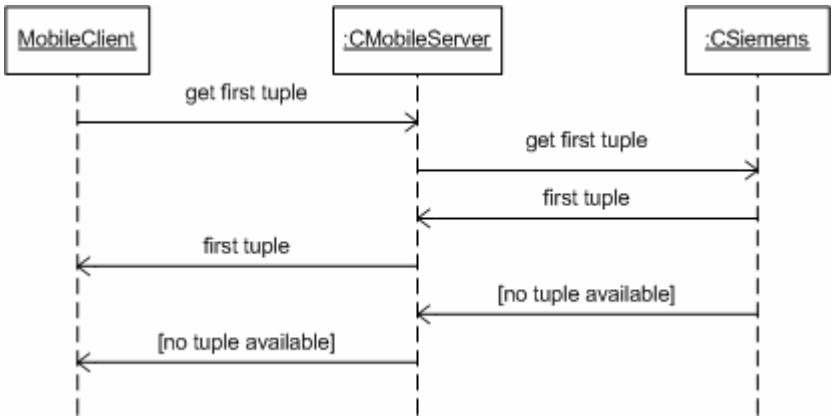
Der erste Schritt, um die Daten aus dem Mobiltelefon auszulesen, besteht darin, eine Verbindung zum Mobiltelefon aufzubauen. Gelingt dieses, so werden alle relevanten Daten im Teilsystem Mobile zwischengespeichert.



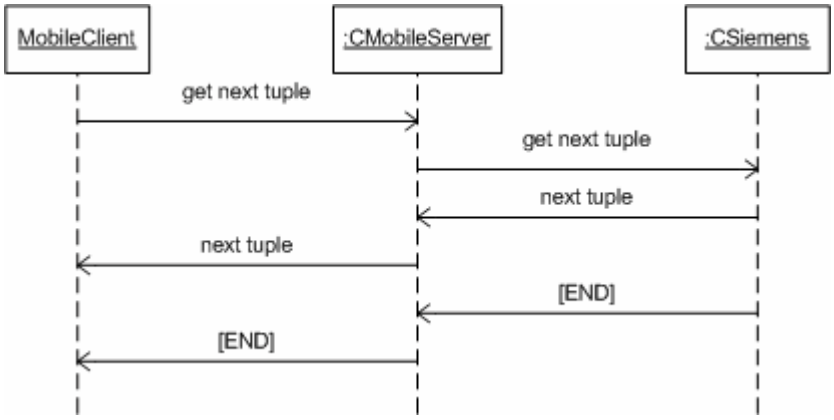
### 5.4.2. Zugriff auf die Datensätze

Insgesamt haben wir vier Möglichkeiten, auf die einzelnen Datensätze zuzugreifen. Hierbei handelt es sich um die Funktionen „GetFirst“, „GetNext“, „GetPrevious“ und „GetLast“. Mit dessen Hilfe können wir uns durch das Datenarray hangeln. Alle Funktionen beziehen sich immer auf den aktuellen Datensatz.

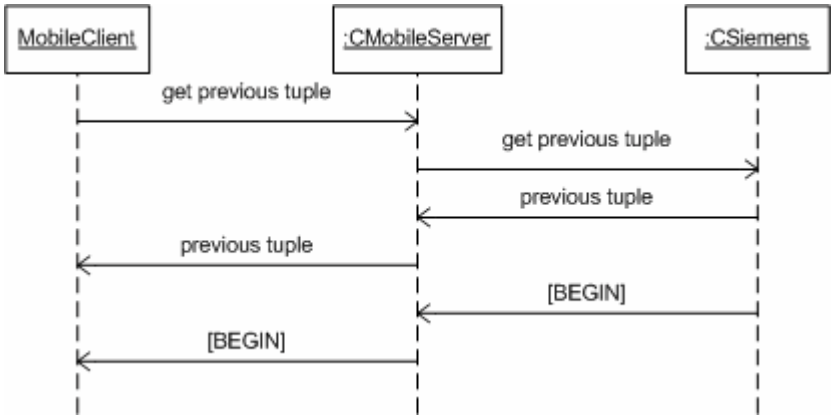
**a.** GetFirst



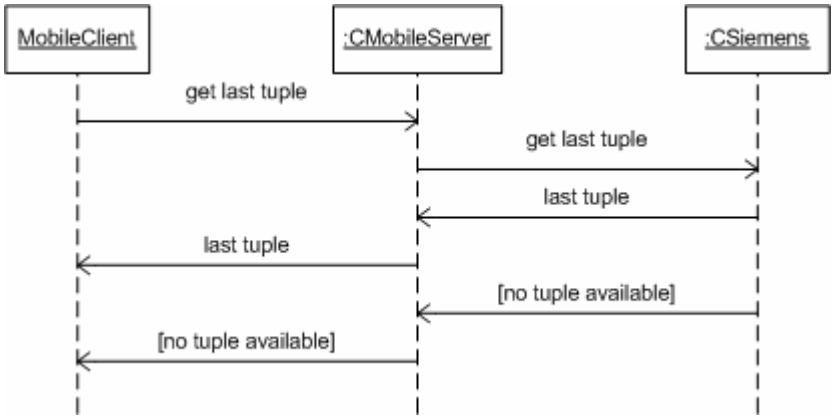

**b.** GetNext



**c.** GetPrevious

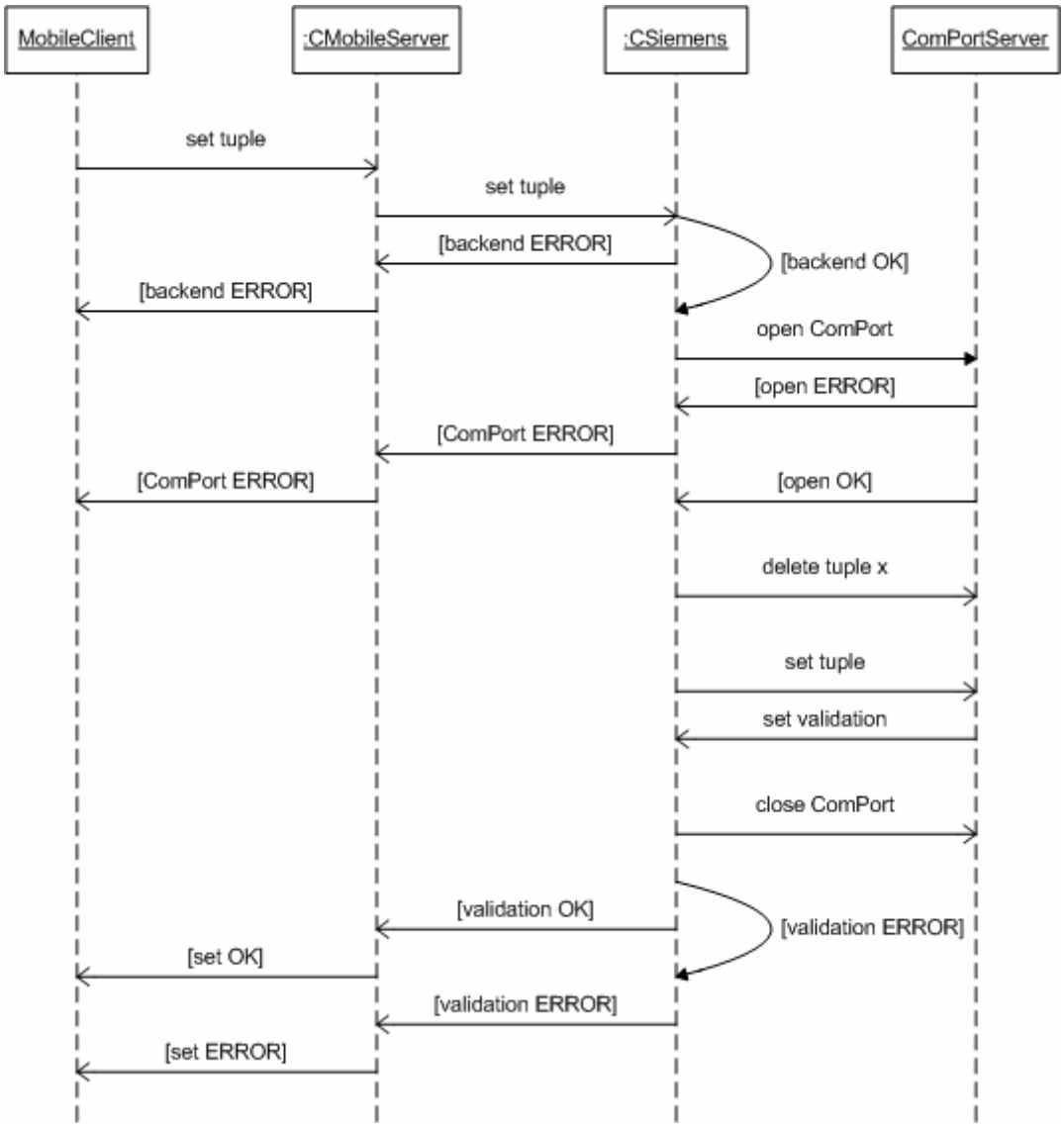



**d.** GetLast



### 5.4.3. Schreiben

Während der Synchronisation ist es erforderlich, Daten in das Mobiltelefon zu schreiben. Hierfür gibt die „Set“ Anweisung. Eine Besonderheit besteht darin, dass immer ein neuer Datensatz im Mobiltelefon an dem ersten freien Platz angelegt wird. Es ist nicht möglich einen bestehenden Datensatz zu ändern. Somit ist es erforderlich, den bestehenden Datensatz im Mobiltelefon vorher zu löschen.



# 6. ComPort

## 6.1. Aufgabenbeschreibung

Im laufe der letzten Jahre sind diverse Schnittstellen am Computer standardisiert wurden. Im Prinzip ist es möglich, an jede einzelne auch ein Mobiltelefon zur Synchronisation anzuschließen. Dies erfordert, dass auch unsere Software mit allen Schnittstellen kommunizieren kann.

## 6.2. Problemidentifikation

### 6.2.1. Verwaltung

Bei Microsoft Windows Betriebssystemen ist es nicht möglich, direkt auf eine Hardwarekomponente zuzugreifen. Die Kommunikation mit der gewünschten Schnittstelle geschieht über das Betriebssystem mit Hilfe der API.

### 6.2.2. Antrag

So wie wir eine Schnittstelle beim Betriebssystem beantragen, ist dieses für andere Anwendungen gesperrt. Es ist selbstverständlich, dass wir eine Schnittstelle nur dann belegen, wenn wir sie wirklich benötigen und sie auch so früh wie möglich wieder freigeben.

### 6.2.3. Initialisierung

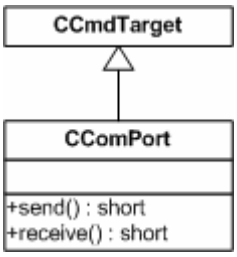
Bevor wir über eine Schnittstelle kommunizieren können, muss diese auch initialisiert werden. Hierzu gehört z.B. das Timing, die Transferrate, etc.

### 6.2.4. Fehlererkennung

Die Fehlererkennung funktioniert bei diesem Teilsystem über das Betriebssystem. Nach jeder Übertragung wird ein Status zurückgegeben.

## 6.3. Klassenmodell

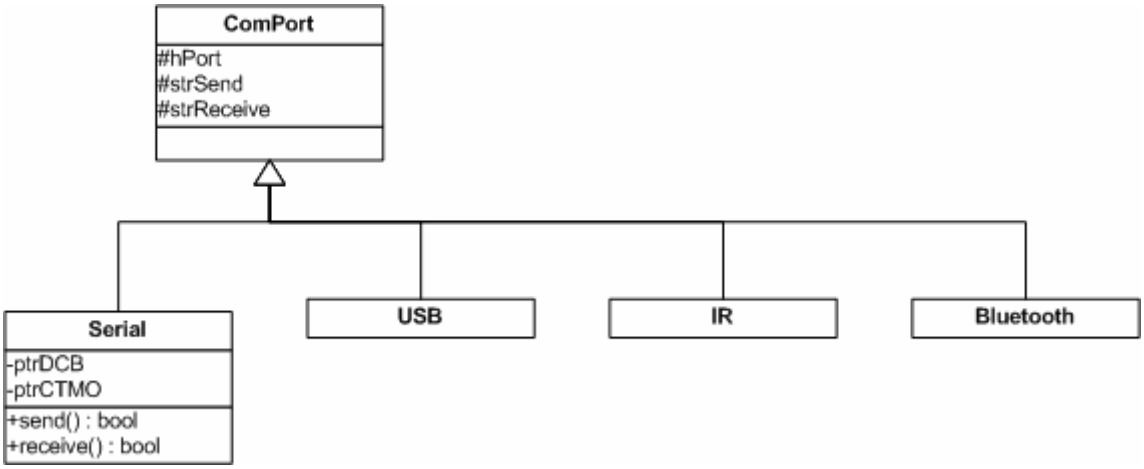
### 6.3.1. Automatisierungsserver




Dieser Server stellt, egal ob seriell, USB, BT oder IR, eine einheitliche Schnittstelle zum Mobiltelefon bereit. Diese besteht aus einer Funktion zum Senden und einer zum Empfangen von Daten.
Wird ein Objekt dieses Servers konstruiert, so wird anhand der vom Benutzer vorgenommenen Einstellungen eine externe Schnittstelle beim Betriebssystem angefordert.

### 6.3.2. Schnittstellen

Egal welche Schnittstelle zur Übertragung genutzt wird. Es ist immer eine Zeichenkette die übertragen wird. Somit können wir diese Informationen in der Basisklasse unterbringen. Auch der vom Betriebssystem veröffentlichte „handle“ ist immer notwendig. Spezielle Eigenschaften werden dann in den abgeleiteten Klassen behandelt.



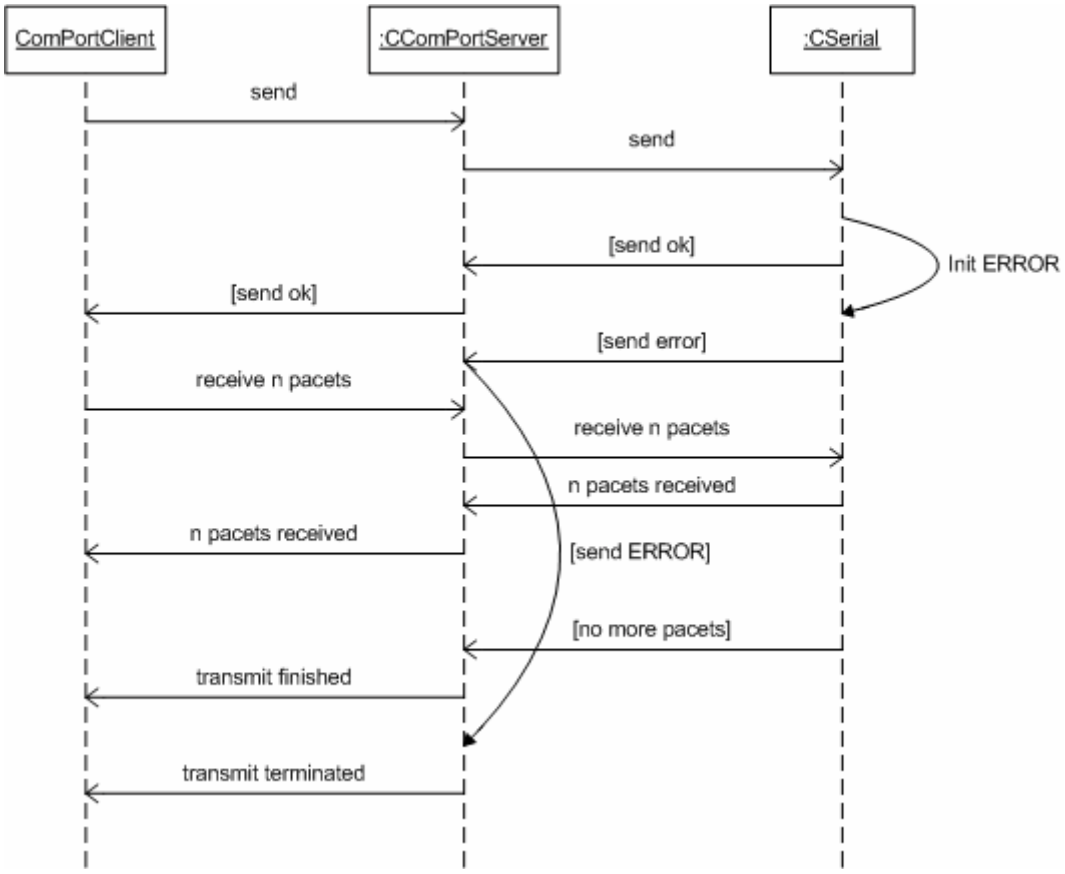

In der ersten Version dieses Teilsystem implementieren wir nur die serielle Schnittstelle. In einer späteren Version werden dann alle weiteren Schnittstellen realisiert.

## 6.4. Reihenfolge-Diagramme

### 6.4.1. Sende- und Empfangszyklus

Das Teilsystem ComPort hat nur sicherzustellen, dass eine bestimmte Anzahl von Zeichen gesendet wurden. Hierbei handelt es sich z.B. um einen GSM-Befehl. Wurden weniger Zeichen gesendet, als der Befehl Zeichen hat (inklusive Steuerzeichen), so ist ein Fehler aufgetreten.

Nachdem ein AT-Command (GSM-Befehl) erfolgreich gesendet wurden ist, wird einmal der Empfangspuffer der Schnittstelle ausgelesen. Der Inhalt wird nicht ausgewertet. Dies obliegt dem ComPortClient.




### 6.4.2. Sendefehler

Es gibt zwei schwerwiegende Fehler, die das Senden eines Commands unmöglich machen. Zum ersten kann es passieren, das die angeforderte Schnittstelle nicht zur Verfügung steht. Somit ist eine Initialisierung dieser nicht möglich. Das Senden wird sofort abgebrochen.

Der zweite schwerwiegende Fehler kann durch ein nicht angeschlossenes Mobiltelefon entstehen. Auch in diesem Fall wird die Übertragung für gescheitert angesehen. Es ist aber nicht Aufgabe dieses Teilsystem über Fehlerroutinen nachzudenken. Es meldet nur den Fehler an die steuernde Schicht.

# 7. Realisierung

Zur Umsetzung der Theorie in einen Programmcode verwenden wir die Entwicklungsumgebung Visual Studio.net der Firma Microsoft. Diese bietet mit umfangreichen Assistenten eine sehr gute Hilfe bei der Codeerzeugung. Zusammen mit dem Buch Visual C++ .NET von Dr. Susanne Wigard [1] ist es ein leichtes, den vorgefertigten Code auf unsere Bedürfnisse anzupassen.

## 7.1. ComPort

Der erste Quellcode den, wir erzeugen werden ist der ComPort Automatisierungsserver. Nachdem wir mit Hilfe des Erstellungsassistenten einen Automatisierungsserver auf Dialogbasis erstellt haben, müssen wir Teile des automatisch generierten Codes wieder deaktivieren. Hierzu gehört das Dialogfeld, welches wir nicht benötigen. Im nächsten Schritt implementieren wir unsere Klasse „CComPort“ die folgende Funktionsweise beinhaltet.

### 7.1.1. Konstruktor / Destruktor

Insgesamt implementieren wir drei Pointer. Diese werden im Konstruktor auf NULL gesetzt. Sollten sie bei der zerstörung eines Objektes nicht mehr auf NULL zeigen, so wird delete angewandt.

### 7.1.2. Initialisierung der Schnittstelle

```
Init()
{
    get HANDLE();
    wenn ok     dann init DCB;
    wenn ok     dann init CTMO;
}
```

Nachdem wir die Schnittstelle erfolgreich initialisiert haben, können wir über diese senden und empfangen.

### 7.1.3. Senden

```
Send(strToSend)
{
    ist strToSend gültiger string;

    wenn ok     Sende- und Empfangspuffer leeren;
                senden;
    wenn nicht  Send abbrechen;

    Rückgabe Anzahl der gesendeten Zeichen;
}
```

### 7.1.4. Empfangen

Nach jedem Senden erwarten wir eine Antwort des Mobiltelefons. Hierzu rufen wir die Funktion Receive auf, die den Empfangspuffer des HANDLEs ausliest.

### 7.1.5. Receive

```
Receive( &strResult)
{
    init Buffer;

    solange Zeichen empfangen werden
          lese Zeichen bis Buffer voll;
          wenn Buffer voll
                leeren;

    Rückgabe Anzahl der Empfangenen Zeichen;
}
```

## 7.2. Mobile

Nachdem wir den ComPort Automatisierungsserver implementiert haben, ist es nun möglich hierauf aufzubauen. Wir erstellen wieder mit Hilfe eines Assistenten einen Automatisierungsserver auf Dialogbasis.

Dieses Teilsystem besteht aus der Basisklasse CGSM, die drei allgemeingültige AT-Commands beinhaltet und den abgeleiteten speziellen Klassen für jedes Mobiltelefon (CSiemens).

### 7.2.1. CGSM

**a.** Konstruktor / Destruktor

Auch hier haben wir einen Zeiger auf die ComPort Schnittstelle der erzeugt und gelöscht werden muss.

**b.** ComPort öffnen

Bevor wir überhaupt mit der seriellen Schnittstelle kommunizieren können, müssen wir den ComPort Automatisierungsserver starten.

```
open()
{
     starte ComPort();

     wenn ok     return true;
     sonst       return false;
}
```

**c.** ComPort schließen

Solange der ComPort offen ist, belegen wir auch die vom Betriebssystem verwaltete Schnittselle. Es gehört zum guten Ton, diese nur zu belegen, wenn sie wirklich benötigt wird.

```
close();
{
     close ComPort();
     Zeiger auf NULL;
}
```

**d.** Hersteller auslesen

```
GetVendor()
{
     ComPort bereit?

     wenn ok     Send(at+cgmi\r\n);
     wenn ok     receive();
     wenn ok     Filter Steuerzeichen heraus;

     sonst       retrun NO_VENDOR_AVAILABLE
}
```

**e.** Modell auslesen

```
GetModel()
{
     ComPort bereit?

     wenn ok     Send(at+cgmmr\n);
     wenn ok     receive();
     wenn ok     Filter Steuerzeichen heraus;

     sonst       retrun NO_MODEL_AVAILABLE
}
```

**f.** Datum-Zeit auslesen

```
GetDateTime()
{
     ComPort bereit?

     wenn ok     Send(at+cclk\r\n);
     wenn ok     receive();
     wenn ok     Filter Steuerzeichen heraus;

     sonst       retrun NO_DATETIME_AVAILABLE
}
```

### 7.2.2. CSiemens

CSiemens ist eine von CGSM abgeleitete Klasse. Sie beinhaltet die speziellen Befehle für die Siemens-Familie, die von dem GSM-Standard abweichen.

**a.** Konstruktor / Destruktor

Diese Klasse speichert alle Datensätze aus dem Mobiltelefon in einem Array. Der Konstrukto initialisiert jedes Feld des Arrays mit einem leeren String. Gleichzeitig werden alle Marken, die später den Zugriff auf das Array regeln mit „0“ initilaisiert.

**b.** Datenfilter

Alle empfangenen Daten beinhalten auch Steuerzeichen diese werden mit Hilfe eines Automaten herausgefiltert.

```
bool DataFilter(Data)
{
     lese jedes Zeichen von Data
           wenn Steuerzeichen dann ignorieren;
           sonst merken;

     Automat im Endzustand;
           wenn ok     strData = strTmp;
                       return true;
           sonst       false;
}
```

**c.** Backend Mobile

Das Mobiltelefon sendet die Daten hexdezimal kodiert. Somit benötigen wir einen Automaten, der die Daten konvertiert.

```
ConvertHexToChar(Data)
{
     lese jedes Zeichen von Data
           konvertiere hex to char;
}
```

**d.** Frontend Mobile

Wenn wir Daten zum Mobiltelefon senden möchten, ist es natürlich notwendig diese dann hexadezimal zu senden.

```
ConvertHexToChar(Data)
{
     lese jedes Zeichen von Data
          konvertiere char to hex;
}
```

**e.** Frontend Sync

Daten, die zum Teilsysten Synchronisation übermittelt werden, müssen im Zwischencode vorliegen.

```
ConvertMobileToSync(Data)
{
     finde Schlüsselwörter
          merke die dazugehörigen Daten;
}
```

**f.** Backend Sync

Daten, die vom Teilsystem Synchronisation empfangen werden sind von der Zwischensprache in die Sprache des Mobiltelefons zu konvertrieren.

```
ConvertSyncToMobile(Data)
{
     finde Schlüsselwörter
          merke die dazugehörigen Daten;
}
```

**g.** open

Wie oben erwähnt speichert das Teilsystem Mobile alle im Mobiltelefon befindlichen Daten. Dies wird in der Funktion „open“ realisiert.

```
open()
{
     öffne ComPort();
     wenn ok     überprüfe ob Mobile present();
     wenn ok     lese alle Kontakte ein.

     schließe ComPort();
}
```

**h.** Get First

Der Datenzugriff erfolgt über die Get-Funktionen

```
GetFirst()
{
     Array-Zeiger auf 0;
     solange ArrayFeld leer;
           wenn ja dann Zeiger++;
           sonst return String;
}
```

**i.** Get Next

```
GetNext()
{
     Array-Zeiger++;
     solange ArrayFeld leer;
           wenn ja dann Zeiger++;
           wenn ArrayEnde erreicht  return END;
           sonst return String;
}
```

**j.** Get Previous

```
GetPrevious()
{
     Array-Zeiger--;
     solange ArrayFeld leer;
           wenn ja dann Zeiger--;
           wenn ArrayAnfang erreicht  return BEGIN;
           sonst return String;
}
```

**k.** Get Last

```
GetLast()
{
     Array-Zeiger auf Ende;
     solange ArrayFeld leer;
           wenn ja dann Zeiger--;
           sonst return String;
}
```

**l.** Set

Alle Get Funktionen arbeiten sozusagen offline. Dies ist bei Set nicht möglich. Es wird also erst die Verbindung über ComPort zum Mobiltelefon wieder aufgebaut und anschließend werden die Daten übertragen.

```
GetLast()
{
     öffne ComPort();
     konvertiere Daten von SyncToMobile();
     konvertiere Daten von CharToHex();

     lösche bestehenden Eintrag();

     zerteile Daten in mehrere Pakete();
     sende alle Pakete();
           wenn ok     weiter;
           sonst       return false;

     schließe ComPort();
}
```

**m.** Delete

Über delete können wir einen existierenden Datensatz im Mobiltelefon löschen.

```
Delete()
{
     öffne ComPort();
     Send(delete-command, Platz-Nr.);
     schließe ComPort();
}
```

**n.** Close

Der letzte Schritt besteht darin, nach der Synchronisation das Array wieder zu löschen und alle Läufer wieder auf deren Anfangsposition zu setzen.

```
Close()
{
     lösche jedes Array-Element;
     setze Läufer zurück;
}
```

## 7.3. PIM

Der Automatisierungsserver für Microsoft Outlook wird auf dieselbe Art und Weise wie die beiden vorherigen Teilsysteme erzeugt. Allerdings bedient dieses Teilsystem ausschließlich Microsoft Outlook und keine weiteren PIMs. Die Funktionalität ist in der Klasse COutlook untergebracht. Da wir direkt auf die Automatisierungsserver von Outlook zugreifen, ist es notwendig sogenannte Typbiliotheken mit einzubinden. Diese wiederum werden dann als Klassen im Projekt PIM mit eingebunden.

### 7.3.1. Outlook Klassen

Die genauen Funktionalitäten jeder einzelnen Klasse entnehmen Sie bitte dem Buch [3] Microsoft Outlook 2002 / Visual Basic Schritt für Schritt.

**a.** CApplication
**b.** CNameSpace
**c.** CItems
**d.** CContactItems
**e.** CAppointmentItem

### 7.3.2. COutlook

Diese Klasse dient der Synchronisation zum vereinfachten Zugriff auf Outlook. Wie schon im Teilsystem Mobile gibt es die vier Get-Funktionen, eine Set- und eine Delete-Funktion. Hinzu kommt noch eine Create-Funktion, die es ermöglicht in Outlook ein neues Item anzulegen. Ein Item kann z.B. ein Kontakt oder ein Termin sein.

**a.** Frontend
Alle Daten für die Synchronisation müssen in der Sprache des Zwischencodes übergeben warden.

```
ConvertOutlookToSync()
{
     leere String Data;

     addiere Outlook.Name zu Data;
     addiere Outlook.LastName zu Data;
     ...
     addiere Outlook.WebPage zu Data;

     return Data;
}
```

**b.** Backend

```
ConvertSyncToMobile(Data)
{
     suche Name in Data
          wenn gefunden schreibe Name nach Outlook;
     suche LastName in Data
          wenn gefunden schreibe LastName nach Outlook;
     ...
     suche WebPage in Data
          wenn gefunden schreibe WebPage nach Outlook;
}
```

**c.** Open

Über die open-Funktion steuern wir, ob wir nun mit Adressen oder mit Terminen arbeiten möchten. Dies bezieht sich insbesondere auf die Get-Funktionen.

```
open(typ)
{
     öffne Outlook;
     öffen NameSpace;
     wenn typ == Contact
          dann öffne Contact;
          sonst       Calendar;
}
```

**d.** GetFirst

```
GetFirst()
{
     hole erste Element von Outlook();
     konvertiere es to Sync();
     return String;
}
```

**e.** GetNext

```
GetNext ()
{
     hole nächstes Element von Outlook();
     konvertiere es to Sync();
     return String;
}
```

**f.** GetPrevious

```
GetPrevious()
{
     hole vorheriges Element von Outlook();
     konvertiere es to Sync();
     return String;
}
```

**g.** GetLast

```
GetLast()
{
     hole letztes Element von Outlook();
     konvertiere es to Sync();
     return String;
}
```

**h.** Create

```
Create(new Tuple)
{
     erzeuge neues Element in Outlook();
     konvertiere Tuple in Format Outlook();
     schreibe Tuple;
}
```

**i.** Set

```
Set(Tuple)
{
     konvertiere Tuple in Format Outlook();
     schreibe Tuple;
}
```

**j.** Delete

```
Delete()
{
     lösche aktuelles Item();
}
```

## 7.4. Synchronisation

Auch hier nutzen wir wieder den Projektassistenten von Visual Studio.net. Allerdings lassen wir uns eine einfache Dialogbasierte Anwendung erstellen, die als Add-In generiert wird.

```
Sync()
{
    open Mobile();
    open Outlook();

    wenn beide offen
          durch Liste Mobile hangeln
                Eintrag in Liste Outlook suchen
                      Eintrag gefunden
                      Wenn ja     synchronisieren;
                                  etwas geändert
                                  wenn ja     beide set;
                      wenn nein   Anwender fragen löschen
                                  oder neu anlegen;

          durch Liste Outlook hangeln
                Eintrag in Liste Mobile suchen
                      Eintrag gefunden
                      Wenn ja     synchronisieren;
                                  etwas geändert
                                  wenn ja     beide set;
                      wenn nein   Anwender fragen löschen
                                  oder neu anlegen;

    close Mobile;
    close Outlook;
}
```

# 8. Resümee

## 8.1. Mobiltelefone

Damit Systeme unterschiedlicher Hersteller miteinander kommunizieren können, hat die Industrie für viele Systeme einen so genannten Standard geschaffen. Auch im Bereich der Mobiltelefone gibt es die standardisierten AT-Commands. Leider bringt ein solcher Standard überhaupt nichts, wenn die Hersteller diesen überhaupt nicht implementieren oder diesen firmenspezifisch erweitern. Ebenso unsinnig ist ein Standard der nicht mit der Zeit geht. Sicherlich bringt es nichts diesen alle zwei Jahre neu zu definieren, aber es muss doch möglich sein diesen zu erweitern.

Erschwerend kommt noch hinzu, dass die Hersteller der Mobiltelefone Datenbankfunktionalitäten zwar im Handy integriert haben, diese aber nur halbherzig implementiert sind. Als Beispiel nehmen wir das Siemens ME45. Dieses Mobiltelefon bietet ein komplettes Adressbuch, aber es ist nicht möglich einem Eintrag einen Primary Key zuzuordnen. Hier ist noch erheblicher Nachholbedarf.

## 8.2. Outlook

Der Datentransfer mit Microsoft Outlook ist ohne größere Probleme möglich. Leider sind aber die meisten Dokumentationen nur in VBA verfügbar.

Outlook bietet sogar die Möglichkeit jedem Datensatz einen Art Primary Key zuzuweisen. Der Nachteil besteht nur darin, dass jeder Programmierer diesen ändern kann. Somit ist dieses Feld für den PK nicht geeignet. Dabei wäre es doch ein leichtes ein eindeutigen PK für jeden DS zu generieren (siehe GUI-ID)

## 8.3. Synchronisation

Die Synchronisation ist mit dem Stand der heutigen Technik zwar möglich, aber mit Vorsicht zu genießen. Es müssen aufwendige Prozeduren geschrieben werden, da wie schon erwähnt, Grundlegende Datenbankfunktionalitäten nicht implementiert sind. Dies erhöht die Wahrscheinlichkeit der Programmfehler erheblich.

## 8.4. Zukunft

Die erste Version unserer Synchronisationssoftware kann nur mit einem Mobiltelefon der „besseren“ Siemens-Klasse kommunizieren. In Zukunft soll dies auf andere Hersteller ausgedehnt werden. Auch Outlook soll durch andere PIMs ersetzt werden können.

Diese Software wurde unter Windows XP entwickelt. Ziel ist es diese auch auf andere BS zu portieren.

Das Ziel ist, dass durch die neue Technik „Bluetooth“ sich alle Gerät eine Anwenders selbstständig finden und deren Datenbestand abgleichen. Es wäre somit nicht mehr notwendig, dass ein Anwender die Synchronisation anstößt.

# 9. Anhang

## 9.1. Zwischencode

Die Kommunikation zwischen den Teilsystemen Mobile und Synchronisation erfolgt über eine eigene Sprache, dem Zwischencode. Der Zwischencode kann mehrere Tupel beinhalten. Jedes Tupel besitzt einen Ident und optional auch ein oder mehrere Datenelemente.

G = ($V_T$, $V_N$, P, S)

$V_T$ = alle Symbole des Latin Zeichensatzes

$V_N$ = (<tuple>, <ident>, <data>, <typ>, <subtyp>, <mod>, <vers> ,<attrib> ,<text> ,<char> ,<maxl> ,<date> ,<day> ,<month> ,<year> ,<time> ,<std> ,<min> ,<sec> ,<number> ,<digit>)

S = <tupel>

P:

| | | |
|---|---|---|
| <tupel> | ::= | <ident><data><tupel> \| <ident><data> |
| <ident> | ::= | <typ><subtyp><mod><vers> |
| <data> | ::= | <attrib><text><maxl> \| <attrib><text><maxl><data> |
| <typ> | ::= | contact \| termin |
| <subtyp> | ::= | <number> \| ε |
| <mod> | ::= | <date><time> |
| <vers> | ::= | number \| number.number \| 0 |
| <attrib> | ::= | N \| FN \| STR \| PC \| TWN \| TEL \| CELL |
| <text> | ::= | <char><text> \| <char> |
| <char> | ::= | LATIN - Zeichensatz |
| <maxl> | ::= | <number> |
| <date> | ::= | <day><month><year> |
| <day> | ::= | 0 … 31 |
| <month> | ::= | 0 … 12 |
| <year> | ::= | 0 … 9999 |
| <time> | ::= | <std><min><sec> |
| <std> | ::= | 0 … 24 |
| <min> | ::= | <sec> |
| <sec> | ::= | 0 … 60 |
| <number> | ::= | <digit><number> \| <digit> |
| <digit> | ::= | 0 \| 1 \| 2 \| 3 \| 4 \| 5\| 6 \| 7 \| 8 \| 9 |

## 9.2. Zeichenätze

### 9.2.1. Latin

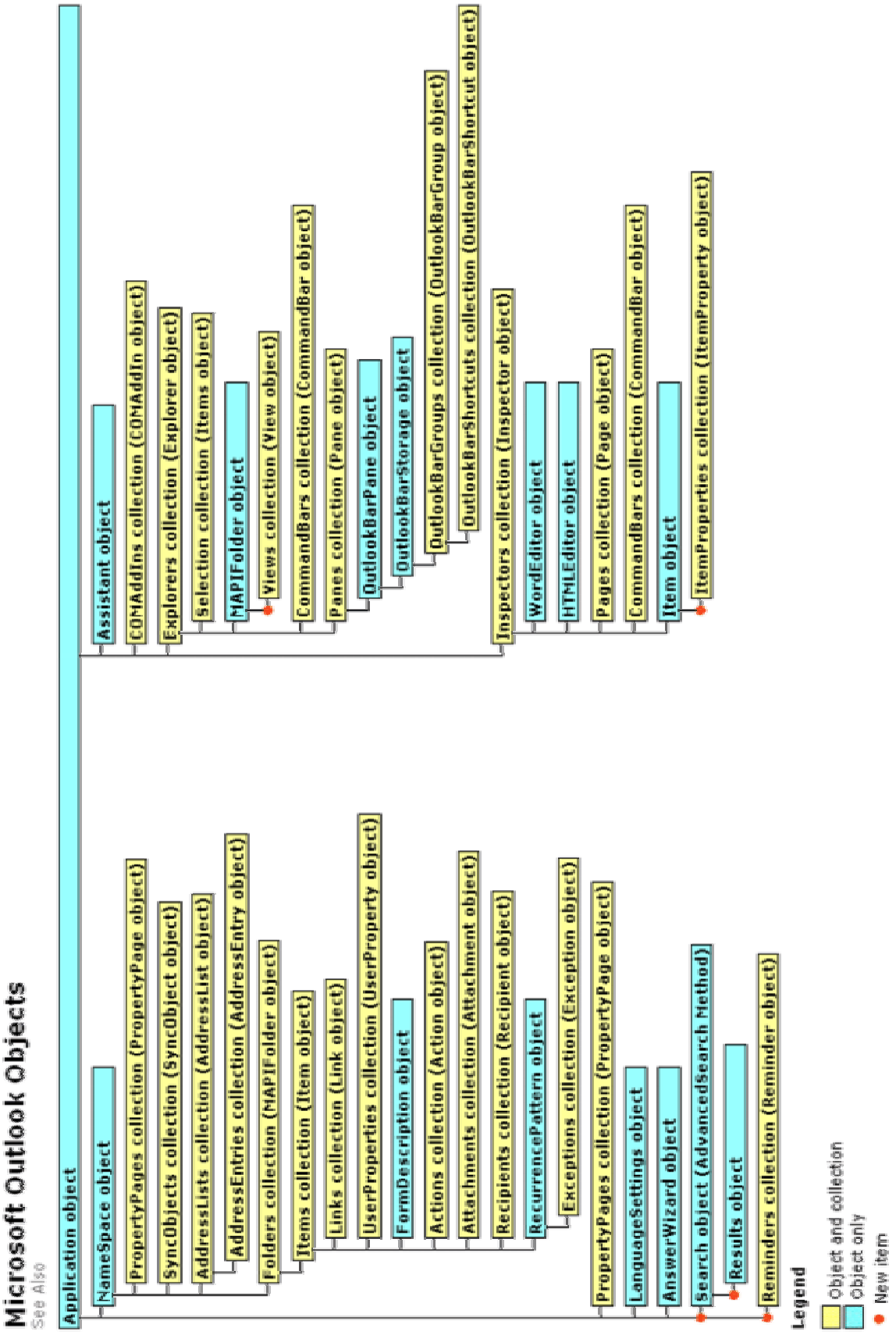
| Hex | Zeichn. | Hex | Zeichn. | Hex | Zeichn. | Hex | Zeichn. | Hex | Zeichn. | Hex | Zeichn. |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | NUL | 30 | 0 | 60 | ` | 90 | frei | C0 | À | F0 | ð |
| 1 | SOH | 31 | 1 | 61 | a | 91 | frei | C1 | Á | F1 | ñ |
| 2 | STX | 32 | 2 | 62 | b | 92 | frei | C2 | Â | F2 | ò |
| 3 | ETX | 33 | 3 | 63 | c | 93 | frei | C3 | Ã | F3 | ó |
| 4 | EOT | 34 | 4 | 64 | d | 94 | frei | C4 | Ä | F4 | ô |
| 5 | ENQ | 35 | 5 | 65 | e | 95 | frei | C5 | Å | F5 | õ |
| 6 | ACK | 36 | 6 | 66 | f | 96 | frei | C6 | Æ | F6 | ö |
| 7 | BEL | 37 | 7 | 67 | g | 97 | frei | C7 | Ç | F7 | ÷ |
| 8 | BS | 38 | 8 | 68 | h | 98 | frei | C8 | È | F8 | ø |
| 9 | HT | 39 | 9 | 69 | i | 99 | frei | C9 | É | F9 | ù |
| A | LF | 3A | : | 6A | j | 9A | frei | CA | Ê | FA | ú |
| B | VT | 3B | ; | 6B | k | 9B | frei | CB | Ë | FB | û |
| C | FF | 3C | < | 6C | l | 9C | frei | CC | Ì | FC | ü |
| D | CR | 3D | = | 6D | m | 9D | frei | CD | Í | FD | ý |
| E | SO | 3E | > | 6E | n | 9E | frei | CE | Î | FE | þ |
| F | SI | 3F | ? | 6F | o | 9F | frei | CF | Ï | FF | ÿ |
| 10 | DLE | 40 | @ | 70 | p | A0 |  | D0 | Đ |  |  |
| 11 | DC1 | 41 | A | 71 | q | A1 | ¡ | D1 | Ñ |  |  |
| 12 | DC2 | 42 | B | 72 | r | A2 | ¢ | D2 | Ò |  |  |
| 13 | DC3 | 43 | C | 73 | s | A3 | £ | D3 | Ó |  |  |
| 14 | DC4 | 44 | D | 74 | t | A4 | ¤ | D4 | Ô |  |  |
| 15 | NAK | 45 | E | 75 | u | A5 | ¥ | D5 | Õ |  |  |
| 16 | SYN | 46 | F | 76 | v | A6 | \| | D6 | Ö |  |  |
| 17 | ETB | 47 | G | 77 | w | A7 | § | D7 | × |  |  |
| 18 | CAN | 48 | H | 78 | x | A8 | " | D8 | Ø |  |  |
| 19 | EM | 49 | I | 79 | y | A9 | © | D9 | Ù |  |  |
| 1A | SUB | 4A | J | 7A | z | AA | ª | DA | Ú |  |  |
| 1B | ESC | 4B | K | 7B | { | AB | « | DB | Û |  |  |
| 1C | FS | 4C | L | 7C | \| | AC |  | DC | Ü |  |  |
| 1D | GS | 4D | M | 7D | } | AD |  | DD | Ý |  |  |
| 1E | RS | 4E | N | 7E | ~ | AE | ® | DE | Þ |  |  |
| 1F | US | 4F | O | 7F | frei | AF | ÿ | DF | ß |  |  |
| 20 | space | 50 | P | 80 | € | B0 | ° | E0 | à |  |  |
| 21 | ! | 51 | Q | 81 | frei | B1 | ± | E1 | á |  |  |
| 22 | " | 52 | R | 82 | frei | B2 | ² | E2 | â |  |  |
| 23 | # | 53 | S | 83 | frei | B3 | ³ | E3 | ã |  |  |
| 24 | $ | 54 | T | 84 | frei | B4 | ' | E4 | ä |  |  |
| 25 | % | 55 | U | 85 | frei | B5 | µ | E5 | å |  |  |
| 26 | & | 56 | V | 86 | frei | B6 |  | E6 | æ |  |  |
| 27 | ' | 57 | W | 87 | frei | B7 | • | E7 | ç |  |  |
| 28 | ( | 58 | X | 88 | frei | B8 | ¸ | E8 | è |  |  |
| 29 | ) | 59 | Y | 89 | frei | B9 | ¹ | E9 | é |  |  |
| 2A | * | 5A | Z | 8A | frei | BA | º | EA | ê |  |  |
| 2B | + | 5B | [ | 8B | frei | BB | » | EB | ë |  |  |
| 2C | , | 5C | \ | 8C | frei | BC | ¼ | EC | ì |  |  |
| 2D | - | 5D | ] | 8D | frei | BD | ½ | ED | í |  |  |
| 2E | . | 5E | ^ | 8E | frei | BE | ¾ | EE | î |  |  |
| 2F | / | 5F | _ | 8F | frei | BF | ¿ | EF | ï |  |  |

### 9.2.2. GSM

| Nieder-wertigeres Byte (Hex) | Höherwertiges Byte (Hex) | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
| 0 | @ | | SP | 0 | | P | | p |
| 1 | £ | _ | ! | 1 | A | Q | a | q |
| 2 | $ | | " | 2 | B | R | b | r |
| 3 | YEN | | # | 3 | C | S | c | s |
| 4 | é | | | 4 | D | T | d | t |
| 5 | è | | % | 5 | E | U | e | u |
| 6 | ù | | & | 6 | F | V | f | v |
| 7 | ì | | ' | 7 | G | W | g | w |
| 8 | ò | | ( | 8 | H | X | h | x |
| 9 | | | ) | 9 | I | Y | i | y |
| A | LF | | * | : | J | Z | j | z |
| B | Ø | 1) | + | ; | K | Ä | k | ä |
| C | ø | Æ | , | < | L | Ö | l | ö |
| D | CR | æ | - | = | M | Ñ | m | ñ |
| E | Å | ß | . | > | N | Ü | n | ü |
| F | å | É | / | ? | O | § | o | á |

## 9.3. Objekt-Modelle

### 9.3.1. Microsoft Outlook



## 9.4. Quellen

### 9.4.1. Literaturverzeichnis

[1] Visual C++ .NET von Dr. Susanne Wigard
DAS bhv TASCHENBUCH

[2] Microsoft Outlook 2002 von Rita Wendel
Microsoft Press Deutschland

[3] Microsoft Outlook 2002 / Visual Basic Schritt für Schritt
Microsoft Press Microsoft Press

[4] Administering NDS (Nancy Cadjan und Jeffrey L. Harris)
Computing McGrill-Hill