Diplomarbeit im Studiengang Audiovisuelle Medien
Fachbereich Electronic Media

Bluetoothbasiertes Informationssystem

Konzeption und Realisation eines Prototypen für einen Stadtinformationsdienst

1. Prüfer: Prof. Walter Kriha
2. Prüfer: Nina Köstering

vorgelegt von Thomas Suchy

an der Fachhochschule Stuttgart
Hochschule der Medien
am 10. April 2004

Erklärung

Hiermit versichere ich, die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst zu haben. Alle verwendeten Quellen sind im Text oder im Anhang nachgewiesen.

Thomas Suchy
Stuttgart, den 08. April 2004

Kurzfassung

Diese Diplomarbeit befasst sich mit einem Informationsdienst, der auf Basis der Funktechnologie Bluetooth Daten übermittelt. Hierzu kommuniziert ein stationärer Informationsprovider mit einem mobilen Endgerät.
Zunächst werden mögliche Komponenten und Technologien für ein solches System erörtert. Anschließend wird ein Einblick in die verwendete Übertragungstechnologie Bluetooth gegeben, sowie die Grundlagen und Konzepte der beiden eingesetzten Betriebssysteme beschrieben.
Der Hauptteil der Diplomarbeit beschreibt die Ausarbeitung des praktischen Teils, in dem ein Prototyp für einen Stadtinformationsdienst entwickelt und umgesetzt wurde. Hierbei wird auf die Konzeption des Gesamtsystems und die Umsetzung der Anwendungen auf den einzelnen Komponenten eingegangen.
In den Schlussbetrachtungen folgt ein Fazit über den Prototypen und mögliche Erweiterungen werden aufgezeigt. Abschließend wird ein Ausblick auf die vielfältigen Einsatzmöglichkeiten der verwendeten Technologien gegeben.

Danksagung

Dank gilt in erster Linie meinen Eltern, die mir mein Studium ermöglicht und mich jederzeit voll unterstützt haben.

Susanne Faber danke ich für Ihre große Unterstützung und ihre hilfreichen Kommentare beim Durchsehen dieser Arbeit. Ebenso danke ich hierfür Jan Schulze, Christoph Nufer und Stephan Schöbel.

Tobias Frech danke ich für seine Hilfestellungen bei der Entwicklung des Prototypen, ebenso Ansgar Gerlicher, Dennis Kühn und Jakob Meister.

Zu guter Letzt danke ich Professor Walter Kriha und Nina Köstering für die Betreuung dieser Diplomarbeit und ihre hilfreichen Hinweise.

Inhaltsverzeichnis

1 Einleitung

Sie sind überall - mobile Endgeräte, die uns den Alltag erleichtern. Fast ständig führen wir Laptops, PDAs und Mobiltelefone bei uns und nutzen sie zur Kommunikation und Datenverarbeitung.
Vor allem durch den Austausch von Informationen gewinnen diese mobilen Geräte an Bedeutung. Unterschiedliche Technologien zur Datenübertragung finden Verwendung und auch Sprache wird mittlerweile in Form von digitalen Datenpaketen übermittelt. Bald wird die Größe eines Gerätes nur noch durch die gewünschte Größe der Anzeige bestimmt sein, später werden die Informationen vielleicht sogar in den Raum projiziert oder auf andere Art dem Benutzer vermittelt.
Doch keines der Geräte wird in der Lage sein, alle gewünschten Informationen zu speichern, diese müssen auch von außen übermittelt werden können. Und um möglichst unabhängig zu sein geschieht dies drahtlos, ohne physikalische Verbindung zum bereitstellenden System.

Was versteht an unter mobilen Endgeräten und welche Übertragungsmöglichkeiten für den Informationsaustausch gibt es?

1.1 Mobile Endgeräte

„Mobil" heißt „beweglich". Mobilität ist daher vor allem eine Frage von Größe, Gewicht und Unabhängigkeit bezüglich Stromversorgung und Verbindungen.
Unter mobilen Endgeräten werden in dieser Diplomarbeit Geräte verstanden, die diesen Anforderungen gerecht werden: Sie sind portabel, besitzen eine eigene Hardware mit entsprechender Stromversorgung und geeignetem Betriebssystem und verfügen über die notwendigen Schnittstellen, um drahtlos kommunizieren zu können.

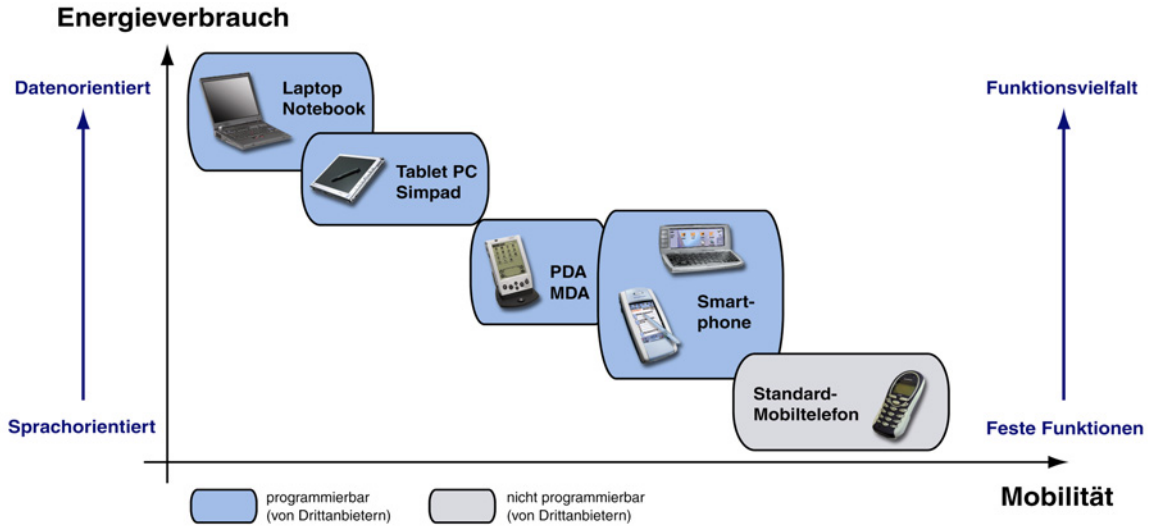


Abbildung 1-1: Überblick Mobile Endgeräte

Um mobile Endgeräte für den Markt interessant zu machen, sollten diese programmierbar sein. So können auch komplexe Anwendungen von Drittanbietern entwickelt werden. Solche Anwendungen für mobile Geräte werden als „Mobile Applikationen" bezeichnet. Soll die Anwendung auf mehreren Gerätearten eingesetzt werden, so muss sie auf das entsprechende Betriebssystem, sowie die Darstellungs- und Eingabemöglichkeiten des Zielgerätes angepasst werden. Immer wichtiger wird dabei die plattformunabhängige Programmiersprache Java. Unterschiedliche Java-Versionen wurden für die unterschiedlichen Einsatzgebiete entwickelt, für programmierbare Mobiltelefone z. B. J2ME (Java 2 Micro Edition). Auf Java-Programmierung soll in dieser Diplomarbeit jedoch nicht weiter eingegangen werden, da der Prototyp aufgrund der derzeitigen technischen Voraussetzungen nicht in Java realisiert wurde.

Programmierbare mobile Endgeräte

Die drei wichtigsten programmierbaren mobilen Endgeräte sind Laptops, PDAs/MDAs und Smartphones (s. Abbildung 1-1). Eher exotische Vertreter wie Tablet-PCs oder SimPads spielen für den Massenmarkt keine große Rolle und sollen daher hier nicht weiter betrachtet werden.

Laptops sind sehr leistungsfähig und haben ein großes Display, sind aber auch recht teuer. Für einen bluetoothbasierten Informationsdienst erscheinen sie gut geeignet, haben jedoch den großen Nachteil, dass sie aufgrund ihrer Größe und des Strombedarfs nur bedingt portabel sind.
PDAs/MDAs sind wesentlich kleiner und dank eines ansprechend großen Displays auch gut in einem Informationssystem einsetzbar. Gegen sie spricht die eher geringe Verbreitung im Consumer-Bereich.
Smartphones sind klein und portabel. Sie verfügen über ein leistungsfähiges Betriebssystem und ein ansprechend großes Display. Smartphones sind momentan noch nicht allzu sehr verbreitet. Aufgrund der Nähe zum Standard-Mobiltelefon ist jedoch zu erwarten, dass sie sich in den nächsten Jahren als Massenmarktprodukt durchsetzen werden.

1.2 Übertragungstechnologien

Informationsaustausch wird immer wichtiger und gerade für mobile Endgeräte ist es entscheidend, dass dieser drahtlos geschehen kann. In diesem Abschnitt soll daher auf die wichtigsten drahtlosen Übertragungstechnologien eingegangen werden, die bei der Programmierung von mobilen Applikationen eingesetzt werden können.

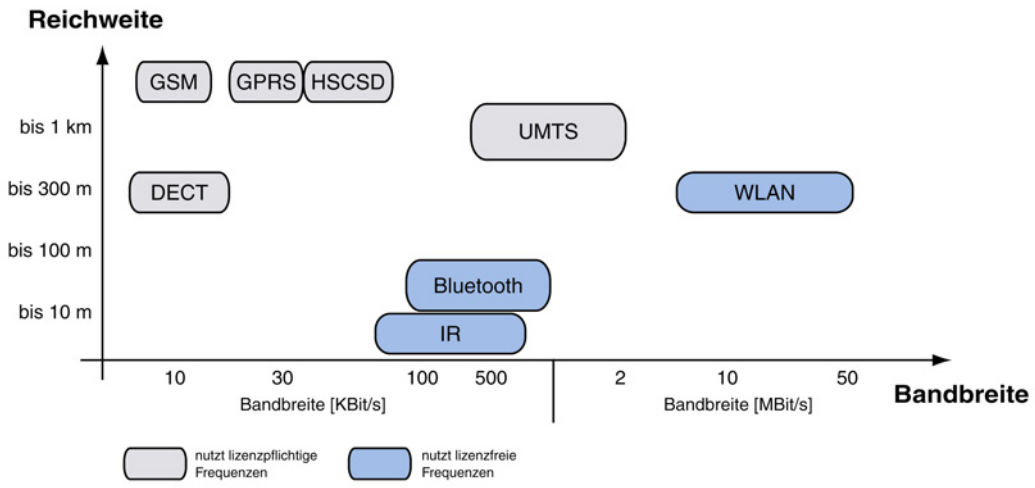


Abbildung 1-2: Überblick drahtlose Übertragungstechnologien

Grundsätzlich bieten sich zur drahtlosen Kommunikation zwei Möglichkeiten an:
Die Kommunikation per Infrarot oder per Funk.

IR (Infrarot)

Bei Infrarot werden Daten optisch übertragen. Die Wellenlängen sind hierbei länger als die für das menschliche Auge sichtbare und kürzer als Funkwellenlängen.[1] Sender und Empfänger müssen bei Infrarot-Übertragungen Sichtkontakt haben, was den großen Nachteil dieser Technologie darstellt und den Einsatz in vielen Bereichen uninteressant macht.
Infrarot ist dennoch sehr verbreitet und findet vor allem bei Fernbedienungen Anwendung. Aber auch Laptops und Mobiltelefone sind oft mit Infrarot ausgerüstet und können so beispielsweise ihre Daten synchronisieren. Infrarot ist günstig in Herstellung und Stromverbrauch, hat aber auch nur eine recht geringe Reichweite von etwa 5 Metern.

Funk

DECT (Digital European Cordless Telecommunication)

DECT ist ein etablierter Standard für digitale Funkübertragung und wird hauptsächlich bei Schnurlostelefonen eingesetzt. DECT arbeitet in einem reservierten Frequenzband und ermöglicht Reichweiten zwischen 50 Metern in Gebäuden und 300 Metern im Freien. Ein großer Vorteil von DECT besteht in seiner geringen Störanfälligkeit. Ein Nachteil ist jedoch, dass weltweit unterschiedliche Frequenzen verwendet werden.[2]

GSM (Global System for Mobile Communication)

Das GSM-Netz unterstützt Sprach- und Datentransport. Hauptsächlich Mobiltelefone nutzen dieses weit verbreitete Netz der so genannten „Second Generation" (2 G).[3] Die so genannte CSD-Verbindung (Circuit Switched Data) ist leitungsorientiert und überträgt bis zu 9,6 kBit/s. Beim GSM-Netz können zur Erhöhung der Transferrate auch mehrere Kanäle gebündelt werden. Diese Technik nennt sich dann HSCSD oder „High Speed Circuit Switched Data".[4]

GPRS (General Packet Radio Service)

GPRS ist eine Verbesserung des GSM-Standards für Datenübertragungen und stellt eine Brückentechnologie dar für so genannte 2.5 G-Netzwerke. GPRS ist paketorientiert und unterstützt Konzepte wie virtuelle Verbindungen.[5] Da GPRS vor allem nach Volumen und nicht nach Zeit berechnet wird, ist für den Nutzer auch eine ständige Verbindung zum Provider möglich, ohne die Kosten entscheidend zu erhöhen. GPRS kann bis zu 115 kBit Daten pro Sekunde übertragen und ist damit deutlich schneller als GSM.

UMTS (Universal Mobile Telecommunication System)

UMTS ist ein Mobilfunkstandard der dritten Generation (3 G) und bietet vor allem höhere Transferraten von bis zu 2 MBit/s. Derzeit wird UMTS aufgrund der hohen Kosten noch selten eingesetzt. Außerdem gibt es nur sehr wenige Endgeräte, die UMTS unterstützen und der Dienst ist bei weitem nicht flächendeckend verfügbar.

[1] Vgl. Jipping (2002), S. 52.
[2] Vgl. Frech (2003), S. 16.
[3] Vgl. Jipping (2002), S. 380.
[4] Vgl. Frech (2003), S. 17.
[5] Vgl. Jipping (2002), S. 380.

WLAN (Wireless Local Area Network)

WLAN definiert mehrere Versionen des bekannten Funkstandards IEEE 802.11. Fast alle WLAN-Produkte arbeiten im Bereich von 2,4 GHz. Die Technologie wird hauptsächlich zur drahtlosen Vernetzung von Computern eingesetzt, wobei die Transferrate bei der schnellsten Version bis zu 54 MBit/s beträgt. Die Reichweite liegt zwischen 30 Metern in Gebäuden und etwa 300 Metern im Freien. In einer Richtfunkvariante können sogar bis zu 20 Kilometer im Freien erreicht werden.[6]

Bluetooth

Bluetooth ist ein offener Funkstandard für Kurzstrecken, der wie auch WLAN im freien Frequenzband von 2,4 GHz arbeitet. Bluetooth unterstützt Sprach- und Datenübertragungen und kann sehr vielfältig eingesetzt werden. So kann beispielsweise ein Computer mit seinen Peripherie-Geräten oder ein Mobiltelefon mit seinem Headset drahtlos über Bluetooth verbunden werden. Die Transferrate bei asymmetrischen Verbindungen liegt bei bis zu 721 kBit/s und die Reichweite eines Gerätes kann bis zu 100 Meter erreichen.[7] Auf Bluetooth wird in Kapitel 2 ausführlich eingegangen.

1.3 Abgrenzung des Themas

Diese Diplomarbeit beschreibt ein bluetoothbasiertes Informationssystem anhand eines Prototypen für einen Stadtinformationsdienst. Folgende Bereiche sind daher zu betrachten:

- Ein stationäres Gerät, das die Informationen bereitstellt.
- Ein mobiles Endgerät, das die Informationen abruft und darstellt.
- Die Verbindung der beiden Geräte über Bluetooth.

Stationäres Gerät

Als stationäres Gerät könnte im Grunde jeder PC dienen. Anforderungen an das stationäre Gerät beim Stadtinformationsdienst waren neben Unterstützung von Bluetooth jedoch vor allem eine einfache Programmierbarkeit und die einfache Installationsmöglichkeit am jeweiligen Standort. In dieser Diplomarbeit wird daher ein kleines, programmierbares Embedded Device beschrieben, das beim Prototypen eingesetzt wurde. Es unterstützt Bluetooth und hat ein eigenes Betriebssystem namens HyNetOS. Das eingesetzte Gerät ist weitaus kleiner als ein PC, was der Installation am Standort sehr entgegenkommt.

Mobiles Endgerät

Welche mobilen Endgeräte sich in der Zukunft durchsetzen werden, ist nicht mit Sicherheit vorherzusagen. Doch die Größe und damit die Portabilität wird ein entscheidender Faktor sein. Und unter den mobilen Endgeräten haben Mobiltelefone momentan die größte Verbreitung.[8] Der zu entwickelnde Stadtinformationsdienst stellte einige Anforderungen an das eingesetzte Mobiltelefon. Wichtige Auswahlkriterien waren die Unterstützung von Bluetooth, umfangreiche Programmier-barkeit und Benutzerfreundlichkeit. Daher wurde ein Bluetooth-Smartphone mit dem leistungsfähigen Symbian-Betriebssystem verwendet. Das eingesetzte Gerät ist mit einem ansprechend großen Display ausgestattet und bietet umfangreiche Bedienmöglichkeiten.

[6] Vgl. PDF-Quelle IZT, S. 49.
[7] Vgl. Merkle (2002), S. 68-69.
[8] Vgl. Internetquelle Statistisches Bundesamt.

Verbindung über Bluetooth

Als Technologie für die Verbindung der Geräte wird Bluetooth näher erläutert. Für die Entwicklung des Stadtinformationsdienstes war es wichtig, dass die Informationen problemlos an den jeweiligen Standorten lokal bereitgestellt werden können. Bluetooth kann im Kurzstreckenbereich unlizenziert eingesetzt werden und ist hinsichtlich der Reichweite von bis zu 100 m ebenfalls für einen solchen Dienst geeignet. Gegen den Einsatz von Infrarot sprach neben dessen ungenügender Reichweite auch die recht hohe Störanfälligkeit.
Ein weiteres Auswahlkriterium war die Kostenkontrolle für den Benutzer, wie sie beim Einsatz der Technologien aus dem Mobiltelefonbereich wie etwa GSM oder GPRS nicht gegeben gewesen wäre. Bluetooth ist weiterhin aufgrund seiner stromsparenden Eigenschaften im Gegensatz zu WLAN schon in vielen mobilen Endgeräten integriert.

Aufbau der Diplomarbeit

Bluetooth ist eine relativ neue Funktechnologie. Im folgenden Kapitel werden daher deren Grundlagen erläutert. Auch Symbian OS und HyNetOS sind recht neue und vor allem spezielle Betriebssysteme mit eigenen Konzepten. Um ein grundlegendes Verständnis zu ermöglichen, wird in den Kapiteln drei und vier auf diese beiden Betriebssysteme eingegangen.
Den Hauptteil der Diplomarbeit stellt der praktische Teil dar, in dem ein Prototyp für einen bluetoothbasierten Stadtinformationsdienst entwickelt und realisiert wurde. Dieses Projekt, das für die Firma Alcatel SEL AG in Stuttgart durchgeführt wurde, wird im Folgenden auch kurz als „Stadtinfo-Projekt" bezeichnet. Nach eingehender Beschreibung von Analyse, Konzeption und Umsetzung des Projektes folgen einige Schlussbetrachtungen.

2 Bluetooth

Die Geschichte von Bluetooth beginnt mit einer Studie der Firma Ericsson im Jahre 1994.[9] Es sollten mittels eines kostengünstigen und energiesparenden Systems über Funk Daten zwischen Mobiltelefonen und deren Zubehör ausgetauscht werden. Nachdem Ericsson das Potential des Konzeptes bewusst wurde, gründeten sie 1998 die „Bluetooth Special Interest Group" (Bluetooth SIG).[10] Diese sollte die Entwicklung einer Technologie für drahtlose Übermittlung von Sprache und Daten per Funk übernehmen. Weitere Gründungsmitglieder waren IBM, Intel, Nokia und Toshiba. Heute sind über 2500 Firmen, hauptsächlich als Lizenznehmer, vertreten. Im Juli 1999 wurde die Version 1.0 der Bluetooth-Spezifikation veröffentlicht.
Wofür steht Bluetooth?

- Bluetooth steht für Netzwerke für Sprach- und Datenübertragung:
  Da Bluetooth sowohl Sprache als auch Daten übertragen kann, ist es eine ideale Technologie für Geräte wie Mobiltelefone, die genau dies benötigen. Außerdem kann Bluetooth spontan Netzwerke aufbauen, um zwei oder mehrere Geräte miteinander zu verbinden.
- Bluetooth kann weltweit im Kurzstreckenbereich eingesetzt werden:
  Bluetooth arbeitet im Frequenzbereich des 2,4 GHz ISM-Bandes (Industrial Scientific Medical Band). Dieser wurde weltweit für Industrie, Wissenschaft und Medizin vorgesehen und ist daher nahezu überall gebührenfrei und ohne Lizenz verfügbar. Bluetooth-Geräte haben, je nach Klasse, eine Reichweite zwischen 3 und 100 Metern.
- Bluetooth ist ein offener Industriestandard:
  Die Standards sind kostenlos verfügbar.[11]

Was aber ist das Besondere an Bluetooth? Bisherige Techniken sind recht spezialisiert: z. B. GSM für das Mobiltelefonnetz, WLAN für Computernetzwerke, DECT für schnurlose Telefone oder Infrarot für Fernbedienungen. Bluetooth dagegen versucht, ein wesentlich größeres Spektrum abzudecken. Das zeigt sich schon in der Anzahl an unterschiedlichen Klassen und Unterklassen der Bluetooth-Spezifikation. Die unterschiedlichsten Geräte sollen sich gegenseitig spontan erkennen können, so dass keine Konfiguration von Seiten des Benutzers notwendig ist – quasi ein Plug and Play, nur über Funk.

---

[9] Vgl. Merkle (2002), S. 5-23.
[10] Der Name „Bluetooth" geht zurück auf Harald Blatand, einen dänischen König des 10. Jhdts. Ihm gelang die Vereinigung sich bekämpfender Parteien des heutigen Norwegens, Schwedens und Dänemarks. Vgl. Internetquelle Bluetooth-Geschichte.
[11] Kostenloser Download der Bluetooth-Spezifikation unter Internetquelle Bluetooth-Spezifikation.

Bereits heute gibt es eine Vielzahl an Geräten auf Bluetooth-Basis und der Standard bleibt offen für Erweiterungen: Headsets, Mobiltelefone, Computerdongles, Computermaus/-tastatur, Bluetooth Access Points und vieles mehr.
Bluetooth kann beispielsweise eingesetzt werden, um Zeichnungen, die mit einem mit Bluetooth ausgerüsteten Filzstift an eine herkömmliche Tafel gemalt werden, direkt auf die bluetoothfähigen Geräte beispielsweise von Schulungsteilnehmern zu übertragen.[12]
Toshiba rüstet Waschmaschinen mit Bluetooth aus[13], SonyEricsson stellt ein per Mobiltelefon fernsteuerbares Spielzeugauto her[14] und auch Pulsdaten können mittlerweile per Bluetooth übertragen werden[15]. Auch die Automobilindustrie hat die neue Funktechnologie für sich entdeckt: Im Toyota Prius ist z. B. eine auf Bluetooth basierende Freisprecheinrichtung integriert.[16]

Abbildung 2-1: Nokia N-Gage

Vor allem wird die Technologie jedoch bei mobilen Geräten eingesetzt, da Bluetooth kostengünstig ist und sich durch einen geringen Stromverbrauch auszeichnet. Auch die Fähigkeit, spontan Netzwerke aufbauen zu können, scheint für Mobiltelefonhersteller zunehmend interessant zu werden. So hat Nokia das „N-Gage" entwickelt (s. Abbildung 2-1): Ein Mobiltelefon basierend auf dem Symbian-Betriebssystem, das als Spielekonsole genutzt werden kann. N-Gage-Besitzer können so über Funk im Netzwerk miteinander spielen. Angesichts des Booms in der LAN-Party-Szene ein durchaus erfolgversprechendes Konzept.

2.1 Technik

Bluetooth-Geräte werden hinsichtlich ihrer Sendeleistung in drei Klassen unterteilt:

Klasse	***Maximale Sendeleistung (Pmax)***	***Reichweite***
3	1 mW (0 dBm)	etwa 3m
2	2,5 mW (4 dBm)	etwa 10m
1	100 mW (20 dBm)	etwa 100m

Tabelle 2-1: Bluetooth-Klassen[17]

Wie bereits erwähnt arbeitet Bluetooth in dem Frequenzbereich des 2,4 GHz ISM-Bandes, da dieses für Geräte mit geringer Sendeleistung weltweit lizenzfrei zur Verfügung steht.
Dieser Frequenzbereich wird daher aber auch von vielen anderen Geräten genutzt. Diese sind oft lokal installiert und auf eine feste Frequenz eingestellt.
Um Störern entgegenzuwirken, entschied man sich zu einem so genannten „Frequenzsprung-verfahren". Die Frequenz, auf der gesendet und empfangen wird, springt bei Bluetooth in einer pseudozufälligen Sequenz 1.600 mal in der Sekunde zwischen insgesamt 79 Kanälen. So erreicht

[12] Vgl. Internetquelle Bluetooth-E-Beam.
[13] Vgl. Internetquelle Bluetooth-Waschmaschine.
[14] Vgl. Internetquelle Bluetooth-Spielzeugauto.
[15] Vgl. Internetquelle Bluetooth-Pulsmessgerät.
[16] Vgl. Internetquelle Bluetooth-Toyota.
[17] Vgl. PDF-Quelle Bluetooth-Spezifikation, S. 21.

man eine sehr gute Störunanfälligkeit.[18] Gehen trotzdem Daten verloren, regeln bestimmte Ebenen im Protokoll, dass diese erneut gesendet werden.
Auch das verbreitete WLAN nutzt das 2,4 GHz-Frequenzband. Wie sich in Untersuchungen mit Bluetooth und WLAN-IEEE 802.11b herausgestellt hat, sind Interferenzen zwischen den beiden Systemen und damit eine Verringerung der Datenrate jedoch akzeptabel, solange diese in einem angemessenen Abstand zueinander betrieben werden.[19]

Netze

Eine Besonderheit von Bluetooth ist die Art und Weise, wie Netzwerke spontan und selbstständig gebildet werden können. Man spricht hier von so genannten „Ad-hoc-Netzen". Für diese ist keine fest installierte Infrastruktur notwendig. Die einzelnen Geräte sind anhand ihrer weltweit eindeutigen Bluetooth-ID identifizierbar.[20]

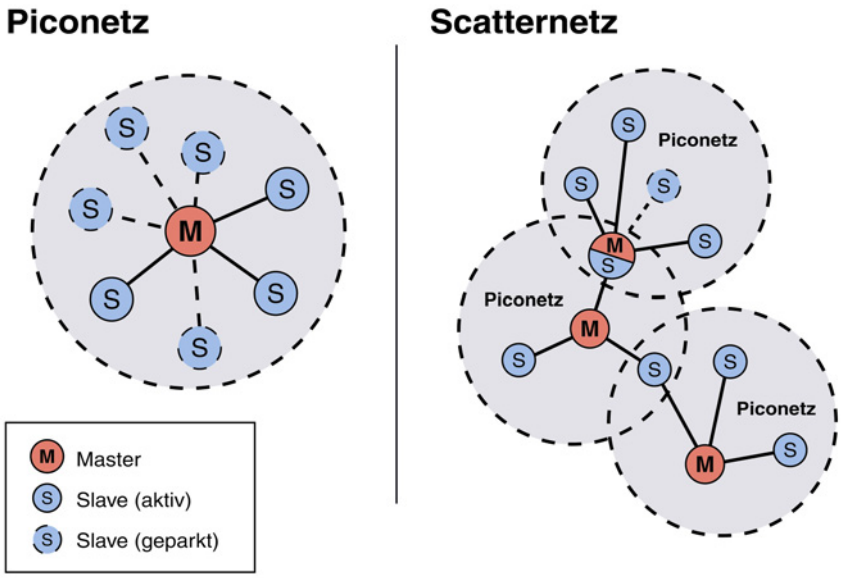


Abbildung 2-2: Bluetooth-Netze

Verbinden sich zwei oder mehr Bluetooth-Geräte miteinander, so spricht man von einem „Piconetz" (s. Abbildung 2-2). Aufgrund des Frequenzsprungverfahrens müssen die beiden Geräte synchronisiert werden, um eine Kommunikation zu ermöglichen. Diese Aufgabe übernimmt die Einheit, die den Verbindungsaufbau eingeleitet hat: die so genannte „Master"-Einheit. Der Master kann sich gleichzeitig mit bis zu sieben aktiven und 255 geparkten „Slaves" verbinden. Ein PC, der beispielsweise Maus, Tastatur, Drucker und Scanner per Bluetooth anspricht, ist ein solcher Master in einem Piconetz. Piconetze können aber auch untereinander kommunizieren, indem ein Master oder ein Slave des einen Piconetzes gleichzeitig Slave eines zweiten Piconetzes wird. Dann spricht man von einem „Scatternetz". Der jeweilige Master legt die individuelle Frequenzsprungfolge fest. Die Wahrscheinlichkeit einer Kollision der Piconetze ist daher relativ gering.

[18] Vgl. Kapitel 2.2.1.
[19] Vgl. Merkle (2002), S. 271-284.
[20] Vgl. Kapitel 2.1.2.

Stromverbrauch

Bluetooth-Geräte in einem Piconetz können vom Master in verschiedene Zustände versetzt werden, die in einem unterschiedlich hohen Stromverbrauch resultieren:

- Active:
  Im aktiven Zustand erwartet der Slave Übertragungen vom Master. Alle Pakete werden ausgewertet, und bei einer Sendeerlaubnis wird mit der Übertragung begonnen. Die Stromaufnahme im aktiven Zustand ist hoch, die Reaktionszeit dafür aber kurz. Anwendung: Aktive Kommunikation.
- Sniff:
  Der Slave wird nur periodisch aktiv, sonst ist er abgeschaltet. Das Intervall der aktiven Phase kann flexibel eingestellt werden. Je nach Einstellung sind Reaktionszeit und Strombedarf entsprechend kurz bzw. hoch.
  Anwendung: Der Slave soll relativ schnell angesprochen werden können, es ist aber noch nicht bekannt, wann wieder kommuniziert werden muss.
- Hold:
  Der Slave wird für eine festgelegte Zeit abgeschaltet. Diese Zeit ist dem Master und dem Slave bekannt und kann flexibel eingestellt werden. Danach ist der Slave wieder aktiv. Je nach Einstellung kann die Reaktionszeit im Vergleich zum Sniff-Zustand schlechter sein, die durchschnittliche Stromaufnahme ist jedoch geringer.
  Anwendung: Es ist bekannt, dass für eine bestimmte Zeit keine Kommunikation notwendig sein wird. So lange kann der Slave abgeschaltet werden.
- Park:
  Der Slave ist nicht aktiv, lediglich die Synchronisation mit dem Master wird aufrecht erhalten. Dies erfolgt mit dem so genannten „Beacon-Schlitz-Verfahren": In einem dem Master und Slave bekannten Zeitraum schaltet der Slave kurz auf empfangsbereit. Danach schläft er wieder, sollte er kein Aktivierungssignal vom Master erhalten haben. Bis zu 255 Slaves eines Piconetzes können sich neben den sieben aktiven Geräten im geparkten Zustand befinden. Park bedeutet eine sehr geringe Stromaufnahme, aber auch eine relativ hohe Reaktionszeit.
  Anwendung: Der Slave soll im Piconetz eingebunden sein, um aktiviert werden zu können, derzeit ist aber keine aktive Kommunikation notwendig.

Bluetooth-Geräte, die nicht in ein Piconetz eingebunden sind, befinden sich in einem Standby-Zustand. Darin suchen sie im Abstand von 1,28 s nach möglichen Übertragungen in ihrer Umgebung. Dies ist der Zustand mit der geringsten Stromaufnahme.

Um weiterhin Strom zu sparen, können Geräte ihre Sendeleistung gegenseitig beeinflussen. Befinden sich zwei Geräte beispielsweise direkt nebeneinander, so ist die Empfangsqualität, gemessen mit dem so genannten „RSSI"-Wert (Received Signal Strength Indicator), sehr gut. Der Empfänger kann den Sender in einem solchen Fall auffordern, seine Sendeleistung zu reduzieren, um die Stromaufnahme nicht unnötig zu erhöhen. Der Master führt diese individuelle Leistungsanpassung für jeden Slave gesondert durch. Diese werden also unter Umständen alle mit einer anderen Sendeleistung angesprochen.[21]

[21] Vgl. Merkle (2002), S. 68-75.

Eine Systemübersicht über Bluetooth wurde nun gegeben. Dabei wurde auf die verwendete Übertragungstechnik sowie auf die möglichen Verbindungsnetze der Geräte untereinander eingegangen. Im Folgenden sollen der Bluetooth-Protokoll-Stack und seine verschiedenen Protokolle eingehender beschrieben werden.

2.2 Bluetooth-Protokoll-Stack

Bluetooth soll die Kommunikation unterschiedlichster Geräte von verschiedenen Herstellern ermöglichen. Um dies zu erreichen, sind einheitliche Regeln notwendig, an die sich alle halten. Diese werden von der Bluetooth SIG formuliert und ständig erweitert. Die so genannte Bluetooth-Spezifikation kann in der jeweils aktuellen Version von deren Internetseite kostenlos heruntergeladen werden.[22]
In der Bluetooth-Spezifikation wird ein Protokoll-Stack festgelegt, der von den Herstellern je nach Bedarf auf seinen Geräten implementiert wird. Dieser Protokoll-Stack verwendet weitest möglich schon bestehende Protokolle. So kann die Einarbeitungszeit für Entwickler gering gehalten werden und es ist einfacher, bestehende Anwendungen auf Bluetooth zu adaptieren.

Bluetooth-Protokoll-Stack

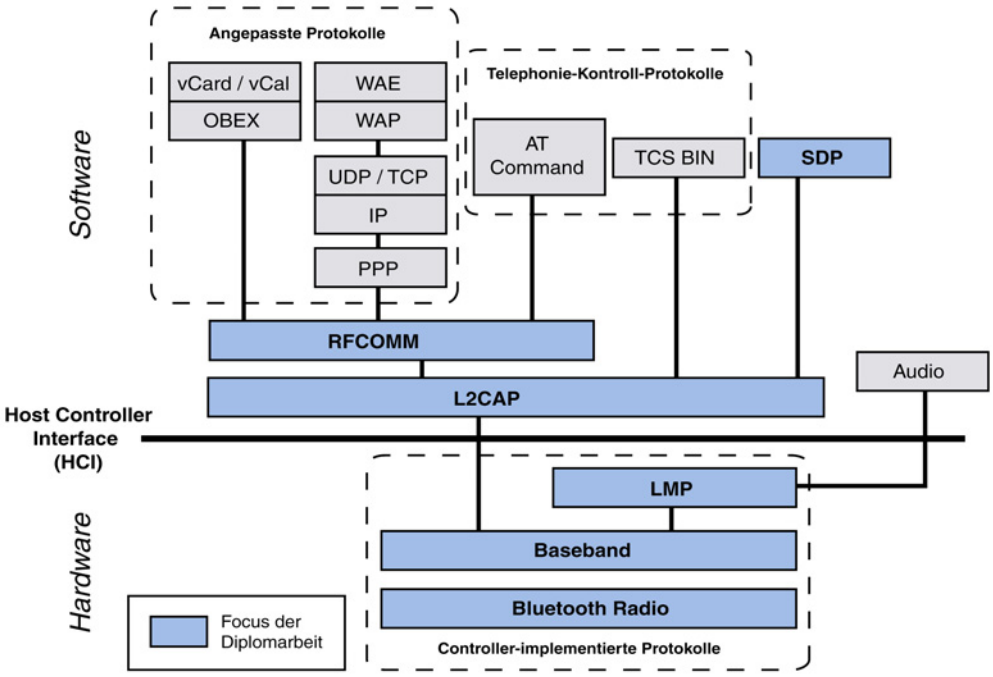


Abbildung 2-3: Der Bluetooth-Protokoll-Stack

[22] Vgl. Internetquelle Bluetooth-Spezifikation.

Die Protokolle des Bluetooth-Protokoll-Stacks können in vier Gruppen unterteilt werden (s. Abbildung 2-3):

- Bluetooth-Kern-Protolle (Bluetooth Core Protocols): Bluetooth Radio, Baseband, LMP, L2CAP und SDP.
  Dies sind die für Bluetooth exklusiven und wichtigsten Protokolle.
- Kabelersatz-Protokoll (Cable Replacement Protocol): RFCOMM.
  Mit RFCOMM (Radio Frequency Oriented Emulation of the Serial COM Ports) werden serielle Schnittstellen emuliert.
- Telefonie-Kontroll-Protokolle (Telephony Control Protocols): TCS-Binary, AT-Commands.
  Sie basieren auf Protokollen aus der Telefon-Industrie.
- Angepasste Protokolle (Adopted Protocols): PPP, UDP, TCP, IP, WAP, OBEX.
  Diese Protokolle wurden lediglich für Bluetooth angepasst.

Das Host Controller Interface (HCI) ist die Schnittstelle zwischen den Protokollen, die auf dem Bluetooth Controller integriert sind wie Baseband und LMP und Protokollen, die softwareseitig auf dem Host implementiert sind, wie beispielsweise L2CAP oder SDP.

Vergleich von Bluetooth mit dem ISO/OSI-Modell [23]

Der Bluetooth-Protokoll-Stack referenziert das ISO/OSI-Modell, stimmt jedoch nicht exakt damit überein. In Abbildung 2-4 kann die Verbindung der beiden erkannt werden.

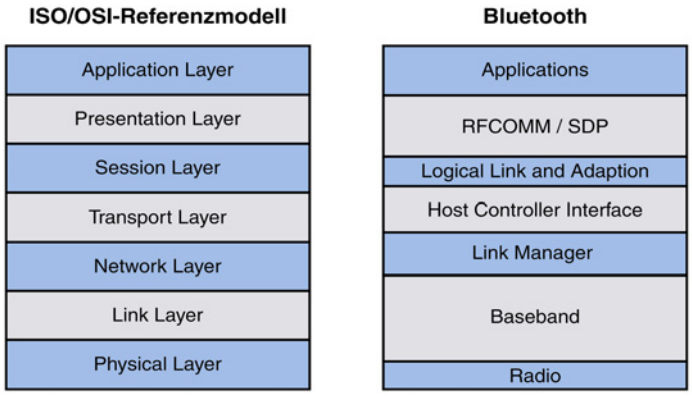


Abbildung 2-4: Vergleich ISO/OSI-Modell mit dem Bluetooth-Protokoll-Stack

In den folgenden Abschnitten soll nur auf die Bluetooth-Kern-Protokolle und das RFCOMM-Protokoll näher eingegangen werden. Zum einen sind dies die wichtigsten und für Bluetooth spezifischen Protokolle, zum anderen wurden sie direkt oder indirekt im Stadtinfo-Projekt verwendet.

[23] Vgl. Merkle (2002), S. 79.

2.2.1 Bluetooth Radio

Bluetooth-Geräte sind naturgemäß nicht per Kabel miteinander verbunden. Bluetooth Radio ist die unterste Ebene des Protokoll-Stacks und repräsentiert die physikalische Ebene des OSI-Referenzmodells.

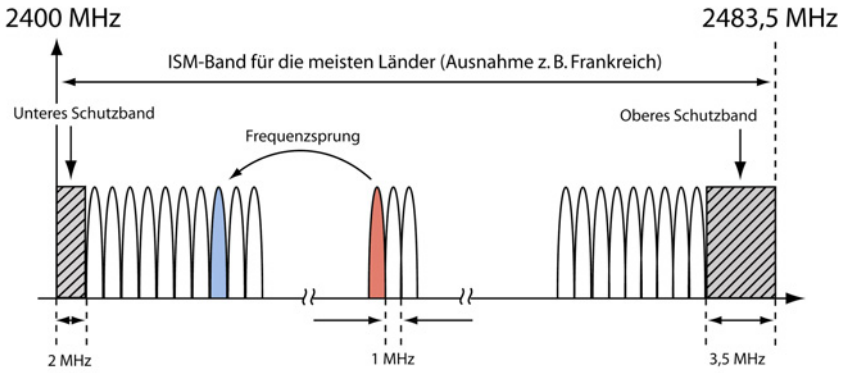


Abbildung 2-5: Bluetooth-Frequenzband und Kanaleinteilung

Das freie ISM-Band im Bereich von 2,4 GHz hat eine Bandbreite von insgesamt 83,5 MHz. Diese wurde bei Bluetooth in 79 Kanäle mit je 1 MHz Bandbreite eingeteilt. Außerdem wurden zwei Schutzbänder vorgesehen, um Interferenzen mit benachbarten Funksystemen zu vermeiden (s. Abbildung 2-5).[24] Zwischen diesen 79 Kanalfrequenzen springen alle Teilnehmer eines Piconetzes gleichzeitig 1.600 mal in der Sekunde. Die Frequenzsprungfolge wird von der weltweit eindeutigen Bluetooth-Geräteadresse des Masters abgeleitet und ist somit ebenfalls einzigartig.[25] Gegenüber schmalbandigen, festfrequenten Störsignalen ist Bluetooth daher recht robust, da durch die vielen Sprünge insgesamt gesehen nur recht wenig Daten im gestörten Frequenzbereich übertragen werden (vgl. Abbildung 2-6).

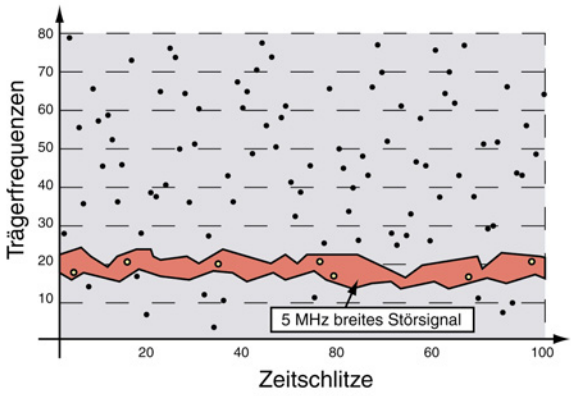


Abbildung 2-6: Frequenzsprünge mit Störfrequenz

Die Daten werden auf die jeweilige Trägerfrequenz mit dem so genannten Frequency Shift Keying-Verfahren aufmoduliert. Das bedeutet, dass eine binäre Eins durch eine positive Frequenzabweichung von der Trägerfrequenz übertragen wird, eine binäre Null durch eine negative Abweichung. Genauer gesagt wird das GFSK (Gaussian Frequency Shift Keying) eingesetzt: Um die Bandbreite des Signals noch weiter zu reduzieren, wird das Datensignal vor der Modulation noch durch einen Gaußfilter gefiltert. Das Frequenzspektrum wird so noch effektiver ausgenutzt und die Beeinflussung von benachbarten Kanälen weiter reduziert.[26]

[24] Da in Frankreich nicht das komplette 2,4 GHz ISM-Band freigegeben wurde, wird hier mit nur 23 Kanälen gearbeitet. Vgl. Merkle (2002), S. 85.
[25] Vgl. PDF-Quelle Bluetooth-Spezifikation, S. 126-137.
[26] Vgl. Merkle (2002), S. 86-88.

Bluetooth soll sowohl Daten, als auch Sprache übertragen. Diese verschiedenen Anwendungszwecke stellen unterschiedliche Anforderungen. Es wurden daher zwei Arten von physikalischen Verbindungen definiert, die auch kombiniert werden können:[27]

- SCO-Link (synchronous connection-oriented link):
  Ein SCO-Link ist eine synchrone, verbindungsorientierte Verbindung. Ein Master kommuniziert mit einem bestimmten Slave in einer symmetrischen Punkt-zu-Punkt-Verbindung. Verloren gegangene SCO-Pakete werden nicht erneut übertragen. Ein Master kann bis zu drei SCO-Links verwalten. Anwendung: Zeitkritische Übertragungen wie z. B. Sprache.
- ACL-Link (asynchronous connection-less link):
  Eine asynchrone, verbindungslose Verbindung. Ein Master kommuniziert mit seinen Slaves in einer Punkt-zu-Multipunkt-Verbindung. Verlorene Pakete werden wiederholt versendet. Anwendung: Zeitunkritische, paketorientierte Übertragungen wie z. B. beim Laden einer Internetseite.

2.2.2 Baseband

Die Baseband-Ebene ist die wohl wichtigste Komponente des Bluetooth-Protokoll-Stacks.
Sie hat eine Reihe von Aufgaben:

- Steuerung der physikalischen Funkverbindung
- Verpacken/Entpacken der Datenpakete
- Festlegung der Frequenzsprung-Sequenz
- Fehlerkorrektur
- Datentransfer
- Verwaltung der Verbindungen
- Adressierung
- Authentisierung, Autorisation und Verschlüsselung

Um diese Aufgaben zu erfüllen, kommunizieren Bluetooth-Geräte auf Baseband-Ebene in Form von Datenpaketen. Auf dessen Aufbau und unterschiedliche Arten soll daher zunächst kurz eingegangen werden.

Aufbau des Baseband-Standardpaketes

Abbildung 2-7: Baseband-Standardpaket

Ein Standardpaket besteht aus drei Teilen (s. Abbildung 2-7):

- Access Code (68 oder 72 Bit):
  Der Access Code dient der Synchronisation und Identifikation. Ein Bluetooth-Gerät überprüft diesen Teil des empfangenen Paketes und verwendet es nur, wenn es für das Gerät bestimmt ist. Alle Pakete eines Piconetzes haben den gleichen Access Code.

[27] Vgl. PDF-Quelle Bluetooth-Spezifikation, S. 45.

- Header (54 Bit):
  Der Header enthält in den ersten 10 Bit wichtige Steuerinformationen. Hier wird z. B. die Adresse der aktiven Slaves und die Art der Übertragung (SCO oder ACL) festgelegt. Danach werden 8 Bit von einem CRC-Code belegt, der den fehlerfreien Empfang des Headers feststellt. Diese 10+8 Bit werden drei mal übertragen, wodurch die Länge des Headers auf 54 Bit anwächst. Die korrekte Übertragung des Headers ist also mehrfach abgesichert.
- Payload (0-2745Bit):
  Im Payload sind die eigentlichen Daten des Paketes enthalten, wie beispielsweise Sprachdaten bei einer SCO-Verbindung oder Daten einer zu übertragenden Datei bei einer ACL-Verbindung.

Arten des Baseband-Standardpaketes

Die Baseband-Standardpakete werden in drei unterschiedliche Arten eingeteilt:

- Steuerpakete:
  Sie dienen der Steuerung und Fehlerkorrektur der Übertragung und enthalten keine Informationsdaten des Nutzers.
- ACL-Pakete:
  ACL-Pakete enthalten die Nutzerdaten bei einer ACL-Verbindung und sind je nach notwendiger Fehlerkorrektur und Übertragungsgeschwindigkeit unterschiedlich aufgebaut.
- SCO-Pakete:
  SCO-Pakete haben eine feste Länge und enthalten, je nach erforderter Qualität der zu übertragenden Daten, die Möglichkeit zur Fehlerkorrektur. Verlorene SCO-Pakete werden nicht erneut gesendet.

Fehlerkorrektur

Ein Funkkanal ist, verglichen mit einer Kabelverbindung, grundsätzlich sehr störanfällig. Im Baseband werden daher unterschiedliche Fehlerkorrekturverfahren eingesetzt, auf die im Rahmen dieser Diplomarbeit jedoch nicht weiter eingegangen werden kann.[28]

Scrambling

Wie eben beschrieben gibt es eine Vielzahl unterschiedlicher Pakete. Bevor diese Pakete übermittelt werden, werden sie umgeordnet, was man auch als „Scrambling" bezeichnet. Bei Bluetooth dient dies weniger dem Schutz vor unerlaubtem Abhören der Verbindung als vielmehr einer Verbesserung der Übertragungseigenschaften. Eine Übertragung digitaler Daten in Form einer Schwingung ist immer dann für Fehler anfällig, wenn viele Nullen oder Einsen aufeinander folgen. Durch das Scrambling-Verfahren wird dieser Gleichspannungsanteil minimiert. Auf Empfängerseite werden die Pakete dann wieder zurückgeordnet.

[28] Weitergehende Informationen vgl. PDF-Quelle Bluetooth-Spezifikation, S. 66-74.

Synchronisation

Um eine problemlose Kommunikation zwischen Bluetooth-Geräten zu gewährleisten, müssen diese exakt synchronisiert werden. Jede Bluetooth-Einheit hat hierfür einen internen Zeitgeber. In Piconetzen passen die Slaves ihre interne Systemzeit an die des Masters an. Regelmäßig wird ein Offset bestimmt, damit alle Geräte gleichzeitig die Frequenz wechseln und zur richtigen Zeit empfangsbereit werden. Dieser Zeitgeber sollte laut Spezifikation mit 3,2 kHz getaktet sein. Bei einer Untersuchung mit einem Bluetooth-Analyzer wurde hingegen festgestellt, dass Geräte teilweise erheblich von dieser Vorgabe abweichen können. So beispielsweise bei einem getesteten SonyEricsson P800-Mobiltelefon, dessen interner Takt mit 3,4 kHz gemessen wurde.[29] Eine Datenübertragung mit dem Gerät war trotzdem problemlos möglich.

Bluetooth-Adresse

LAP (Lower Address Part): herstellerspezifische Identifikationsnummer
UAP (Upper Address Part): firmeninterne Nummer
NAP (Non-significant Address Part): firmeninterne Nummer

Abbildung 2-8: Bluetooth-Adresse

Eine weitere wichtige Aufgabe des Baseband ist die Adressierung. Ein Bluetooth-Gerät hat eine weltweit eindeutige Adresse (s. Abbildung 2-8). Diese besteht aus insgesamt 48 Bit und enthält unter anderem eine herstellerspezifische ID sowie eine firmeninterne Nummer. Bei 48 Bit sind so 2^{48} verschiedene Adressen möglich. Bei einer Weltbevölkerung von angenommenen 6,5 Milliarden Menschen entspricht dies über 40.000 möglichen Bluetooth-Adressen pro Mensch. Einem Problem vergleichbar mit dem der IP-Adressen wurde so aller Voraussicht nach ausreichend vorgebeugt.
Die Bluetooth-ID wurde beim Stadtinfo-Projekt unter anderem eingesetzt, um die Geräte eindeutig identifizieren zu können, für die bereits für den Informationsdienst bezahlt wurde.

Sicherheit

Sicherheit ist bei einer Funkverbindung ungleich wichtiger als bei Kabelverbindungen. Schon das Frequenzsprungverfahren gewährleistet eine gewisse Sicherheit vor ungewolltem Abhören einer Verbindung. Das genügt jedoch nicht, wenn sensible Daten übermittelt werden sollen. Werden in einer Firma beispielsweise Daten per Bluetooth an einen Drucker gesendet, so soll es nicht möglich sein, diese von außen abzuhören. Bluetooth trägt diesen Anforderungen Rechnung, indem drei Sicherheitsstufen definiert wurden:[30]

- Security Mode 1 (non-secure):
  Keinerlei Sicherheitsmaßnahmen auf Baseband-Ebene.
- Security Mode 2 (service level enforced):
  Erst nach erfolgreichem Verbindungsaufbau werden erste Sicherheitsmaßnahmen ergriffen. Je nach angefordertem Service kann dann beispielsweise eine Verschlüsselung erfolgen.
- Security Mode 3 (link level enforced):
  Je nach Einstellung können Geräte schon vor Verbindungsaufbau abgelehnt werden.

[29] Dies ergab eine Untersuchung mit einem Bluetooth-Analyzer an der Hochschule der Medien, Stuttgart, im Juli 2003.
[30] Vgl. Merkle (2002), S. 134-147.

2.2.3 Der Link Manager

In den vorangegangenen Kapiteln wurden die beiden untersten Schichten des Protokoll-Stacks beschrieben. Bluetooth Radio ermöglicht die physikalische Übertragung der einzelnen Bits und Baseband regelt das gesicherte Senden und Empfangen von Datenpaketen. Ein Verbindungsaufbau beispielsweise kann aber mit diesen Funktionen noch nicht erreicht werden.
Der Link Manager übernimmt nun diese und andere wichtige Aufgaben, die hauptsächlich die Verbindung selbst betreffen:

- Verbindungsaufbau und –abbau
- Generierung, Austausch und Überprüfung der Sicherheitsschlüssel
- Powermodes und Leistungsregelung
- Aushandeln der Paketgrößen des Baseband
- Quality of Service (QoS)
- Link-Überwachung

Link Manager Protocol

Der Link Manager ist, wie auch Bluetooth Radio und Baseband, auf dem Bluetooth Controller implementiert und kommuniziert in erster Linie mit dem Link Manager des jeweils anderen Gerätes. Das dabei verwendete Protokoll wird als Link Manager Protocol (LMP) bezeichnet. Das LMP überträgt keine Anwenderdaten. Die zwischen zwei Link Managern ausgetauschten Nachrichten werden im Payload von ACL-Paketen übertragen.[31]

Verbindungsaufbau und -abbau

Als Voraussetzung für einen Verbindungsaufbau durch den Link Manager muss bereits ein ACL-Link auf Baseband-Ebene vorhanden sein. Durch den Link Manager erfolgt dann eine Vielzahl aufeinander folgender Schritte, von denen die wesentlichen hier beschrieben werden sollen:[32]

1. Als erstes wird der Offset zur Systemzeit des Masters bestimmt. Dieser wird nach jedem empfangenen Paket aktualisiert, um eine optimale Synchronität zu gewährleisten.
2. Dann wird die Version des LMP-Protokolls ermittelt. Dies ist notwendig, da Unterschiede in den LMP-Versionen bestehen.
3. Als nächstes wird ausgetauscht, welche Eigenschaften das jeweilige Gerät tatsächlich unterstützt, da nicht alle Eigenschaften von Baseband und Bluetooth Radio obligatorisch sind. Ein Bluetooth-USB-Dongle, der an einen PC angeschlossen ist, benötigt beispielsweise nicht die optionalen Stromsparzustände Hold, Park oder Sniff.
4. Nun wird der benutzerfreundliche Name ausgetauscht, den jede Bluetooth-Einheit haben kann. Dieser darf laut Spezifikation bis zu 248 Byte lang sein.
   Offset, LMP-Version, unterstützte Eigenschaften und Name der Gegenstelle sind nun bekannt. Sollte nach Auswertung dieser Daten keine Verbindung mehr erwünscht werden, wird der Verbindungsaufbau hier beendet.
5. Soll eine Verbindung hergestellt werden, fordert der Master nun den eigentlichen Verbindungsaufbau an. Wird diese Anforderung erfolgreich bestätigt, werden die Verbindungseigenschaften bezüglich Pairing, Authentifikation, Verschlüsselung und Quality of Service (QoS) ausgehandelt.
   Der Aufbau einer ACL-Verbindung ist nun abgeschlossen.

[31] Vgl. PDF-Quelle Bluetooth-Spezifikation, S. 191-192.
[32] Weiterführende Informationen vgl. Merkle (2002), S. 152-162.

Wird eine SCO-Verbindung gewünscht, folgen noch weitere Schritte, bei denen die Parameter für eine SCO-Verbindung festgelegt werden. Dies kann z. B. der Audio-Codec sein, der für die Übermittlung von Sprachdaten eingesetzt wird.
Weitere LMP-Prozeduren dienen dem Einstellen der Stromsparzustände, einem Master/Slave-Rollentausch etc.

Auch der Verbindungsabbau erfolgt in einer strengen Abfolge von ausgetauschten Nachrichten. Hierbei wird unter anderem der Grund für die Einleitung des Verbindungsabbaus angegeben.

2.2.4 Host Controller Interface (HCI)

In diesem Kapitel soll auf die Schnittstelle zwischen der Hardware des Bluetooth Controllers und der Software des Host eingegangen werden. Die Bluetooth Hardware wird über diese Schnittstelle mit so genannten HCI-Kommandos angesprochen werden.

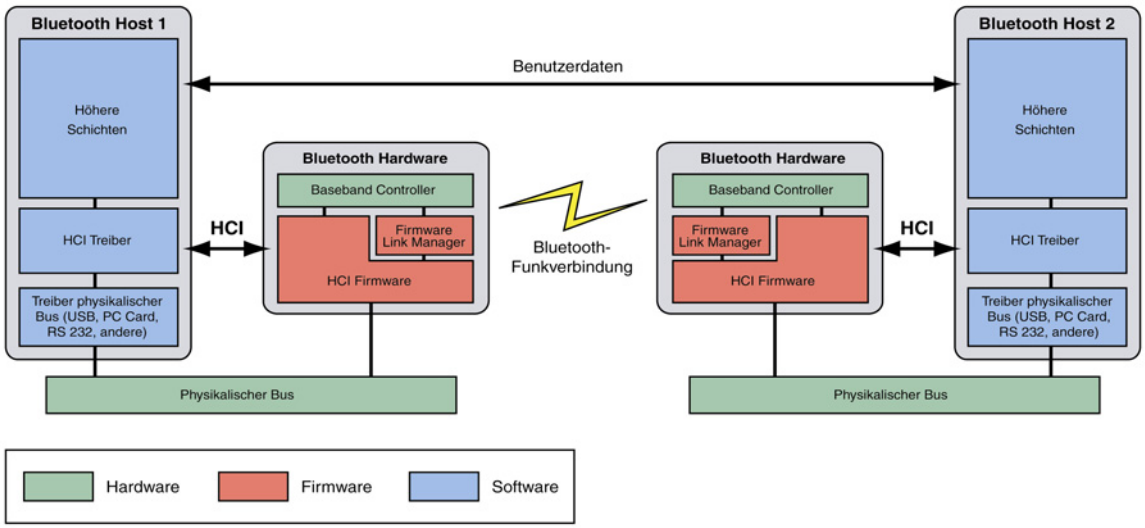


Abbildung 2-9: Überblick Bluetooth-Kommunikation mit Focus HCI[33]

Damit ein Gerät den Bluetooth-Chip nutzen kann, muss es physikalisch mit diesem verbunden sein. Dies geschieht über ein so genanntes Bussystem. Derzeit werden bei Bluetooth drei Bussysteme unterstützt: Universal Serial Bus (USB) wie bei USB-Dongles, PC-Card (PCMCIA) und serielle Schnittstellen.[34]
Ist diese physikalische Schnittstelle vorhanden, kommuniziert der Host über seinen HCI-Gerätetreiber mit der auf dem Controller des Bluetooth-Chips integrierten HCI Firmware.
Diese Kommunikation ermöglicht auch eine Flusskontrolle für ACL-Daten: Der Host kann den zur Verfügung stehenden Puffer der Bluetooth Hardware ermitteln und den Datenfluss zum Bluetooth-Chip dementsprechend regulieren. Das ist beispielsweise notwendig, wenn das Gegengerät nicht mehr antwortet.
Im Embedded-Bereich werden vornehmlich serielle Schnittstellen eingesetzt, sei es RS232, UART oder das proprietäre Transportprotokoll BCSP. BCSP wurde von der Firma Cambridge Silicon Radio (CSR) entwickelt und steht für BlueCore Serial Protocol.

[33] Vgl. PDF-Quelle Bluetooth-Spezifikation, S. 544.
[34] Auf den Datenaustausch über das USB- und PCMCIA-Bussystem soll hier nicht näher eingegangen werden.

Beim Einsatz von BCSP ist es möglich, auch die Schichten L2CAP, RFCOMM und SDP im Controller des Bluetooth-Chips zu integrieren und damit in der Hardware. So muss weit weniger Aufwand für die softwareseitige Implementierung dieser Protokolle auf dem entsprechenden System betrieben werden. BCSP setzt auf UART auf, ermöglicht jedoch zusätzlich noch eine Fehlerkorrektur und hat eine höhere Transferrate als UART.[35]
Das beim Stadtinfo-Projekt eingesetzte Bluetooth-Modul ist mit dem Bluetooth-Chip „BlueCore01b" der Firma CSR bestückt.[36] Das BCSP-Protokoll wird hier allerdings nicht eingesetzt, da die zusätzliche Fehlerkorrektur bauartbedingt nicht notwendig ist und eine Einschränkung der Leistungsfähigkeit zur Folge gehabt hätte.[37] Das Modul nutzt daher das UART-Protokoll.

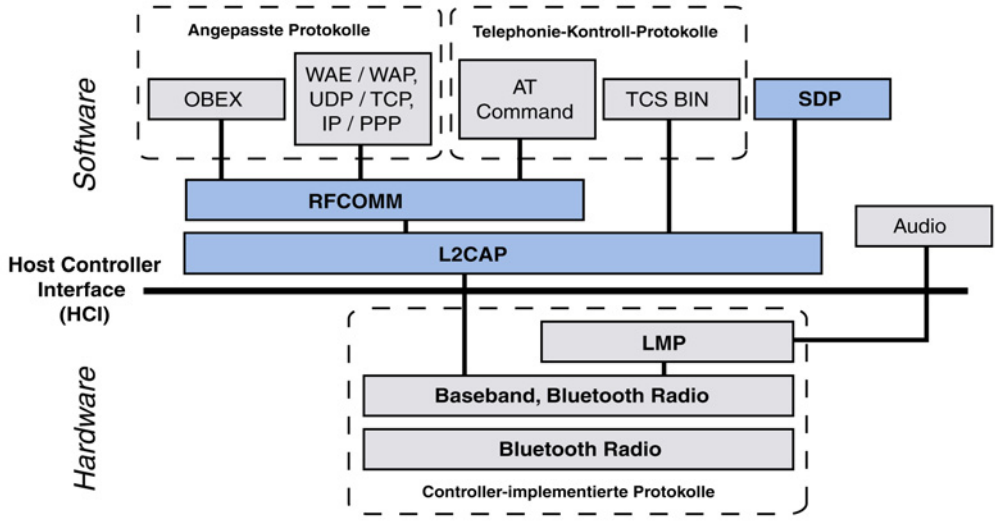


Abbildung 2-10: Bluetooth-Protokoll-Stack mit Focus L2CAP/RFCOMM/SDP

Die Protokolle des Controllers und die Schnittstelle zum Host wurden nun erläutert. In den folgenden Kapiteln wird auf die wichtigsten Protokolle des Host eingegangen (s. Abbildung 2-10): L2CAP, RFCOMM und SDP. RFCOMM kommt in dieser Diplomarbeit besondere Bedeutung zu, da das Mobiltelefon und das Bluetooth-Modul im Stadtinfo-Projekt auf dieser Ebene kommunizieren. SDP stellt ein interessantes Protokoll dar, das es Bluetooth-Geräten ermöglicht, die gegenseitig angebotenen Dienste zu erkennen. SDP könnte daher auch in der Finalversion des Stadtinfo-Projektes eingesetzt werden.

2.2.5 Logical Link Control and Adaption Protocol (L2CAP)

Das L2CAP-Protokoll empfängt Daten von den höheren Protokollschichten und sendet diese weiter an die unter ihr liegenden Schichten (s. Abbildung 2-10). L2CAP unterstützt ausschließlich ACL-Verbindungen, überträgt also in der Regel kein Audio.
Die Aufgaben der L2CAP-Ebene kann man folgendermaßen zusammenfassen:

- Multiplexen von höheren Protokollen
- Segmentierung/Desegmentierung von großen Datenpaketen
- Gruppenmanagement
- Verwaltung des Quality of Service für die höheren Protokollschichten

[35] Vgl. Merkle (2002), S. 163-182.
[36] Vgl. PDF-Quelle MBT-Info.
[37] Auskunft per Mail vom 04.03.2004 von Offner, Georg, SND.

Kommunikation über Kanäle

Die L2CAP-Schichten der verschiedenen Bluetooth-Geräte kommunizieren über so genannte Kanäle. Jeder Kanal hat zwei Endpunkte. Diesen Endpunkten wird je nach Kanalart eine so genannte Channel ID (CID) zugeordnet (s. Abbildung 2-11). Es gibt verbindungsorientierte und verbindungslose Kanäle. Letztere werden beispielsweise für unidirektionale Kommunikation wie das Senden einer Broadcast-Nachricht verwendet und haben die CID 2.

Neben den Kanälen für den Datenaustausch gibt es in diesem Kanalprinzip auch noch die so genannten Signalisierungskanäle mit der CID 1. Pro Verbindung kann es immer nur einen Signalisierungskanal geben.

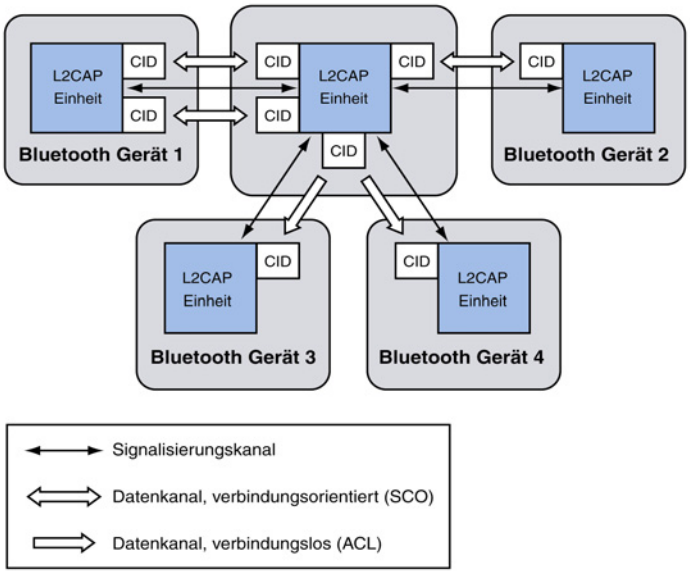


Abbildung 2-11: L2CAP-Entities und Kanäle[38]

Wie bereits erwähnt, empfängt die L2CAP-Ebene Daten von höheren Ebenen. Dies sind die Ebenen RFCOMM, SDP und TCS. In den Paketen der Signalisierungskanäle wird angegeben, welche darüber liegende Ebene das Protokoll nutzt und auf welchen Kanal die Rückantwort geleitet werden soll. Wird eine Verbindung vom anderen Bluetooth-Gerät abgelehnt, wird in den Signalisierungskanälen dafür ein Grund angegeben, wie z. B. Sicherheitsgründe oder mangelnde Kapazitäten.

Nach erfolgreichem L2CAP-Verbindungsaufbau muss der Übertragungskanal noch konfiguriert werden. Dabei wird angegeben

- wie lange ein L2CAP-Paket maximal sein darf. Dies wird auch Maximum Transmission Unit (MTU) genannt. Die MTU kann für die beiden Bluetooth-Geräte unterschiedlich sein.
- wie oft versucht werden soll, ein Paket zu versenden, wenn es nicht erfolgreich übertragen wurde. Standardmäßig geschieht dies so oft, bis die Verbindung abbricht.
- ob das Best-Effort-Prinzip oder das Quality of Service-Prinzip verwendet werden soll.

Nachdem diese Konfiguration erfolgt ist, kann mit der eigentlichen Datenübertragung begonnen werden. Dabei kann es vorkommen, dass die höheren Protokollschichten größere Pakete an die L2CAP-Ebene schicken als Bluetooth an einem Stück verarbeiten kann. Solche Pakete werden in der L2CAP-Ebene zerlegt und paketweise an die unteren Schichten weitergeleitet. Im Baseband kann dann unter Umständen eine weitere Segmentierung der Pakete erfolgen.

[38] Vgl. PDF-Quelle Bluetooth-Spezifikation, S. 262.

2.2.6 Service Discovery Protocol (SDP)

Begibt man sich mit einem Bluetooth-Gerät in eine Umgebung, in der noch andere bluetoothfähige Geräte vorhanden sind, so möchte man unter Umständen deren Dienste nutzen. Die Dienste eines jeden Bluetooth-Gerätes sind in der so genannten Service Discovery-Datenbank eingetragen. Jedes Bluetooth-Gerät hat genau einen so genannten SDP-Server, der auf die Einträge seines Gerätes zugreifen kann. Möchte man also wissen, welche Dienste von einem Gerät angeboten werden, so startet der SDP-Client des eigenen Gerätes eine Anfrage an den SDP-Server des externen Gerätes, der dann die angebotenen Dienste übermittelt. Dadurch kann man noch nicht auf die Dienste zugreifen, man weiß aber, welche Dienste angeboten werden.
Informationen über den jeweiligen Dienst sind in Service-Einträgen gespeichert. Jeder Eintrag ist eine Instanz einer so genannten Service-Klasse, wobei ein Dienst auch zu mehreren dieser Klassen gehören kann. Ein Farbdrucker kann beispielsweise den Service-Klassen Drucker, Postscript-Drucker und Farb-Postscript-Drucker angehören.
Jeder Service-Klasse ist gemäß Vorgaben der Bluetooth SIG eine eindeutige Nummer zugeordnet. So haben Service-Klassen wie „Cordless Telephony" oder „Fax" ihren eigenen so genannten Universally Unique Identifier (UUID).[39]
Es gibt zwei Arten, wie man nach Diensten auf anderen Bluetooth-Geräten suchen kann:

- Suchen nach einer speziellen UUID. So kann man beispielsweise herausfinden, ob das andere Gerät den Drucker-Dienst anbietet und dann versuchen, diesen Dienst zu nutzen.
- Abfragen des SDP-Servers nach allen von dem Bluetooth-Gerät angebotenen Diensten. Dies wird als „browsen" bezeichnet.

2.2.7 RFCOMM (Radio Frequency Oriented Emulation of the Serial COM Ports)

Wie bereits erwähnt, wurde bei der Entwicklung von Bluetooth aus verschiedenen Gründen versucht, vorhandene, ausgereifte Protokolle zu nutzen.
Bluetooth soll bei vielen Geräten der Computer- und Telekommunikationswelt Kabelverbindungen ersetzen. Bei PDAs, Notebooks, Mobiltelefone und Druckern werden oft standardisierte serielle Schnittstellen eingesetzt. RFCOMM emuliert daher genau diese seriellen Schnittstellen und wird deswegen auch „Cable Replacement Protocol" genannt. Es unterstützt theoretisch bis zu 60 gleichzeitige Verbindungen zwischen zwei Bluetooth-Geräten, in der Realität hängt die Anzahl von der Implementierung des jeweiligen Gerätes ab.

[39] Diese Liste der UUIDs wird ständig erweitert. Sie kann kostenlos heruntergeladen werden von Internetquelle UUIDs.

Emulation serieller Ports auf RFCOMM-Ebene

Jedem der 60 möglichen Verbindungen wird ein so genannter Data Link Connection Identifier (DLCI) zugewiesen (s. Abbildung 2-12). Der ersten Schnittstelle wird die Nummer 2, der letzten die Nummer 61 zugewiesen. Die Nummer 0 ist für den Signalisierungskanal reserviert, die Nummer 1 ist ebenfalls anderweitig reserviert.
Die im Stadtinfo-Projekt eingesetzten Geräte kommunizieren auf der RFCOMM-Ebene, wobei der DLCI hier willkürlich auf fünf festgelegt wurde.

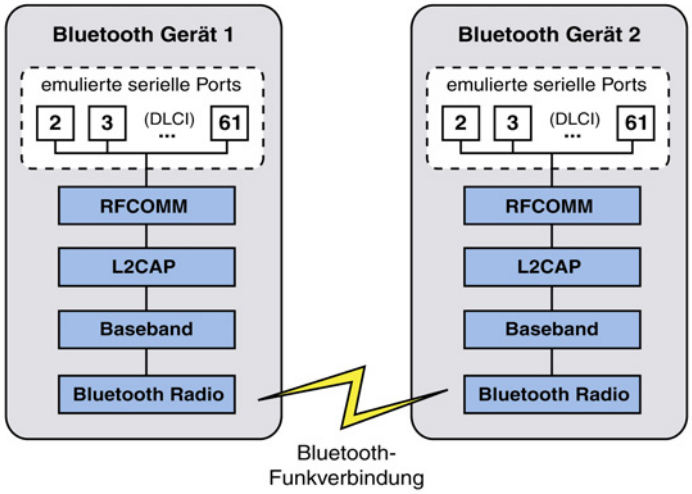


Abbildung 2-12: Emulation serieller Ports auf RFCOMM-Ebene[40]

Damit ein Dienst, der auf RFCOMM aufsetzt, genutzt werden kann, muss dieser auch in der Service Discovery Datenbank eingetragen werden.[41] Hierbei wird mindestens ein Dienstname und eine Kanalnummer für die Übertragung angegeben.

[40] Vgl. PDF-Quelle Bluetooth-Spezifikation, S.401.
[41] Vgl. Kapitel 2.2.5.

2.3 Bluetooth-Profile

Im vorangegangen Kapitel wurden die wichtigsten Bluetooth-Protokolle beschrieben. Nicht jedes Gerät muss jedoch alle Protokolle beherrschen. Ein Drucker beispielsweise benötigt kaum das für Telefonie notwendige TCS-Protokoll. Bestimmte Geräteklassen sollten also bestimmte Protokoll-Pakete implementieren. Die Bluetooth SIG definiert daher so genannte Bluetooth-Profile. Jedes Profil definiert die notwendigen Bluetooth-Protokolle und legt die darauf aufbauenden Regeln für die Kommunikation fest. Durch die Festlegung von Bluetooth-Geräten auf bestimmte Profile wird auch die Entwicklung von Anwendungen wesentlich vereinfacht.

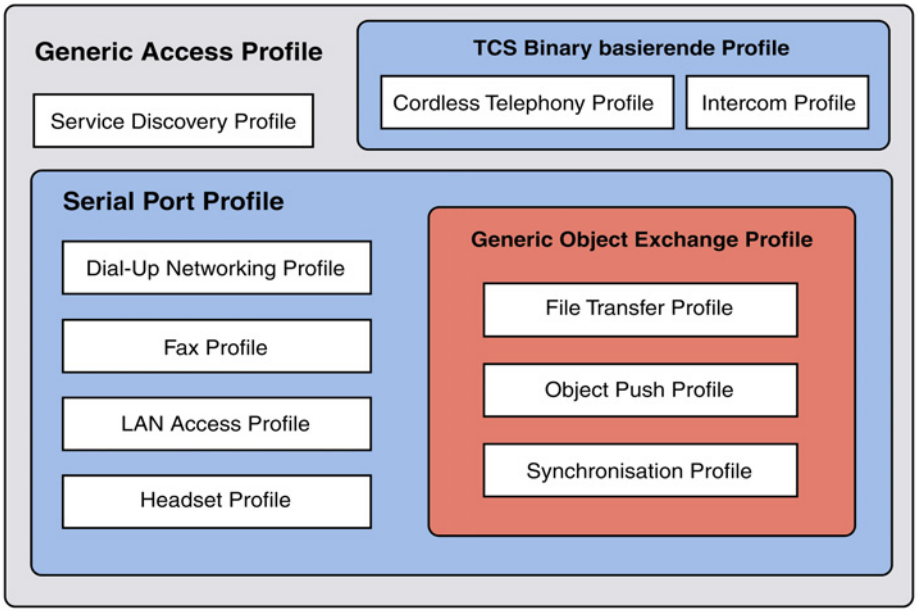


Abbildung 2-13: Bluetooth-Profile

Abbildung 2-13 gibt einen Überblick über die wichtigsten Bluetooth-Profile:[42]

- Generic Access Profile (GAP):
  Dieses Profil ist grundlegend für alle anderen Profile, hier wird die Erkennung von Bluetooth-Geräten durch den Link Manager beschrieben.
- Service Discovery Profile (SDP):
  Um die Dienste eines anderen Gerätes herauszufinden, müssen verschiedene Abläufe definiert werden. Diese sind im SDP festgelegt.
- Cordless Telephony Profile (CTP):
  Soll ein Mobiltelefon mit einer Basisstation kommunizieren, die beispielsweise dann einen Festnetz-Telefonanschluß anspricht, wird dieses Profil implementiert.
- Intercom Profile (INTP):
  Ähnlich wie CTP, jedoch besteht hier eine direkte Verbindung von z. B. zwei Mobiltelefonen. Dieser Modus wird auch „Walkie-Talkie-Modus" genannt.

[42] Vgl. Merkle (2002), S. 80-82.

- Serial Port Profile (SPP):
  Für die Emulation einer seriellen Kabelverbindung wird das SPP verwendet. Das Kabel wird durch eine virtuelle serielle Schnittstelle ersetzt. Dieses Profil wurde beim Stadtinfo-Projekt verwendet.
- Headset Profile (HSP):
  Das HSP definiert die notwendigen Abläufe zwischen einem Audio-Headset und beispielsweise einem Mobiltelefon.
- Dial-Up Networking Profile (DUNP):
  Dieses Profil stellt dar, welche Protokolle für einen Netzzugang eines Computers per Mobiltelefon implementiert werden müssen. Dazu gehört auch die Möglichkeit, als Server zu fungieren und angewählt werden zu können.
- Fax Profile (FAXP):
  Das Fax Profile definiert die notwendigen Abläufe und Protokolle für die Nutzung drahtloser Faxgeräte.
- LAN Access Profile (LAP):
  Dieses Profil dient dem Zugang zu einem LAN über das PPP-Protokoll.
- General Object Exchange Profile (GOEP):
  Das GOEP enthält die wesentlichen Protokolle für den generellen Austausch von Objekten.
- Object Push Profile (OPP):
  Für die Übertragung von Objekten von einem Client an einen Server wurde das OPP definiert. Ein Objekt könnte hierbei z. B. eine elektronische Visitenkarte sein (vCard). Diese wird in die Inbox des Servers gelegt.
- File Transfer Profile (FTP):
  Dieses Profil wird zur Übertragung von Dateien genutzt oder um die Verzeichnisstruktur eines anderen Bluetooth-Gerätes zu durchsuchen.
- Synchronisation Profile (SP):
  Sollen beispielsweise Daten wie Adressbuch oder Kalender auf verschiedenen Geräten synchronisiert werden, kommt dieses Profil zum Einsatz.

Die Definition von Profilen wird stetig erweitert. So wird es auch möglich sein, für zukünftig entwickelte Geräte passende Profile zu definieren.[43] Auf der Internetseite der Bluetooth SIG finden sich beispielsweise auch die Definition für ein Profil für ISDN-Dienste über Bluetooth (Common ISDN Access Profile) oder das Human Interface Device Profile für z. B. Joysticks, Thermometer oder Fernbedienungen.

[43] Vgl. Internetquelle Bluetooth-Spezifikation.

3 Symbian OS

„OS" steht für „Operating System", also „Betriebssystem". Ein Betriebssystem dient der Steuerung und Verwaltung der Komponenten eines Computersystems.[44]
Im Hinblick auf die Entwicklung des Stadtinformationsdienstes sollen in dieser Diplomarbeit die beiden Betriebssysteme Symbian OS und HyNetOS in ihren Grundlagen sowie wesentlichen Konzepten beschrieben werden. Dieses Kapitel gibt einen Einblick in das in erster Linie für Mobiltelefone entwickelte Betriebssystem Symbian OS.

3.1 Einführung

Die Ursprünge des Betriebssystem Symbian OS sind auf das von der Firma Psion Computers entwickelte Betriebssystem SIBO zurückzuführen („Sixteen Bit Organizer" oder „Single-Board" – je nach hardware- oder softwarenaher Betrachtung).
Psion hatte sich zum Ziel gesetzt, die Leistungsfähigkeit eines Desktop PCs mit der Mobilität eines Organizers zu verbinden und brachte im Jahre 1990 erstmals ein auf SIBO basierendes Gerät auf den Markt. In den nächsten Jahren folgten weitere Geräte, 1991 beispielsweise die „Series 3", dessen Display bereits die halbe VGA-Auflösung erreichte. Alle Geräte waren sehr erfolgreich, unter anderem da sie sehr stromsparend waren, leistungsfähige Anwendungen beinhalteten und einfach mit anderen Geräten wie z. B. PCs kommunizieren konnten.
Das Betriebssystem SIBO hatte jedoch einige Einschränkungen:

- Es war ein 16-Bit-Betriebssystem, konnte also nur Zahlen bis 65.536 verwalten.
- Es unterstützte weder Maus noch Stift als Eingabemedium.
- SIBO war in C geschrieben. Grafische Benutzeroberflächen und komplexe Anwendungen lassen sich jedoch wesentlich besser mit moderneren, objektorientierten Sprachen programmieren.

1994 entschied sich Psion daher, ein neues Betriebssystem zu entwickeln, welches 1997 unter dem Namen „EPOC" (für „Epoche") erstmals auf Psions „5mx" eingesetzt wurde.
Während der Weiterentwicklung von EPOC suchte Psion nach Wegen, das Betriebssystem auf möglichst vielen interessanten Plattformen einzusetzen und entdeckte den Markt der Mobiltelefone. Die Hersteller waren hier schon seit längerem auf der Suche nach einem leistungsfähigen Betriebssystem für die nächste Generation ihrer Geräte.

[44] Vgl. Balzert (1999), S. 17.

Psion gründete daher 1998 die unabhängige Firma „Symbian". Symbian sollte die Weiterentwicklung von EPOC übernehmen, das nun in Symbian OS umbenannt wurde.[45]
An Symbian beteiligten sich außerdem die vier größten Mobiltelefonhersteller Nokia, Ericsson, Motorola und Panasonic. Mittlerweile besteht die Firma aus einer Reihe weiterer Teilhaberfirmen und einer Vielzahl an Lizenznehmern.
Doch auch der Software-Konzern Microsoft will auf dem Markt der Mobiltelefon-Software Fuß fassen. Sein Konkurrenzprodukt „Windows Mobile OS" scheint zwar momentan noch nicht ausgereift zu sein[46], doch aufgrund der Macht und Finanzkraft von Microsoft bleibt abzuwarten, welches der beiden Betriebssysteme sich durchsetzen wird und ob eine Koexistenz möglich ist.
Vor diesem Hintergrund gewinnt es an Bedeutung, dass Symbian seit der Übernahme der Psion-Anteile durch Nokia im Februar 2004[47] mittlerweile fest in der Hand des weltweit größten Hersteller für Mobiltelefone[48] ist.

Die Flexibilität der Benutzeroberfläche ist eines der Schlüsselkonzepte von Symbian. Symbian teilt die Zielgeräte seiner Lizenzteilnehmer dahingehend in drei Familien ein:[49]

- „Pearl": Mobiltelefone mit der üblichen numerischen Mobiltelefon-Tastatur.
- „Crystal": Mobiltelefone mit einer kompletten Tastatur als hauptsächliches Eingabemedium.
- „Quartz": Mobiltelefone mit Touchscreen und Zeichenstift als mögliche Eingabemedien.

Gemeinsam ist allen Geräten ein grafikfähiges Display, zahlreiche leistungsfähige Anwendungen und die Erweiterbarkeit durch die Installation neuer Applikationen.[50]

Übersicht Symbian-Familien

			
Familie	**"Pearl"**	**"Crystal"**	**"Quartz"**
Orientierung	Sprache	Information	Information
Bedienung	Ein-Hand-Bedienung	Tastatur	Touchscreen / Stift
Abbildung	Nokia 6600	Nokia 9210	SonyEricsson P800

Abbildung 3-1: Übersicht Symbian-Familien

[45] Vgl. Tasker (2000), S. 9-33.
[46] Vgl. Internetquelle Windows Mobile OS.
[47] Vgl. Internetquelle Symbian-Anteilübernahme.
[48] Vgl. Internetquelle Mobiltelefon-Hersteller und Internetquelle Symbian-Teilhaber.
[49] Vgl. Internetquelle Symbian-Familen.
[50] Unter Internetquelle Symbian-Mobiltelefone finden sich alle Symbian-Mobiltelefone, die momentan oder bald im Handel verfügbar sind.

3.2 System-Architektur

Symbian wollte keine Kompromisse eingehen, die bei der Anpassung eines bereits bestehenden PC-Betriebssystems entstanden wären, sondern entwickelte basierend auf EPOC ein komplett neues Betriebssystem.
Im Folgenden wird ein kurzer Überblick über Hardware und Software-Architektur von Symbian-Systemen gegeben.

3.2.1 Hardware

Die Grundbestandteile der Hardware bei Symbian-Mobiltelefonen sind überall gleich:[51]

- Ein Prozessor (CPU):
  32 Bit-Prozessor, Little-Endian, multitaskingfähig mit integrierter MMU (Memory Management Unit).[52]
- System ROM-Speicher:
  Der ROM enthält das Betriebssystem und alle eingebaute Software. Programme werden direkt im ROM ausgeführt und sind schreibgeschützt.[53]
- System RAM-Speicher:
  Der RAM wird zum einen vom Kernel und aktiven Programmen genutzt, zum anderen als „Festplattenspeicher".[54]
- Ein- und Ausgabeschnittstellen wie Zeichenstift, Tastatur, Compact-Flash-Karten-Slot, Bluetooth, Infrarot-Port, etc.
- Stromquellen wie Akku, Batterien oder ein externer Stromanschluss.

Die Hardware-Ressourcen bei einem Mobiltelefon sind jedoch begrenzt:
Die CPU besteht aus einem typischerweise preisgünstigen, stromsparenden und auch eher langsam getakteten Chip. Symbian OS in Version 7.0 unterstützt z. B. die CPU-Architekturen ARMv4(T) und ARMv5T(J).[55] In einem Mobiltelefon ist typischerweise auch keine Festplatte integriert. Es verfügt daher über nur recht wenig Speicher, welcher jedoch durch Speicherkarten erweiterbar ist.

3.2.2 Software-Architektur

Die Firma Symbian konzentrierte sich von Anfang an auf den Markt der Mobiltelefone.[56] Um der Dynamik dieses noch jungen Massenmarktes Rechnung zu tragen, entwickelte Symbian ein sehr modular aufgebautes Betriebssystem. So können beispielsweise neue Übertragungstechnologien relativ einfach integriert werden. Außerdem sind bei Symbian OS Kern-Betriebssystem und Benutzeroberfläche streng getrennt, so dass Hersteller ihre Produkte durch eine eigenes Look-and-Feel von anderen Anbietern differenzieren können.

[51] Vgl. Tasker (2000), S.82.
[52] Vgl. Internetquelle Symbian-Telefon-Herstellung.
[53] Vgl. Kapitel 3.3.5.
[54] Vgl. Kapitel 3.3.5.
[55] Vgl. Internetquelle Symbian-Telefon-Herstellung.
[56] Vgl. Internetquelle Symbian-White Papers.

Modulare Software-Architektur

Abbildung 3-2 zeigt den strukturellen Aufbau der Symbian OS Version 7.0[57], die auch im SonyEricsson P800 eingesetzt wurde, für das der Stadtinfo-Prototyp programmiert wurde.

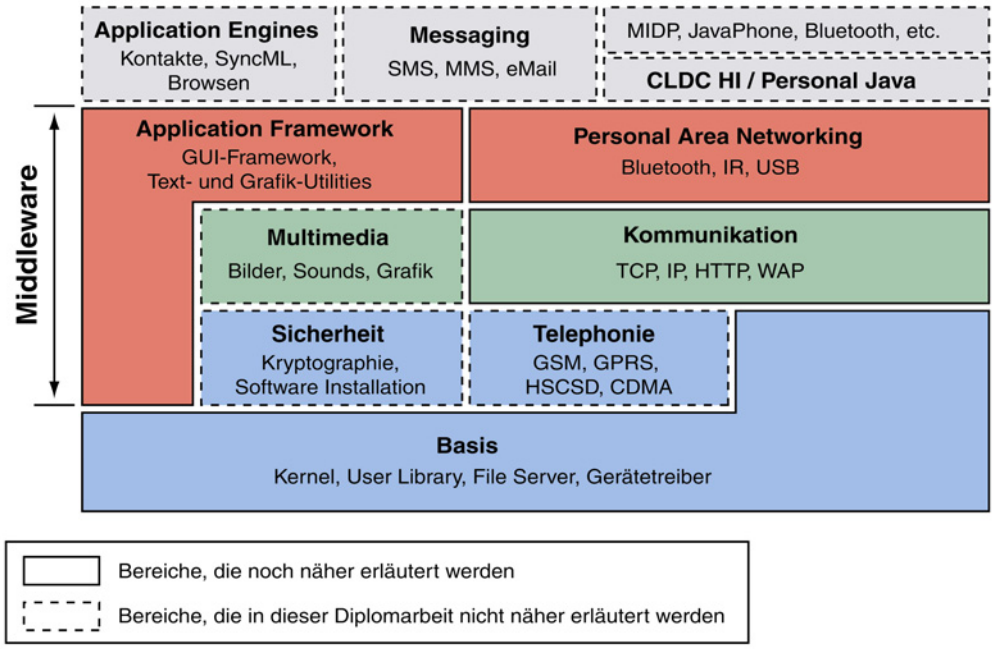


Abbildung 3-2: Software-Architektur von Symbian OS Version 7.0

Die Basis beinhaltet die absolut notwendigen Programme des Systems. Diese sollen die Robustheit, Performanz und effizientes Power Management des Systems sicherstellen.

Darüber liegt die Middleware für verschiedenste Systemdienste. Die Programme der Middleware sind meistens in Form eines Servers implementiert (z. B. der Windows Server). Das Application Framework Sub-System ermöglicht es den Lizenznehmern und Partnern von Symbian, ihre eigene Benutzeroberfläche zu entwerfen. Es bildet die Grundlage für die Programmierung neuer Anwendungen mit grafischer Oberfläche, wie dies auch die Stadtinfo-Anwendung ist.[58] Neue Protokolle beispielsweise können im Kommunikationsmodul integriert werden.

Über der Middleware befinden sich Application-Engines, das Messaging-Subsystem und Java-verwandte Komponenten. Application-Engines wie z. B. die Kontakte-Engine sind wie die Middleware meist als Server implementiert. Sie verwalten aber eher Anwendungsdaten und –dienste, keine systemorientierten Daten und Dienste.

In dieser Diplomarbeit sollen mehrere Bereiche näher betrachtet werden:
Die Basis mit ihren System-Grundlagen wird im folgenden Kapitel näher beschrieben. Im Hinblick auf den Stadtinfo-Prototypen wird später das Application Framework erläutert und auf die Kommunikationsarchitektur mit Personal Area Networking, insbesondere bezüglich Bluetooth, näher eingegangen werden.

[57] Vgl. PDF-Quelle Symbian-OS Version 7.0, S. 5.
[58] Vgl. Kapitel 3.4.5.

3.3 System-Grundlagen

Symbian OS ist ein neues und spezialisiertes Betriebssystem. Wie jedes Betriebssystem besitzt es bestimmte grundlegende Konzepte für das Prozess-Management, die Speicherverwaltung oder die Kommunikation der Komponenten. Dieses Kapitel gibt Einblicke in diejenigen Grundlagen von Symbian OS, die für das Verständnis der folgenden Kapitel notwendig erscheinen.

3.3.1 Komponenten und Grenzen

Abbildung 3-2 zeigte den modularen Aufbau von Symbian OS und die Systemabhängigkeit der Bereiche untereinander. Hier sollen die Komponententypen des Betriebssystems hinsichtlich der zwischen ihnen bestehenden Grenzen dargestellt werden.

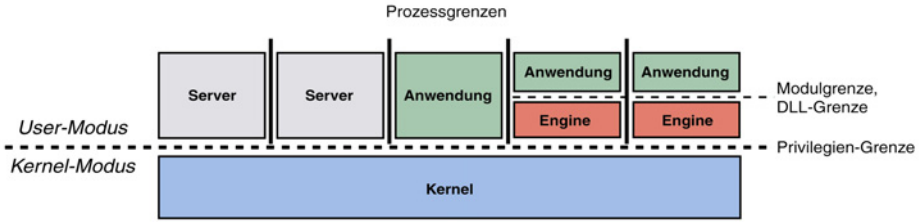


Abbildung 3-3: Software-Komponenten und Grenzen[59]

Wie aus Abbildung 3-3 ersichtlich gibt es in Symbian OS vier unterschiedliche Komponenten (Kernel, Server, Anwendungen und Engines) und drei unterschiedliche Grenzen (Prozess-, Privilegien- und Modul-Grenze).

Grenzen

Die Grenzen sollen folgende Aufgaben erfüllen:

- Abgrenzung der Speicherbereiche der Komponenten untereinander (Prozessgrenzen).
- Schutz der Hardware vor unerlaubtem Zugriff durch Komponenten (Privilegiengrenze).
- Kapselung von Programmteilen zur Übersichtlichkeit (Modulgrenzen).

Es sind unterschiedlich viele Instruktionen notwendig, um diese Grenzen zu überschreiten und dies kostet daher auch unterschiedlich viel Zeit. Die DLL-Grenze zu überschreiten ist sehr kostengünstig, eine Überschreitung der Privilegien-Grenze schon etwas teurer. Prozessgrenzen jedoch sind am teuersten zu überschreiten, da diese die Integrität der einzelnen Anwendungen garantieren und den RAM-Speicher voneinander isolieren sollen.

Komponenten

Kernel

Der Kernel ermöglicht und kontrolliert den Zugriff anderer Softwarekomponenten auf die Hardware-Ressourcen wie RAM, Ein-/Ausgabe-Geräte usw.
Er selbst besitzt Privilegienrechte, um auf die Hardware zuzugreifen. Diese Privilegienrechte müssen von der CPU unterstützt werden. Die CPU führt also bestimmte privilegierte Instruktionen nur für den Kernel im so genannten Kernel-Modus aus. Alle anderen Programme laufen im sogenannten User-Modus und können nur über die Kernel-API auf Hardwareressourcen zugreifen.[60]

[59] Vgl. Tasker (2000), S. 84.
[60] Vgl. Kapitel 3.3.2.

Anwendung (Bsp. Texteditor)

Eine Anwendung ist ein Programm mit Benutzeroberfläche.
Jede Anwendung läuft in einem eigenen Prozess mit eigenem virtuellen Adressraum. So kann eine Anwendung keine Daten anderer Anwendungen aus Versehen oder absichtlich verändern.

Server (Bsp. Font Server)

Ein Server ist ein Programm ohne Benutzeroberfläche und verwaltet eine oder mehrere Ressourcen. Ein Server besitzt eine API, über die seine Clients auf seine Dienste zugreifen können. Clients können andere Anwendungen oder auch andere Server sein. Der Font Server beispielsweise ermöglicht einem Texteditor den Zugriff auf die System-Schriftarten.
Die Client-/Server-Architektur wird in Symbian äußerst häufig eingesetzt, um Dienste zur Verfügung zu stellen, die bei anderen Betriebssystemen vom Kernel oder Gerätetreibern angeboten werden.

Engine (Bsp. Mp3-Engine)

Oft kann Software in grafische Oberfläche und Engine aufgeteilt werden. Die Engine interagiert nicht direkt mit dem Nutzer. Ist eine Anwendung groß genug, lohnt es sich, Software-Funktionen zu kapseln und als Module oder DLLs auszulagern.
Ein Beispiel hierfür ist ein Audio-Player, der beispielsweise eine Mp3-Datei in seiner Engine dekodiert und unkomprimiert als Audiostream an die Anwendung zurückgibt.

3.3.2 Der Kernel

Der Kernel eines Betriebssystems ist das grundlegendste und wichtigste Programm und soll die Stabilität und Performanz des Systems garantieren. Auf den Kernel von Symbian OS soll daher hier näher eingegangen werden.
Der Kernel in Symbian wurde speziell auf die Anforderungen eines Betriebssystems für ein Mobiltelefon angepasst. Um ihn möglichst klein, effektiv und flexibel gegenüber der Peripherie zu halten ist er in Form eines Mikrokernels realisiert, das heißt nur die allerwichtigsten Funktionen sind im Kernel implementiert:

- Management von Prozessen und Threads (Implementierung der Warteschlangen-Politik)[61]
- Interprozesskommunikation und Client-/Server-Architektur[62]
- Speicherverwaltung[63]
- Eventbasiertes Multitasking mit Active Objects[64]
- Error Handling und Cleanup Framework[65]
- Deskriptoren[66]
- Power Management
- Gerätetreiber

Der Kernel in Symbian läuft als einziges Programm im Privilegien-Modus und hat direkten Zugriff auf die Hardware. Ihm kommt daher auch die Aufgabe eines effektiven Power Managements zu. Die CPU beispielsweise wird vom Kernel abgeschaltet, sobald sich alle Threads im Wartezustand befinden, das Display wird sobald wie möglich gedimmt oder ausgeschaltet.

[61] Vgl. Kapitel 3.3.3.
[62] Vgl. Kapitel 3.3.4.
[63] Vgl. Kapitel 3.3.5.
[64] Vgl. Kapitel 3.4.1.
[65] Vgl. Kapitel 3.4.2.
[66] Vgl. Kapitel 3.4.3.

Der Kernel beinhaltet weiterhin die wichtigsten Gerätetreiber, kann aber auch durch DLLs erweitert werden, die dynamisch mit dem Kernel verlinkt werden.[67]
Der Kernel in Symbian OS ist recht klein: In Version 7.0 ist die Kerndatei beispielsweise nur 200 KB groß und unterstützt doch ein präemptives Multitasking-Betriebssystem.[68]

User Library

Eine wichtige Aufgabe kommt der User Library namens „`EUser.lib`" zu: Sie bildet die Schnittstelle zwischen dem Kernel und der restlichen Software und ist somit die unterste Ebene im User-Modus.

3.3.3 Prozesse und Threads in Symbian OS

Der Prozess ist ein Schlüsselkonzept jedes Betriebssystems. Im Prinzip ist ein Prozess ein Programm in Ausführung.[69]
Jedem Prozess in Symbian wird sein eigener virtueller Adressraum zugeordnet: eine Liste von Speicherzellen, die er nutzen darf. Die Übersetzung der virtuellen Adressen eines Prozesses in die physischen Adressen in ROM und RAM übernimmt eine spezielle Einheit des Prozessors, die sogenannte MMU. Sie führt hierzu eigene Tabellen.

Oft jedoch macht es Sinn, mehrere Aufgaben innerhalb eines Prozesses parallel zu bearbeiten.
Jeder Prozess kann daher mehrere Threads (Thread = leichtgewichtiger Prozess) haben, die unabhängig voneinander ausgeführt werden, aber auf den gleichen Adressraum zugreifen können.
Bei Betriebssystemen gibt es grundsätzlich zwei Möglichkeiten, Threads zu verwalten:[70]

- Threads werden vom Prozess verwaltet und der Kernel wechselt nur zwischen den Prozessen.
- Threads werden vom Kernel verwaltet. Dies ist flexibler, es ist aber teurer, Threads zu erzeugen und zu zerstören, da ein Systemaufruf notwendig ist.

Bei Symbian OS wurde die zweite Möglichkeit gewählt:
Threads werden vom Kernel verwaltet: gestartet, angehalten, wieder geweckt, beendet, etc.
Dies geschieht mit präemptivem Multitasking, d.h. Threads mit höherer Priorität können andere Threads unterbrechen. Diese werden dann in eine Warteschlange gestellt. Threads mit gleich hoher Priorität werden mit dem Round-Robin-Verfahren behandelt. Beim Round-Robin-Verfahren darf ein Thread die CPU für eine festgelegte Zeit nutzen[71], danach wird die CPU dem nächsten Thread zugeteilt usw. Auf einem ARM-Prozessor geschieht diese abwechselnde Zuteilung der CPU zu einem Thread z. B. mit 64 Hz.[72]
In Symbian OS verwendet jede Anwendung typischerweise ihren eigenen Prozess und nur einen Thread. Anwendungen oder Server sind daher eher als Thread, denn als Prozess, anzusehen. Ein Wechsel zwischen unterschiedlichen Prozessen ist sehr teuer, da dabei unter anderem die MMU aktualisiert werden muss oder Hardware-Speicher neu gefüllt werden müssen. In Symbian OS sind

[67] Vgl. PDF-Quelle Symbian-OS Version 7.0, S.18.
[68] Vgl. Internetquelle Symbian-Mobiltelefon-Herstellung.
[69] Vgl. Tanenbaum (2002), S. 48
[70] Vgl. Tanenbaum (2002), S. 106-110.
[71] Vgl. Tanenbaum (2002), S. 159-160.
[72] Vgl. Tasker (2000), S. 91.

daher manche Server, die eng zusammen arbeiten, in einem einzigen Prozess zusammengefasst. So laufen beispielsweise alle wichtigen Kommunikations-Server (Seriell-, Socket-, Telefon-Server, etc.) in einem gemeinsamen Prozess ab.

In anspruchsvolleren Anwendungen müssen oft Hintergrundaufgaben erledigt werden. Damit Anwendungen und Server, die in einem einzigen Thread ablaufen, dennoch effektiv arbeiten können, hat Symbian auf ein gutes Event Handling-System Wert gelegt, das auf Active Objects aufgebaut ist.[73]

3.3.4 Server und Clients

Die meisten Programme in Symbian OS nutzen die Client-/Server-Architektur. Der Server verwaltet eine oder mehrere Ressourcen und ein oder mehrerer Clients nutzen den Server, um auf diese Ressourcen zuzugreifen.
Server und Client laufen typischerweise in einem unterschiedlichen Prozess ab. Die Kommunikation zwischen ihnen basiert auf einem vom Kernel unterstützten Message-Passing-System. So können über die Prozessgrenze hinweg Nachrichten und Daten ausgetauscht werden.[74]

Die wichtigsten Server in Symbian OS sind der File Server und der Windows Server. Der File Server ist für Zugriffe auf das Dateisystem zuständig und ist in der Software-Architektur daher als grundlegender Bestandteil in der Basis angesiedelt worden. Der Windows Server behandelt die Eingaben des Benutzers und die Darstellung auf dem Screen.

3.3.5 Speicher

Der Systemspeicher RAM und ROM wird von der MMU organisiert. Ein Mobiltelefon hat typischerweise nur sehr begrenzten Speicher zur Verfügung.

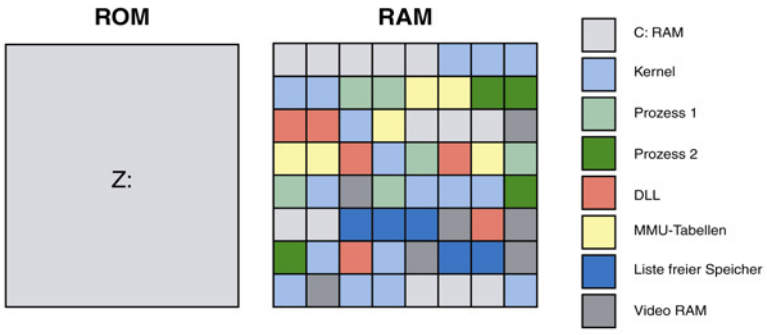


Abbildung 3-4: Speicheraufteilung unter Symbian OS

[73] Vgl. Tasker (2000), S. 82-86.
[74] Genaue Erläuterungen zum Message-Passing finden sich unter Tasker (2000), S. 599-624.

RAM/ROM

- Der System-ROM enthält neben dem Betriebssystem alle eingebaute Middleware und Anwendungen. Die Verwaltung ist recht einfach, da er nur zum Lesen genutzt werden kann. Die Daten werden über feste Adressen angesprochen und können direkt im ROM ausgeführt werden, ohne den File Server bemühen zu müssen. Programme müssen so nicht wie etwa bei einem PC beim Bootvorgang von der Festplatte in den RAM geladen werden. Der ROM-Speicher wird als Laufwerk „Z:" angesprochen und liegt in der Größenordnung von etwa 12 MB.
- Der System-RAM-Speicher wird zum einen von Kernel und aktiven Programmen als Speicher genutzt, zum anderen als „Festplatte" zur Datenspeicherung unter dem Laufwerksbuchstaben „C:". Er liegt in der Größenordnung von 16 MB. Der RAM wird von der MMU in 4 KB große Bereiche aufgeteilt, die mit den unterschiedlichen Daten gefüllt werden (s. Abbildung 3-4).

Speicher für Prozesse und Threads

Jeder Prozess, jeder Thread benötigt Speicher. Es gelten symbianspezifische Limitierungen, deren Beachtung bei der Programmierung unerlässlich sind. Hier wird kurz auf die Speicherbelegung eingegangen, die beim Starten eines Programms abläuft.
Sobald eine „`.exe`"[75] gestartet wird, wird ein neuer Prozess mit einem einzigen Haupt-Thread erzeugt. Der Adressraum dieses Prozesses enthält Bereiche für:

- Systembezogenen Speicher (System-ROM und shared DLLs im RAM).
- Prozessbezogenen Speicher (z. B. das `.exe`-Image des Programms und dessen beschreibbare statische Daten).
- Speicher für jeden Thread (in Symbian enthält wie erwähnt jeder Prozess typischerweise nur einen Thread).

Der Speicher für jeden Thread teilt sich wiederum in zwei Arten auf:

- Stack:
  Für den Stack wird nur einmal am Anfang Speicher belegt von typischerweise nur 12KB. Er kann nicht erweitert werden und ist gedacht für temporär erzeugte, lokale Variablen.
- Heap:
  Der Heap ist dynamisch und kann bis auf typischerweise 2 MB anwachsen. Hier werden z. B. alle Variablen angelegt, die mit dem Befehl „`new`" erzeugt werden. Da jeder Thread seinen eigenen Heap besitzt, ist die Speicherbelegung und –freigabe sehr effektiv. Wird beispielsweise eine neue Variable angelegt, ohne dass der Speicherbereich des Heap erweitert werden muss, sind hierzu nur wenige Instruktionen nötig. Keine Privilegien-Grenze muss überschritten werden und keine Synchronisation mit der Speicherbelegung von anderen Threads ist notwendig.[76]

Speichererweiterung

Für große Dateien des Nutzers wie z. B. Mp3-Dateien oder Videodateien steht in der Regel über eine Erweiterungskarte zusätzlicher Speicher zur Verfügung. Beim SonyEricsson P800 beispielsweise wird standardmäßig eine 16 MB große Compact-Flash-Karte mitgeliefert. Die Dateien auf der Erweiterungskarte werden dann über den File Server zugängig gemacht.

[75] Vgl. Kapitel 3.4.4.
[76] Vgl. Tasker (2000), S. 93.

3.4 Programmierung unter Symbian OS

Ein Mobiltelefon stellt ganz besondere Anforderungen an sein Betriebssystem, die die Konzeption des Betriebssystems Symbian OS beeinflussten:

- Stabilität:
  Der Nutzer eines Mobiltelefons ist in der Regel weniger tolerant bzgl. Systemabstürzen (z. B. während eines Telefonates) als ein PC-Nutzer, der Absturz und Neustart seines Rechners eher gewohnt ist.
- Effektiver Einsatz der begrenzten Ressourcen:
  Mobilgeräte haben recht wenig Speicher, eine eher geringe Prozessorleistung und nur begrenzte Energie zur Verfügung.
- Vorausbestimmbare und schnelle Antwortzeiten:
  Mobilfunknetze erfordern festgelegte und möglichst schnelle Antwortzeiten, ähnlich denen eines Echtzeitsystems. Besonders aber auch der Nutzer erwartet, dass das System auf seine Eingaben sofort reagiert.

Wie bei der Implementierung des Kernels darauf geachtet wurde, diese Anforderungen zu erfüllen, wurde in den vorigen Kapiteln beschrieben. Doch auch in der Programmierung sollten diese Anforderungen beachtet und die Anwendungen möglichst stabil und ressourcenschonend programmiert werden. Weiterhin sollte man stets daran denken, dass die Stromzufuhr jederzeit unterbrochen werden kann wie beispielsweise bei einem leeren Akku. Der Datenverlust ist in einem solchen Fall möglichst gering zu halten.
Dem Programmierer werden einige Konzepte an die Hand gegeben, um die Programmierarbeit zu erleichtern. In den folgenden beiden Kapiteln soll auf zwei Konzepte näher eingegangen werden, die auch vom Kernel unterstützt werden:

- Active Objects als neues Konzept für eventbasiertes Multitasking
- Error Handling und Cleanup: Symbian sieht besonders für die Fehlerbehandlung von Speicherproblemen bestimmte Konzepte in der Programmierung vor.

Anschließend werden weitere wichtige Programmiergrundlagen in Symbian OS beschrieben, wie z. B. das Application Framework oder der Einsatz von Ressourcen-Dateien.

3.4.1 Event Handling und Active Objects

Die vielleicht fundamentalste Design-Entscheidung bei Symbian war es, das Betriebssystem auf ein effizientes Event Handling hin zu optimieren. Symbian stellt für die Behandlung von Events das Framework der Active Objects zur Verfügung.
Zwei wesentliche Gedanken führten zum Konzept der Active Objects:

- Moderne Betriebssysteme wie Symbian sind multitaskingfähig. Da nur ein Prozessor zur Verfügung steht, müssen die Tasks quasi-parallel bearbeitet werden. Jeder Wechsel zwischen den einzelnen Threads kostet jedoch Zeit. Symbians Implementierung beruht auf präemptivem Multitasking mit prioritätenbasierten Threads, d.h. eine Anfrage von einem Thread mit höherer Priorität kann einen Thread mit einer geringeren Priorität unterbrechen. Der Windows Server kann z. B. eine laufende Anwendung unterbrechen, wenn ein Tastatur-Event behandelt werden muss.

- Früher hatte die Hauptfunktion „`main()`" die Kontrolle über ein Programm, heute hat diese der Nutzer. Viele Betriebssysteme legen sehr viel Wert auf gute Synchronisation von Threads und Prozessen, vernachlässigen aber ein gutes Event Handling. Dies wird aber angesichts der großen Nutzer-Interaktion immer wichtiger, vor allem bei Systemen mit grafischen Oberflächen. Selbst das recht moderne Java nutzt auf fundamentaler Ebene Threads anstatt eines Event Handling Frameworks.[77]

Ideal wäre folglich ein System, in dem innerhalb nur eines Threads auf mehrere mögliche Events schnell reagiert werden kann, wie es in einem eventgesteuerten System verlangt wird. So können Kontextwechsel zwischen Threads vermieden werden. In Symbian OS wird dies durch Active Objects verwirklicht.

Wie funktionieren Active Objects?

Alle Threads in Symbian sind eigentlich Event-Handler mit einem so genannten „Active Scheduler" pro Thread. Dieser hat ein oder mehrere Active Objects. Man kann sich den Active Scheduler auch als Mini-Kernel vorstellen und ein Active Object als Mini-Thread.

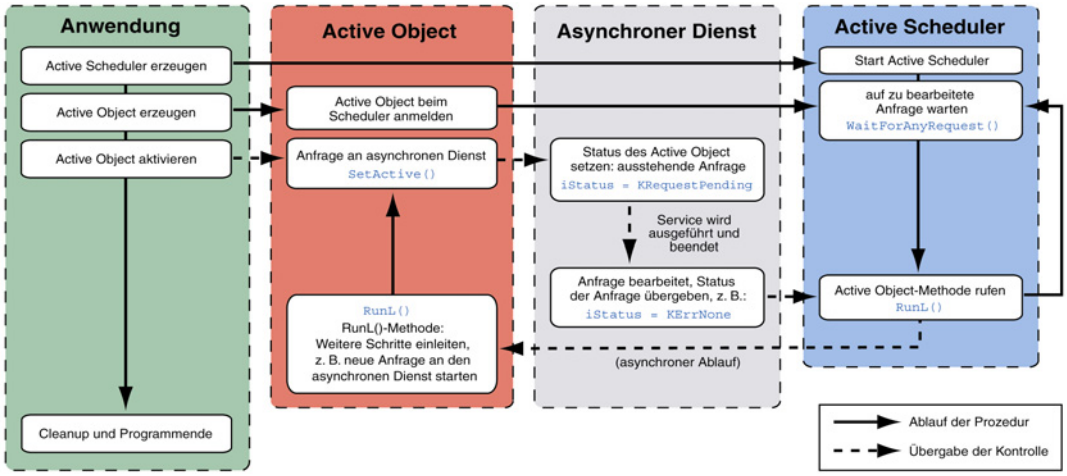


Abbildung 3-5: Ablauf Active Objects[78]

Abbildung 3-5 zeigt den groben Ablauf bei Active Objects:
Die Anwendung erzeugt Active Scheduler und ein oder mehrere Active Objects. Jedes Active Object meldet sich bei Erzeugung beim Active Scheduler an. Dann aktiviert die Anwendung ein Active Object, indem sie eine Methode des Active Objects ruft. Ein Beispiel hierfür wäre ein Objekt, das auf einkommende Daten über Bluetooth warten soll und mit „`IssueRead()`" aktiviert wird. Das Active Object startet nun eine Anfrage an einen asynchronen Dienst, beispielsweise eine Anfrage an den Socket Server mit dem Befehl „`Read()`" und dem anschließenden Befehl „`SetActive()`". Der Status des Active Objects wird daraufhin auf „ausstehende Anfrage" (`iStatus=KRequestPending`) gesetzt. Die Anwendung läuft weiter, wird also nicht durch diese ausstehende Anfrage blockiert. Sobald die Anfrage ausgeführt wurde, wird der Status des Active Objects geändert. Der Active Scheduler ruft daraufhin die `RunL()`-Methode des Active Objects. Hier werden dann weitere Schritte eingeleitet, wie z. B. eine erneute Anfrage an den asynchronen Dienst gestartet.

[77] Vgl. Tasker (2000), S. 96.
[78] Vgl. Internetquelle Symbian-Active Objects.

Active Objects sind wie Threads auch prioritätenbasiert, jedoch innerhalb eines Thread-Kontextes nicht präemptiv. Das heißt, dass ein Active Object während der Abarbeitung der `RunL()`-Methode von keinem anderen Active Object des gleichen Threads unterbrochen werden, auch nicht von einem Active Object mit höherer Priorität.
In Symbian OS wird Multitasking vorzugsweise durch Active Objects verwirklicht und nicht durch Multithreading. Prozesse benötigen daher typischerweise nur einen einzigen Thread.
Durch Active Objects gewinnt man an Bequemlichkeit, da man weiß, dass die `RunL()`-Methode nicht durch andere Active Objects unterbrochen werden kann. Verfahren mit Mutexes, Semaphoren, kritischen Sektionen, etc. um sich gegen Eingriffe anderer Active Objects im eigenen Thread zu schützen, sind daher überflüssig.
Es sind auch keine Systemaufrufe notwendig, um Active Objects zu erzeugen oder zu beenden.

Active Objects werden vom Kernel unterstützt und bieten eine interessante und ressourcenschonende Lösung für ein eventbasiertes Multitasking. Für die Stabilität eines Symbian-Systems muss jedoch auch ein effektives Error Handling eingesetzt werden, vor allem im Hinblick auf den begrenzten Speicher eines Mobiltelefons. Im Folgenden soll hierauf näher eingegangen werden.

3.4.2 Error Handling und Cleanup

Ein Mobiltelefon sollte unter Umständen monatelang ohne Neustart laufen können. Aufgrund der begrenzten Speicherkapazitäten sind daher eine gute Fehlerbehandlung und sauberes Cleanup auf Mobiltelefonen enorm wichtig.
Bei jeder einzelnen Anweisung, die Speicher belegen kann, muss man sich darüber bewusst sein, dass ein „Out-of-Memory"- Fehler entstehen kann. Das Framework von Symbian kann zwar weitaus mehr als nur Out-of-Memory-Fehler behandeln. Da bei anderen Fehlern (Schreib-/ Lesefehler, Eingabefehler, etc.) jedoch ebenfalls oft Speicher neu belegt werden muss, sind Out-of-Memory-Fehler die wohl wichtigsten und häufigsten Fehler in einem Symbian-System.
Die nun folgenden Abschnitte beschäftigen sich eingehend mit der Fehlerbehandlung in Symbian OS und mit den damit verbundenen Vorgehensweisen, die bei der Symbian-Programmierung angewandt werden.[79]
Die Handhabung von Fehlern beim Start eines Programms und den damit verbundenen Speicherbelegungen ist einfach: Das Programm wird beendet.
Eine größere Herausforderung hingegen sind Speicherprobleme, die während der Benutzung einer Anwendung entstehen, z. B. in einem Textverarbeitungsprogramm. Wird hier beispielsweise ein neuer Buchstaben eingegeben, muss unter Umständen neuer Speicher belegt werden. Sollte nicht mehr genügend Speicher zur Verfügung stehen, so muss dieser Fehler abgefangen werden, um Datenverlust und Programmabsturz zu vermeiden.
Symbian OS führte zur Behandlung von Fehlern ein eigenes Verfahren ein, das dem Try-Catch-Verfahren in Java sehr ähnlich ist:[80] Sobald ein Fehler entsteht wird die entsprechende Funktion beendet, was in Symbian OS als „`Leave`" bezeichnet wird. Nun wird die erste Funktion gesucht, die den Fehler mit „`TRAP()`" oder „`TRAPD()`" abfängt. Ein entsprechender Fehlercode wie beispielsweise „`KErrNotFound`" wird mitübergeben.

[79] Vgl. Tasker (2000), S. 156-193.
[80] Anm.: Das C++-native Exception-Handling war zur Zeit der Entwicklung von Symbian mit GCC noch nicht verfügbar und mit MS Visual C++ nicht zuverlässig, vgl. Tasker (2000), S. 173.

Für die korrekte Fehlerbehandlung in Symbian OS sind mehrere Dinge nötig:

1) Objekterzeugung mit dem Zusatz „ELeave":

Damit ein eventuell entstehendes Leave abgefangen werden kann, muss ein Objekt mit dem Zusatz „ELeave" erzeugt werden:

```
CObjectX* x = new (ELeave) CObjectX
```

Symbian wollte dieses Verhalten ursprünglich als Standard-Verhalten bei „new" einführen, verwarf den Gedanken aber aufgrund der daraus resultierenden schlechten Portierbarkeit anderer Programme auf Symbian OS. Jetzt wird bei „new" ohne „ELeave" wie in Standard C++ ein Fehler erzeugt (Exception).
Oft ist es nicht notwendig, den Fehler selbst mit TRAP abzufangen, da das Framework dies übernimmt und das Cleanup durch andere Mittel gewährleistet wird.

2) Einsatz des Cleanup-Stacks:

Wird ein Objekt mit obigem Befehl ohne Fehler angelegt, so wird Speicher auf dem Heap belegt. Die Variable x ist der Zeiger auf diesen Speicherbereich. Sollte nun aber anschließend in der Funktion ein Fehler auftreten und die Funktion mit Leave verlassen werden, so geht dieser Zeiger verloren und der Speicher kann nicht freigegeben werden.
Hier kommt der Cleanup-Stack zum Einsatz. Objekte auf dem Cleanup-Stack werden zerstört, sobald ein Leave entsteht. Der Cleanup-Stack sollte also genutzt werden, um Zeiger auf heap-basierte Objekte zwischenzuspeichern:

```
1: CObjectX* x = new (ELeave) CObjectX;
2: CleanupStack::PushL(x);          // Kopie des Zeigers wird auf dem
                                    // Cleanup-Stack gespeichert.
3: x->DoSomethingL();               // Dies ist die Funktion, die das
                                    // Leave erzeugen könnte.
4: CleanupStack::PopAndDestroy();   // Objekte auf dem Cleanup-Stack
                                    // werden zerstört.
```

Entsteht nun ein Fehler bei der Funktion, die in Zeile drei aufgerufen wird, so können alle Objekte gelöscht und der Speicherbereich korrekt freigegeben werden. Entsteht kein Fehler, so wird x in Zeile vier vom Cleanup-Stack gelöscht und anschließend zerstört.

Trotz allem kann ein Problem entstehen. Und zwar schon in der ersten Zeile, falls im Konstruktor von x ein Leave entsteht. Zur Erklärung: Die Zeile

```
CObjectX* x = new (ELeave) CObjectX;
```

wird in C++ aufgelöst in:

```
1: CObjectX* x;
2: CObjectX* temp = User::AllocL(sizeof(CObjectX)); // Speicher allokieren
3: temp->CObjectX::CObjectX();                      // Konstruktor
4: x=temp;
```

In der dritten Zeile wird der Konstruktor gerufen. Dieser könnte evtl. auch so aussehen:

```
1: CObjectX::CObjectX()
2: {
3:    iY = new (ELeave) CObjectY;      // Speicher für iY allokieren
4: }
```

Entsteht an dieser Stelle ein Fehler, wird das `Leave` erzeugt, ohne dass der Zeiger von `x` vorher auf den Cleanup-Stack gepusht werden konnte und der Speicherbereich kann folglich nicht korrekt freigegeben werden.
Für die Programmierung unter Symbian OS wurde daher folgende Regel eingeführt:
Konstruktoren dürfen kein `Leave` erzeugen, dürfen also keine neuen Objekte anlegen oder Funktionen rufen, die ein `Leave` erzeugen könnten. Da aber bei der Erzeugung von Objekten genau dies oft notwendig ist, wurde in Symbian ein Workaround eingeführt: Die Erzeugung von Objekten in zwei Phasen, die so genannte „Two-Phase-Construction".

Two-Phase-Construction

Objekte sollten unter Symbian OS in zwei Phasen erzeugt werden:

```
1: ObjectX* x = new (ELeave) CObjectX;        // Phase 1
2: CleanupStack::PushL(x);
3: x->ConstructL();                           // Phase 2
4: x->DoSomethingL();
5: CleanupStack::Pop();
```

Da der Konstruktor in Zeile 1 nun kein `Leave` mehr erzeugt, kann der Zeiger von `x` in Zeile 2 auf den Cleanup-Stack zwischengespeichert werden.[81] Die in Zeile 3 gerufene `ConstructL()`-Methode enthält nun den Code, den vorher der Konstruktor enthielt:

```
1: CObjectX::ConstructL()
2: {
3:      iY = new (ELeave) CObjectY;    // Speicher für iY allokieren
4: }
```

Sollte nun hier das `Leave` entstehen, so kann der Speicher, der für das `CObjectX` belegt wurde, korrekt freigegeben werden, da eine Kopie des `CObjectX` auf dem Cleanup-Stack vorhanden ist.
So können durch das Two-Phase-Construction-Prinzip nach der eigentlichen Erzeugung des Objektes mit dem Standardkonstruktor in der `ConstructL()`-Methode alle weiteren bei Objekterzeugung notwendigen Schritte durchgeführt werden.

Diese Vorgehensweise der Objekterzeugung in zwei Phasen zieht sich durch die komplette Symbian-Programmierung. Um die Programmierung zu vereinfachen, wird die Two-Phase-Construction jedoch oft in einer neuen Methode namens „`NewL()`" gekapselt. Objekte werden dann folgendermaßen erzeugt:

```
CObjectX* x = CObjectX::NewL();
```

[81] Anm.: Die Anweisung `CleanupStack::PushL()` könnte zwar ebenfalls „`leaven`", dies wird aber abgefangen, indem immer ein freier Bereich auf dem Cleanup-Stack gehalten wird, in dem das Object zunächst abgelegt wird. Dann wird versucht, einen neuen Bereich anzulegen. Sollte hierbei ein Fehler entstehen, wird das Objekt gelöscht.

Die `NewL()`-Methode könnte dann folgendermaßen aussehen:

```
1: CObjectX* CObjectX::NewL()
2: {
3:      ObjectX* self = new (ELeave) CObjectX;
4:      CleanupStack::PushL(self);
5:      self->ConstructL();            // könnte ein "Leave" erzeugen
6:      CleanupStack::Pop();
7:      return self;
8: }
```

Die Aufgabe der `NewL()`-Methode erinnert hier an eine Factory-Methode, eine statische Funktion, die als eine Art Konstruktor wirkt.[82] Es ist allerdings nicht notwendig, die Erzeugung immer in `NewL()` zu kapseln. Dies macht nur Sinn, wenn das Objekt oft verwendet wird. Ansonsten genügt es, den Cleanup-Stack einzusetzen und „`new (ELeave)`", gefolgt von „`ConstructL()`" zu verwenden.

3.4.3 Strings und Deskriptoren

Wie in Java oder C++ gibt es auch bei Symbian OS die üblichen Datentypen wie integer, float, boolean, etc. in ähnlicher Form. Für die Verarbeitung von Strings und Binärdaten gibt es unter Symbian OS hingegen ein neues Konzept.
Strings können auf zwei unterschiedliche Weisen angelegt werden. Zum einen als so genannte „Literals". Diese werden fest in den Binär-Programmcode eingebaut. Auf Literals soll hier nicht genauer eingegangen werden.
Zum anderen können Strings auch als so genannte „Deskriptoren" angelegt werden, die sich in drei Typen unterteilen:[83]

- Buffer-Deskriptoren, die auf dem Stack angelegt werden und sich daher für kleine Strings mit fester Länge oder Zeiger eignen.
- Heap-Deskriptoren, die auf dem Heap angelegt werden. Diese sind geeignet für größere Strings, deren Länge variieren kann.
- Pointer-Deskriptoren, bei denen die Daten getrennt vom Deskriptor abgelegt sind.

Neu ist das Konzept deshalb, weil Deskriptoren für Strings und Binärdaten gleichermaßen eingesetzt werden. Zum Vergleich: Deskriptoren sind komfortabler als die Stringverarbeitung in C, erlauben aber im Gegensatz zu Java die volle Kontrolle über den Speicherverbrauch eines Strings. Durch diese Gleichbehandlung muss sich der Code für Strings und Binärdaten nicht unterschieden. Außerdem können Daten so aus einer Mischung von Strings und Binärdaten bestehen.
Ein Beispiel hierfür ist die Bluetooth Engine im Stadtinfo-Projekt: Einkommende Daten bestehen aus Binärdaten (Audiodaten) oder Text, können aber beide mit Hilfe des gleichen Deskriptors weiterverarbeitet werden.

[82] Vgl. Gamma et al. (1995), S. 107-116.
[83] Vgl. Internetquelle Symbian-Deskriptoren.

Unicode

Der internationale Einsatz von Symbian OS war von Anfang an ein Teilaspekt bei der Konzeption des Betriebssystems. Um alle Sprachen und Zeichensätze zu unterstützen, war auch an die Unicode-Unterstützung gedacht worden. Diese wurde von Anfang an vorgesehen, jedoch erst später implementiert.
Deskriptoren speichern Text und Binärdaten. Symbian führte daher ein „allgemeines" Symbol ein wie z. B. „`HBufC`", sowie eine 8-Bit-Version („`HBufC8`") und eine 16-Bit-Version („`HBufC16`"). Unicode benötigt aufgrund der großen Menge an Zeichen die 16-Bit-Version. Der Nachteil ist hierbei, dass die 16-Bit-Version im RAM für offene Dokumente etwa 30-50% mehr Speicher belegt.[84]
Verwendet man im Code das allgemeine Symbol, so entscheidet das Setzen eines `_UNICODE`-Makros des Compilers, welche Version für Strings verwendet wird.[85] So wird in den aktuellen Symbian-Versionen immer die 16-Bit-Version für Text verwendet, um Unicode zu unterstützen.
Das hat aber auch zur Folge, dass aufgrund der Gleichbehandlung von Strings und Binärdaten explizit die 8-Bit-Version verwendet werden muss, wenn Binärdaten verarbeitet werden sollen.

Die Arbeit mit Deskriptoren stellte sich beim Stadtinfo-Projekt als eher umständlich heraus. Die einkommenden Daten waren oft gemischt aus Strings und Binärdaten, wurden aber grundsätzlich als 8-Bit-Pakete übertragen. Je nach Verwendung musste deswegen am Ende teilweise umkopiert werden, da Klassen zur Textdarstellung in Symbian OS die 8-Bit-Version des Deskriptors nicht annahmen.

Wichtige Prinzipien der Programmierung wurden nun beschrieben. In den folgenden Kapiteln soll erklärt werden, wie eine Symbian-Anwendung erstellt wird.

3.4.4 Ausführbare Programme in Symbian OS

In Symbian OS gibt es grundsätzlich zwei Arten von ausführbaren Programmen:[86]

- Ein „`.exe`" – Programm:Dieses Programm hat nur einen einzigen Einstiegspunkt: „`E32Main()`". Beim Starten einer „`.exe`" wird vom System zunächst ein neuer Prozess mit einem Thread erzeugt. Der Einstiegspunkt wird dann in diesem Thread aufgerufen.
- Eine DLL:
  Dies ist eine Bibliothek an Programmcode mit mehreren möglichen Einstiegspunkten. Das System lädt eine DLL in den Kontext eines existierenden Threads.

DLLs werden noch einmal unterschieden in zwei Typen:

- Eine „Shared Library DLL" kann von mehreren Programmen gleichzeitig genutzt werden. Sie wird zur Laufzeit zusammen mit dem ausführbaren Programm, ein „`.exe`"-Programm oder eine andere DLL, geladen. Diese Dateien enden auf „`.dll`" wie z. B. „`Mp3Decoder.dll`".

[84] Vgl. Tasker (2000), S. 150.
[85] Tasker (2000), S. 148.
[86] Vgl. Tasker (2000), S. 86-87.

- Eine „polymorphische DLL" wird in der Regel explizit von dem Programm geladen, das sie nutzen möchte und hat normalerweise nur einen Einstiegspunkt. Ein Druckertreiber oder Socketprotokolle sind beispielsweise polymorphische DLLs. Diese Dateien enden auf „`.prn`", „`.prt`", etc. wie z. B. „`bt.prt`" (Bluetooth-Protokoll).

Eine Symbian-Anwendung wie beispielsweise die Stadtinfo-Anwendung ist ebenfalls eine polymorphische DLL mit dem Haupteinstiegspunkt namens „`NewApplication()`" und endet auf „`.app`". Wird diese vom Anwender gestartet, so wird eigentlich ein kleines Programm namens „`apprun.exe`" aufgerufen. Diesem werden die Bezeichnung der Anwendung und der Name der Anwendungsdatei als Befehlszeilenparameter übergeben. `Apprun.exe` erzeugt dann wiederum einen neuen Prozess und lädt die Anwendung als polymorphische DLL. Im folgenden Kapitel soll nun auf die Erstellung dieser polymorphischen DLL mit der Endung `.app` eingegangen werden. Diese wird im Folgenden auch als Application-Datei bezeichnet.

3.4.5 Das Application Framework

Das Application Framework unter Symbian OS gibt mindestens vier Klassen vor:

- Application
- Document
- Application UI
- Mindestens einen Application View

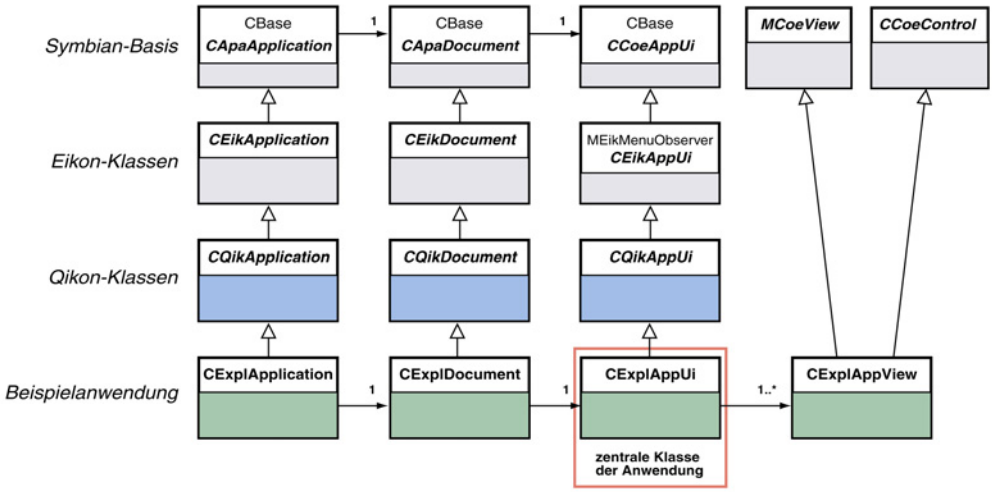


Abbildung 3-6: Application Framework einer Symbian-Qikon-Anwendung

Vertikale Abhängigkeiten

Die Beispielanwendungs-Ebene wird vom Anwendungsprogrammierer erzeugt. Da Symbian OS in unterschiedlichen Versionen existiert, sind auch die Eingabemöglichkeiten unterschiedlich. Abbildung 3-6 zeigt den Zusammenhang der Klassen bei einer so genannten Qikon-Anwendung. Qikon ergänzt die Standard-Architektur (Eikon-Implementierung) um drei spezielle Qikon-Klassen

und unterstützt so neben den Standard-Eingabemöglichkeiten beispielsweise Toolbars und Touchscreen. Die Qikon-Implementierung verleiht der Beispielanwendung das so genannte UIQ-Look-and-Feel,[87] wie es auch auf dem SonyEricsson P800 eingesetzt wird. Alle weiteren Klassen gehören zur Uikon-Kern-API.

Horizontale Abhängigkeiten

Wie in der Abbildung zu sehen ist, werden auf Anwendungsebene in horizontaler Richtung Instanzen der jeweiligen Klassen erzeugt. Im Folgenden soll kurz auf die einzelnen Klassen eingegangen werden.
Die Klasse „Application" hat lediglich zwei Aufgaben. Zum einen legt sie die eindeutige Identifikation (UID) der Anwendung fest. Diese unterscheidet die Anwendung von allen anderen Anwendungen auf dem System. Zum anderen implementiert sie die Methode `NewApplication()`, die bei Programmstart vom Framework gerufen wird. Durch diese Methode wird die Applikation erzeugt und das „Document" angelegt.
Die Klasse „Document" repräsentiert das Datenmodell der Anwendung. Hier werden die Daten gespeichert. Diese Klasse erzeugt in der Methode „`CreateAppUiL()`" die zentrale Klasse der Anwendung, die „Application UI". Diese übernimmt die Steuerung des gesamten Programms und behandelt in der Methode „`HandleCommandL()`" Events wie sie z. B. bei Menü-Eingaben erzeugt werden. Außerdem erzeugt sie einen oder mehrere „Application Views".
Ein „Application View" stellt die Anwendungsdaten auf dem Display dar und ermöglicht dem Benutzer somit die Interaktion mit dem Programm.

Die notwendigen Klassen zur Erstellung einer Application-Datei wurden nun beschrieben. Grundsätzlich können neben dieser polymorphischen DLL noch zwei weitere Dateien auf das Zielgerät übertragen werden:

- Eine Ressourcen-Datei, die die grafischen Elemente und bestimmte Texte enthält und auf „`.rsc`" endet.
- Eine optionale Datei, die unter anderem das Icon der Anwendung enthält. Diese hat die Endung „`.aif`".[88]

Das folgende Kapitel geht nun auf die Verwendung von Ressourcen-Dateien unter Symbian OS ein.

3.4.6 Ressourcen-Dateien

Ressourcen-Dateien spielen bei Symbian OS eine wichtige Rolle. Sie enthalten unter anderem Definitionen für GUI-Elemente wie Dialogfenster, Menüs, etc. oder Texte, die zur Laufzeit von der Anwendung eingesetzt werden. Jeder Ressource, wie z. B. einem Menütext, wird eine eindeutige ID zugewiesen.
Diese Trennung vom eigentlichen Programmcode hat einige Vorteile:

- Ressourcen sind dynamisch, werden also erst zur Laufzeit geladen, sofern sie benötigt werden.

[87] UIQ ist ein anpassbares User Interface für Symbian-Mobiltelefone mit Zeichenstifteingabe, vgl. Internetquelle Symbian-UIQ und PDF-Quelle Symbian Designing For UIQ.
[88] Auf die `.aif`-Datei soll nicht näher eingegangen werden. Sie wurde beim Stadtinfo-Projekt zwar für das Icon verwendet, ist jedoch für das weitere Verständnis unwichtig.

- Bei Veränderungen in der Ressourcen-Datei muss nicht noch einmal der komplette Programmcode neu kompiliert werden.
- Texte der Anwendung können getrennt verändert werden, was für mehrsprachige Anwendungen recht komfortabel ist.

Die Ressourcen-Datei in Symbian wird von einem eigenen Ressourcencompiler erstellt und kann auch getrennt von der eigentlichen Anwendung auf das Zielgerät übertragen werden. Dies mag Vorteile haben was die Portierung Symbians auf neue Plattformen angeht. Für den Programmierer ist dies aber eher umständlich. Auch bei der Stadtinfo-Anwendung wurden alle Texte für den Benutzer in Ressourcen ausgelagert, um eine Finalversion auf einfache Weise an mehrere Sprachen anpassen zu können. Bei der Arbeit mit den Ressourcen stellte sich jedoch heraus, dass oft erst nach Neukompilierung und Übertragung der Ressourcen-Datei samt Application-Datei die Texte wieder richtig angezeigt werden konnten.

3.4.7 Namenskonventionen

In Symbian gibt es sehr viele Namenskonventionen, die, sofern sie eingehalten werden, das Verständnis des Quellcodes für einen geübten Symbian-Programmierer erleichtern. Die Lesbarkeit des Codes hingegen nimmt deutlich ab.
Folgend einige der Konventionen und die damit verbundene Bedeutung für die Programmierung:

- Klassen, die mit „`C`" beginnen sind von der Klasse `CBase` abgeleitet. Objekte dieser Klassen werden auf dem Heap angelegt wie z. B. „`CActive`"-Objekte.
- Klassen, die mit „`M`" beginnen sind Interface-Klassen. Die Funktionen werden in der erbenden Klasse implementiert. (Bsp.: „`MEikMenuObserver`")
- Member-Variablen beginnen mit einem kleinen „`i`" – müssen also im Destruktor gelöscht werden. (Bsp.: „`iMember`")
- Funktionen, die auf „`L`" enden können direkt oder indirekt ein `Leave` erzeugen.[89] Man muss bei Verwendung dieser Funktionen folglich beide Fälle beachten; wie soll reagiert werden, wenn ein `Leave` entsteht und wenn nicht. (Bsp.: „`NewL()`")

[89] Vgl. Kapitel 3.4.2.

3.5 Symbian OS und Bluetooth

Informationen nützten wenig ohne Kommunikation. Symbian OS unterstützt eine Vielzahl an Kommunikationsarten, z. B. Telefonie, SMS, IR, TCP/IP, WAP, Sockets allgemein oder auch Bluetooth.
Hier soll zunächst die Kommunikationsarchitektur unter Symbian OS vorgestellt werden. Darauf aufbauend wird anschließend auf die Übertragungstechnologie Bluetooth eingegangen werden, da beim Stadtinfo-Projekt symbianseitig ausschließlich Bluetooth eingesetzt wurde.

Kommunikationsarchitektur unter Symbian OS

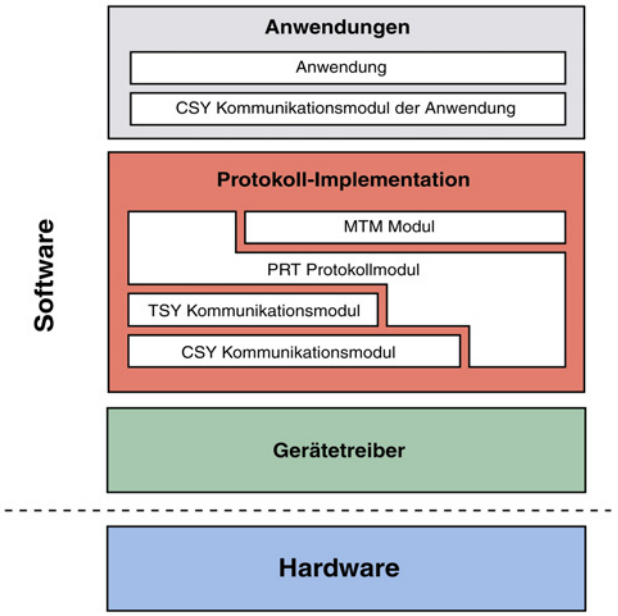


Abbildung 3-7: Kommunikationsarchitektur unter Symbian OS

In der Symbian-Kommunikationsarchitektur gibt es vier Hauptkomponenten:[90]

- Hardware:
  Die Hardware besteht zum Beispiel aus dem Bluetooth-Transmitter, der in einem Mobiltelefon eingebaut ist.
- Gerätetreiber:
  Gerätetreiber bilden die hardwarespezifische Software-Ebene und stellen den oberen Ebenen eine standardisierte Schnittstelle für Hardware-Zugriffe zur Verfügung.
- Protokolle:
  Symbian unterstützt eine Vielzahl an Protokollen. Je nach der Art der Kommunikation werden die in dieser Ebene enthaltenen Module kombiniert:
  - CSY: Kommunikations-Server, unterste Ebene der Protokoll-Implementierung. (Bsp.: das Protokoll für den seriellen Gerätetreiber `ECUART.CSY`)
  - TSY: Telefonie-Server, z. B. `GSM.TSY`.
  - PRT: Zentrale Module für die Protokoll-Implementierung wie z. B. Bluetooth (Bsp. `BT.PRT`, `TCPIP.PRT`).
  - MTM: Message Type Module, spezielle Module für Messaging (SMS, …).

[90] Vgl. Jipping (2002), S. 33.

Die Modularität bringt einige Vorteile mit sich. Während das Bluetooth-Protokoll direkt auf den Geräteteiber zugreift, nutzt die WAP-Implementierung beispielsweise ein Telefon- und ein Kommunikations-Modul. Wird nun eine neue Telefontechnologie entwickelt und soll WAP diese nutzen, so muss lediglich das Telefon-Modul ausgetauscht werden.

- Anwendung:
  Die Anwendung nutzt die Kommunikations-Infrastruktur, indem es dem System sagt, welches Gerät benutzt werden soll. Sie agiert hier als Client gegenüber einem Kommunikations-Server wie z. B. dem seriellen Kommunikations-Server, der dann die richtigen Module lädt.

Bluetooth-Protokolle unter Symbian OS

Grundsätzlich unterstützt Symbian OS alle wichtigen Bluetooth-Protokolle (s. Abbildung 3-8):[91]

- Bluetooth Radio, Baseband, LMP, L2CAP und SDP
- Audio
- RFCOMM und die darauf aufsetzenden anderen Transportprotokolle
- TCS-BIN: Telephonie-Kontroll-Protokolle

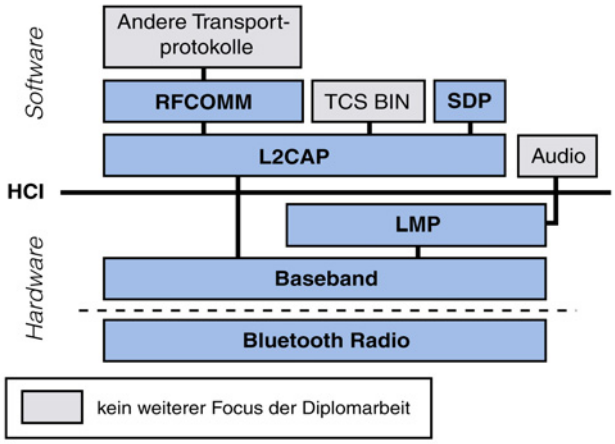


Abbildung 3-8: Bluetooth-Protokolle unter Symbian OS

Die Kommunikation im Stadtinfo-Projekt erfolgte auf RFCOMM-Ebene. RFCOMM simuliert hierbei eine serielle Schnittstelle. Um eine Verbindung auf RFCOMM-Ebene aufbauen zu können, sind mehrere Schritte notwendig, die im Kapitel 6.1.1 genauer beschrieben werden. Zunächst jedoch soll auf die Verwendung von Bluetooth unter Symbian OS allgemein eingegangen werden.

[91] Vgl. Jipping (2002), S. 236.

Verwendung von Bluetooth unter Symbian OS

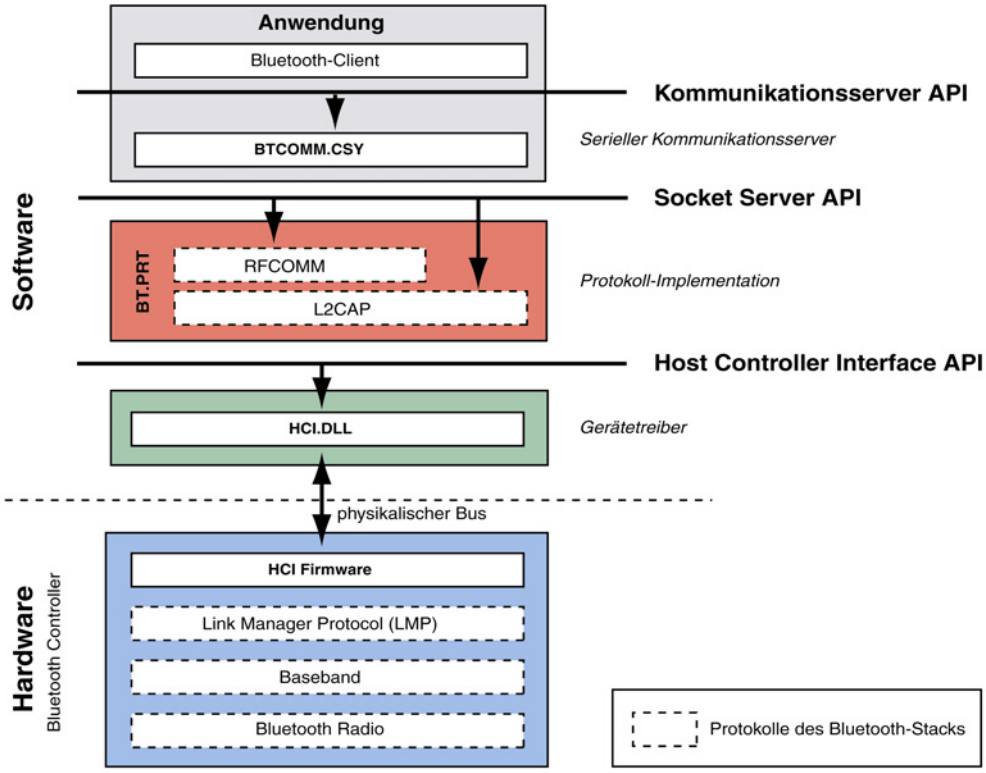


Abbildung 3-9: Verwendung von Bluetooth unter Symbian OS

Unter Symbian OS wird für Bluetooth weder ein eigener Kommunikations-Server noch eine eigene Bluetooth API zur Verfügung gestellt (s. Abbildung 3-9). Als Server muss der serielle Kommunikations-Server eingesetzt und das bluetoothspezifische Kommunikationsmodul „`BTCOMM.CSY`" geladen werden.

Die Bluetooth-Kommunikation selbst beruht auf einer Socketverbindung. Um einen Datenaustausch zwischen Bluetooth-Geräten zu ermöglichen, wird daher der Socket Server eingesetzt. Über dessen API werden auch die Protokolle konfiguriert.

Für die Anmeldung des Bluetooth-Services in der Service-Datenbank oder die Konfiguration der Sicherheitseinstellungen müssen weitere APIs genutzt werden, auf die hier nicht näher eingegangen werden soll. [92]

Bluetooth-Profile unter Symbian OS

Symbian-Geräte unterstützen unterschiedliche Profile. Das SonyEriccson P800 beispielsweise implementiert die Profile GAP, SPP, Generic OBEX, Dial-Up Networking, Object Push, Headset.[93]

In der Stadtinfo-Anwendung wurde lediglich das SPP-Profil genutzt, da das eingesetzte Embedded Device keine höheren Profile unterstützt.

[92] Wie die erwähnten APIs genau genutzt werden wird in Kapitel 6.2.1 genauer erklärt.

[93] Vgl. Internetquelle SonyEricsson P800

3.6 Programmierumgebung

Damit sich ein gut designtes Betriebssystem auch unter Anwendungsprogrammierern durchsetzen kann, sollte es möglichst komfortabel zu programmieren sein. Dazu gehören neben einer guten Dokumentation auch gute Software Development Kits (SDKs) mit sinnvollen und einfachen APIs und eine leicht zu bedienende Programmierumgebung.

Software Development Kits

Die Programmierung für die einzelnen Symbian-Endgeräte unterscheidet sich je nach Version des Symbian-Betriebssystems und nach produktspezifischen Eigenarten wie Eingabemöglichkeiten, Schnittstellen, etc. Die Hersteller stellen daher auf ihre Geräte und Serien angepasste SDKs zur Verfügung, wie z. B. auf den Homepages von Nokia und SonyEricsson.[94] Diese können in der Regel kostenlos heruntergeladen werden und enthalten alle notwendigen Tools, um eine Anwendung erstellen zu können. Dies sind bei Symbian OS für C++ unter anderem der Ressourcen-Compiler für die Erstellung der Ressourcen-Dateien, der C++-Compiler und Linker zur Erstellung des ausführbaren Programms, sowie weitere optional einzusetzende Tools, um Bitmaps und Icons in das spezielle Format für Symbian-Geräte konvertieren zu können.[95] Mit den SDKs wird auch eine Dokumentation zur Verfügung gestellt. Beim SDK für das SonyEricsson P800 z. B. wird diese mit Hilfe eines Java-Applets ständig per Internet erweitert und aktualisiert.
Um eine installierbare Symbian-Anwendungen erstellen zu können, sind neben den eigentlichen Quellcodedateien und Header-Dateien viele projektbeschreibende Dateien notwendig, die mit den Tools verarbeitet werden. Auf die einzelnen Dateien und den Verarbeitungsprozess soll hier nicht näher eingegangen werden, da die Tools in der Regel in eine Programmierumgebung eingebunden werden, die einen Großteil der Vorgänge abnimmt.

Programmierumgebungen

Derzeit gibt es drei wichtige Programmierumgebungen für Symbian OS, auch „IDE"s (Integrated Development Environment) genannt:

- Codewarrior for Symbian von Metrowerks
- Borland Mobile Studio[96]
- Microsoft Visual Studio C++

Der Codewarrior for Symbian ist besser auf die Symbian-Programmierung spezialisiert als das Visual Studio von Microsoft und ist schon wesentlich länger erhältlich als Borlands Mobile Studio. Die Metrowerks-Software wurde daher auch im Stadtinfo-Projekt eingesetzt.

[94] Vgl. Internetquellen Symbian-SDK Nokia und Symbian-SDK SonyEricsson.
[95] Vgl. Digia Inc. (2003), S. 66
[96] Diese IDE kombiniert Borland JBuilder und C++ Builder Mobile Edition. Das Mobile Studio wurde Ende Februar 2004 veröffentlicht, vgl. Internetquelle Borland Mobile Studio.

4 HyNetOS

4.1 Einführung

Das Betriebssystem HyNetOS wurde speziell für Embedded Devices mit Netzwerkfähigkeiten entwickelt, und zwar von der nahe Düsseldorf ansässigen Firma SND (Smart Network Devices). SND wurde 1999 gegründet und vertreibt das Betriebssystem zusammen mit den von ihnen hergestellten Entwicklerboards und deren Hard- und Softwareerweiterungen. Neben zwei Modulen, die auf Web-Anwendungen spezialisiert sind vertreibt SND ein bluetoothfähiges Gerät, das sogenannte „MicroBlueTarget" (MBT, s. Abbildung 4-1). Dieses Modul wurde im Stadtinfo-Projekt eingesetzt. SND arbeitet eng mit der Hyperstone AG in Konstanz zusammen, die die auf den Modulen eingesetzten Prozessoren herstellt und auch Kernel, Compiler, Linker, etc. programmiert hat. Die drei Module sind zwischen 3,2 x 4,2 cm und 3,2 x 5,9 cm groß und mit entsprechend unterschiedlichen Bauteilen bestückt. Die Modul-Platine wird auf das sogenannte „Shuttle2"-Board aufgesteckt, das die Stromversorgung liefert und an das unterschiedliche Peripheriegeräte angeschlossen werden können. Die wichtigsten Schnittstellen hierbei sind die Ethernet-Schnittstelle und eine oder mehrere serielle Schnittstellen. Der Ethernet-Anschluß wird genutzt, um die programmierte Software etc. auf das Board zu übertragen und das System zu warten, die serielle Schnittstelle dient unter anderem dem Debuggen. Über den Ethernet-Anschluß werden auch die Ausgaben des Programms an den angeschlossenen Rechner weitergeleitet und dort in einem MS-DOS-Fenster dargestellt.[97]

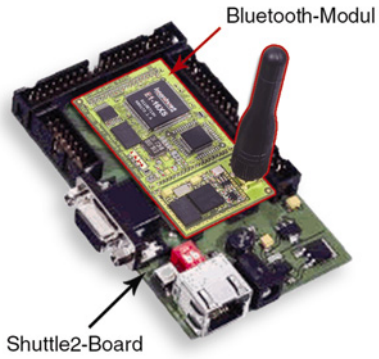


Abbildung 4-1: MicroBlueTarget (MBT)

Die beiden auf Web-Anwendungen spezialisierten Boards unterstützen kein Bluetotoh und sind daher für diese Diplomarbeit uninteressant. In den folgenden Abschnitten wird das Betriebssystem HyNetOS daher im Hinblick auf das bluetoothfähige MBT beschrieben.

[97] Vgl. Internetquelle SND

4.2 System-Architektur

Wie bereits erwähnt, ist HyNetOS ein spezialisiertes Betriebssystem für Embedded Devices. Solche Systeme sind meist auf geringe Größe, einen kleinen Arbeitsspeicher und geringen Stromverbrauch hin optimiert.[98] Wie bei Symbian OS entschieden sich die Designer daher gegen eine Modifikation eines bestehenden Betriebssystems, das für größere Systeme entwickelt wurde. Im Folgenden wird die Hardware des MBT beschrieben, auf die HyNetOS unter anderem ausgelegt ist.

4.2.1 Hardware

Das MBT besteht im wesentlichen aus folgenden Hardwarekomponenten:[99]

- Ein Prozessor (CPU): 32-Bit RISC/DSP-Prozessor der Firma Hyperstone AG mit 118 MHz.
- Speicher:
  - 8 MB SDRAM für alle flüchtigen Daten.
  - Insgesamt 2 MB Flash Memory für Bootloader, Betriebssystem und Dateisystem.[100]
- Schnittstellen:
  - 10/100 Mbit/s Ethernet
  - Serielle Schnittstelle UART bis 1 MBit/s
  - Bluetooth Klasse 2[101]
- Stromanschluss 3,3 V (5V tolerant)

4.2.2 Software-Architektur

Das HyNet-Betriebssystem ist wie die meisten anderen Betriebssysteme in mehreren Ebenen aufgebaut. Die Ebenen verhalten sich in vertikaler Richtung als Dienst-Anbieter (Service Provider) und/oder Dienst-Nutzer (Service User).
Betrachtet man die Kommunikation zweier Systeme in horizontaler Richtung, so besteht nur auf der untersten, der physischen Ebene eine, echte Verbindung zwischen den Systemen. Dies kann zum Beispiel ein Ethernet-Kabel sein. Alle anderen Verbindungen werden daher als „virtuell" bezeichnet.

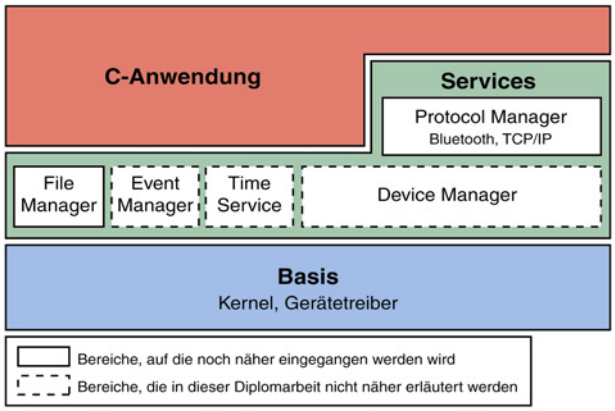


Abbildung 4-2: Software-Architektur unter HyNetOS

[98] Vgl. Tanenbaum (2002), S. 32.
[99] Vgl. PDF-Quelle MBT.
[100] Vgl. Kapitel 4.3.4.
[101] Vgl. Kapitel 2.2.

Abbildung 4-2 zeigt den groben Aufbau des HyNetOS und die Verbindung zu einer in C programmierten Anwendung. Java-Programmierung ist derzeit für das MBT noch nicht verfügbar.
Man kann das System grob in folgende Komponenten unterteilen:
Basis, Services und Anwendungen.

Basis

Sie besteht aus dem Kernel und den Gerätetreibern. Der Kernel ist fester Bestandteil des Betriebssystems, Gerätetreiber hingegen können beliebig hinzugefügt werden.

Kernel

Der Kernel des HyNetOS wurde von der Hyperstone AG geschrieben, die auch die Prozessoren herstellt. „HyRTK" steht für Hyperstone Real-Time-Kernel und bezeichnet damit auch schon eine der wesentlichen Eigenschaften. Der Kernel ist wie auch der Kernel des Symbian-Betriebssystems ein Mikrokernel, dessen Code und Daten hier sogar nur 32 KB in Anspruch nimmt.[102]

Gerätetreiber

Die Gerätetreiber bilden über ihren Hardware Abstraction Layer (HAL) die Schnittstelle zu den einzelnen Geräten wie Bluetooth, Ethernet oder Flash Memory. Sie werden von Anwendungen indirekt über den Device Manager angesprochen. Zum anderen reagieren sie auf physische Interrupt-Signale, die z. B. bei eingehenden Signalen über den Bluetooth-Funktransmitter erzeugt werden.

Services

Services sind Systemdienste, die nicht aktiv agieren, sondern sich beim Systemstart registrieren und dann auf Anfragen von Anwendungen oder anderen Diensten warten. Der Protocol Manager nimmt hierbei eine Sonderstellung ein.
Die wichtigsten Services unter HyNetOS sind:

- File Manager Service:
  Er bietet Zugriff auf die lokalen Speicher RAM oder Flash-ROM. Über die Netzwerk-Schnittstellen sind auch Zugriffe auf externe Dateien möglich. Der Zugriff kann sowohl synchron als auch asynchron erfolgen.
- Event Handler Service:
  Dieser Service kapselt den asynchronen Zugriff auf Dateien, Netzwerk oder Geräte und stellt ihn nach außen hin synchron dar. Dies ist beispielsweise wichtig für synchron ablaufende Prozesse innerhalb einer Java Virtual Machine.
- Time Service:
  Er bietet die Timer-Funktionalität für Anwendungen und die anderen Services.
- Device Manager Service:
  Der Device Manager bietet alle notwendigen Dienste, um Geräte über die Gerätetreiber ansprechen zu können.

[102] Vgl. Kapitel 4.3.1.

- Protocol Manager Service:
  Der Protocol Manager ist eine Art Container für alle möglichen Protokolle und bietet Protokolldienste für beispielsweise TCP/IP-Protokolle oder Bluetooth an. Dieser Service wird bei Systemstart automatisch mit dem Device Manager verlinkt und nimmt daher eine gewisse Sonderstellung ein: Anwendungen können entweder direkt auf den Device Manager zugreifen oder über den Protocol Manager. Der Unterschied besteht darin, dass bei direktem Zugriff das entsprechende Gerät ausschließlich für diese Anwendung reserviert wird. Das macht beispielsweise Sinn bei einer Tastatur. Wird ein Gerät jedoch indirekt über den Protocol Manager angesprochen, so kann es von mehreren Anwendungen genutzt werden, wie beispielsweise die Ethernet-Schnittstelle.

Anwendungen

Anwendungen stellen die oberste Ebene dar und bieten daher auch keine Dienste an, sondern nutzten die darunter liegenden Ebenen.[103]

4.3 System-Grundlagen

Im vorigen Kapitel wurde eine Übersicht über die Software-Architektur gegeben. Nun sollen die wichtigsten Grundlagen und Konzepte von HyNetOS beschrieben werden:

- Der Kernel und seine Aufgaben
- Prozess-Synchronisation mit Guards
- Interprozess-Kommunikation mit Messages
- Speicher-Aufbau und das Flash-Dateisystem

4.3.1 Der Kernel

Der Kernel ist der wichtigste Teil des Systems und übernimmt in HyNetOS folgende Aufgaben:

- Task Management:
  Der Kernel unterstützt präemptives Multitasking. Bis zu 32 Prozesse können gleichzeitig aktiv sein, die je nach Priorität präemptiv behandelt werden.
- Synchronisation und Interrupt Services:
  Die Synchronisation der unterschiedlichen Prozesse erfolgt mit Hilfe von so genannten Guards. Interrupts starten so genannte Interrupt Level Tasks.[104]
- Messaging System-Funktionalität:
  Daten zwischen Prozessen werden mit „Messages" ausgetauscht.[105]
- Speicher Management[106]
- Time Management:
  Der Kernel stellt 31 virtuelle Timer zur Verfügung, sowie weitere Timer für Interrupt Level Tasks.[107]

[103] Vgl. Kapitel 4.4.
[104] Vgl. Kapitel 4.3.2.
[105] Vgl. Kapitel 4.3.3.
[106] Vgl. Kapitel 4.3.4.
[107] Auf das Time Management wird nicht näher eingegangen, da es zum weitern Verständnis nicht notwendig ist.

4.3.2 Tasks und Task-Synchronisation

Es gibt grundsätzlich zwei unterschiedliche Arten von Tasks:

- Stack Level Task: Standard-Prozess des Nutzers wie z. B. eine Anwendung. Ein Stack Level Task kann im Vordergrund oder Hintergrund laufen.
- Interrupt Level Task: Ein Interrupt Level Task ist typischerweise ein System-Prozess, der im Hintergrund läuft, z. B. als Interrupt-Service-Routine

Die Tasks werden vom Kernel präemptiv behandelt, das heißt Tasks höherer Priorität unterbrechen Tasks mit niederer Priorität. Alle Tasks haben eine eindeutige Priorität.
Interrupts bewirken, dass automatisch ein Interrupt Level Task gestartet wird. Um unnötige Kontextwechsel zu vermeiden nutzt dieser den Stack des gerade aktiven Tasks. Auf Interrupt Level Tasks soll nicht näher eingegangen werden, da diese in der Regel nur vom System erzeugt und nicht aktiv von einer Anwendung gestartet werden.

Task-Synchronisation

Tasks können im Grunde unabhängig voneinander ausgeführt werden. Greifen sie jedoch auf gleiche Ressourcen zu, müssen sie synchronisiert werden. Es gibt verschiedenste Möglichkeiten, Tasks zu synchronisieren. Eine bekannte Variante ist die der Semaphoren. Hierbei wird eine Variable eingeführt, die die Anzahl der Weckrufe für einen Task speichert.[108] HyNetOS nutzt eine andere Möglichkeit der Task-Synchronisation, so genannte Guards. Der Umgang mit Guards ist in der Programmierung recht einfach:
Jeder Guard hat einen Guard-Status, der nur vom Kernel geändert werden kann. Tasks warten darauf, dass sie aktiv werden können, indem sie die Funktion `Waitguard()` für einen bestimmten Guard aufrufen. Wird der Guard-Status vom Kernel auf 0 gesetzt, so wird der Task aktiv.
Ein Guard ist keinem speziellen Task zugeordnet. Mehrere Tasks können daher auf denselben Guard warten. Ein einzelner Task jedoch kann zu einer bestimmten Zeit immer nur auf einen einzigen Guard warten, der sich aber innerhalb des Ablaufes des Tasks ändern kann.
HyNetOS versucht, den Kontextwechsel zwischen Tasks aus Performanzgründen zu vermeiden. Statt dessen werden bei einem Webserver beispielsweise mehrere „Connections" zu den verschiedenen Clients geöffnet, alle innerhalb des gleichen Tasks. Auch beim Stadtinfo-Projekt besteht die Anwendung aus lediglich einem Task mit mehreren Connections.
Neben dem Anwendungs-Task werden bei Programmstart verschiedene System-Tasks gestartet wie z. B. der Protocol Manager. Um Nachrichten und Daten zwischen diesen Tasks austauschen zu können, nutzt HyNetOS ein Messaging-System, das die eben beschriebenen Guards nutzt.

4.3.3 Intertask-Kommunikation mit Messages

Daten und Nachrichten zwischen Tasks werden in so genannten „Messages" ausgetauscht, wobei dieser Vorgang rein asynchron abläuft. Trifft eine Message für einen bestimmten Task ein, so wird diese in dessen so genannte „Message Queue" eingelinkt. Eine Message Queue ist also lediglich eine verkette Liste von Messages. Der Zugriff auf Messages in der Message Queue wird nun wiederum mit Hilfe des so genannten „Message Guard" geregelt. Ein Task hat in der Regel genau eine Message Queue, um Nachrichten zu empfangen, kann aber Nachrichten an beliebige Message Queues anderer Tasks schicken.

[108] Vgl. Tanenbaum (2002), S. 127.

4.3.4 Speicher und Dateisystem

Wie bereits erwähnt, besitzt das MBT zwei unterschiedliche Speicherarten: SDRAM-Speicher[109] und Flash-ROM-Speicher[110].

SDRAM-Speicher

Der flüchtige SDRAM-Speicher wird als Stack für die einzelnen Tasks genutzt, sowie für die Systemdateien wie Anwendung und Betriebssystem. Die Systemdateien werden entweder vom PC über Ethernet oder beim Systemstart vom Bootloader vom Flash-ROM in den SDRAM-Speicher geladen.

Flash-ROM-Speicher

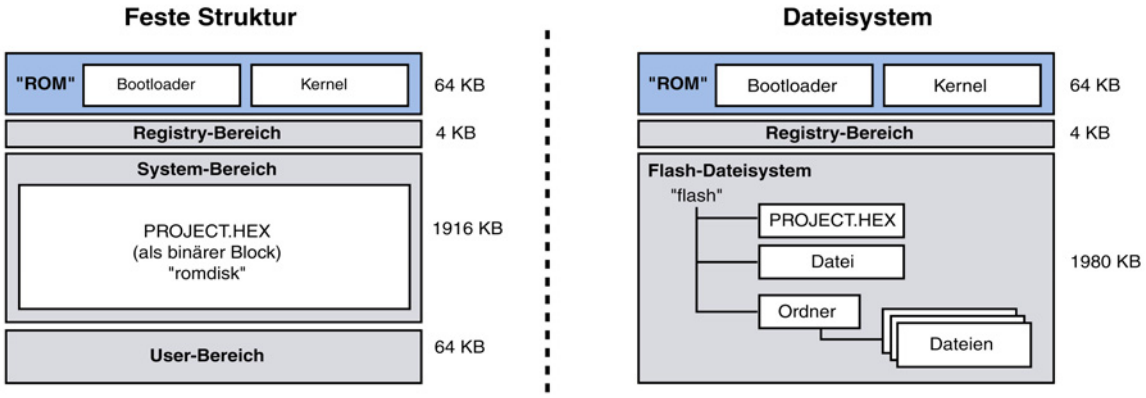


Abbildung 4-3: Das Flash-Speichersystem unter HyNetOS

Der Flash-ROM-Speicher kann auf zwei Arten genutzt werden (s. Abbildung 4-3):

- Feste Struktur:
  Der Hauptteil des Flash-Speichers enthält nur eine Datei mit der Bezeichnung „`PROJECT.HEX`". In dieser sind das System und alle anderen Dateien in Form von binären Daten abgelegt als so genannte „Romdisk". Der Zugriff auf Dateien des virtuellen Dateisystems erfolgt über das Verzeichnis „`/romdisk/dateiname`".
- Dateisystem:
  Die `PROJECT.HEX`-Datei ist nur eine von mehreren Dateien in dem virtuellen Dateisystem. Dieses kann eine beliebige Ordnerstruktur enthalten. Auf die Dateien wird über das Verzeichnis „`/flash/ordnername/dateiname`" zugegriffen.

Standardmäßig befinden sich in den obersten 64 KB des Flash-Speichers zum einen der Kernel HyRTK und zum anderen der Bootloader. Das eigentliche System, repräsentiert durch die `PROJECT.HEX`-Datei, muss immer erst in den SDRAM-Speicher geladen werden. Es enthält neben den Treibern etc. auch die programmierte Anwendung. Entdeckt der Bootloader die `PROJECT.HEX`-Datei im Flash-Memory, sei es als Binär-Image oder als Datei, so lädt er diese automatisch und startet das System. Im Stadtinfo-Projekt wurde die Variante mit Dateisystem eingesetzt. Neben der `PROJECT.HEX`-Datei wurden die Audio- und Textinformationen in einer entsprechenden Ordnerstruktur abgelegt. So konnte komfortabel auf die Dateien zugegriffen und diese auf das Mobiltelefon übertragen werden.

[109] Beim MBT sind dies 8MB.
[110] Beim MBT sind dies insgesamt 2MB.

4.4 Programmierung unter HyNetOS

Dieses Kapitel beschäftigt sich mit der Programmierung von Anwendungen. Auf die Implementierung von Treibern soll in diesem Rahmen nicht eingegangen werden. Wie bereits erwähnt, ist es bei HyNetOS-Anwendungen selten notwendig, mehrere Tasks zu verwenden und Messages zwischen diesen auszutauschen. Dieser Fall wird daher ebenfalls nicht erläutert.
Anwendungen auf dem MBT werden in C programmiert, alle Funktionen der Standard-Library werden unterstützt und zusätzliche APIs für die Programmierung des jeweiligen Moduls mitgeliefert.
Eine Anwendung implementiert mindestens zwei Funktionen. Zum einen die `Main()`-Funktion als Einstiegspunkt für das Programm und zum anderen eine Funktion für den Anwendungs-Task selbst.

Main()

Im Main-Programm werden zunächst alle notwendigen Tasks gestartet:

```
CreateTask(&TaskName);
```

In der Regel sind dies der Device Manager, der Protocol Manager, der Event Handler und der File Manager.
Auf die gleiche Weise wird der Stack Level Task der programmierten Anwendung gestartet. Die System-Tasks haben eine höhere Priorität als die Anwendung, was sich in einem niedrigeren Wert der Prioritätsvariablen zeigt. Alle diese Tasks sind Stack Level Tasks, keine Interrupt Level Tasks.

Anwendungs-Task

Der Anwendungs-Task ist meist in einer anderen Datei implementiert, die im Folgenden genauer beschrieben werden soll.
Jede Anwendung beginnt mit der Definitionen einiger Ressourcen, normalerweise sind dies ein Guard, eine Message Queue und ein Zeiger auf eine sogenannte "Connection Table", in der die verschiedenen Verbindungen und ihre IDs abgelegt werden. Im folgenden Beispielcode werden außerdem noch zwei weitere benötigte Ressourcen definiert: eine Verbindung zum File Manager Service (`cFileManager`) und zum Bluetooth Service (`cBTManager`):

```
static GuardType          TaskNameGuard = { 1 };
static MsgQType           TaskNameMessageQueue;
static ConnInfoType       *ConnectionTable = NULL;
static ConnInfoType       *cFileManager = NULL;
static ConnInfoType       *cBTManager = NULL;
```

Der Kernel selbst speichert die Informationen über den Task in einem sogenannten Task Control Block (TCB). Mit dem Aufruf

```
StackLevelTCB (Taskname, Priorität, OnCreate, ...)
```

werden so unter anderem Taskname und Priorität festgelegt. Hier wird auch übergeben, welche Methode der Anwendung bei Erzeugung des Tasks (`OnCreate`) gerufen werden soll. Diese Methode beinhaltet nun den eigentlichen Code der Anwendung.

Anwendungscode

Messages können nur gesendet und empfangen werden, wenn ein oder mehrere andere Tasks die Message Queue der Anwendung kennen. Der Anwendungs-Task muß daher zunächst einmal angemeldet werden. Hierbei wird neben dem Tasknamen die Message Queue, der Guard und die Connection Table mitübergeben:

```
task_init("TaskName", &TaskNameMessageQueue, &TaskNameGuard, &ConnectionTable);
```

Um eine oder mehrere logische Verbindungen aufbauen zu können, werden anschließend eine oder mehrere Connections zu anderen Diensten gestartet. Die erfolgreichen Connections erhalten eine eindeutige ID und werden in der Connection Table gespeichert. Über diese ID können sie später eindeutig angesprochen und verwendet werden. Bei erfolgreichem Aufbau einer Connection zu einem Systemdienst wird beispielsweise über das Messaging-System die Nachricht „CONN_ACK" an die Anwendung zurückgeschickt, was für „Connection acknowledged" steht.
Mit folgenden Befehlen werden beispielsweise Verbindungen zum File Manager oder Bluetooth Protocol Manager initiiert:

```
cBTManager = connect_bluetooth_service("Bluetooth Manager", …);
cFileManager = fileman_connect("File Manager", …);
```

Nun kommt die eigentliche Programmlogik. In einer Endlos-Schleife werden alle eingehenden Messages nacheinander abgearbeitet:

```
1: while (1)
2: {
3:     WaitGuard(&TaskNameGuard);
4:     message = get_msg(&TaskNameMessageQueue);
5:     {
6:             connection = find_conn_by_id(message->id);
7:
8:             if (connection == cFileManager)
9:             {
10:                    switch (message->type)
11:                    {
12:                            case CONN_ACK: do_something()… break;
13:                            case DATA_IND: do_something_else()… break;
14:                            …
15:                    }
16:            }
17:            if (connection == cBTManager)
18:            {
19:                    switch (message->type)
20:                    {
21:                            case CONN_ACK: do_something()… break;
22:                            case EVT_IND:  handle_event()… break;
23:                            …
24:                    }
25:            }
26:    }
27: }
```

Sobald eine Message für den Task eintrifft wird der Guard-Status vom Kernel geändert und die Anwendung beginnt zu laufen (Zeile 3).
Zunächst wird die Message gelesen (Zeile 4). Für die anschließende Verarbeitung des Inhaltes der Message muss herausgefunden werden, welche Connection angesprochen wird. Dies geschieht durch Auslesen der in der Message angegebenen Connection-ID in Zeile 6. Je nach Connection erfolgt nun die weitere Verarbeitung (Zeile 8 oder 17):
Innerhalb der Connection wird der Typ aus der Message ausgelesen, wie in den Zeilen 10 oder 18. Je nach Message-Typ erfolgen nun die entsprechenden weiteren Befehle. Wird z. B. ein erkanntes Bluetooth-Gerät mit „`EVT_IND`" angezeigt, so kann dieser Event mit „handle_event()" weiter verarbeitet werden, wie in Zeile 21 angedeutet.
Wird der Guard-Status vom Kernel wieder auf einen Wert ungleich 0 gesetzt wartet der Task, bis beispielsweise wieder eine neue Message eintrifft.
Eine Message enthält neben der angesprochenen Connection ID und dem Message-Typ natürlich auch optionale Daten. Diese sind jedoch bei jeder Message anders aufgebaut und leider nicht von der Firma SND dokumentiert. Besonders diese unzureichende Dokumentation erschwerte die Arbeit mit diesem eigentlich einfachen, geradlinig programmierten Betriebssystem. Die enthaltenen Daten einer Message konnten beispielsweise nur durch genaues Studieren der mitgelieferten Beispiel-Anwendungen herausgelesen werden.

Ein Überblick über die Struktur einer Anwendung unter HyNetOS wurde nun gegeben. Innerhalb eines Tasks können mehrere Verbindungen genutzt werden. Die Prozess-Synchronisation erfolgt durch Guards. Wird eine Message an die Anwendung geschickt, so wird der Guard vom Kernel auf 0 gesetzt und die Message kann von der Anwendung in ihrer Endlos-Schleife verarbeitet werden.

4.5 HyNetOS und Bluetooth

Nachdem die Software-Architektur und der grobe Aufbau einer Anwendung dargestellt wurden, sollen hier noch kurz die Bluetooth-Fähigkeiten des MBT erklärt werden. Das MBT wurde im Stadtinfo-Projekt unter anderem als Bluetooth-Client verwendet, um Daten an das Mobiltelefon zu übertragen.

Verwendung von Bluetooth auf dem MBT

SND vertreibt ihr Modul MBT mit der dazu gehörigen Software-Erweiterung, die dem Protocol Manager alle notwendigen Bluetooth-Protokolle hinzufügt.

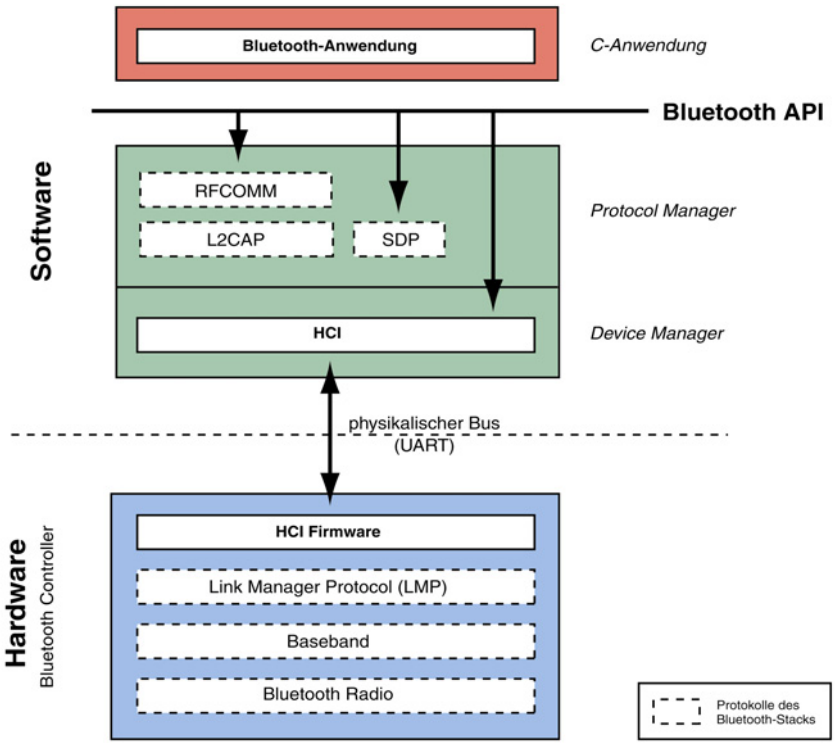


Abbildung 4-4: Verwendung von Bluetooth auf dem MBT

Wie in Abbildung 4-4 ersichtlich, sind RFCOMM, L2CAP und SDP unabhängige Protokollinstanzen, die vom MBT behandelt werden. Der Protocol Manager fungiert als eine Art Container für die Protokolle.
Im Gegensatz zu Symbian OS können die Bluetooth-Funktionen unter HyNetOS mit der mit dem MBT mitgelieferten Bluetooth API recht einfach angesprochen werden. Einige Beispiele hierzu:

- Mit „`BT_hci_retrieve_devices()`" kann nach Bluetooth-Geräten in Reichweite gesucht werden. Jedes gefundene Gerät wird dann über das Messaging-System mit einem Event namens „`EVT_GAP_INQUIRY_RESULT`" an die Anwendung zurückgeliefert.
- Mit „`BT_sdp_search_service()`" kann auf SDP-Ebene nach Diensten auf anderen Bluetooth-Geräten gesucht werden.

- Soll eine Datenverbindung zu einem anderen Bluetooth-Gerät aufgebaut werden, wird dies mit „`BT_spp_connect()`" initiiert und geschieht mit der zur Verfügung gestellten API grundsätzlich auf RFCOMM-Ebene. Hierbei muss neben der Bluetooth-Adresse auch der simulierte Kanal der seriellen Schnittstelle mit angegeben werden. Dieser unter HyNetOS als „SppServerChannel" bezeichnete Kanal entspricht dem DLCI und muss für einen erfolgreichen Verbindungsaufbau auf den Kanal des anderen Bluetooth-Gerätes eingestellt werden.
- Steht eine Verbindung, so können mit dem Befehl „`BT_spp_read_data()`" die ankommende Daten ausgelesen und weiter verarbeitet werden.

Bluetooth-Profile auf dem MBT

In der Bluetooth-Spezifikation wird eine Vielzahl an Profilen definiert, die auf Basis der unterschiedlichen Protokolle arbeiten. Das MBT unterstützt jedoch nur die grundlegenden Profile GAP, SDP und SPP.
Das SPP definiert die Anforderungen, um eine Bluetooth-Kommunikation zwischen zwei Geräten auf RFCOMM-Basis zu ermöglichen. Der Datenaustausch zwischen MBT und Mobiltelefon erfolgte im Stadtinfo-Prototypen folglich auf der RFCOMM-Ebene. Höhere Profile wie z. B. das OBEX-Profil sind derzeit auf dem MBT noch nicht implementiert, konnten beim Stadtinfo-Projekt also auch nicht zur Datenübertragung eingesetzt werden.

4.6 Programmierumgebung

Nachdem eingehend erklärt wurde, wie das Betriebssystem aufgebaut ist und welche Konzepte bei der Programmierung von Anwendungen zu beachten sind soll, nun kurz auf die mitgelieferten Tools eingegangen werden. Diese sind als Befehlszeilentools zu verwenden, eine Programmierumgebung mit grafischer Benutzeroberfläche wird nicht mitgeliefert.

Das System ist denkbar einfach zu installieren. Der komplette Ordner HyNetOS kann einfach auf die Festplatte kopiert werden. In den entsprechenden Unterordnern befinden sich die Tools, Bibliotheken, etc. Hier werden auch die selbst programmierten Anwendungen erstellt, damit die mitgelieferten Batch-Files eingesetzt werden können. Diese nutzen relative Pfadangaben, um die mitgelieferten Tools aufzurufen. Die Netzwerk-Konfiguration ist gleich bei der Installation anzupassen, kann aber auch nachträglich in einer Konfigurationsdatei geändert werden.

Für den Einsatz des MBT werden Build-Tools und Runtime-Tools zur Verfügung gestellt:

Build-Tools

Mit den Build-Tools wird die `PROJECT.HEX`-Datei erstellt, die alle nötigen Komponenten enthält, um die programmierte Anwendung auf dem System ausführen zu können. Hierzu muss der Code kompiliert und in Assembler-Code umgewandelt werden. Dies übernehmen der Compiler „hyC" und der Assembler „hyMasm". Die Objekte werden dann mit dem Linker „hyLink" entsprechend verlinkt. Oft genutzte Bibliotheken können mit dem Library Manager „hyLib" erstellt werden.
Auf diese Tools soll hier nicht weiter eingegangen werden. Umfangreiche Informationen dazu werden von der Dokumentation der Hyperstone AG geliefert.

Zu den Build-Tools gehören außerdem die beiden Tools für den Flash-ROM-Speicher, um die beiden Möglichkeiten der Speichereinteilung nutzen zu können: Mit „`CreateRamDisk`" wird die `PROJECT.HEX`-Datei als binärer Block erzeugt. Eine mit „`CreateFlashDisk`" erzeugte Datei repräsentiert ein komplettes Dateisystem und enthielt im Falle der Stadtinfo-Anwendung auch die `PROJECT.HEX`-Datei, damit das System autonom lauffähig war.

Runtime-Tools

Diese Tools werden verwendet, um das Modul zu booten, zu konfigurieren und zu beobachten.
Folgende Tools werden mitgeliefert:

- Das Administrations-Tool HyFlash:
  Mit HyFlash wird das Flash-Memory auf dem Modul neu beschrieben, um Dateien permanent auf dem Modul zu speichern. Eine `PROJECT.HEX`-Datei kann aber auch über HyLoad per Ethernet auf das Modul übertragen und die Anwendung so getestet werden, ohne gleich den Flash-Speicher neu beschreiben zu müssen.
- Der Bootloader HyLoad:
  Mit HyLoad wird die System-Datei geladen und das System gestartet.
- Der Tracer HyTrace:
  Mit HyTrace wird die Ausgabe des Moduls angefragt. Der Befehl „`printf`" in einer programmierten Anwendung gibt also seine Ausgabe über das Ethernet an den PC weiter, der HyTrace aufruft. Die Ausgabe kann opt ional zusätzlich auch in eine Log-Datei mitgeschrieben werden.
- Der Monitor HyMon:
  HyMon wird selten benötigt, hiermit können u.a. Informationen des Moduls abgefragt werden wie z. B. das Dateisystem angezeigt werden.
- Der Debugger HyDebug:
  Der Debugger HyDebug nimmt eine Sonderstellung ein, da er eine grafische Oberfläche bietet. Seit der im Dezember 2003 veröffentlichten Version läuft er auch unter aktuellen Windows-Versionen. Leider wurde die CD mit dem neuen Debugger erst Mitte Januar zur Verfügung gestellt, so dass dieser nicht eingesetzt wurde und daher auch nicht näher auf seine Funktionen eingegangen werden soll.

5 Stadtinfo-Projekt: Analyse und Konzeption

Aufgrund der anwachsenden Informationsvielfalt gewinnt es heutzutage zunehmend an Bedeutung, auf die richtigen Informationen zum richtigen Zeitpunkt zugreifen zu können. Wie in der Einleitung schon angedeutet, wird daher auch der Austausch von Informationen an Bedeutung gewinnen. Mobiltelefone sind das ideale Medium, um Informationen abzurufen. Sie sind portabel, werden immer leistungsfähiger und legen naturgemäß großen Wert auf umfangreiche Kommunikationsmöglichkeiten.
In dieser Arbeit wird ein Projekt beschrieben, bei dem Informationen mittels eines Mobiltelefons abgerufen werden. In diesem speziellen Fall handelt es sich um Informationen zu Sehenswürdigkeiten, die ein Tourist an den für ihn interessanten Orten zur Verfügung gestellt bekommt.

5.1 Idee und Anforderungen

Jeder kennt das Gefühl, sich in einer fremden Stadt zurecht finden zu müssen. Immer weniger Menschen haben jedoch die Zeit, vor Eintreffen vor Ort Reiseführer und Karten zu studieren und die interessanten Informationen herauszufiltern. Außerdem soll die Spontaneität nicht zu kurz kommen. Oft ist es am interessantesten, eine Stadt dadurch zu erkunden, indem man sich einfach mit den Menschen treiben und die Eindrücke auf sich wirken lässt.
Aus diesen Überlegungen heraus entstand bei der Firma Alcatel im Herbst 2003 die Idee zu einem Stadtinformationsdienst, der gegen Bezahlung lokale Informationen über Sehenswürdigkeiten bereitstellt. In einem exemplarisch für die Stadt Stuttgart entwickelten Prototypen sollte Funktionalität und Benutzerführung getestet werden. Der Dienst soll bei erfolgreicher Realisation Vertretern verschiedener Städte vorgeführt werden, für die dieses System interessant sein könnte, z. B. auch im Hinblick auf die Fußball-Weltmeisterschaft 2006 in Deutschland. Das Konzept und die Kombination der Komponenten erscheint über einen Stadtinformationsdienst hinaus jedoch auch für viele andere Bereiche interessant und sollte später eventuell zu einer flexibel einsetzbaren Finalversion weiterentwickelt werden.

Use Cases des Stadtinformationsdienstes

Betrachtet man die Nutzungsmöglichkeiten des Touristen gegenüber dem Stadtinformationsdienst als Gesamtes, so ergeben sich folgende Use Cases:

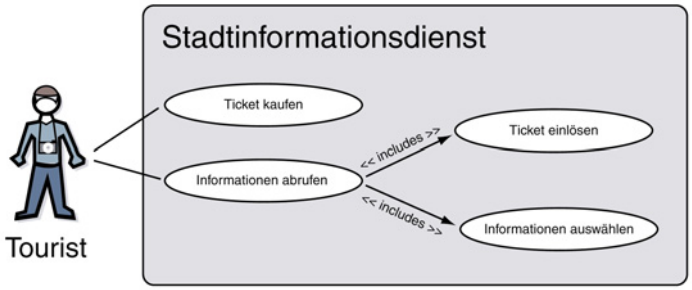


Abbildung 5-1: Use Cases Stadtinformationsdienst

Der Tourist muss zunächst ein Ticket kaufen, um den Dienst nutzen zu können. Um Informationen abrufen zu können, muss er dieses Ticket an einer Sehenswürdigkeit einlösen. Die Informationen sollen dann automatisch heruntergeladen werden, so dass er die entsprechenden für ihn interessanten Informationen auswählen kann.

Die Anforderungen an das zu entwickelnde System wurden folgendermaßen definiert:

- Einfaches Bezahlsystem
- Volle Kostenkontrolle für den Nutzer
- Automatische Bereitstellung der lokalen Informationen
- Einfache Benutzerführung
- Erweiterbarkeit und Portierbarkeit

Die Grundidee eines solchen Dienstes ist nicht neu. In Museen beispielsweise kann man oft gegen Gebühr mittels eines portablen Gerätes Informationen zu den Ausstellungsstücken abrufen.
Die Idee, Touristen per Mobiltelefon lokal Informationen über Sehenswürdigkeiten anzubieten wurde in ähnlicher Form ebenfalls schon verfolgt. In Landsberg am Lech z. B. wurde über einen Zeitraum von drei Jahren von verschiedenen Firmen und Forschungseinrichtungen bereits ein mobiler Stadtführer entwickelt. Dieser fokussiert sich jedoch eher auf vorgegebene Stadttouren.[111]
Das System weist noch weitere entscheidende Unterschiede zu dem in dieser Diplomarbeit entwickelten Prototypen auf: Die Bezahlung im Fremdenverkehrsbüro beschränkt den Nutzer auf festgelegte Öffnungszeiten und die Datenübertragung, die in Landsberg teilweise auch über GPRS erfolgt, erschwert eine Kostenkontrolle. Des weiteren sind die Informationen zentral gespeichert und müssen über ein komplexes System von einem Content-Server abgerufen werden.
Im Folgenden soll nun ein Überblick über das Gesamtkonzept des Stadtinfo-Projektes gegeben werden.

[111] Vgl. Hannemann (2003), S. 158.

5.2 Konzeption

5.2.1 Gesamtarchitektur

Abbildung 5-2 zeigt die Gesamtarchitektur des Stadtinformationsdienstes. Das System kann grob in zwei Bereiche eingeteilt werden:

- Ticketkauf
- Informationsabruf

Der Informationsabruf wird wiederum in zwei Schritte unterteilt:

- Anmeldung bei erstmaligem Einsatz
- Abrufen der lokalen Informationen

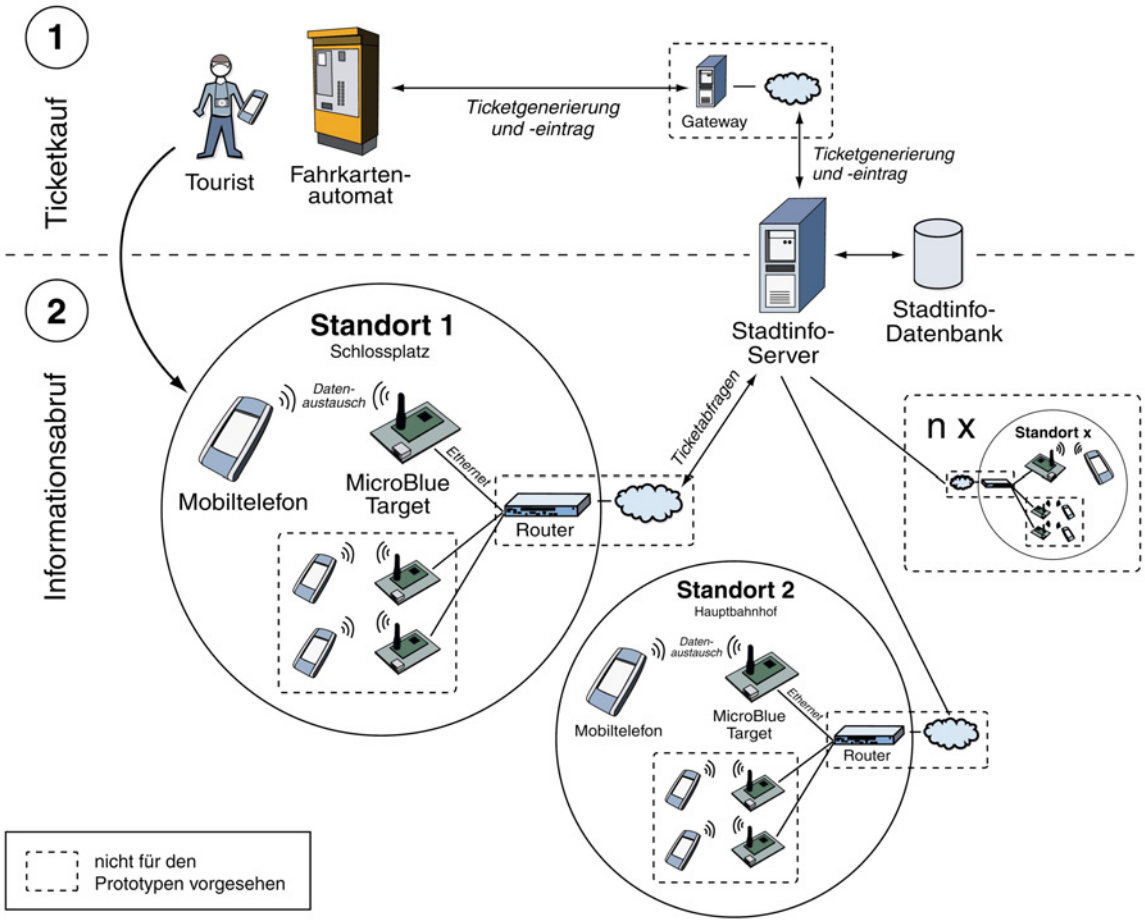


Abbildung 5-2: Stadtinfo-Gesamtarchitektur

Ticketkauf

Für das Bezahlsystem des Stadtinformationsdienstes sind viele Varianten denkbar, beispielsweise per Internet. Viele Anwender sind einem solchem Bezahlsystem gegenüber jedoch skeptisch. Daher wurde die Verwendung eines bereits bestehendes System angedacht, das jedem bekannt ist und vielerorts zur Verfügung steht: Das System für öffentliche Verkehrsmittel, im Falle der Stadt Stuttgart also das VVS-System. Die meisten Touristen erreichen Stuttgart per Zug oder Flugzeug oder bewegen sich innerhalb der Stadt ohnehin mit öffentlichen Verkehrsmitteln. Ein Kontakt mit den Fahrkartenautomaten kann daher vorausgesetzt werden. An einem solchen Fahrkarten-automaten kann der Tourist ein „Ticket" für den Stadtinformationsdienst erwerben, exemplarisch für einen Tag oder eine Woche. Die Ticketnummer wird vom Stadtinfo-Server generiert und nach Bezahlung am Fahrkartenautomaten ausgedruckt. Außerdem wird es in der Datenbank des Stadtinfo-Servers eingetragen als ein vergebenes, aber noch nicht aktiviertes Ticket. Den Zeitpunkt, wann der Tourist dieses Ticket einlösen möchte, kann er so selbst bestimmen. Frühestens geschieht dies bei Erreichen der ersten Sehenswürdigkeit. Durch dieses Bezahlsystem und keine weiteren Kosten bei der späteren Datenübertragung kann eine einfache, vollständige Kostenkontrolle garantiert werden.
Als Stadtinformations-Server wurde ein PC mit dem Windows 2000 Professional-Betriebssystem eingesetzt. Die Datenbank wurde mit MySQL realisiert, die Datenbankabfrage erfolgt über PHP.

Informationsabruf

Die interessanten Daten an den Sehenswürdigkeiten sollen automatisch heruntergeladen und bereit gestellt werden.
Als Übertragungstechnik wurde, wie bereits erwähnt, Bluetooth eingesetzt. Bluetooth bietet mit bis zu 100 Metern Reichweite und 721 kBit/s Transferrate ansprechende Werte. Die Informationen werden lokal von den bereits beschriebenen Bluetooth-Modulen der Firma SND bereitgestellt. Das MBT ist aufgrund seiner Größe einfach zu installieren und bietet die notwendigen Schnittstellen Ethernet und Bluetooth. Die Möglichkeit zur lokalen Datenspeicherung ist ebenfalls ausreichend vorhanden. Als mobiles Endgerät soll das Mobiltelefon des Touristen zum Einsatz kommen. So muss er zum Informationsabruf kein zusätzliches Gerät und dessen Bedienung erlernen. Beim Stadtinfo-Projekt wurde hier aufgrund des leistungsfähigen Betriebssystems und der umfangreichen Bedien- und Darstellungsmöglichkeiten das SonyEricsson P800 eingesetzt, im Folgenden kurz „P800" genannt.

Anmeldung bei erstmaligem Einsatz

Erreicht der Tourist nach Kauf des Tickets zum ersten Mal eine Sehenswürdigkeit, so wird er vom MBT erkannt und durch sein Mobiltelefon mittels Vibration und Signalton informiert. Dies wird im Folgenden als „Stadtinfo-Notifikation" bezeichnet. Nach dieser Notifikation kann er seine Ticketnummer eingeben und wird bei korrekter Eingabe angemeldet. Hierbei wird die bereits in der Datenbank vorhandene Ticketnummer mit der Bluetooth-Adresse des Mobiltelefons verknüpft. Die Ticketgültigkeit wird ausschließlich auf dem Stadtinfo-Server verwaltet, so dass eine Manipulation durch Veränderung der lokalen Daten des Mobiltelefons ausgeschlossen werden kann.

Abrufen der lokalen Informationen

Sobald sich der Tourist erfolgreich beim Stadtinformationsdienst angemeldet hat, werden ihm die vor Ort interessanten Informationen über die Sehenswürdigkeit übermittelt und als Dateien auf dem Mobiltelefon gespeichert. Dies erfolgt in zwei Schritten:

- Übermittlung einer XML-Datei:
  Diese beschreibt, welche Informationen am Standort verfügbar sind.
- Übermittlung der so genannten Content-Dateien:
  Nach Auswertung der XML-Datei durch die Anwendung auf dem Mobiltelefon werden die Informationen nacheinander heruntergeladen. Sobald die erste Content-Datei erfolgreich gespeichert wurde, kann der Tourist diese in einer Liste auswählen und darstellen lassen.

Nach Download aller Informationen wird die Verbindung zwischen MBT und Mobiltelefon wieder getrennt. Betritt der Tourist einen neuen Standort und besitzt ein noch gültiges Ticket, so wird dies anhand der Bluetooth-Adresse erkannt. Die Daten des alten Standortes werden gelöscht und die neuen Informationen automatisch an ihn übermittelt.

5.2.2 Installation der Anwendung und Verbindungsaufbau

Installation der Anwendung auf dem Mobiltelefon

Um den Stadtinformationsdienst nutzen zu können, muss die Anwendung auf dem Mobiltelefon des Touristen installiert sein. Hierfür gibt es eine Reihe von Möglichkeiten. Die Anwendung könnte per Internet vom Touristen schon zu Hause heruntergeladen werden oder per GPRS vor Ort. Eine bessere Möglichkeit wäre hingegen das kostenlose Bereitstellen der Anwendung vor Ort per Bluetooth. Mehrere Bluetooth-Profile stehen zur Verfügung, um dies zu ermöglichen. Keines jedoch unterstützt eine automatische Installation der Anwendung. Eine Möglichkeit wäre das Object-Push-Profil, beruhend auf OBEX. Leider unterstützen die MBT-Module dieses Profil nicht. Möglich wäre auch, die Anwendung per FTP zur Verfügung zu stellen. Der Tourist müsste dann am Fahrkartenautomat sein Mobiltelefon auswählen und einen Download auslösen. Auch dies wird vom MBT momentan nicht unterstützt.
Eine vollständig befriedigende Lösung für die Übertragung der Stadtinfo-Applikation konnte also aufgrund der momentanen technischen Voraussetzungen nicht gefunden werden. Es wird daher vorausgesetzt, dass die Anwendung bereits auf dem Mobiltelefon des Anwenders installiert wurde.

Verbindungsaufbau

Eine wichtige Entscheidung in der Konzeptionsphase betraf die Initiierung der Bluetooth-Verbindung. Im Vorfeld der Diplomarbeit wurden schon Erfahrungen gesammelt mit Verbindungen, die das P800 zum MBT initiierte. Das P800 agierte hier als Client, das MBT als Server. Dieses Vorgehen hätte bei der Stadtinfo-Anwendung jedoch erhebliche Nachteile mit sich gebracht: Das Mobiltelefon hätte in periodischen Abständen aktiv nach MBTs in Reichweite suchen müssen, was hinsichtlich des damit verbundenen Stromverbrauchs nicht akzeptabel war.
Der Tourist hätte die Anwendung folglich selbst beenden müssen, hätte er den Stadtinformationsdienst zeitweise nicht nutzen wollen. Dies widerspricht jedoch zum einen dem Konzept

des Stadtinformationsdienstes und ist zum anderen auf dem P800 auch gar nicht erwünscht: Die Philosophie der Symbian-Version auf dem P800 sieht vor, dass eine Anwendung nicht vom Nutzer beendet werden können soll. Einen Task-Manager gibt es auf dem P800 standardmäßig ebenfalls nicht.[112]
Die Lösung bestand folglich darin, dass das stationäre MBT aktiv nach Bluetooth-Geräten in seiner Reichweite sucht und die Verbindung herstellt. Eine Verbindung mit dem MBT als Client konnte jedoch trotz intensiver Versuche zunächst nicht hergestellt werden. Nach Rücksprache mit der Herstellerfirma SND wurde jedoch ein Fehler in der Bluetooth-Protokoll-Implementierung behoben. Mit der neuen Software-Release war dann ein Verbindungsaufbau vom MBT aus möglich.

5.2.3 Prototyp und Finalversion

Für den Prototypen wurde das System auf zwei exemplarische Sehenswürdigkeiten beschränkt, an denen jeweils genau ein MBT installiert ist. Des weiteren werden die in Reichweite befindlichen Geräte nacheinander mit Informationen versorgt und nicht gleichzeitig bedient. Diese Einschränkungen sollen in einer Finalversion wegfallen. Für die finale Version sind auch weitere Möglichkeiten angedacht, wie beispielsweise das Anlegen von Profilen, in denen Sprache, Interessengebiete usw. eingestellt werden können.[113]
Der Fahrkartenautomat wurde im Prototypen durch ein Web-Interface simuliert, über das ein virtuelles Ticket gekauft werden kann, das 5 bzw. 35 Minuten gültig bleibt.

5.2.4 Ablauf des Informationsabrufs

Um den korrekten Informationsabruf zu ermöglichen, ist eine Interaktion der Komponenten untereinander notwendig. Abbildung 5-3 auf der folgenden Seite gibt einen Überblick über diesen Ablauf.

Vorbedingungen

- Die Stadtinfo-Anwendung muss auf dem P800 installiert und gestartet sein.
- Die Bluetooth-Funktion des P800 muss eingeschaltet sein.
- Der Tourist sollte ein Ticket erworben haben.

Ablauf

1. Der Tourist nähert sich einer Sehenswürdigkeit.
2. Das MBT sucht laufend nach Bluetooth-Geräten in seiner Reichweite und sendet bei erfolgreicher Verbindung die Stadtinfo-Notifikation, die auch die Standort-ID enthält.
3. Das P800 erkennt den Standort.

4a. Sind bereits alle Dateien des aktuellen Standorts geladen, werden diese angezeigt.[114]

4b. Sind nicht alle Daten des aktuellen Standorts vorhanden, schickt das P800 die so genannte „Ticketstatusabfrage" an das MBT, um abzufragen, ob beim Stadtinfo-Server ein gültiges Ticket eingetragen ist.

5a. Ist kein gültiges Ticket vorhanden, wird der Tourist aufgefordert, eine Ticketnummer einzugeben.

---

[112] Vgl. PDF-Quelle Symbian Designing For UIQ, S. 6.
[113] Vgl. Kapitel 7.2.
[114] Der Tourist hat bereits für diese Informationen bezahlt und sollte diese abrufen können, auch falls sein Ticket in der Zwischenzeit abgelaufen sein sollte.

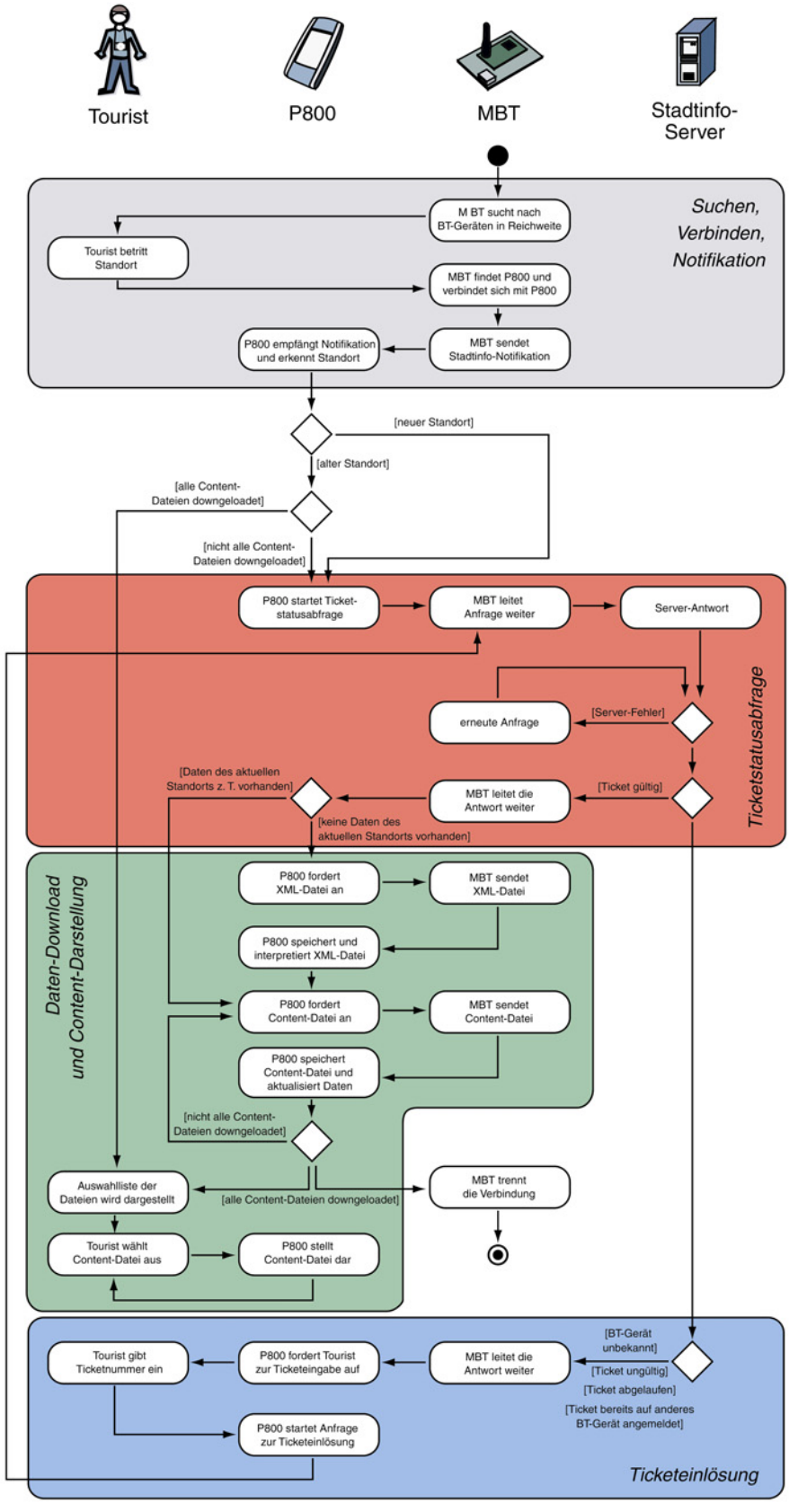


Abbildung 5-3: Activity-Diagramm Informationsabruf

5b. Sobald ein gültiges Ticket vorhanden ist, werden die notwendigen Dateien übertragen. Befinden sich noch keine Dateien des aktuellen Standorts auf dem Mobiltelefon, ist dies zunächst die XML-Datei, die beschreibt, welche Content-Dateien an der betreffenden Sehenswürdigkeit bereitgestellt werden. Ansonsten werden gleich die noch fehlenden Content-Dateien übertragen.
6. Waren schon Dateien auf dem Mobiltelefon vorhanden oder wurde eine Content-Datei erfolgreich heruntergeladen und gespeichert, so kann der Tourist diese ansehen oder im Falle von Audiodateien anhören.
7. Wurden alle Dateien heruntergeladen, so wird die Verbindung getrennt.[115]

5.2.5 Use Cases P800 / MBT

Das Stadtinfo-System besteht aus einer Reihe von miteinander interagierenden Systemen: Das P800, das MBT, der Stadtinfo-Server und der Fahrkartenautomat. Jedes dieser Systeme hat wiederum Anwenderziele[116] ein anderes System betreffend, ist also auch Akteur. Die zwei Hauptsysteme und –akteure sind das P800 und das MBT, die in den folgenden beiden Abbildungen isoliert betrachtet werden sollen. Die Umsetzung der Anwendungen auf dem P800 und dem MBT wird in den Kapiteln 6.1 und 6.2 eingehend beschrieben.

Use Cases P800

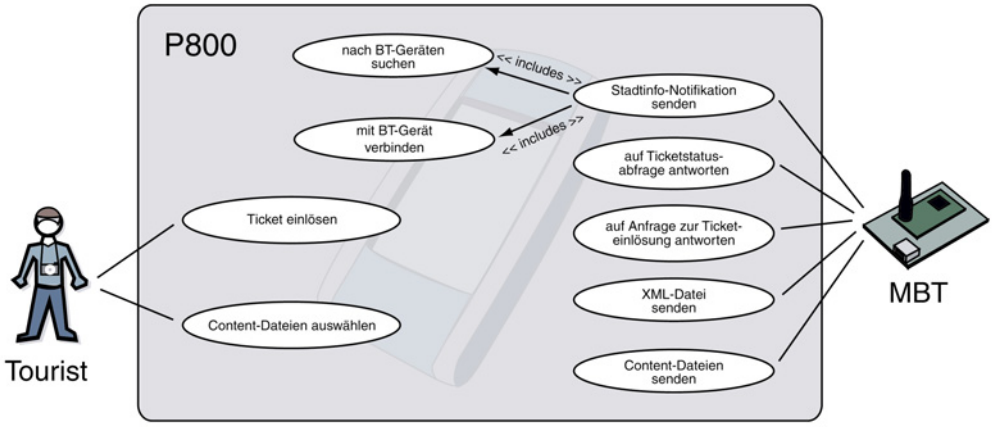


Abbildung 5-4: Use Cases P800

Das P800 bildet für den Touristen nach dem Ticketkauf am Fahrkartenautomaten die Schnittstelle zum Stadtinformationsdienst. Um die Bedienung möglichst einfach zugestalten, gibt es für ihn nur zwei Möglichkeiten: Das Einlösen eines Tickets und das Auswählen der Content-Dateien.
Das MBT übernimmt die Notifikation des P800 am Standort. Hierfür muss es zunächst nach Bluetooth-Geräten suchen und dann die Verbindung einleiten. Anschließend reagiert das MBT nur noch auf Anfragen vom P800: Es antwortet auf Ticketstatusabfragen, Anfragen zur Ticketeinlösung und sendet XML- und Content-Dateien.

[115] Genauer: Das P800 fordert das MBT auf, die Verbindung zu trennen. Erklärungen hierzu später.
[116] Vgl. Fowler (1998), S. 54.

Use Cases MBT

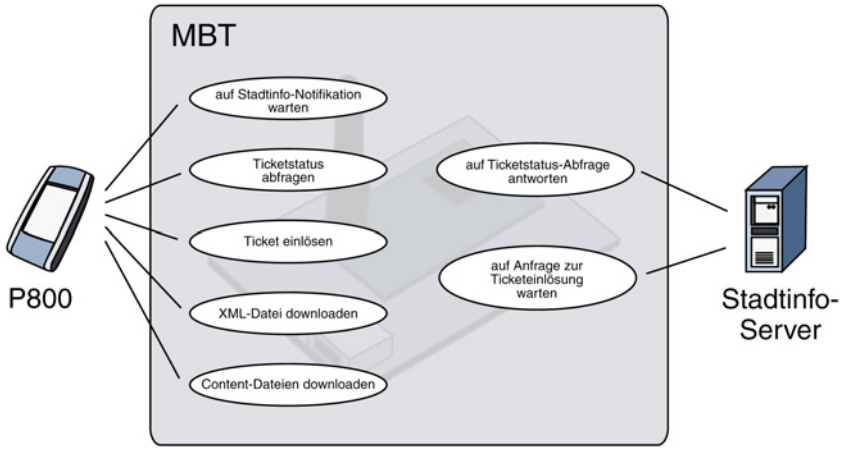


Abbildung 5-5: Use Cases MBT

Das MBT bildet die Schnittstelle zwischen P800 und Stadtinfo-Server. Das P800 übernimmt zunächst eine passive Rolle gegenüber dem MBT und wartet auf dessen Notifikation. Sobald diese erfolgt ist und die Verbindung steht, ergeben sich vier mögliche Use Cases: Die Ticketstatus-abfrage, die Ticketeinlösung und der Download von XML-Datei und Content-Dateien.
Der Stadtinfo-Server muss gegenüber dem MBT auf die entsprechenden Anfragen antworten.

5.3 Projektablauf

Konzeptionsphase und Testphase

Die größten Herausforderungen bei der Umsetzung ergaben sich daraus, dass fast alle Komponenten des Projektes noch nicht voll ausgereift sind. Das P800 z. B. gibt es erst seit etwa einem Jahr auf dem Markt, Symbian OS wird ständig weiterentwickelt und ausgebessert. Mit Bluetooth wurden ebenfalls noch nicht genügend Erfahrungen gesammelt. Das MBT ist ein Entwicklerboard, dessen neueste Software-Release Mitte Dezember 2003 herausgegeben wurde und einige Fehler, unter anderem in der Implementierung der Bluetooth-Protokolle, behob. Angesichts dieser Unbekannten war es unumgänglich, vor Beginn der eigentlichen Programmierung einige Tests durchzuführen. Sie ermöglichten gleichzeitig eine allgemeine, sowie projektbezogene Einarbeitung in die Betriebssysteme und Entwicklungsumgebungen. Die Tests beeinflussten wiederum die Konzeption, da herausgefunden werden musste, was nicht nur theoretisch, sondern auch im praktischen Einsatz machbar war.
Die vorab durchgeführten Untersuchungen umfassten folgende Gebiete:

- HyNetOS/Symbian OS:
  Grundlagen und Konzepte verstehen, Fähigkeiten und Profile herausfinden.
- Bluetooth:
  Umfangreiche Tests zu Verbindungsaufbau und Kommunikation zwischen P800 und MBT.

Darüber hinaus wurden während dieser Phase einige Testinhalte in Form von Texten und Audiodateien erstellt.

Programmierung

Nach Abschluß der grundlegenden Tests wurde das Gesamtsystem entsprechend der beteiligten Komponenten in mehreren Schritten implementiert:

- Stadtinfo-Anwendung auf dem P800:
  Das P800 stellte die komplexeste der zu erstellenden Anwendungen dar. Die Programmierung des P800 nahm daher nach der Konzeptionsphase die meiste Zeit in Anspruch. Die Funktionalität der einzelnen Klassen wurde mit entsprechenden Testklassen überprüft, die zunächst statische Werte zurücklieferten. Die Kommunikation mit dem MBT beispielsweise wurde anfangs innerhalb einer Test-Bluetooth-Engine simuliert, um die Zeit für Verbindungsaufbau und Datenaustausch zu sparen.
- Stadtinfo-Anwendung auf dem MBT:
  Auch die Anwendung auf dem MBT wurde in mehreren Schritten entwickelt. Da der Server anfangs noch nicht verfügbar war, wurden zunächst von einer Testklasse statisch Werte zurückgegeben, die die Ticketgültigkeit regelten. Je nach Gültigkeit wurden dann die Content-Dateien übermittelt. Erst am Ende wurde der Stadtinfo-Server eingebunden und der Gesamtablauf des Systems getestet.

Debugging

Gerade bei neuen und unbekannten Systemen nimmt ein komfortables Debugging einen hohen Stellenwert ein. Der Emulator für das P800, der vom SDK zur Verfügung gestellt und von der Programmierumgebung eingebunden wird, liefert bezüglich der Funktionalität auf dem Endsystem jedoch leider keine zuverlässigen Ergebnisse. Auch das Remote-Debugging auf dem P800 war aufgrund seiner Instabilität nicht einsetzbar.
Für die Anwendung auf dem P800 wurde daher ein Logger implementiert, der wichtige Informationen während des Programmablaufes dokumentierte. Diese konnte anschließend mit einem einfachen Texteditor ausgewertet werden. Während des Projektes stellte sich jedoch heraus, dass auch dieses Vorgehen teilweise zu Problemen führte. Erfolgte ein Schreibvorgang beispielsweise in der `RunL()`-Methode eines Active Objects der Bluetooth Engine, so kam es vor, dass diese nicht mehr korrekt arbeitete: Mit dem Logger wurde ein anderer Thread aufgerufen und das System war daraufhin mit der Verarbeitung der gleichzeitig über Bluetooth eingehenden Daten überfordert. Auch der Logger wurde am Ende folglich eher vorsichtig eingesetzt.
Das für das MBT mitgelieferte Debugging-Tool steuert das Modul auf der seriellen Schnittstelle an und hat auch eine grafische Oberfläche. Leider war es bis kurz vor Projektende nur auf Windows 98 lauffähig, weshalb es auf dem zur Verfügung stehenden System nicht eingesetzt werden konnte. Das war jedoch kein allzu großer Nachteil. Kontrollausgaben konnten im Anwendungscode einfach eingefügt und über Ethernet auf einem angeschlossenen PC ausgegeben werden.

Abschließende Testphase und Fehlerbehebung

Nach Abschluss jeder Programmierphase wurden die einzelnen Komponenten überprüft. Am Ende des Projektes erfolgten dann abschließende Tests mit mehreren P800-Mobiltelefonen und zwei MBTs, die exemplarisch die Sehenswürdigkeiten Schlossplatz und Hauptbahnhof darstellten und per Ethernet mit dem Stadtinfo-Server verbunden waren. In dieser letzten Phase konnten noch einige Fehler behoben und das System weiter verbessert werden.
So wurde in Tests mit mehreren Sehenswürdigkeiten erkannt, dass alte Audiodateien teilweise nicht korrekt gelöscht wurden, da der Medienserver den Zugriff auf diese nicht frei gab. Dieser Fall trat nur dann auf, wenn der Nutzer zu einem bereits besuchten Standort zurückkehrte und dort auch vorher die Audiodatei angehört hatte. Die Freigabe der Datei konnte durch Löschen des Medienplayer-Objektes erreicht und der Fehler so behoben werden.
Auch die Serverlogik wurde weiter verbessert. So bleiben beispielsweise die Bluetooth-Adressen nun nicht mehr dauerhaft in der Datenbank gespeichert, so dass die Server-Antwort „Bluetooth-Gerät unbekannt" auch bei identischem Vorführgerät bei jeder Demonstration gegenüber Interessierten möglich ist und nicht nur ein abgelaufenes Ticket signalisiert wird.

6 Stadtinfo-Projekt: Umsetzung

Fahrkartenautomat und Stadtinfo-Server

Die Infrastruktur für das Bezahlsystem per VVS-Fahrkartenautomat in der Stadt Stuttgart besteht bereits. Für die Entwicklung des Prototypen war auch kein Zugriff darauf notwendig. Der Ticketkauf wurde daher über ein Web-Interface simuliert.
Der Stadtinfo-Server verwaltet Tickets und angemeldete Geräte und legt bei Kauf eines Tickets einen neuen Datensatz an. Dieser enthält für das Funktionieren des Systems notwendigerweise eine Ticketnummer, die Gültigkeitsdauer, ein freies Feld für die Bluetooth-Adresse und ein freies Feld für das Einlösedatum. Ticketabfragen werden per HTTP-Request über Ethernet vom MBT gestartet und per PHP-Skript verarbeitet.
Auf die Simulation des Fahrkartenautomaten und die Implementierung des Stadtinfo-Servers soll in dieser Diplomarbeit nicht weiter eingegangen werden, da dies von einem externen Mitarbeiter der Firma Alcatel übernommen wurde.

MBT und P800

Bezüglich der Komplexität der zu programmierenden Anwendungen stellen MBT und P800 die Hauptkomponenten des Systems dar. In den folgenden beiden Kapiteln soll daher eingehend auf die Umsetzung dieser beiden Applikationen eingegangen werden.

Darauf aufbauend wird in einem Gesamtüberblick das Szenario beim erstmaligen Erreichen einer Sehenswürdigkeit durch den Touristen dargestellt und anschließend auf das zur Kommunikation der Komponenten entworfene Stadtinfo-Protokoll, sowie den Aufbau der XML-Datei eingegangen werden. Eine wichtige Anforderung an den Stadtinformationsdienst war die einfache Benutzerführung. Kapitel 6.5 zeigt schließlich einige Screenshots der Stadtinfo-Anwendung auf dem P800.

6.1 Die Stadtinfo-Anwendung auf dem MBT

Das MBT initiiert Verbindungsaufbau und ermöglicht dem P800 dadurch den Zugang zum Stadtinformationsdienst. Es ist daher ständig aktiv und übernimmt folgende Aufgaben:

- Aktives Suchen von Bluetooth-Geräten in Reichweite und Verbindungsaufbau
- Schnittstelle zum Stadtinfo-Server für Ticketabfragen
- Bereitstellen von Informationen

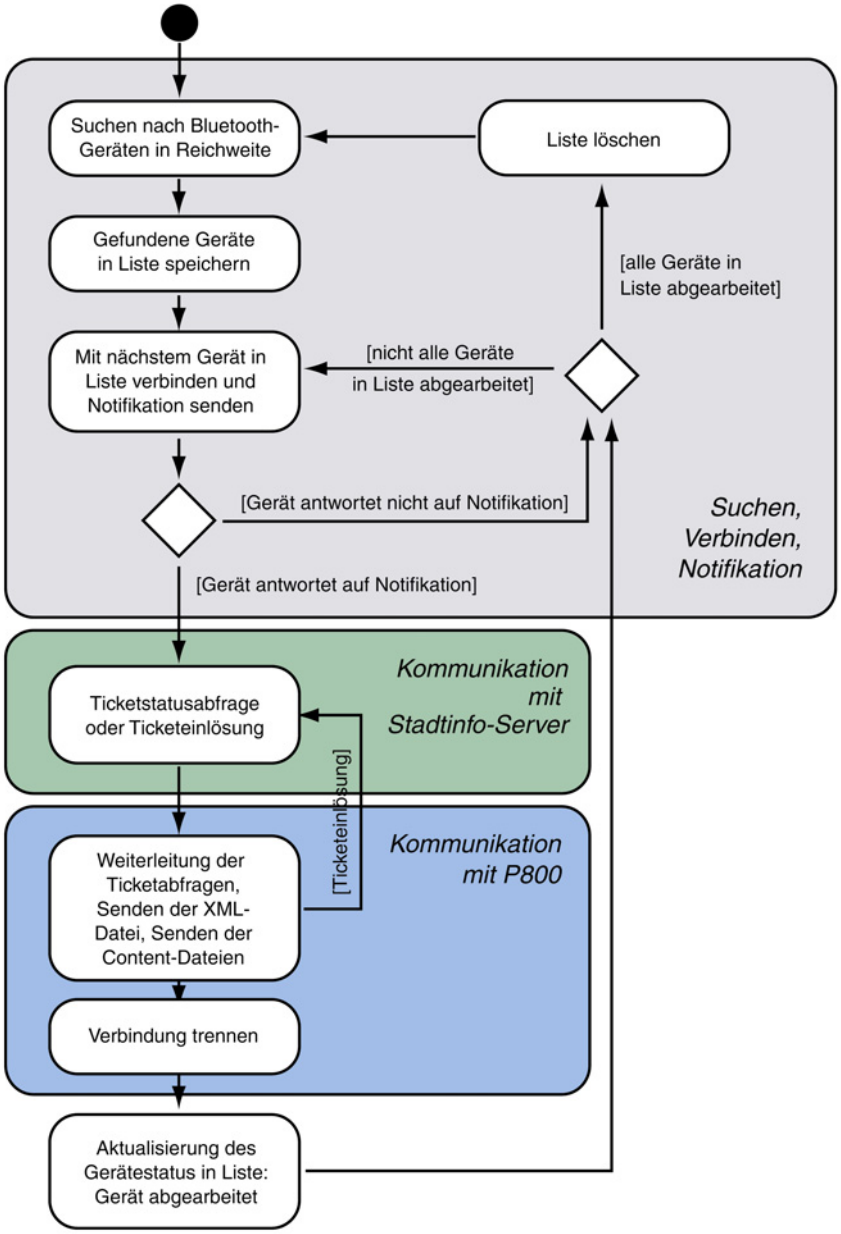


Abbildung 6-1: Activity-Diagramm MBT

Abbildung 6-1 gibt einen Überblick über den groben Ablauf der Anwendung auf dem MBT.
Für den Prototypen war vorgesehen, dass nur ein Mobiltelefon gleichzeitig mit dem MBT verbunden sein soll. Die Anwendung wurde daher in Form eines einzigen Tasks implementiert, der asynchron mehrere Verbindungen nutzt:[117]

- Eine Verbindung zum Bluetooth Protocol Manager zum Suchen und Verbindungsaufbau.
- Eine Bluetooth-Daten-Verbindung für Notifikation und Datenaustausch mit dem P800.
- Eine HTTP-Verbindung für die Kommunikation mit dem Stadtinfo-Server.
- Eine Verbindung zum File Manager, um die Dateien vom virtuellen Flash-Dateisystem lesen zu können.
- Eine Verbindung zum Time Service, um Timeouts definieren zu können wie z. B. die Zeit, wie lange auf eine Antwort auf die Notifikation gewartet werden soll.

Auf den genauen Ablauf der Kommunikation und auf die Datenübermittlung soll nun genauer eingegangen werden.

Aktives Suchen von Bluetooth-Geräten in Reichweite und Verbindungsaufbau

Die Bluetooth API des MBT bietet mit der Funktion „`BT_hci_retrieve_devices()`" die Möglichkeit, aktiv nach Bluetooth-Geräten zu suchen. Jedes gefundene Gerät wird mit dem Event `EVT_GAP_INQUIRY_RESULT` angezeigt. Die Anwendung speichert das Gerät daraufhin in einer Liste. Diese Liste enthält außer der Bluetooth-Adresse noch eine Flag namens „`processed`". Diese Flag wird aktualisiert, sobald das Gerät abgearbeitet wurde.
Nach einer vom System vorgegebenen Zeit wird der Suchvorgang beendet. Nun versucht das MBT, sich mit „`BT_spp_connect(...)`" nacheinander mit den Geräten zu verbinden.
Um eine erfolgreiche Verbindung herstellen zu können, muss der beim MBT so genannte „SppServerChannel" mit den serverseitigen Einstellungen auf dem P800 übereinstimmen.[118] Bei der Stadtinfo-Anwendung wurde beispielsweise der Channel mit der ID fünf verwendet. So kann schon von vorne herein eine ganze Reihe von Geräten ausgeschlossen werden, die auf diesem Kanal keine Verbindungen annehmen.
Wurde eine erfolgreiche Verbindung aufgebaut, so wird eine Message des Typs `CONN_ACK` für die neue Bluetooth-Daten-Verbindung generiert. Daraufhin sendet das MBT die Stadtinfo-Notifikation.[119] Wird auf diese innerhalb einer eingestellten Zeit nicht geantwortet, so wird die Verbindung wieder abgebaut und das nächste Gerät in der Liste kontaktiert. So können weiterhin alle Geräte vom Stadtinformationsservice ausgeschlossen werden, die zwar eingehende Verbindung auf dem richtigen Kanal entgegennehmen, aber auf die dann eingehende Notifikation nicht korrekt antworten.
Auch ein bereits mit Daten versorgtes P800 wird am gleichen Standort erneut notifiziert. Erkennt dieses aufgrund der mit der Notifikation übermittelten Standort-ID, dass es bereits alle Informationen des aktuellen Standorts heruntergeladen hat, fordert es mit einer entsprechenden Nachricht das MBT auf, die Verbindung abzubauen.[120]

[117] Vgl. Kapitel 4.3.2.
[118] Vgl. Kapitel 6.2.1.
[119] Vgl. Kapitel 6.4.1.
[120] Vgl. Kapitel 6.2.1.

Schnittstelle zum Stadtinfo-Server für Ticketabfragen

Sollen Informationen vom aktuellen Standort heruntergeladen werden, so antwortet das kontaktierte Gerät auf die Notifikation mit einer Ticketstatusabfrage. Das MBT leitet hierauf über die Ethernet-Schnittstelle eine HTTP-Verbindung zum Stadtinfo-Server ein. Für das MBT steht dafür eine recht einfach einzusetzende API zur Verfügung. Per HTTP-Request („`Get`“) wird dann auf dem Server ein PHP-Skript aufgerufen.
Die Anfrage des MBT an den Server sieht folgendermaßen aus:

```
http11_get_resource(url, Stadtinfo_Srv_IPAddr, Stadtinfo_Srv_Port, ...);
```

Interessant hierbei ist die angegebene URL. Sie setzt sich im Prototypen aus dem Pfad des PHP-Skriptes und den Übergabeparametern zusammen:

```
"/mbt/mbt.php?btaddr=00:06:A5:33:07:8B?pin=123456?locid=2"
```

Hier wird beispielsweise die Bluetooth-Adresse des P800 angegeben, mit dem das MBT gerade verbunden ist, sowie die vom Nutzer eingegebene Ticketnummer („`pin`“) und die Standort-ID „`2`“. Eine erfolgreiche Antwort des Servers wird dem Stadtinfo-Task mit „`DATA_IND`“ angezeigt und anschließend mit dem Befehl „`http11_dispatch_reply(...)`“ aus der Message Queue ausgelesen. Unabhängig davon, ob es sich bei der Anfrage um eine Ticketstatusabfrage oder eine Anfrage zur Ticketeinlösung handelt gibt es insgesamt sechs mögliche Antworten des Servers, die dem P800 zur weiteren Verarbeitung weitergeleitet werden:

- Bluetooth-Gerät unbekannt:
  Diese Antwort wird gegeben, wenn der Tourist erstmalig eine Sehenswürdigkeit des Stadtinformationsservice erreicht. Die Bluetooth-Adresse ist in der Datenbank noch nicht eingetragen und der Tourist wird daraufhin gebeten, eine Ticketnummer einzugeben.
- Gültiges Ticket:
  Berechtigt zum Daten-Download. Das P800 fordert nun die gewünschten Daten an.
- Ungültiges Ticket:
  Die eingegebene Ticketnummer wurde in der Datenbank nicht gefunden, wurde also auch nicht bezahlt. Der Tourist wird erneut zur Eingabe einer Ticketnummer aufgefordert.
- Ticket ist abgelaufen:
  Ein Ticket ist immer nur eine begrenzte Zeit lang gültig. Der Tourist wird aufgefordert, eine neue Ticketnummer einzugeben.
- Ticket ist bereits auf ein anderes Bluetooth-Gerät angemeldet:
  Die Bluetooth-Adresse ist weltweit eindeutig. Dies soll verhindern, dass beispielsweise ein Ticket zweimal von unterschiedlichen Touristen benutzt werden kann.
- Fehler des Servers:
  Diese Antwort wird nicht sofort an das P800 weitergeleitet, sondern die Anfrage an den Server noch bis zu drei Mal wiederholt. Sollte der Server jedes Mal mit einem Server-Fehler antworten, so wird der Tourist mit einer entsprechenden Meldung darüber informiert. Er kann nun versuchen, sein Ticket erneut einzugeben oder den Vorgang abbrechen.

Bereitstellen von Informationen

Wird ein gültiges Ticket signalisiert, so kann der Tourist die lokal auf dem MBT gespeicherten Informationen abrufen. Je nach bereits auf dem P800 vorhandenen Daten wird entweder die XML-Datei oder eine Content-Datei angefordert. Bei der Stadtinfo-Anwendung sind die Dateien in einer virtuellen Ordnerstruktur abgelegt, wobei sich die XML-Datei „`Stadtinfo.xml`" immer im Root-Verzeichnis „`/flash/`" befindet. Diese XML-Datei enthält alle notwendigen Angaben über die verfügbaren Content-Dateien, welche je nach Medientyp in Ordnern abgelegt sind, wie z. B. Text oder Audio.[121] Um die angeforderte Datei zu senden, wird die Verbindung zum File Manager genutzt. Zunächst wird die Länge der Datei ermittelt, die Datei dann geöffnet, gelesen und dann über die Bluetooth-Daten-Verbindung an das P800 versandt.
Während der Entwicklung des Prototypen wurde festgestellt, dass beim Versenden größerer Dateien über RFCOMM Probleme entstehen können. Normalerweise sollte die Segmentierung und Desegmentierung von der L2CAP-Ebene des Protokoll-Stacks übernommen werden. Dabei gingen jedoch leider Pakete verloren. Die Dateien werden daher nun auf Anwendungsebene paketweise vom Dateisystem gelesen und über Bluetooth versandt. Das P800 setzt die empfangenen Pakete wieder zu einer Datei zusammen.

Timeouts

Auch der Time Service des MBT wurde häufig eingesetzt, um Timeouts definieren zu können, damit die Anwendung nicht zum Stillstand kommt. Für folgende Vorgänge wurden daher angemessene Zeiten festgelegt:

- Verbindungsherstellung zu dem zu kontaktierenden Gerät
- Antwort auf die Stadtinfo-Notifikation
- Timeouts vor Versenden von Dateien und vor Verbindungsabbau:
  Eine Anfrage vom P800 resultiert oft in einer direkten Antwort des MBT. Es stellte sich heraus, dass das P800 in diesen Fällen nicht korrekt erkannte, dass die Anfrage vom MBT bereits angenommen wurde und der Nachrichtenaustausch abgeschlossen ist. Die einzige Lösung dieses Problems bestand nun darin, eine kurze Verzögerung einzuführen, bevor beispielsweise die XML-Datei oder eine Content-Datei auf die Anfrage des P800 hin versandt werden.
  Um genau herauszufinden, ob der Fehler in einer inkorrekten Protokoll-Implementierung des MBT oder beim P800[122] liegt, müsste der Bluetooth-Datenaustausch mit einem Bluetooth-Analyzer[123] untersucht werden, was aus Zeitgründen noch nicht durchgeführt werden konnte. Um die Stabilität einer Finalversion garantieren zu können wird dies jedoch unerlässlich sein.

Weitere Zeiten werden bereits vom System vorgegeben, wie z. B. beim Suchvorgang nach Bluetooth-Geräten oder Verbindungsaufbau der HTTP-Verbindung und erzeugen eine entsprechende Message, auf die reagiert werden kann. Die Einführung manueller Timeouts war in diesen Fällen folglich nicht notwendig.

[121] Vgl. Kapitel 6.4.2.
[122] Anfragen auf dem P800 sind in Form von Active Objects implementiert, vgl. Kapitel 6.2.1.
[123] Vgl. Internetquelle Bluetooth-Analyzer-Vergleich.

6.2 Die Stadtinfo-Anwendung auf dem P800

Das P800 ist für den Nutzer nach dem Ticketkauf die zweite und wichtigste Schnittstelle zum Stadtinformationsdienst. Die Ticketeinlösung und die Darstellung der Informationen sollte für den Nutzer so einfach wie möglich gestaltet werden. Die Kommunikation mit dem MBT sollte für den Nutzer unbemerkt ablaufen und ihn in der Bedienung nicht einzuschränken.
Das P800 wird also von zwei externen Systemen beeinflusst und muss darauf entsprechend reagieren. Diese beiden Vorgänge laufen teilweise asynchron ab und stellen damit gewisse Anforderungen an das System. Konzeption und Implementierung des P800 nahm somit den Großteil der Zeit in Anspruch und soll daher in den folgenden Kapiteln eingehend erläutert werden.

Die Grundlagen, Konzepte und Möglichkeiten des Betriebssystems Symbian OS hatten starken Einfluss auf die Konzeption und Implementierung der Anwendung:

- Objekte beinahe aller Klassen wurden nach dem Two-Phase-Construction-Prinzip erzeugt.
- Active Objects wurden häufig eingesetzt, unter anderem beruht die Bluetooth Engine komplett auf diesem Konzept. So war es möglich, Nutzerinteraktion und Bluetooth-Kommunikation in nur einem Anwendungs-Thread zu behandeln.
- Deskriptoren fanden bei fast jeglicher Datenverarbeitung Verwendung, speicherintensive Objekte wurden auf dem Heap angelegt, kleine Objekte auf dem Stack.
- Sämtliche Texte wurden in einer Ressourcen-Datei ausgelagert, um später eine mehrsprachige Umsetzung zu erleichtern.
- Die Application-Datei wurde als polymorphische DLL gemäß dem Application Framework erstellt.

Model-View-Controller-Prinzip (MVC)

Das Symbian-Framework gibt für die Erstellung einer Anwendung mit grafischer Benutzeroberfläche mindestens vier Klassen vor (Application, Document, Application UI, mindestens ein View).[124] Die Stadtinfo-Anwendung wurde darauf aufbauend nach dem Model-View-Controller-Prinzip erstellt:[125]

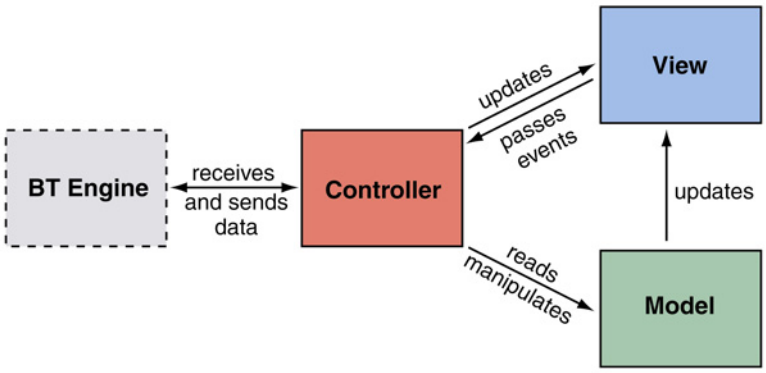


Abbildung 6-2: MVC-Prinzip bei der Stadtinfo-Anwendung auf dem P800

[124] Vgl. Kapitel 3.4.5.
[125] Vgl. Digia Inc. (2003), S. 38-41.

Dabei werden die Aufgaben der Software auf drei Komponenten verteilt:

- Model: Das Model enthält und bearbeitet die Daten des Programms.
- View(s): Ein View stellt dem Nutzer die Daten des Models dar.
- Controller: Legt fest, wie auf eingehende Events reagiert werden soll und steuert die Anwendung.

Dieses Prinzip bringt einige Vorteile mit sich:

- Übersichtlichkeit der Anwendung:
  Die Aufgaben sind klar definiert und die Kommunikation unter den Komponenten erfolgt über klar definierte Schnittstellen.
- Portierbarkeit auf andere Systeme:
  Durch das MVC-Prinzip ist das User-Interface von der Programmlogik klar getrennt, so dass ein Großteil der Anwendung bei der Anpassung auf ein anderes Symbian-Mobiltelefon beispielsweise nicht geändert werden muss.

Das MVC-Prinzip bei der Stadtinfo-Anwendung auf dem P800

Gerade beim stark eventbasierten Symbian OS spielt das Nutzer-Interface eine große Rolle. Der View übernimmt in der Stadtinfo-Anwendung daher mehr Aufgaben wahr als nur die Darstellung der Daten. Eingabemöglichkeiten wie Touchscreen oder das Rad an der Seite des Mobiltelefons, sowie Software-Bedienelemente wie die Toolbar sind versionsstyle-spezifisch, sollten also innerhalb des Views behandelt werden. Alle Eingaben, die beispielsweise über den Touchscreen erfolgen, werden im View entgegengenommen und müssen bei einem Mobiltelefon ohne Touchscreen entsprechend anders gelöst werden. Die Weiterleitung der Daten wie z. B. bei Eingabe der Ticketnummer erfolgt an den Controller, der dann für die weitere Programmlogik zuständig ist.
Wie in Abbildung 6-2 ersichtlich, wurde das MVC-Prinzip in Richtung Bluetooth-Kommunikation erweitert: Der Controller in der Stadtinfo-Anwendung reagiert nicht nur auf Nutzereingaben, sondern auch auf einkommende Daten über Bluetooth.

Kommunikation der Klassen

Die Kommunikation der Klassen erfolgt meist über ein dem Observer-Pattern ähnliches Prinzip.[126]

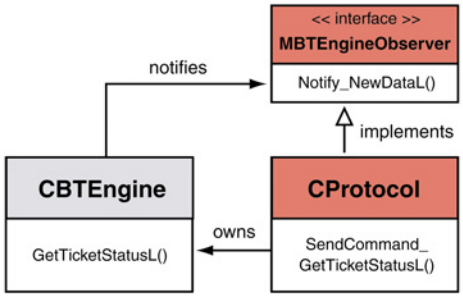


Abbildung 6-3:
Kommunikation der Klassen

Ein Beispiel: Soll der Ticketstatus erfragt werden (`SendCommand_GetTicketStatusL`), so ruft die Protokollklasse die Methode `WriteL()` der Bluetooth Engine und übergibt die zu sendende Nachricht. Die Bluetooth Engine „schreibt" diese dann auf die Bluetooth-Verbindung. Empfängt die Bluetooth Engine dann die Antwort als neu eintreffende Daten, notifiziert sie über den Befehl `BT_Notify_NewDataL()` den Bluetooth Engine Observer. Die Protokollklasse wiederum implementiert diese Methode und wertet die eingehenden Daten aus.

[126] Vgl. Gamma et al. (1995), S. 293-394 und vgl. Digia Inc. (2003), S. 42-43.

Software-Gesamt-Design P800

Abbildung 6-4 gibt einen Überblick über die wichtigsten Elemente der Stadtinfo-Anwendung auf dem P800. Die Hauptsteuerung des Programms übernimmt der Main Controller: Einkommende Daten über die Protokollklasse werden meist an das Model weitergegeben und der View entsprechend notifiziert, Nutzereingaben vom View Controller werden z. B. an die Protokollklasse weitergeleitet.

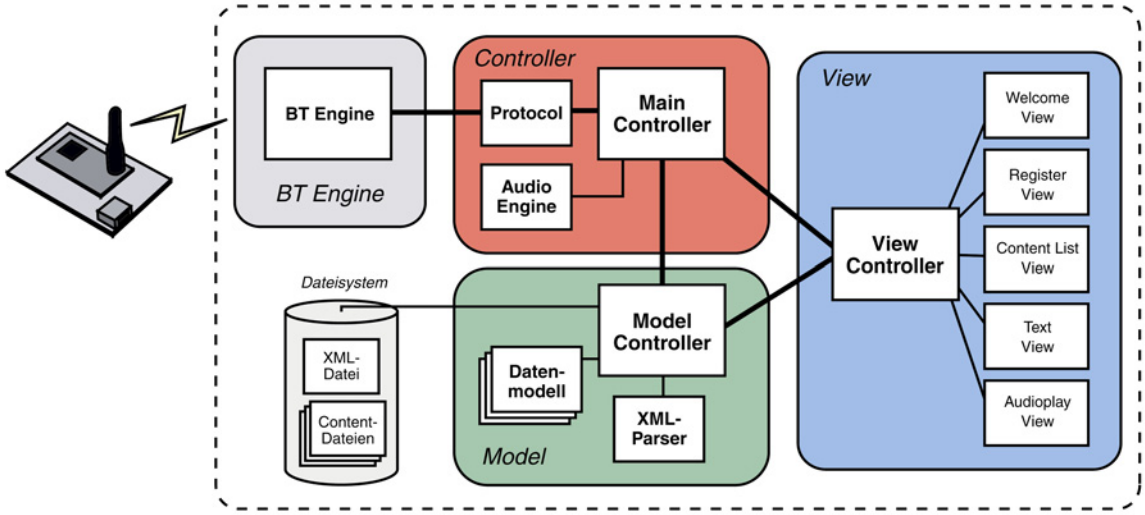


Abbildung 6-4: Software-Gesamt-Design P800

Ein Klassendiagramm mit den wichtigsten Funktionen befindet sich im Anhang.
Im Folgenden sollen nun die einzelnen Klassen genauer beschrieben werden.

6.2.1 Bluetooth Engine

Die Bluetooth Engine ist die Schnittstelle des Programms zum MBT und damit für die Bluetooth-Kommunikation zuständig. Die Bluetooth Engine besteht aus drei Klassen (Bluetooth Engine, Bluetooth Reader und Bluetooth Writer) und übernimmt folgende Aufgaben:

- Vornahme der Grundeinstellungen für Bluetooth
- Warten auf eine eingehende Verbindung
- Lesen der einkommenden Daten und Weiterleitung an die Protokollklasse
- Senden von Daten an das MBT
- Zurücksetzen der Verbindung

Dies sind hauptsächlich asynchrone Vorgänge. Das Warten auf eingehende Verbindungen, das Lesen und das Senden von Daten wurden daher mit Hilfe von Active Objects implementiert.
Wie bereits erwähnt, stellt Symbian OS für Bluetooth weder einen eigenen Kommunikations-Server noch eine eigene Bluetooth API zur Verfügung. Es müssen daher unterschiedliche APIs eingesetzt werden, wie es in den folgenden Absätzen beschrieben wird. Die wichtigste API hierbei ist die Socket Server API.

Grundeinstellungen für Bluetooth

Die Bluetooth-Kommunikation unter Symbian OS basiert auf dem Modell der Socket-Kommunikation. Sockets sind die Kommunikationsendpunkte, auf denen die Verbindung aufsetzt. Sockets haben gegenüber z. B. einer seriellen Verbindung den Vorteil, dass eine Fehler- und Flusskontrolle integriert werden kann.
Um eine Bluetooth-Verbindung herstellen zu können, muss zunächst eine Verbindung zum Socket Server hergestellt werden:

```
iSocketServ.Connect();
```

Der Socket Server unterstützt eine Vielzahl von Protokollen für Clients und bietet Funktionen für die Initialisierung, Konfiguration und Nutzung von Sockets. Nachdem die Verbindung zum Socket Server hergestellt wurde, wird Bluetooth über die API des so genannten Bluetooth-Managers konfiguriert und in der Service-Datenbank angemeldet:

- Festlegen der Channel ID für den Bluetooth-Service:

  ```
  iBtService.SetChannelID(iChannelID);
  ```

  Diese Channel ID entspricht bei RFCOMM dem DLCI, also dem Kanal einer emulierten seriellen Verbindung. Sie muss für einen erfolgreichen Verbindungsaufbau mit dem MBT mit dem dort eingestellten so genannten „SppServerChannel" übereinstimmen.
- Festlegen des Protokolls:

  ```
  iBtService.SetProtocolID(KSolBtRFCOMM);
  ```

  Das RFCOMM-Protokoll wird geladen, da auf dessen Ebene der spätere Datenaustausch beruhen soll. Die Kombination Channel-ID und Protokoll kann es immer nur einmal geben.
- Festlegen der Sicherheitseinstellungen:
  Über die API des so genannten Security Manager wird festlegt, welche Sicherheits-anforderungen eine eingehende Verbindung bezüglich Authentifizierung, Autorisierung und Verschlüsselung erfüllen muss.

Warten auf eine eingehende Verbindung

Wie bereits erwähnt, wird der Verbindungsaufbau vom MBT eingeleitet. So muss das P800 nicht aktiv nach Geräten in Reichweite suchen, was unnötig Ressourcen beanspruchen würde. Das P800 agiert also als Server und wartet auf eine Verbindungsanfrage. Dieses passive Warten wurde mit Hilfe des Active Object Frameworks realisiert, um währenddessen keine anderen laufenden Anwendungen zu blockieren.
Die Bluetooth-Adresse des MBTs ist dem P800 zunächst unbekannt. Um eine erfolgreiche Verbindung mit einem Gerät mit unbekannter Adresse herstellen zu können, werden zwei Sockets benötigt:

- Ein „Listening Socket" mit dem gewünschten Protokoll:
  Zunächst wird ein Socket mit dem RFCOMM-Protokoll geöffnet und mit dem Socket Server verbunden. Dieser Socket dient als „Listener". Er hört auf einem bestimmten so genannten „Port" auf eine oder mehrere Verbindungen. Der Port und die Anzahl der Verbindungen können eingestellt werden. Für eine erfolgreichen Datenaustausch muss der Port der vorher eingestellten Service Channel ID entsprechen.
- Ein „freier" Socket ohne Protokollkonfiguration:
  Über diesen Socket erfolgt später der eigentliche Datenaustausch.

Mit „Accept()“ wird dann vom Listening Socket eine eingehende Verbindung vom MBT angenommen, mit dem freien Socket verbunden und der Socket Channel erzeugt. Der Quellcode dazu sieht wie folgt aus:

```
1: iListenSocket.Open(iSocketServ, _L("RFCOMM"));
2: iListenSocket.SetLocalPort(KPort);   // KPort muss der Channel ID des
3:                                      // Bluetooth-Service entsprechen
4: iListenSocket.Listen(1);  // es soll nur eine Verbindung angenommen werden
5: iBTSocket.Open(iSocketServ);   // freier Socket ohne Protokollkonfiguration
6: iListenSocket.Accept(iBTSocket, …);
7: SetActive();
```

Wie am letzten Befehl zu sehen ist, werden Active Objects eingesetzt. Sobald der Befehl `Accept()` durch eine eingehende Verbindung ausgeführt wurde, wird in der `RunL()`-Methode der Bluetooth Engine entsprechend reagiert. Bei erfolgreichem Verbindungsaufbau wird dann beispielsweise das Lesen von der Verbindung eingeleitet.

Lesen der einkommenden Daten und Weiterleitung an die Protokollklasse

Für das Lesen der Daten wurde bei Erzeugung der Bluetooth Engine ein weiteres Active Object erzeugt: der „Bluetooth Reader“. Sobald eine Verbindung besteht, wird dieser zum ersten Mal aktiviert.
Es gibt unter Symbian OS mehrere Möglichkeiten, eingehende Daten zu lesen, immer muss jedoch ein Puffer mit fester Größe angelegt werden, der die Daten aufnimmt:

- Mit dem Befehl „`Read()`“ wird so lange auf Daten gewartet, bis dieser Puffer gefüllt ist und dann die `RunL()`-Methode des Bluetooth Readers gerufen. Diese Möglichkeit wurde zuerst verwendet. Bei der Entwicklung des Protokolls für den Stadtinformationsdienst wurde daher vorgesehen, in einem ersten Feld mit definierter Größe die Länge der zu übermittelnden Daten zu senden. Anhand dieser dann bekannten Länge sollte der Puffer für die restlichen Daten angelegt und gefüllt werden. Nach einigen Versuchen stellte sich jedoch heraus, dass der in seiner Größe eigentlich fest eingestellte Deskriptor für die eingehenden Daten vom System teilweise erweitert wurde. Somit konnte nie sicher gesagt werden, ob die Daten korrekt eingelesen wurden oder der Deskriptor eventuell mehr als nur das erste Feld enthielt. Dies machte eine Auswertung unmöglich und führte zur Verwendung eines anderen Befehls und der Einführung einer Flusskontrolle.
- Im Prototypen wird nun der Befehl „`RecvOneOrMore()`“ eingesetzt. Mit diesem Befehl werden so viele Bytes vom Bluetooth-Eingang gelesen wie eben verfügbar sind und dann die `RunL()`-Methode gerufen. In der `RunL()`-Methode wird dann untersucht, ob ein zur Flusskontrolle eingeführtes Trennzeichen in den Daten enthalten ist. So kann die Länge der zu übermittelnden Daten sicher ermittelt werden. Alle weiteren eingehenden Daten nach dem Trennzeichen werden dann so lange in den Deskriptor geschrieben, bis alle Bytes angekommen sind. Dieser Puffer wird daraufhin zur Auswertung an die Protokollklasse übergeben.

Wie beim Warten auf eingehende Verbindungen wurde auch beim Lesen das Active Object Framework eingesetzt. So kann der Bluetooth Reader während einer bestehenden Verbindung ständig im Hintergrund bereit sein, ohne andere Vorgänge zu blockieren.

Senden von Daten an das MBT

Das Senden von Daten an das MBT übernimmt der so genannte „Bluetooth Writer". Dieser Vorgang wird immer von der Protokollklasse ausgelöst, die den Bluetooth Writer über die Bluetooth Engine aktiviert. Auch das Senden ist ein asynchroner Vorgang und mit Hilfe von Active Objects realisiert, das heißt nach erfolgreichem Datenversand wird die `RunL()`-Methode des Bluetooth Writers gerufen.
Der Bluetooth Writer verursachte am meisten Probleme bei der Umsetzung der Stadtinfo-Anwendung. Das Senden sollte spätestens dann beendet sein, wenn der gesamte übergebene Puffer versandt wurde. Dies war jedoch nicht der Fall. Die Daten kamen auf dem MBT an und führten zu einer prompten Antwort des MBTs. Der Status des Bluetooth Writers wurde jedoch immer noch als aktiv angesehen und somit die `RunL()`-Methode auch nicht gerufen. Versuche, das Rufen der `RunL()`-Methode mit speziellen Befehlen zu erzwingen, führten ebenfalls nicht zum Erfolg, ebenso wenig wie das Verwenden des alternativen Befehls „`Send()`" statt „`Write()`". Der Status des Active Objects wurde vom Framework nicht zurückgesetzt.
Dieses Problem führte zu der bereits beschriebenen eher uneleganten Lösung: Nach Eintreffen der Daten auf dem MBT wird die entsprechende Antwort nun nicht sofort geschickt, sondern erst nach einer bestimmten Zeit. Durch Einführung dieser Verzögerung erkennt das P800 nun das erfolgreiche Schreiben an und die `RunL()`-Methode wird gerufen. Die Verzögerung ist oft schon vom System her bedingt: Bei einer Ticketabfrage beispielsweise erfolgt ohnehin erst eine Anfrage an den Stadtinfo-Server. Dies dauert lange genug, so dass das P800 den Schreibvorgang als beendet ansieht. An anderer Stelle greift die künstliche Verzögerung jedoch immer noch, wie beispielsweise bei Anfragen zum Download von Content-Dateien.

Zurücksetzen der Verbindung

Sind alle Daten ausgetauscht, wird die Verbindung wieder freigegeben, damit sich das MBT mit dem nächsten Gerät verbinden kann. Hierbei muss die Anwendung auf dem P800 das Zurücksetzen der Verbindung einleiten, da nur diese erkennen kann, ob alle Daten erfolgreich heruntergeladen wurden. Das Trennen der Verbindung vom P800 aus erforderte jedoch, wie in Versuchen herausgefunden wurde, recht viel Zeit. In dieser Zeit war das Mobiltelefon für den Benutzer nur eingeschränkt bedienbar. Es wurde daher eine andere Lösung gefunden: Das P800 sendet nun einen speziellen Steuerbefehl an das MBT, das daraufhin die Verbindung abbaut. Dieses Vorgehen führte zu keinen Einschränkungen für den Benutzer und wurde daher im Prototypen eingesetzt. Nach Absenden dieser Aufforderung zum Verbindungsabbau werden die Sockets wieder geschlossen.

Die Bluetooth Engine kapselt also Verbindungsauf- und abbau, das Lesen und Senden von Daten. Da beim Stadtinformationsdienst recht viele Anfragen und Dateien zwischen MBT und P800 ausgetauscht werden, wurde ein spezielles Stadtinfo-Protokoll entworfen.[127] Die Umsetzung dieses Protokolls auf dem P800 übernimmt die ebenfalls zum Controller-Bereich gehörende Protokoll-klasse.

[127] Vgl. Kapitel 6.4.1.

6.2.2 Protokollklasse

Die Protokollklasse übernimmt die Protokoll-Implementierung und hat somit folgende Aufgaben:

- Auswertung der von der Bluetooth Engine übermittelten Daten:
  Die von der Bluetooth Engine weitergeleiteten Daten werden gemäß dem Stadtinfo-Protokoll interpretiert und die entsprechenden Befehle an den Main Controller weitergeleitet. So wird beispielsweise der Main Controller benachrichtigt, wenn das MBT die Gültigkeit des Tickets signalisiert hat. Bei der XML-Datei und den Content-Dateien werden nur die reinen Daten weitergeleitet.
- Umwandlung der Befehle vom Main Controller in ausgehende Nachrichten:
  Befehle vom Main Controller, die an das MBT gesandt werden sollen, werden in die entsprechenden Nachrichten des Stadtinfo-Protokolls umgesetzt. So wird zum Beispiel eine Ticketaktivierungsanfrage in den richtigen Steuerbefehl umgesetzt oder die nächste Content-Datei angefordert.

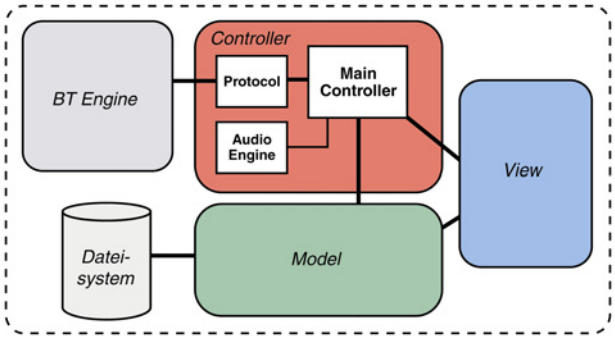


Abbildung 6-5: P800-Softwaredesign mit Focus Controller

6.2.3 Main Controller

Der Main Controller steuert den Ablauf der gesamten Stadtinfo-Applikation. Er übernimmt daher folgende Aufgaben:

- Initialisierung des Models
- Starten des „Listening" der Bluetooth Engine
- Steuerung des Informationsabrufes über Bluetooth
- Benachrichtigung des Nutzers mit Vibration und Signalton
- Einleitung des Verbindungsabbaus

Initialisierung des Models

Bei Programmstart wird die Initialisierung des Models eingeleitet. Hierbei wird die für das Programm benötigte Ordnerstruktur im Dateisystem angelegt bzw. abgefragt, welche Dateien schon geladen wurden. Die Daten des Models werden dementsprechend angelegt.

Starten des Listenings der Bluetooth Engine

Der Main Controller steuert die Bluetooth Engine und leitet das Warten auf eingehende Verbindungen ein. Dies geschieht zum einen beim Start der Anwendung, zum anderen nach erfolgreichem Download aller Dateien an einem Standort. Der Main Controller nutzt hierzu einen Timer, der als Active Object implementiert ist und ihn über einen Observer notifiziert, sobald die vorgegebene Zeit abgelaufen ist. Daraufhin startet der Main Controller das Listening der Bluetooth Engine erneut, damit eine neue Stadtinfo-Notifikation empfangen werden kann.

Steuerung des Informationsabrufes über Bluetooth

Der Informationsabruf erfolgt in der Regel in drei Schritten:

1. Überprüfung des Ticketstatus:
   Wurde eine Stadtinfo-Notifikation empfangen oder wird vom View Controller eine Ticketnummer übergeben, so leitet der Main Controller die entsprechende Anfrage ein und reagiert auf die Server-Antworten. Wird beispielsweise ein Fehler gemeldet, so wird die passende Meldung an den View Controller weitergeleitet, der diese dann dem Touristen anzeigt. Als Fehlertext wird der optional mitübertragene Text angezeigt oder ein Standard-Fehlertext aus der Ressourcendatei der jeweiligen Sprache gelesen.[128] So können zumindest einsprachig auch Fehler angezeigt werden, die bei einer Finalversion erst nach Installation der Anwendung auf dem Mobiltelefon des Touristen erkannt werden.
2. Download der XML-Datei:
   Der Main Controller leitet die Anforderung der XML-Datei und leitet diese nach Empfang zur Speicherung und Auswertung an das Model weiter.
3. Download der Content-Dateien:
   Wird erkannt, dass noch nicht alle Content-Dateien des Standortes heruntergeladen wurden und ist ein gültiges Ticket vorhanden, so wird vom Main Controller der Content-Datei-Download gestartet. In der betreffenden Funktion wird zunächst vom Model die nächste Datei erfragt, die noch nicht vom MBT heruntergeladen wurde. Diese Datei wird dann angefordert und nach erfolgreichem Download an das Model zur Speicherung weitergeleitet. Daraufhin wird der Content-Datei-Download erneut gestartet. So werden nacheinander alle Dateien geladen und der Nutzer durch Vibration und Signalton davon benachrichtigt.

Benachrichtigung des Nutzers mit Vibration und Signalton

Der Main Controller übernimmt auch eine Aufgabe, die den Benutzer betrifft. Diese dient jedoch der Information des Benutzers und ist unabhängig von den Eingabemöglichkeiten des Mobiltelefons: Um den Touristen, der sein Mobiltelefon während seiner Stadterkundung am ehesten in einer Tasche hat, zum richtigen Zeitpunkt auf den Stadtinformationsdienst aufmerksam zu machen, kann sowohl der Vibrationsalarm des P800 sowie auch eine akustischer Hinweis eingesetzt werden. In der Finalversion soll der Nutzer dies über Einstellungen festlegen können.
Zur akustischen Rückmeldung wird über die interne Medienplayer-API ein kurzer Signalton abgespielt. Zur Ansteuerung des Vibrationsalarms des P800 stand standardmäßig jedoch zunächst keine API zur Verfügung. Diese wird aber im Internet frei angeboten.[129] Die hierfür notwendige Datei muss daher bei der Installation der Stadtinfo-Anwendung mit auf das P800 übertragen werden.

[128] Im Prototypen wurden zunächst nur zwei Sprachdateien angelegt, für Deutsch und Englisch. Die englische wurde noch nicht übersetzt.
[129] Vgl. Internetquelle Symbian-Vibration-API.

6.2.4 Audio Engine

Der Stadtinformationsdienst soll dem Touristen Informationen aller Art zur Verfügung stellen. Darunter fallen Texte, Bilder, Audio-Dateien und Videos. Im Prototypen wurde die Anforderung auf Text und Ton konzentriert, wobei die Textdarstellung direkt vom View übernommen wird. Diese bleibt jedoch immer auf das Display beschränkt. Audioinformationen können beispielsweise auch per Headset angehört werden, während der Tourist sich die Sehenswürdigkeit ansieht und umherläuft. Auf Audioinformationen wurde daher beim Stadtinformationsdienst besonderer Wert gelegt.
Wichtig für die Stadtinfo-Applikation war folgendes:

- Die Dateien sollten möglichst klein sein. Auf MBT und P800 steht nur wenig Speicher zur Verfügung und auch die Übertragungszeit über Bluetooth sollte so kurz wie möglich sein.
- Die Dateien sollten von ansprechender Qualität sein. Ein Tourist, der für Informationen Geld gezahlt hat, erwartet dafür auch eine entsprechend gute Qualität. Diese kommt besonders zur Geltung, wenn die Dateien nicht über den eingebauten Lautsprecher, sondern über einen Kopfhörer abgehört werden.
- Die Dateien sollten direkt aus der Stadtinfo-Anwendung heraus abgespielt werden können.

Zunächst musste also eine Entscheidung über das Format der Audiodateien gefällt werden. Hier bot sich das weit verbreitete Mp3-Format an. Besonders bei Sprache können Mp3-Dateien stark komprimiert werden, ohne die Qualität entscheidend zu mindern. Das Mp3-Format konnte jedoch leider aus folgenden Gründen nicht eingesetzt werden:

- Auf dem P800 steht keine interne Standard-API zur Verfügung, mit der Mp3-Dateien wiedergegeben werden können. Der Standard-Medienserver unterstützt lediglich Gsm 6.10, AU, WAV, WVE and Raw-Audio-Dateien.[130] Alle diese Formate sind aber hinsichtlich der Dateigröße eher uninteressant.
- Der integrierte Mp3-Player kann aus einer anderen Anwendung heraus nicht gestartet werden. Hierfür wäre entweder eine API für die eingesetzte so genannte „Beatnik"-Audio Engine des Drittanbieters notwendig oder es müsste die Anwendungs-UID und der genaue Aufbau des so genannten DNL (Direct Navigation Link) bekannt sein, um eine Mp3-Datei direkt übergeben und starten zu können.[131] Beides ist aber nicht verfügbar.
- Die Herstellerfirma des integrierten Mp3-Players vertreibt ihre Beatnik-Audio Engine auch gegen Gebühr, verkauft die Lizenzen jedoch lediglich an Mobiltelefonhersteller.[132]
- Der offizielle Fraunhofer Mp3-Codec wurde bisher von Thomson/Fraunhofer noch nicht auf Symbian OS portiert, ein Einsatz wäre aus Kostengründen wohl auch nicht möglich gewesen.[133]

In der Testversion wurden also zunächst Wave-Dateien eingesetzt. Die Übertragung in der Praxis geschah etwa in Echtzeit, war also einigermaßen akzeptabel. Die Wave-Dateien waren jedoch um der Dateigröße willen Mono, mit nur 8 Bit quantisiert und mit 11 kHz gesampelt. Die Qualität ließ folglich stark zu wünschen übrig. Später fand sich eine sehr gute Alternative: Der Ogg-Vorbis-Codec. Dieser Codec ist frei verfügbar und bietet ähnlich dem Mp3-Codec eine sehr gute

[130] Vgl. Internetquelle Symbian-Unterstützte Audio-Formate.
[131] Mit diesen Angaben kann zwischen den Views verschiedener Anwendungen hin- und hergeschaltet und Parameter übergeben werden werden, vgl. Internetquelle Symbian-View Architecture.
[132] Vgl. Internetquelle Beatnik.
[133] Die Lizenzierungskosten für den PC-Software-Bereich liegen bei momentan 0,75 USD pro Lizenz oder einmalig 50.000-60.000 USD, vgl. Internetquelle Mp3-Lizenzierung.

Kompression von Audio-Dateien.[134] Programme zur Konvertierung in „.ogg"-Dateien stehen als Freeware zur Verfügung. Um die so erstellten Dateien in der Stadtinfo-Anwendung wiedergeben zu können, muss der Codec in Form einer DLL eingebunden werden. Die Audio Engine kann dann diese DLL nutzen, um die Ogg-Dateien zu dekodieren und in Form eines Audio-Streams an den internen Medien-Server weiterzuleiten.
Der Ogg-Player konnte aus Zeitgründen noch nicht in die Stadtinfo-Anwendung integriert werden. In einem Testprogramm auf dem P800 konnte aber unter Einbindung einer selbst kompilierten DLL bereits eine Ogg-Datei abgespielt werden. Als Test-Audio-Datei diente eine der erstellten Audiodateien mit Informationen über die Jubiläumssäule am Stuttgarter Schlossplatz. Die Qualität war überzeugend und die Dateigröße konnte gegenüber der qualitativ weitaus schlechteren Wave-Datei um etwa 60% reduziert werden.

6.2.5 Model Controller

Wie bereits erwähnt, ist das Model für die Daten der Anwendung zuständig. Der Model Controller übernimmt hierbei folgende Aufgaben:

- Initialisierung und Aktualisierung des internen Datenmodells
- Aktualisierung des Views
- Zugriff auf das Dateisystem

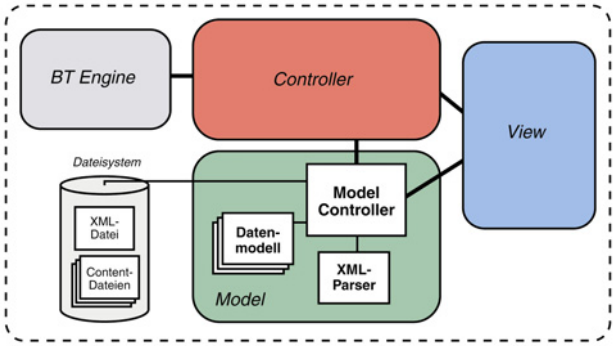


Abbildung 6-6: P800-Softwaredesign mit Focus Model

Initialisierung und Aktualisierung der Daten

Bei Programmstart wird das interne Datenmodell initialisiert. Hierzu werden zunächst im Dateisystem alle notwendigen Ordner angelegt, sofern diese noch nicht vorhanden sind. Ist bereits eine XML-Datei vorhanden, so wird diese mit Hilfe des XML-Parsers ausgewertet und das interne Datenmodell angelegt.
Hierbei werden folgende Daten gespeichert:

- ID des Standortes der gespeicherten XML-Datei:
  Diese ID wird bei eingehender Notifikation mit der Standort-ID der eingehenden Nachricht verglichen. So kann der Main Controller durch Abfrage des Models entscheiden, ob der Tourist sich an einer neuen Sehenswürdigkeit befindet und entsprechend reagieren.
- Name des Standortes:
  Dieser wird zur Darstellung gegenüber dem Touristen verwendet, z. B. „Schlossplatz"

[134] Informationen über das Projekt und Dateien zum Download finden sich unter Internetquelle OggVorbis.

Außerdem werden für jede Content-Datei folgende Daten angelegt:

- ID der Content-Datei: Diese enthält zur einfachen Verarbeitung auf MBT und P800 den Pfad und Dateinamen, z. B. „\Audio\Schlossplatz.wav".
- Name der Content-Datei: Dieser Name wird zur Darstellung verwendet, z. B. „Textinformation Schlossplatz".
- Typ der Content-Datei: Der Typ ist wichtig, um später den richtigen View zu aktivieren. Im Prototypen sind hierfür Textdateien vorgesehen, die im Textview dargestellt werden und Audiodateien, die über den Audioplay-View gesteuert werden.
- Download-Status: Speichert, ob die Datei bereits heruntergeladen wurde. Eine Abfrage des Download-Status der Content-Dateien im Datenmodell dient dem Controller zur Steuerung des Content-Datei-Downloads.
- Viewing-Status: Speichert, ob eine Datei bereits vom Benutzer angesehen bzw. angehört wurde. Der Viewing-Status wurde in der Anwendung noch nicht weiter verwendet.

Nach jedem erfolgreichen Download einer XML- oder Content-Datei werden die Daten entsprechend aktualisiert.
Außerdem werden im Model zwei weitere Variablen angelegt:

- Die Standort-ID der aktuellen Notifikation:
  Diese wird bei jeder neuen Stadtinfo-Notifikation aktualisiert. Durch Vergleich mit der Standort-ID aus der XML-Datei-Auswertung kann so ermittelt werden, ob sich der Tourist an einem neuen Standort befindet.
- Eine Flag:
  Die Flag gibt an, ob der Tourist gerade mit einem MBT verbunden ist („IsInHotSpot"). Diese wird zur Aktualisierung eines Antennensymbols genutzt, das dem Touristen eine aktive Verbindung anzeigt.

Aktualisierung des Views

Wird das interne Datenmodell aktualisiert, so betrifft dies meist auch den Benutzer. Daher benachrichtigt der Model Controller den View Controller, wenn beispielsweise eine Content-Datei erfolgreich gespeichert wurde, damit die Darstellung gegenüber dem Benutzer aktualisiert werden kann.

Zugriff auf das Dateisystem

Da auf dem P800 nur begrenzt Speicher vorhanden ist und immer nur die Informationen des jeweils aktuellen Standorts zur Verfügung stehen sollen, werden die alten Content-Dateien immer dann gelöscht, wenn eine neue XML-Datei erfolgreich heruntergeladen und gespeichert wurde. Auch das Öffnen von Dateien übernimmt der Model Controller. Soll beispielsweise die Audio Engine eine Datei zur Wiedergabe öffnen, so schickt der Main Controller eine entsprechende Anfrage an den Model Controller, der eine Referenz auf die Datei zurückgibt.

6.2.6 XML-Parser

Eine wichtige Aufgabe in der Stadtinfo-Anwendung kommt dem XML-Parser zu, der die „`Stadtinfo.xml`"-Datei auswertet. XML ist ein einheitlicher Standard und wurde daher auch im Stadtinfo-Projekt eingesetzt. So muss keine proprietäre Dateistrukur definiert werden, die von einem speziellen Parser auf dem P800 ausgewertet werden müsste. Der Aufbau von Dateien im XML-Format, sowie das Lesen und Parsen sind weithin bekannt und erleichtert eine Erweiterung des Protoypen für eine eventuelle Finalversion.
Zunächst wurde versucht, den internen XML-Parser zu nutzen. Dieser ist spezialisiert auf das SMIL-Format, welches auch bei MMS-Nachrichten eingesetzt wird.[135] Sollte ein Objekt der entsprechenden Klasse erzeugt werden, führte dies jedoch zu einem Fehler zur Laufzeit, so dass der interne XML-Parser nicht eingesetzt werden konnte.
Der nun programmierte XML-Parser durchsucht die XML-Datei nach Start-Tags, reinem Text und Ende-Tags und deren Attribute und ruft eine entsprechende Methode seines Observers. Das Observerinterface ist im Model implementiert und übernimmt die Auswertung des Tagnamens und der enthaltenen Attribute. In der XML-Datei der Stadtinfo-Anwendung sind nur Start-Tags und deren Attribute interessant, so dass reiner Text und Ende-Tags nicht ausgewertet wurden. Übergibt der Parser z. B. den Tagnamen „Content", so werden die mitübergebenen Attribute nach „ID", „Name" und „Type" untersucht und das Datenmodell vom Model entsprechend angelegt.
Als Vorlage diente hierbei ein in Java programmierter eventgesteuerter XML-Parser.[136]

6.2.7 View Controller

Der dritte Bereich im MVC-Design ist der View. Wie bereits beschrieben, bildet der View die Schnittstelle zum Benutzer. Er stellt die Daten dar und nimmt Benutzereingaben entgegen.

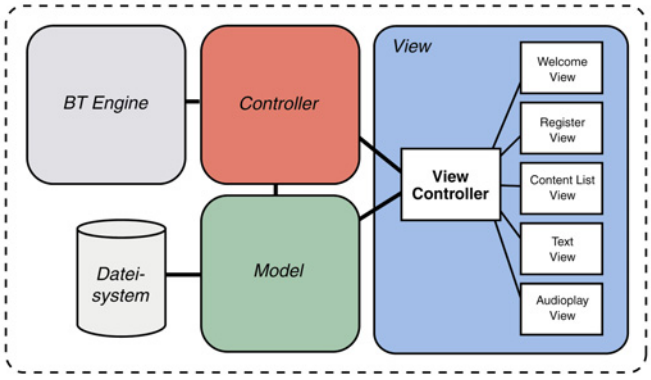


Abbildung 6-7: P800-Softwaredesign mit Focus View

Die Ein- und Ausgabemöglichkeiten des Benutzers sind jedoch abhängig von der Version des Betriebssystems. Im UIQ-Design[137] des P800 werden beispielsweise Toolbars und Zeichenstift-eingabe unterstützt.
Ein wichtiges Ziel der Trennung der Bereiche in Model, View und Controller ist eine einfache Portierbarkeit auf andere Systeme. Der View muss angepasst werden an die jeweilige Plattform, die Logik des Controllers und das Model können weitgehend unverändert bleiben können. Um eine

[135] Vgl. Internetquelle Symbian-SMIL-Parser.
[136] Vgl. Frech (2003), S. 57.
[137] Vgl. Kapitel 3.4.5 und Internetquelle Symbian-UIQ.

gute Portierbarkeit auf andere Symbian-Systeme zu ermöglichen, wurde bei der Stadtinfo-Anwendung daher im View all das integriert, was die Benutzeroberfläche betrifft. Die Benutzereingaben werden innerhalb des Views abgehandelt, so lange es um darstellungs-spezifische Dinge geht, wie das Umschalten zwischen den unterschiedlichen Views.
Betreffen Benutzereingaben das restliche System, so werden diese an den Controller weitergeleitet. Dies geschieht beispielsweise bei der Eingabe der Ticketnummer.
Der View Controller regelt diese Aufteilung und hat folgende Aufgaben:

- Aktivierung und Aktualisierung der Views
- Verarbeitung von Benutzereingaben

Aktivierung und Aktualisierung der Views

Der View Controller erzeugt bei Programmstart die fünf Views der Stadtinfo-Applikation.[138] Zunächst wird der „Welcome View" aktiviert. Der „Register View" wird immer dann aktiviert, wenn eine Ticketnummer eingegeben werden muss. Die Darstellung einer Auswahlliste der angebotenen Informationen übernimmt der „Content List View". Dieser wird im Design von UIQ-Anwendungen als „List View" bezeichnet.[139] Hier kann der Tourist die Information auswählen, die er sehen oder hören möchte und wechselt dazu in einen der „Detail Views". Dies sind zur Textdarstellung der „Text View" und zum Abspielen von Audio-Dateien der „Audioplay View".
Nach jeder erfolgreich heruntergeladenen Content-Datei wird der Content List View aktualisiert und neu gezeichnet. Um den Anwender jedoch nicht zu unterbrechen, wenn er z. B. gerade einen der Texte über den Schlossplatz Stuttgarts liest, wurde eine State Machine eingeführt. In einem Viewing State wird festgehalten, ob gerade einer der Detail Views aktiv ist.

Auf einem P800 können viele Anwendungen parallel laufen. Bekommt der Benutzer beispielsweise einen Anruf während er gerade Stadtinformationen abruft und trägt daraufhin neue Termine in seinen Terminkalender auf dem P800 ein, so sollte beim Zurückschalten zur Stadtinfo-Anwendung wieder die richtige Ansicht aktiviert werden.
Das Framework von Symbian aktiviert immer den View, der als „Default View" definiert wurde. Dieser wird normalerweise einmalig festgelegt. Bei der Stadtinfo-Anwendung wurde jedoch bevorzugt, immer zum letzten View zurückzuwechseln. Um dies zu erreichen, wird der Default View bei jedem Ansichtswechsel neu gesetzt.

Verarbeitung von Benutzereingaben

Beim P800 existieren zahlreiche Eingabemöglichkeiten. Eingaben, die in einem speziellen View getätigt werden, werden zunächst an den View Controller übermittelt. Dieser entscheidet dann über das weitere Vorgehen.
Außer den viewbezogenen Eingaben empfängt der View Controller auch alle anderen Benutzer-Events. So werden z. B. Eingaben, die das Menü oder die Toolbar betreffen, ebenfalls im View Controller behandelt. Im Prototypen sind dies momentan nur zwei Befehle, nämlich zum einen das Zurückschalten von einem der beiden Detail Views zum Content List View und zum anderen das Beenden der Anwendung. In einer Finalversion können dies jedoch weitere Befehle sein, wie z. B. das Aufrufen und Einstellen eines persönlichen Profils.

[138] Vgl. Kapitel 6.2.8.
[139] Vgl. Internetquelle Symbian-UIQ.

6.2.8 Die einzelnen Views

Die Aktivierung der unterschiedlichen Ansichten übernimmt der View Controller. In diesem Kapitel soll kurz auf die einzelnen Views eingegangen werden.[140]

Welcome View
Der Welcome View begrüßt den Touristen. Sobald eine Stadtinfo-Notifikation eingeht, wird zum Register View oder Content List View gewechselt.

Register View
Im Register View wird der Tourist in einem Dialogfeld aufgefordert, eine Ticketnummer einzugeben, die sechsstellig sein muss. Wird der Vorgang vom Benutzer abgebrochen, weil er beispielsweise sein Ticket gerade nicht zur Hand hat, so wird der Welcome View wieder aktiviert, die Verbindung zurückgesetzt und auf eine erneute Stadtinfo-Notifikation gewartet.

Content List View
Der Content List View ist der wichtigste View. Er wird aktiviert, sobald ein gültiges Ticket bestätigt und der Content-Datei-Download gestartet wurde oder wenn der Tourist ein zweites Mal zur selben Sehenswürdigkeit kommt und bereits alle Daten des Standorts vorhanden sind. Während des Downloads der XML-Datei und deren Auswertung wird ein Text eingeblendet, der den Benutzer darüber informiert. Nach Auswertung der XML-Datei wird dann der aktuelle Standort angezeigt. Außerdem enthält der Content List View ein kleines Antennensymbol, das den Benutzer darüber informiert, ob er gerade mit einem Informationspunkt des Stadtinformationsdienstes verbunden ist. Unter Standortangabe und Antennensymbol wird eine Auswahlliste der am Standort bereitgestellten Dateien aufgeführt. Jede Datei wird außerdem mit einem Symbol gekennzeichnet, das angibt, ob sie bereits verfügbar ist. Später soll hier ein Balken integriert werden, der den Download-Fortschritt anzeigt.
Symbian OS stellt für die Darstellung einer Auswahlliste in Spalten eine so genannte „Column List Box" zur Verfügung. Die API hierbei ist jedoch recht rudimentär. Um beispielsweise zu erkennen, welche Datei der Benutzer per Zeichenstift ausgewählt hat, muss der Index des ausgewählten Eintrags erst berechnet werden. Die Berechnung erfolgt aus den Koordinaten des Zeiger-Events unter Einbezug von Höhe der Einträge und Darstellungsbereich der List Box.
Neben der Dateiauswahl per Zeichenstift wurde bei der Stadtinfo-Anwendung auch das seitliche Rad des P800 verwendet. Um die Bedienung für den Touristen komfortabler zu gestalten, kam eine für die Symbian-UIQ-Version spezifische Methode zum Einsatz. Der Tourist kann nun mit dem Rad durch die Liste scrollen und durch Druck auf das Rad die gewünschte Datei auswählen.
Wählt der Tourist nun eine Datei aus, wird anhand des Indexes der passende View aktiviert. Eine Audiodatei wird daraufhin vom Main Controller in seiner Audio Engine abgespielt.

Text View
Der Text View dient der Darstellung der Textinformationen. Im Prototypen wird die Dateibeschreibung angezeigt und in einer Box darunter der eigentliche Informationstext.

Audioplay View
Vom Audioplay View aus soll der Benutzer die Audio-Wiedergabe steuern können. Im Prototypen wird die Datei gleich abgespielt und dabei die Dateibeschreibung angezeigt.

[140] Einige Screenshots finden sich in Kapitel 6.5.

6.3 Gesamtablauf eines Szenarios

Durch das MVC-Prinzip und die damit verbundene Aufgabenverteilung gewinnt die Kommunikation der einzelnen Klassen an Bedeutung. Nachrichten müssen ausgetauscht und der Ablauf richtig gesteuert werden. Abschließend soll hier das wohl interessanteste der möglichen Szenarien in einem Sequenzdiagramm dargestellt werden: Der Tourist betritt einen neuen Standort und ist noch nicht angemeldet.
Von folgenden Vorbedingungen wird ausgegangen:

- Die Bluetooth-Funktion des P800 ist aktiviert.
- Die Stadtinfo-Anwendung ist auf dem P800 installiert und gestartet.
- Ein aktives MBT hat nach Bluetooth-Geräten gesucht und das P800 des Touristen gefunden.

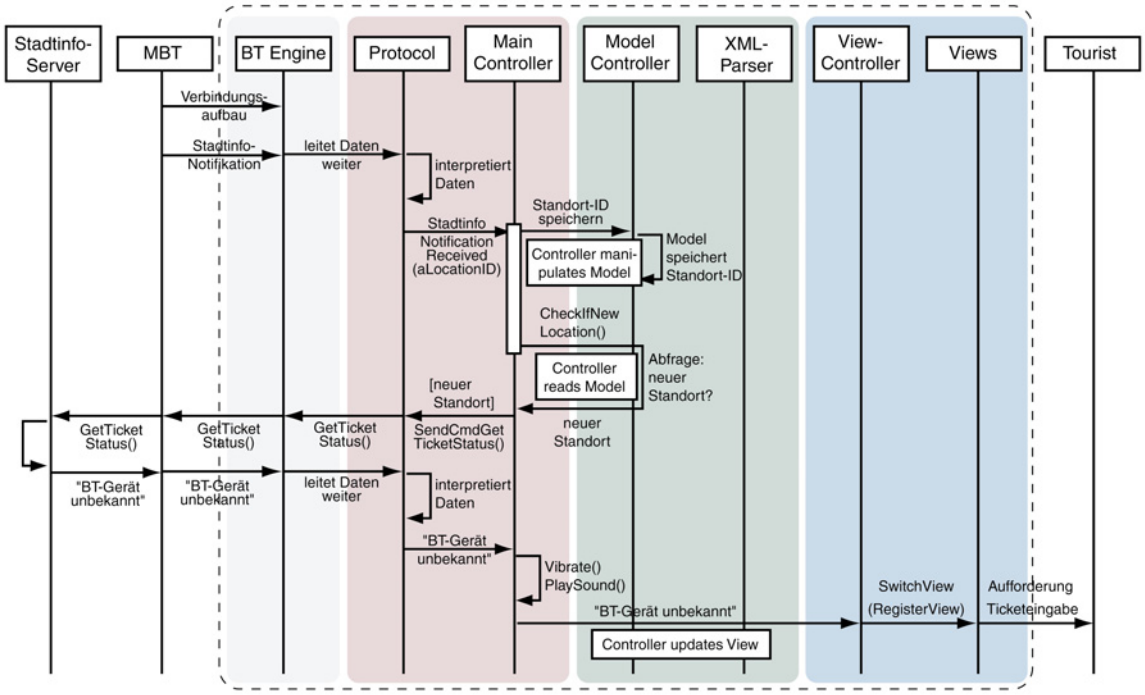


Abbildung 6-8: Sequenzdiagramm 1 (bis Ticketeingabe)

Das MBT verbindet sich mit dem P800 und sendet die Stadtinfo-Notifikation an die Bluetooth Engine. Diese leitet die Daten weiter an die Protokollklasse, die die Nachricht als Notifikation erkennt. Der Main Controller wird davon benachrichtigt und die enthaltene Standort-ID wird übergeben. Zunächst wird die Standort-ID an den Model Controller zur Speicherung im internen Datenmodell weitergeleitet: „Controller manipulates Model". Dann fragt der Main Controller vom Model ab, ob dies die gleiche Standort-ID ist wie eine eventuell bereits abgelegte Standort-ID und erkennt so, dass ein neuer Standort vorliegt: „Controller reads Model". Es müssen also Daten heruntergeladen werden. Dazu muss über die Protokollklasse der Ticketstatus erfragt werden. Das MBT erkennt diese Anfrage. Da das MBT weiß, mit welchem Bluetooth-Gerät es gerade verbunden ist, schickt es die notwendigen Informationen an den Stadtinfo-Server. Dieser antwortet in diesem Fall, dass das Bluetooth-Gerät unbekannt ist. Das MBT schickt diese Nachricht weiter an das P800. Die Protokollklasse interpretiert die Nachricht und leitet sie an den Main Controller weiter. Dieser

benachrichtigt den Touristen mit Vibration und Signalton. Außerdem benachrichtigt er den View Controller, dass das Gerät unbekannt ist, also noch nicht beim Stadtinformationsdienst angemeldet ist. Dieser aktiviert daraufhin die Anmeldungsansicht mit Dialogfenster und dem entsprechenden Text („Controller updates View"). Nun wird gewartet, bis der Tourist eine Ticketnummer eingibt.

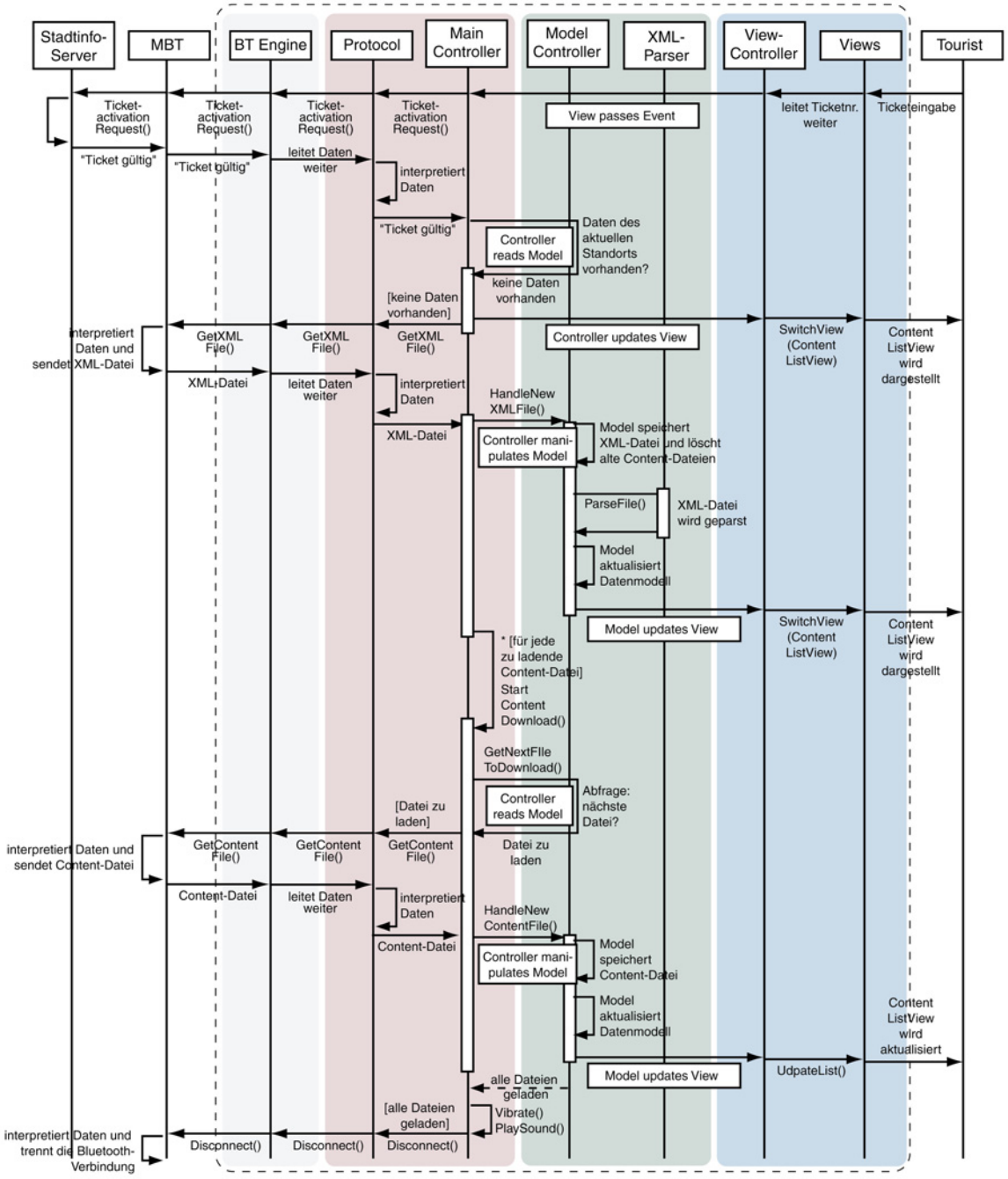


Abbildung 6-9: Sequenzdiagramm 2 (ab Ticketeingabe)

Wurde eine neue Ticketnummer eingegeben, notifiziert der View Controller den Main Controller und übergibt ihm die Ticketnummer: „View passes Event". Bei korrektem Ticket meldet der Stadtinfo-Server den Touristen am System an und benachrichtigt das MBT, dass ein gültiges Ticket vorliegt. Die Protokollklasse interpretiert wiederum die Daten vom MBT und zeigt dem Main Controller ein gültiges Ticket an. Nachdem vom Model erfragt wurde, dass keine Daten des aktuellen Standorts vorhanden sind, notifiziert der Main Controller zunächst den View Controller: „Controller notifies View".[141] Dort wird daraufhin der Content List View aktiviert und der Benutzer informiert, dass Daten geladen werden. Dann fordert der Main Controller die XML-Datei an.[142]
Die XML-Datei wird nach dem Download vom Model Controller im Dateisystem gespeichert und dann alle eventuell vorhandenen alten Content-Dateien gelöscht. Anschließend wird sie mit Hilfe des XML-Parsers ausgewertet und das interne Datenmodell angelegt. Der Model Controller notifiziert anschließend den View Controller: „Model updates View". Dieser aktualisiert den Content List View und zeigt die Content-Dateien an, die nun heruntergeladen werden.
Nun startet der Main Controller den Content-Datei-Download. Hier wird für jede zu ladende Content-Datei immer zuerst vom Model die Content-ID der nächsten zu ladenden Datei erfragt: „Controller reads Model". Die entsprechende Datei wird angefordert und vom MBT übermittelt. Jede eintreffende Content-Datei wird an das Model weitergeleitet: „Controller manipulates Model". Dort wird die Datei gespeichert und das interne Datenmodell aktualisiert. Das Model aktualisiert dann wiederum den View: „Model updates View".
Sind alle Daten geladen, erkennt der Controller dies bei der Abfrage des Models. Er benachrichtigt den Touristen davon mit Vibration und Signalton. Dann fordert er das MBT über die Protokollklasse auf, die Verbindung zu trennen.
Bereits parallel zum Datendownload oder auch anschließend kann der Tourist die Informationen ansehen oder anhören.

[141] Ein gültiges Ticket wird nicht nur angezeigt, wenn der Tourist sich das erste Mal anmeldet und noch keine Daten auf dem P800 vorhanden waren. Ein gültiges Ticket kann z. B. auch dann dem Main Controller gemeldet werden, wenn ein früherer Datendownload nicht beendet werden konnte, weil der Tourist sich beispielsweise aus dem Funkbereich des MBT herausbewegt hat. Folglich erfolgt an dieser Stelle eine Abfrage, ob bereits Daten des aktuellen Standorts vorhanden sind. Dies ist hier nicht der Fall.

[142] Zunächst bestand die Überlegung, bei einem gültigen Ticket gleich die XML-Datei zu schicken. Der Ticketstatus wird aber auch dann abgefragt, wenn z. B. die Daten am gleichen Standort nicht vollständig geladen wurden. Eine XML-Datei muss dann nicht erneut übermittelt werden.

6.4 Kommunikation der Komponenten

Wie aus den bisherigen Ausführungen ersichtlich ist, müssen die Komponenten des Stadtinformationsdienstes untereinander eine Vielzahl von Nachrichten austauschen. Der Hauptteil der Kommunikation läuft hierbei zwischen MBT und P800 ab.
Bei der Konzeption des Prototypen wurde darauf geachtet, dass dieser alle notwendigen Funktionen enthält, um den Erfolg einer eventuellen Finalversion abschätzen zu können. Um später flexibel zu sein, wurden mit einem eigens entworfenen Protokoll und der XML-Datei erweiterbare Konzepte gewählt.

6.4.1 Das Stadtinfo-Protokoll

Um einen einheitlichen Standard für die Kommunikation festzulegen, wurde beim Stadtinfo-Prototyp ein eigenes Protokoll kreiert. Dieses wird an den entsprechenden Stellen interpretiert. Im P800 erledigt dies die Protokollklasse. Beim MBT übernimmt die Auswertung der Daten eine Folge von Funktionen. Das Verfahren zur Interpretation der Daten ist ähnlich, die Felder werden nacheinander in einem Entscheidungsbaum ausgewertet.

Stadtinfo-Standardpaket

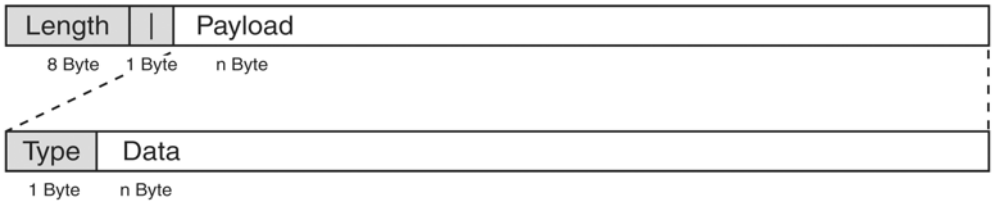


Abbildung 6-10: Stadtinfo-Standardpaket

Feld	***Länge in Bytes***	***Beschreibung***
Length	8	gibt die Länge des Payloads an
senkrechter Strich „|" als Trennzeichen	1	erleichtert die Programmierung der Flusskontrolle
Payload	variabel	enthält den Typ und die eigentlichen Informationen des Pakets

Tabelle 6-1: Aufbau des Stadtinfo-Standardpakets

Die Einführung des ersten Feldes hatte mehrere Gründe:

- Über Bluetooth werden teilweise auch recht große Dateien versandt wie z. B. die Content-Dateien. Beim Versenden größerer Dateien in einem einzigen Paket traten jedoch Probleme auf.
- Die vom MBT versandten Daten wurden vom P800 teilweise als ein, teilweise als mehrere aufeinander folgende Pakete erkannt.

Durch Angabe der Länge des Payloads und das daran anschließende Trennzeichen konnte nun eine korrekte Datenauswertung sichergestellt werden. Die Länge wird ausgewertet und dann die entsprechende Menge an Bytes von der Bluetooth-Verbindung gelesen. Im P800 kapselt die Bluetooth Engine die Auswertung des Trennzeichens und schickt nur die eigentlichen Daten an die Protokollklasse weiter.

Der Payload gibt im ersten Feld zunächst den Typ der übertragenen Daten an (s. Tabelle 6-2), im daran anschließenden Datenfeld folgen die eigentlichen Informationen.

Type	***Beschreibung***
0x00	Systemweite Codes
0x01	Steuerungsbefehle
0x02	XML-Datei
0x03	Content-Datei

Tabelle 6-2: Typen des Stadtinfo-Standardpakets

1) Type 0x00: Systemweite Codes

Pakete des Typs 0x00 enthalten systemweite Codes, die auch bei der Kommunikation zwischen MBT und Stadtinfo-Server verwendet werden. Sie dienen der Ticketabfrage und der Signalisierung systemweiter Fehlercodes. Pakete dieses Typs werden als Antwort auf eine Ticketstatusabfrage versandt.[143]

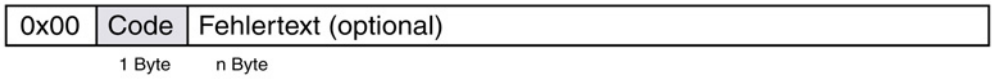


Abbildung 6-11: Stadtinfo-Paket Type 0x00

Code	***Beschreibung***
0x00	kein Fehler
0x01	Ticket ist gültig
0x02	Bluetooth-Gerät unbekannt (-> Ticketnummer anfordern)
0x03	Ticketnummer ungültig (-> erneut anfordern)
0x04	Ticket ist abgelaufen (-> neue Ticketnummer anfordern)
0x05	Ticket ist bereits auf anderes Gerät registriert (-> Ticketnummer anfordern)
0x06	Kommunikationsfehler zwischen P800 und MBT
0x07	Kommunikationsfehler zwischen MBT und Server
0x08	Server-Fehler
...	*nicht belegt*
0xFF	unbekannter Fehler

Tabelle 6-3: Systemweite Codes

Um Mehrsprachigkeit zu unterstützen, werden die Texte der Codes aus der entsprechenden Ressourcen-Datei ausgelesen, die bei der Anwendungsinstallation mit auf das P800 übertragenen wird. Diese Ressourcen-Datei auf dem Mobiltelefon des Anwenders kann jedoch nach Distribution der Software nicht mehr ohne weiteres aktualisiert werden. Ein Paket des Typs 0x00 kann daher neben dem Code jedoch auch noch einen optionalen Fehlertext enthalten. Dieses Feld kann dann genutzt werden, um zumindest einsprachig Meldungen auszugeben, deren Notwendigkeit vor Distribution der Software noch nicht vorhergesehen werden konnte.

[143] Vgl. Type 0x01, Commandtype 0x21

2) Type 0x01: Steuerungsbefehle

Steuerungsbefehle dienen der Steuerung von Verbindung, Ticketabfrage und Datendownload. Sie enthalten als erstes den Befehlstyp und anschließend je nach Befehl notwendige Daten.

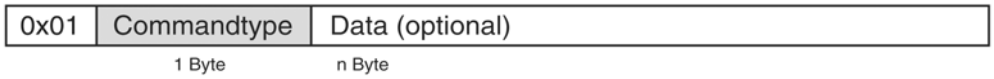


Abbildung 6-12: Stadtinfo-Paket Type 0x01

Commandtype	***Bedeutung***	***Data***
Nachrichten von MBT an P800:		
0x00	Stadtinfo-Notifikation	Standort-ID
Nachrichten von P800 an MBT:		
0x21	Ticketabfrage	evtl. Ticketnummer
0x22	Download-Aufforderung (XML-Datei)	
0x23	Download-Aufforderung (Content-Datei)	ID der Content-Datei
0x24	Disconnect-Aufforderung	

Tabelle 6-4: Steuerungsbefehle des Stadtinfo-Paket Type 0x01

Die Ticketabfrage nimmt eine gewisse Sonderstellung ein. Wird keine Ticketnummer übertragen, so handelt es sich um eine reine Statusabfrage, ansonsten um eine Anfrage zur Ticketeinlösung. Die Disconnect-Aufforderung mit Type 0x24 wurde wie bereits erwähnt eingeführt, da ein Verbindungsabbau vom MBT aus wesentlich schneller erfolgte. Da aber nur das P800 erkennen kann, wann alle Daten heruntergeladen sind, muss es diesen Verbindungsabbau anfordern.

Datenfeld	***Länge in Bytes***	***Beschreibung***
Standort-ID	1	Zahl, z. B. „2"
Ticketnummer	6	6-stellig als String, z. B. „123456"
ID der Content-Datei	variabel	String, z. B. "\Text\Schlossplatz.txt"

Tabelle 6-5: Datenfelder des Stadtinfo-Paket Type 0x01

3) Type 0x02: XML-Datei

Das Datenfeld dieses Pakets enthält die reinen Daten der XML-Datei.

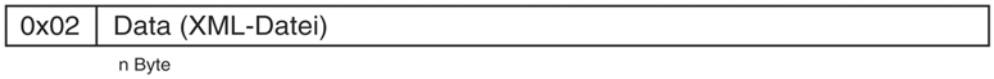


Abbildung 6-13: Stadtinfo-Paket Type 0x02

4) Type 0x03: Content-Datei

Das Datenfeld dieses Pakets enthält die reinen Daten der entsprechenden Content-Datei. Im Prototypen sind dies Text- oder Audiodateien, später können in diesem Pakettyp Dateien jeglicher Medienform wie z. B. Bilder oder Videos übertragen werden.

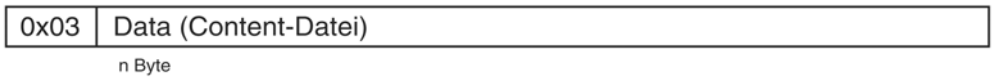


Abbildung 6-14: Stadtinfo-Paket Type 0x03

6.4.2 Die XML-Datei

Der XML-Standard wurde aufgrund seiner großen Verbreitung eingesetzt. Auf einen eigenen Namensraum wurde aufgrund des abgeschlossenen Systems verzichtet. Tabelle 6-6 gibt einen Überblick über die im Prototypen verwendeten und für eine Finalversion vorgesehenen Tags und Attribute:

Tag	***Attribut(e)***	***Beispiel***	***Prototyp***	***Finalversion***
Standort	Standort-ID	„1"	**X**	**X**
	Standort-Name (zur Darstellung)	„Schlossplatz"	**X**	**X**
	Version der XML-Datei	„1"	**-**	**X**
Daten	Sprache	„GER"	**-**	**X**
Content	Content-Datei-ID	"\Text\Schlossplatz.txt"	**X**	**X**
	Datei-Name (zur Darstellung)	"Textinfo Schlossplatz"	**X**	**X**
	Datei-Typ (Text/Audio/...)	„Text"	**X**	**X**
	Metadaten (Interessengebiet/...)	„Kultur", „Sport"	**-**	**X**

Tabelle 6-6: Aufbau der XML-Datei des Stadtinfo-Prototypen

Die Standort-ID des aktuellen Standorts wird bei der Stadtinfo-Notifikation auch bereits in einer Variablen im Datenmodell gespeichert, geht daher aber bei Unterbrechung der Stromzufuhr oder Neustart der Anwendung verloren. Die Standort-ID einer neu eingehenden Notifikation wird deswegen, zur Erkennung eines neuen Standorts, mit der Standort-ID der XML-Datei verglichen. So kann in diesen Fällen das erneute Herunterladen der XML-Datei vermieden werden.
Bei der XML-Datei wurden für die Finalversion weitere Attribute angedacht wie z. B. die Version der XML-Datei für ein Versions-Management, unterschiedliche Sprachversionen der Content-Dateien oder Metadaten wie Interessengebiete von Sport bis Kultur.
Beispiel der „`Stadtinfo.xml`"-Datei am Standort Hauptbahnhof:

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<location Id="2" Name="Hauptbahnhof">
 <Data>
  <Content Id="\Text\Hauptbahnhof.txt" Name="Text Hauptbahnhof" Type="Text" />
  <Content Id="\Audio\Hauptbahnhof.wav" Name="Audio Hauptbahnhof" Type="Audio" />
  <Content Id="\Text\Klettpassage.txt" Name="Text Klettpassage" Type="Text" />
  <Content Id="\Audio\Klettpassage.wav" Name="Audio Klettpassage" Type="Audio" />
 </Data>
</location>
```

6.5 Benutzerführung

Eine wichtige Anforderung an den Stadtinfoprototypen war eine möglichst einfache Benutzerführung. So wurde neben der reinen Funktionalität auch auf Symbole oder zusätzlich angezeigte Informationen Wert gelegt, die dem Benutzer hilfreich sein könnten. Im Folgenden sind einige Screenshots der Stadtinfo-Anwendung auf dem P800 abgebildet, die dies verdeutlichen sollen:[144]

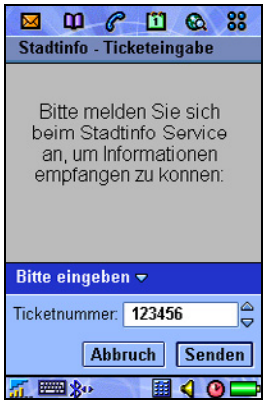
1)

Das Icon fügt sich in das UIQ-Design der anderen Anwendungen ein: Ein hellblauer plastischer Kreis mit deutlicher Umrandung des Stadtinfo-Logos im Inneren.

2)

Begrüßung des Touristen. Hier sollen in einer Finalversion ein persönliches Profil angelegt oder sonstige Einstellungen eingegeben werden können.

3)

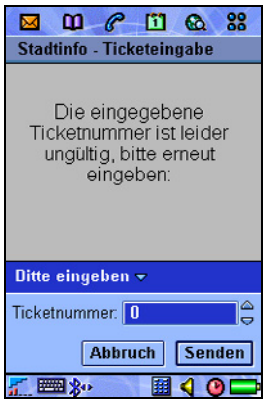


Der Benutzer kann nur eine sechsstellige Ticketnummer eingeben, hat aber auch die Möglichkeit, abzubrechen, sollte er sein Ticket nicht zur Hand haben. Dann wird der Welcome View wieder aktiviert.

4)

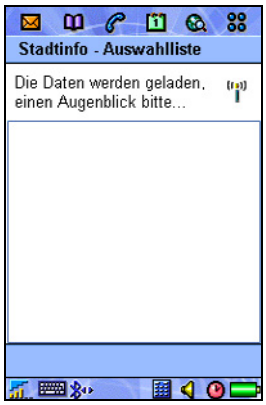
Währen der Ticketabfrage wird dem Benutzer ein neuer Text angezeigt.

5)



Fehler werden angezeigt. Hier wird der Grund für die erneut notwendige Eingabe angegeben.

6)



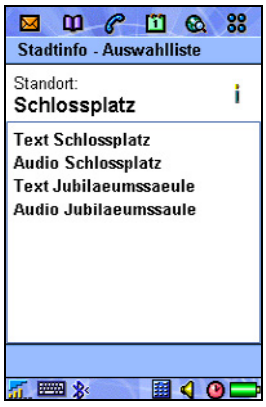
Auch während des Downloads der XML-Datei und deren Auswertung bekommt der Nutzer eine Rückmeldung.

[144] Im Anhang befindet sich eine CD, auf der der Einsatz des Prototypen in einem Video zu sehen ist.

7)

Die Listeneinträge können auch während des Downloads schon mit dem Zeichenstift oder durch einen Druck auf das Rad an der Seite des P800 ausgewählt werden.
In einer rechten Spalte wird angegeben, ob die Datei bereits geladen wurde. Später ist für diese Rückmeldung ein Downloadbalken angedacht.

8)



Wurden alle Dateien geladen, wird der Download-Status nicht mehr angezeigt. Die Antenne rechts oben zeigt hier z. B. an, dass keine Verbindung mehr besteht. Der Tourist kann nun die gewünschte Information auswählen.

9)

Soll eine Audio-Information angehört werden, wird diese abgespielt und die Beschreibung angezeigt. Durch Druck auf das Rad an der Seite des P800 oder den Back-Button rechts unten kehrt der Benutzer wieder zur Auswahlliste zurück.
Später können Steuermöglichkeiten für Play, Pause und Stop sowie Zusatzinformationen über die Länge der Audiodatei oder ähnliches integriert werden.

6.6 Implementierung

Für die Implementierung wurde folgende Software eingesetzt:

Konzeption

- Together 6.0 von TogetherSoft (Klassendiagramme, Activity-Diagramme)
- Visio Standard 5.0 von Visio Soft (Sequenzdiagramme, Übersichts-Grafiken)

Programmierung

- P800: Metrowerks Codewarrior for Symbian Personal Edition 2.0, später 2.5
- MBT: - Texteditor CodeGenie 3.0 von DolfySoft
  - Mitgelieferte Tools von SND (Software Release 2.1, später 2.2)

7 Schlussbetrachtungen

7.1 Fazit

Der Prototyp wurde erstellt, um die Funktionalität eines Stadtinformationsdienstes auf Bluetooth-Basis mit den derzeit auf dem Markt verfügbaren Geräten zu testen. Diese Funktionalität sollte darüber hinaus in eine durchdachte Benutzerführung verpackt werden, um bei der Demonstration des Prototypen abschätzen zu können, ob das System gegenüber dem Endnutzer erfolgversprechend erscheint.
Betrachtet man den fertigen Prototypen, so wurden alle anfangs gestellten Anforderungen erfüllt:

- Der Nutzer hat dank des eingesetzten Bezahlsystems und Bluetooth als verwendete Übertragungstechnologie volle Kostenkontrolle.
- Ticketeinlösung und Informationsabruf auf dem P800 sind einfach, funktionell und dank Symbian OS sehr komfortabel.
- Die Informationen des jeweils aktuellen Standorts werden automatisch vom MBT übermittelt, wobei die Kommunikation zwischen P800, MBT und Stadtinfo-Server komplett im Hintergrund abläuft.
- Aufgrund der Architektur und der eingesetzten Komponenten ist das Gesamtsystem flexibel erweiterbar und voraussichtlich gut auf andere Systeme portierbar.

Der Prototyp basiert fast ausschließlich auf neuen Technologien, Betriebssystemen und Endgeräten. Zu Beginn des Projektes waren daher eine intensive Einarbeitungszeit und umfangreiche Tests notwendig. Das erstellte Konzept konnte in einer eher kurzen Kernprogrammierphase von etwa sechs Wochen umgesetzt werden.
Mitentscheidend für den Erfolg des Systems war es, den Verbindungsaufbau vom MBT aus einleiten zu können. Dies konnte mit der neuen Software-Release des MBT am Ende auch so umgesetzt werden. Auch für das Abspielen der Audiodateien konnte hinsichtlich Qualität und Dateigröße eine überzeugende Lösung gefunden werden.
Einzig bezüglich der Stabilität bleiben noch einige Fragen offen:

- Die Bluetooth-Verbindung selbst scheint relativ stabil, dennoch gingen teilweise Pakete verloren, weshalb eine Segmentierung und zusätzliche Flusskontrolle auf Anwenderebene implementiert werden mussten.
- Der Nachrichtenaustausch auf den unteren Bluetooth-Protokoll-Ebenen scheint nicht korrekt implementiert zu sein, was nur durch Einführung künstlicher Timeouts gelöst werden konnte. Auch eine fehlerhafte Implementierung des Active Object Frameworks in Zusammenhang mit Bluetooth auf dem P800 kann nicht vollständig ausgeschlossen werden.

Der Prototyp ist dank der gefundenen Lösungen funktionstüchtig. Hinsichtlich einer Finalversion sollten oben genannte Fragen durch Analyse der über Bluetooth ausgetauschten Daten jedoch noch geklärt werden.
Bezüglich der eingesetzten Komponenten kann folgendes Fazit gezogen werden:

- Bluetooth:
  Bluetooth ist eine sehr vielfältig und flexibel einsetzbare Technologie und konnte in der Praxis durchaus überzeugen. Die Implementierung auf den einzelnen Geräten lässt jedoch noch genügend Spielraum für Verbesserungen.
- MBT/HyNetOS:
  Das Modul bietet von der Hardware gesehen alles, was für den Prototypen notwendig war. Die Programmierung in C und die Bluetooth API ist ebenfalls relativ einfach aufgebaut, so dass die Anwendung auf dem MBT am Ende alle Anforderungen bezüglich Kommunikation und lokale Datenbereitstellung erfüllen konnte. Die Entwicklung komplexerer Anwendungen erscheint jedoch eher schwierig. Leider war auch die mitgelieferte Dokumentation unvollständig und erschwerte die Arbeit mit dem Modul. Aufgrund der bereits erwähnten Probleme bezüglich Bluetooth-Implementierung sollte der Einsatz in einer Finalversion überdacht werden.
- P800/Symbian OS:
  Symbian OS erwies sich im Einsatz als sehr leistungsfähiges Betriebssystem mit interessanten Konzepten. Aufgrund der Modularität und der uneinheitlichen APIs war die Programmierung jedoch teilweise unkomfortabel. Auch die eingesetzte Programmierumgebung konnte aufgrund des unzuverlässigen Debuggings nicht voll überzeugen. Die Dokumentation des SDKs von SonyEricsson wird dank Java-Applet ständig erweitert und korrigiert, so dass man zuversichtlich sein kann, dass die momentan noch enthaltenen Fehler bald behoben sein werden. Insgesamt gesehen jedoch konnte das Symbian-Betriebssystem, vor allem auch in Verbindung mit den umfangreichen Möglichkeiten des SonyEricsson P800, voll überzeugen.

7.2 Erweiterungsmöglichkeiten des Prototypen

Können auch die letzten technischen Probleme gelöst werden, erscheint das Konzept sehr vielversprechend. Im Folgenden sollen nur einige der angedachten Erweiterungsmöglichkeiten aufgeführt werden, die für den Erfolg einer Finalversion durchaus sinnvoll und reizvoll erscheinen.

Konzeptionelle Erweiterungen

- Eingabe der Ticketnummer:
  Der Tourist sollte diese gleich beim Kauf in die Anwendung eingeben können. Die Ticketnummer kann dann am ersten Standort automatisch oder nach Bestätigung durch den Touristen an das System übermittelt werden.
- Bezahlsystem:
  Bezahlen per so genannter „Premium-SMS". Je nach Anbieter können derzeit bis zu drei Euro per Premium-SMS abgerechnet werden.[145] Eine Antwort-SMS könnte dann beispielsweise die Ticketnummer enthalten, die bei Start der Stadtinfo-Anwendung automatisch aus der SMS-Inbox des Mobiltelefons ausgelesen werden könnte.

[145] Vgl. Internetquelle Premium-SMS.

- Download der Applikation:
  Einrichtung von Bluetooth Access Points, an denen der Tourist die Anwendung für sein Mobiltelefon bequem und kostenfrei herunterladen und installieren kann.
- Profile:
  Der Tourist soll ein persönliches Profil anlegen können, um die angebotenen Informationen automatisch hinsichtlich seiner Interessengebiete filtern lassen zu können.
- Mehrwert:
  Mehrwert durch Landkarten zur Orientierung für den Touristen.
- Marketing:
  Kostenloser Zugang zum Stadtinformationsdienst durch Werbefinanzierung.

Technische Erweiterungen

- Automatisches Generieren der XML-Datei auf dem MBT:
  Die Content-Dateien können jetzt schon bequem über Ethernet aktualisiert werden, die XML-Datei muss momentan jedoch noch manuell erstellt werden. Bei einer Generierung der XML-Datei anhand der auf dem MBT vorhandenen Ordnerstruktur könnte Zeit gespart und eine potentielle Fehlerquelle ausgeschlossen werden.
- SDP:
  Eintragen des Stadtinformationsdienstes als Bluetooth-Service in der Datenbank des Mobiltelefons. Das MBT könnte so Zielgeräte nach dem entsprechenden Eintrag durchsuchen, bevor es versucht, eine Datenverbindung aufzubauen.
- Text-to-Speech:
  Einsatz oder Entwicklung einer leistungsfähigen Text-to-Speech Software. So könnten sehr viel mehr Informationen in Textform übermittelt und dann in Sprachform wiedergegeben werden, ohne viel Speicher auf dem Zielsystem zu belegen.
- Audio-Streaming:
  Bluetooth unterstützt auch Audio-Streaming. Durch Übertragung der Audio-Dateien als Stream würde auf dem System des Benutzers fast kein Speicher mehr benötigt werden.
- Portierung:
  Portierung auf andere Symbian-Systeme und andere Betriebssysteme, um z. B. auch PDAs und Laptops zu unterstützen.
- Umsetzung als Java-Midlet:
  Java-Midlets können auf sehr vielen Mobiltelefonen eingesetzt werden. Leider gibt es derzeit kaum ein Gerät, das gleichzeitig Bluetooth unterstützt.
- Kabellose Anbindung der Informationspunkte:
  Die Informationspunkte könnten an das Airdata-System angebunden werden, das in einigen größeren deutschen Städten angeboten wird.[146]
- Stromversorgung:
  Um neben einer drahtlosen Datenanbindung vollständig unabhängig von Kabelverbindungen zu sein, könnten die Informationspunkte auch per Solar betrieben werden.
- Sicherheit:
  Auf Sicherheitsaspekte wurde bei der Entwicklung des Prototypen kein Wert gelegt, da es sich nicht um sensible Daten handelt. Um den Missbrauch des Systems in einer Finalversion auszuschließen, könnte hingegen die Einführung entsprechender Verschlüsselungsverfahren/Handshake-Verfahren notwendig erscheinen.

[146] Vgl. Internetquelle Airdata.

7.3 Visionen

Der Stadtinformationsdienst ist nur ein Beispiel, wie Bluetooth in Zusammenhang mit mobilen Endgeräten genutzt werden kann. Das Potential dieser Kombination ist damit nicht einmal ansatzweise beschrieben. Durch den Einsatz der kleinen programmierbaren Module erscheint dieses Konzept beispielsweise auch bei Konferenzen oder bei Veranstaltungen im Freien, wie Konzerte oder Produktpräsentationen, sehr erfolgversprechend.
Die vergleichsweise eher kleine Reichweite von Bluetooth kann ein großer Vorteil sein. Andere Systeme müssen diese Ortsbezogenheit erst herstellen. Bei Mobiltelefonen kann der Ort nur recht ungenau bestimmt werden und zwar durch Ermittlung des Sendemasten, mit dem das Gerät gerade verbunden ist. Bei Laptops oder PDAs ist auch dies nicht möglich und die Ermittlung des Standorts muss beispielsweise durch einen zusätzlichem GPS-Empfänger teuer erkauft werden.
Doch gerade ortsbezogene, kostengünstige Dienste sind für den Nutzer interessant.
Unzählige Einsatzorte der in dieser Diplomarbeit verwendeten Technologien sind denkbar. Hier sollen nur einige mögliche Szenarien ausgemalt werden:

- Die Präsentation des neuen Projektes im Konferenzsaal ist beendet. Das Mobiltelefon hat das Angebot bereits dem anwesenden Kunden übermittelt, zusammen mit den Präsentationsdaten, die vom Mobiltelefon aus auf den Beamer übertragen wurden. Der Termin zur Vertragsunterzeichnung wird vereinbart und noch einmal wichtige Kontaktdaten per Bluetooth ausgetauscht.
- Auf dem Heimweg von der Arbeit muss noch eingekauft werden. Die Einkaufsliste wird per Internet vom heimischen Kühlschrank abgefragt und auf das Mobiltelefon übermittelt. Im Supermarkt werden dann die Artikel eingekauft. Dabei kann man als Schnäppchenjäger noch einige Sonderangebote ergattern, die auf Anfrage per Bluetooth ans Mobiltelefon geschickt wurden. Die SIM-Karte des Mobiltelefons ist im Laden bereits angemeldet und nach Eingabe einer PIN-Nummer wird die Rechnung bezahlt und am Monatsende über die Telefonabrechnung beglichen.
- Wieder im Auto angelangt wird das Mobiltelefon erkannt und die Freisprecheinrichtung aktiviert. Ein Freund ruft an und kündigt seinen Besuch für den Abend an. Der Bluetooth-Empfänger am eigenen Haus erkennt die Bluetooth-Adresse des Fahrzeugs und öffnet das Garagentor.
- Im Haus angekommen zeigt das Display des Mobiltelefons eine Auswahl an Profilen an. Nachdem die eingekauften Artikel verstaut sind, wird auf Knopfdruck das Licht gedimmt, der Beamer aktiviert und eine Auswahl von Filmen angezeigt. Ein Film startet auf Knopfdruck, die Fernbedienung erfolgt per Mobiltelefon.
- Ein einkommender Anruf auf dem Festnetz wird auf das Mobiltelefon weitergeleitet, das seit Ankunft im Haus mit der Bluetooth-Telefonstation verbunden ist. Der Termin für die abendliche Verabredung wird bestätigt.
- Der erwartete Besuch klingelt an der Haustüre. Das Bild vom Türeingangsbereich wird an das Mobiltelefon geschickt und die Türe per Bluetooth geöffnet, während der Gastgeber schon einmal eine Flasche Wein aus dem Keller holt. Die drei unterhalten sich und spielen bis spät in die Nacht noch ein spannendes Adventurespiel – jeder auf seinem Gerät, das sich per Bluetooth mit den anderen verbunden hat.

Diese Szenarien sind technisch gesehen heute schon möglich. Was davon wirklich Sinn macht und genutzt werden wird, wird sich zeigen. Der Ideenvielfalt sind jedenfalls keine Grenzen gesetzt.

7.4 Persönliches Fazit

Während meines Studiums der audiovisuellen Medien wurde viel zu den Themen Inhalt, Technik und Gestaltung vermittelt. In den ersten drei Semestern interessierten mich die Vorlesungen der Informatik besonders. Bei der Diplomarbeit bot sich mir dann wieder die Möglichkeit, dieses Wissen zu vertiefen und mit den anderen Studienschwerpunkten zu verbinden. Ich empfand es als große Herausforderung, zwei recht spezielle Betriebssysteme kennen zu lernen und die Kommunikation und Datenübertragung auf Basis von Bluetooth zu untersuchen.
Bei der Entwicklung des Stadtinformationsdienstes konnte ich meine gewonnenen Erkenntnisse in den Bereichen Benutzerführung, sowie Inhaltsdarstellung in Form von Ton, Text und Bild einbringen. Natürlich steht bei einem Prototypen die Funktionalität im Vordergrund und gestalterische Ansprüche beschränkten sich auf das Design von Icons und Logos. Das Screendesign spielte ebenfalls eine eher untergeordnete Rolle. Trotzdem war es mir wichtig, dem Anwender durch eine klare grafische Ausarbeitung der Bedienoberfläche und guter Benutzerführung eine intuitive Bedienung zu ermöglichen.
Aufgrund der Komplexität des Systems und der Kombination der vielen Technologien gewann ich Einblick in viele neue Bereiche – von Konzeptionsdesign mit UML, Kommunikationsdesign mit Protokollentwurf über XML bis hin zu Serverprogrammierung; von hardwarenaher Programmierung eines Entwicklermoduls bis hin zur Anwendungsprogrammierung eines Mobiltelefons.

Abschließend kann ich sagen, dass die Entscheidung für dieses Projekt trotz aller Hindernisse absolut richtig war. Die in dieser Diplomarbeit gewonnenen Erfahrungen brachten mich in meiner persönlichen wie fachlichen Entwicklung wichtige Schritte weiter und werden sicherlich großen Einfluss auf meinen weiteren beruflichen Werdegang haben. Das Thema faszinierte mich vom Anfang bis zum erfolgreichen Abschluß, nicht zuletzt aufgrund des starken Praxisbezuges und der mit mobilen Anwendungen verbundenen Möglichkeiten und Visionen.

8 Anhang

8.1 Klassendiagramm der Stadtinfo-Anwendung auf dem P800

Abbildung 8-1 auf der folgenden Seite zeigt die wichtigsten Klassen, Methoden und Eigenschaften der Stadtinfo-Anwendung auf dem P800. Aus Gründen der Übersichtlichkeit wurden unter anderem folgende Dinge nicht mit eingearbeitet:

- Objekte beinahe aller Klassen werden mit dem Two-Phase-Construction-Prinzip erzeugt, gekapselt in einer `NewL()`-Funktion. Die Methoden `NewL()` und `ConstructL()` wurden daher nicht dargestellt.
- In der Stadtinfo-Anwendung wird von den angegebenen Klassen jeweils nur ein Objekt erzeugt, auf Angabe der Kardinalitäten wurde daher verzichtet.
- Klassen, die von einem Interface erben, implementieren auch dessen Methoden.
- Alle Methoden der Interfaces sind pure virtual deklariert.
- Beinahe alle Klassen nutzen den Logger, der von der `CStadtinfoAppUi` erzeugt wird, um Fehler in eine Log-Datei mitzuschreiben, die im Stadtinfo-Root-Verzeichnis des P800 liegt.
- Die Klasse `CStadtinfoAppUi` entspricht dem View Controller und erbt von der Quikon-Klasse „`CQikAppUi`". Ein Objekt dieser Klasse wird vom Application Framework erzeugt mit dem Aufruf „`CreateAppUiL()`".
- `CStadtinfoAppUi` erzeugt die Views, meldet jeden View mit „`RegisterViewL()`" beim View Server an und erzeugt mit „`AddToStackL()`" einen neuen Control Stack für jeden View.
- Alle Views erben von den Klassen „`CCoeControl`" und „`MCoeView`".
- Alle Views implementieren die `MCoeView`-Methoden „`ViewActivatedL()`", „`ViewDeactivatedL()`", „`ViewId()`" und „`ViewUid()`".
- Alle Views implementieren die `CCoeControl`-Methode „`Draw()`".
- Alle Active Objects erben von der Klasse `CActive` und implementieren neben der „`RunL()`"-Methode auch die Methode „`DoCancel()`".
- Die weitere Vererbungsstruktur wurde nicht dargestellt.

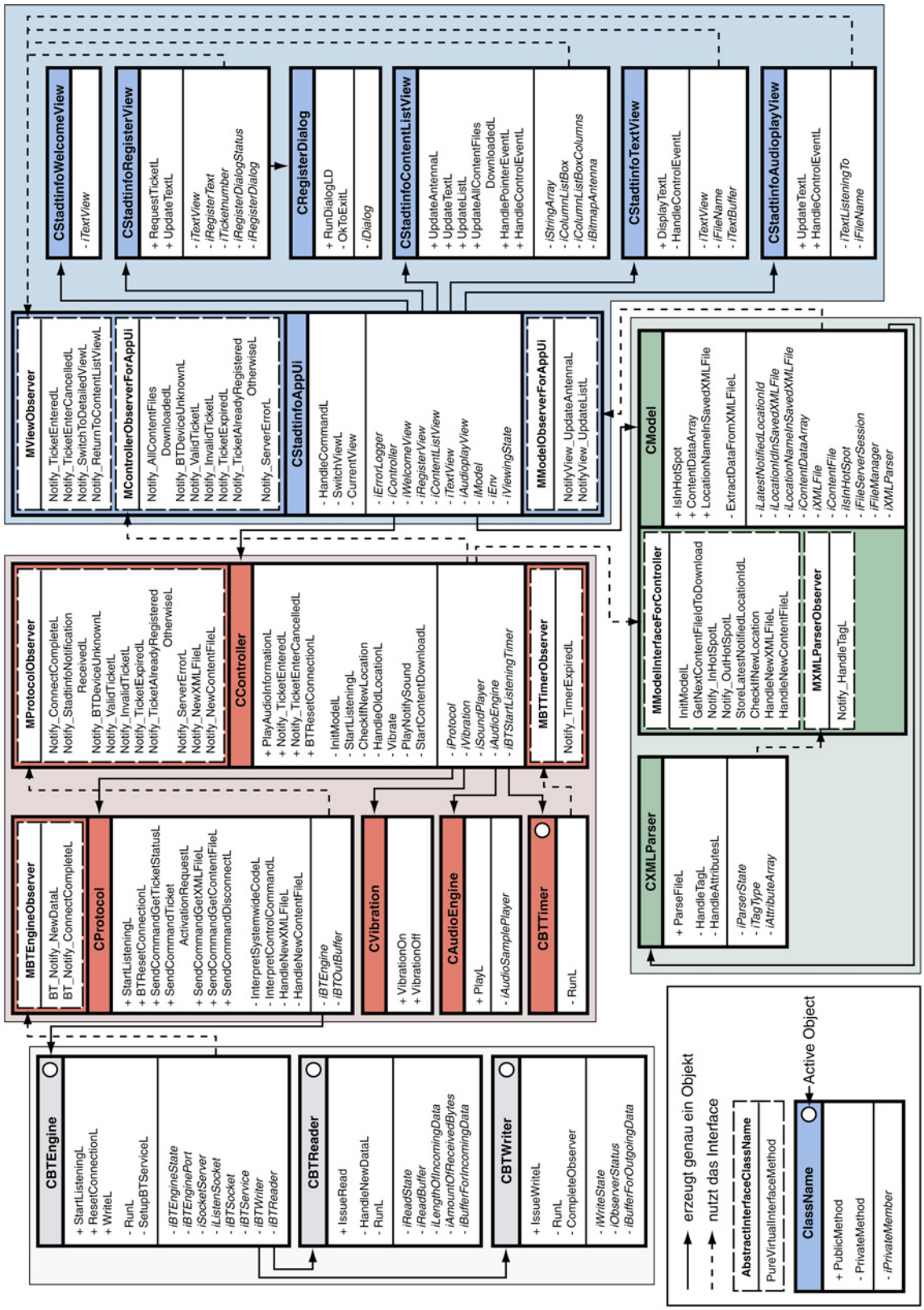


Abbildung 8-1: Klassendiagramm der Stadtinfo-Anwendung auf dem P800

8.2 Abkürzungsverzeichnis

ACL – Asynchronous Connectionless
API – Application Programming Interface

BCSP – BlueCore Serial Protocol
BT – Bluetooth

CID – Channel Identifier
CPU – Central Processing Unit
CRC – Cyclic Redundancy Check
CSD – Circuit Switched Data
CSR – Cambridge Silicon Radio

DECT – Digital European Cordless Telecommunication
DLL – Dynamic Link Library
DNL – Direct Navigation Link

FSK – Frequency Shift Keying

GFSK – Gaussian Frequency Shift Keying
GPRS – General Packet Radio Service
GPS – Global Postitioning System
GSM – Global System for Mobile Communication
GUI – Graphical User Interface

HAL – Hardware Abstraction Layer
HCI – Host-Controller Interface
HSCSD – High Speed Circuit Switched Data
HTTP – Hypertext Transfer Protocol

IDE – Integrated Development Environment
IP – Internet Protocol
IR – Infrarot
ISM – Industrial, Scientific and Medical
ISO – International Standardisation Organisation

J2ME – Java 2 Micro Edition
L2CAP – Logical Link Control and Adaption Protocol

LMP – Link Manager Protocol

MBT – MicroBlueTarget
MDA – Mobile Digital Assistant
MMS – Multimedia Message Service
MMU – Memory Management Unit
MVC – Model View Controller

OBEX – Object Exchange Protocol
OS – Operating System
OSI – Open Systems Interconnection

PCMCIA – Personal Computer Memory Card International Association
PDA – Personal Digital Assistant
PDU – Protocol Data Units
PHP – Hypertext Preprocessor
PPP – Point-to-Point-Protocol

RAM – Read and Write Memory
RFCOMM – Radio Frequency Oriented Emulation of the Serial COM Ports
ROM – Read Only Memory
RSSI – Received Signal Strength Indicator

SCO – Synchronous Connoction Oriented
SDK – Software Development Kit
SDP – Service Discovery Protocol
SIG – Special Interest Group
SMIL – Synchronized Multimedia Integration Language
SND – Smart Network Devices
SQL – Structured Query Language

TCP – Transport Control Protocol
TCS – Telephony Control Protocol Specification

UART – Universal Asynchronous Receiver Transmitter
UDP – User Datagram Protocol
UI – User Interface
UID – Unique Identifier
UML – Unified Modeling Language
UMTS – Universal Mobile Telecommunication System
URL – Uniform Resource Locator
UUID – Universally Unique Identifier

WAE – Wireless Application Environment
WAP – Wireless Application Protocol
WLAN – Wireless Local Area Network

XML – Extensible Markup Language

8.3 Tabellenverzeichnis

8.4 Abbildungsverzeichnis

8.5 Bibliographie

Literatur

Balzert, Helmut (Hg.): *Lehrbuch Grundlagen der Informatik – Konzepte und Notationen in UML, Java und C++, Algorithmik und Software-Technik, Anwendungen*. Heidelberg/Berlin: Spektrum Akademischer Verlag GmbH, 1999.

Digia Inc.: *Programming for the Series 60 Platform and Symbian OS*. Hg.: John Wiley & Sons. Chichester: Wiley, 2003.

Fowler, Martin und Kendall Scott: *UML konzentiert – Die neue Standard-Objektmodellierungssprache anwenden*. Bonn: Addison-Wesley-Longman, 1998.

Frech, Tobias: *Wireless Applcations mit J2ME am Beispiel eines Instant-Messaging-Clients*. Stuttgart: Diplomarbeit an der HdM Stuttgart, MI 13, WS 2002/2003.

Gamma, Erich et al.: *Design Patterns – Elements of Reusable Object-Oriented Software*. Hg.: Brian W. Kerninghan. New York: Addison-Wesley, 1995.

Hannemann, Ulf: „Terminals für Touris". *Focus*. Ausgabe 48, 2003.
Jipping, Michael J.: *Symbian OS Communications Programming*. Hg.: John Wiley & Sons. Chichester: Wiley, 2002.

Merkle, Andreas und Anestis Terzis: *Digitale Funkkommunikation mit Bluetooth*. Poing: Franzis' Verlag GmbH, 2002.

Tanenbaum, Andrew S.: *Moderne Betriebssysteme*. 2. überarbeitete Auflage. München: Pearson Studium, 2002.

Tasker, Martin: *Professional Symbian Programming – Mobile Solutions on the EPOC Platform*. Birmingham, UK: Wrox Press Ltd., 2000.

PDF-Quellen

PDF-Quelle Bluetooth-Spezifikation: Bluetooth SIG: *Specification of the Bluetooth System – Version 1.1.* Februar 2001. Kostenloser Download unter Internetquelle Bluetooth-Spezifikation oder auf beigelegter CD („3_Dokumentation\2_Bluetooth\Bt_V11_Core_22Feb01.pdf").

PDF-Quelle IZT: Institut für Zukunftsstudien und Technologiebewertung: *Entwicklung und zukünftige Bedeutung mobiler Multimediadienste - WerkstattBericht Nr. 49*. Berlin, Dezember 2001. Kostenloser Download unter URL: http://www.izt.de/publikationen/werkstattberichte/index.html [Stand 24.02.2004] oder auf beigelegter CD („3_Dokumentation\1_Einleitung\IZT_WB49.pdf").

PDF-Quelle MBT: SND: *Reference Board Overview*. Kostenloser Download unter Internetquelle SND oder auf beigelegter CD („3_Dokumentation\4_HyNetOS\SND_Boards_Overview.pdf").

PDF-Quelle MBT-Info: SND: *Micro BlueTarget – the system*. Kostenloser Download unter Internetquelle SND oder auf beigelegter CD(„3_Dokumentation\4_HyNetOS\SND_BT_Info_Products.pdf").

PDF-Quelle Symbian Designing For UIQ: Symbian Ltd.: *Designing for UIQ*. Kostenloser Download unter Internetquelle Symbian-Designing For UIQ oder auf beigelegter CD („3_Dokumentation\3_Symbian_OS\Symbian_Designing_For_UIQ.pdf").

PDF-Quelle Symbian-OS Version 7.0: Dixon, Kevin: *Symbian OS Version 7.0s – Functional descrition*. Symbian Ltd., Juni 2003. Kostenloser Download unter Internetquelle Symbian-OS Version 7.0 oder auf beigelegter CD („3_Dokumentation\3_Symbian_OS\Symbian_OS_v7.0s_ functional_description2.1.pdf").

Internetquellen

Internetquelle Airdata:
URL: http://www.airdata.ag [Stand: 17.03.2004]

Internetquelle Beatnik:
URL: http://www.beatnik.com/products/mobilebae.html [Stand: 17.03.2004]

Internetquelle Bluetooth-Analyzer-Vergleich:
URL: http://www.palowireless.com/bluearticles/Bluetoothanalyzercompare1.asp
[Stand: 17.03.2004]

Internetquelle Bluetooth-E-Beam:
URL: http://www.e-beam.com/products/system3w.html [Stand: 18.02.2004]

Internetquelle Bluetooth-Geschichte:
URL: http://www.bluetooth.org/bluetooth/landing/btname.php [Stand: 17.03.2004]

Internetquelle Bluetooth-Pulsmessgerät:
URL: http://www.bluetooth.com/products/prods.details.asp?CPID=1130 [Stand: 18.02.2004]

Internetquelle Bluetooth-Spezifikation:
URL: http://www.bluetooth.org/spec [Stand 20.02.2004]

Internetquelle Bluetooth-Spielzeugauto:
URL: http://www.bluetooth.com/products/prods.details.asp?CPID=1183&CAT=19
[Stand: 18.02.2004]

Internetquelle Bluetooth-Toyota:
URL: http://www.toyota.com/prius [Stand: 18.02.2004]

Internetquelle Bluetooth-Waschmaschine:
URL: http://www.bluetooth.com/products/prods.details.asp?CPID=1171 [Stand: 18.02.2004]

Internetquelle Borland Mobile Studio:
URL: http://www.sonyericsson.com/developer/site/global/newsandevents/latestnews/newsfeb04/p_borland_mobile_studio.jsp [Stand 26.02.2004]

Internetquelle Mobiltelefon-Hersteller:
URL: http://www.symbian.com/about/vision.html [Stand: 10.02.2004]

Internetquelle Mp3-Lizenzierung:
URL: http://www.mp3licensing.com [Stand: 10.03.2004]

Internetquelle OggVorbis:
URL: http://sourceforge.net/projects/symbianoggplay [Stand: 20.01.2004]

Internetquelle Premium-SMS:
URL: http://www.telespiegel.de/html/mehrwert_sms___premium_sms.html [Stand: 17.03.2004]

Internetquelle SND:
URL: http://www.smartnd.com/webseite/01_start/start.html [Stand: 16.02.2004], im Download-Bereich werden zahlreiche Dokumente zu den Produkten zur Verfügung gestellt.

Internetquelle SonyEricsson P800:
URL: http://www.sonyericssonp800.com [Stand: 10.02.2004]

Internetquelle Statistisches Bundesamt:
URL: http://www.destatis.de/basis/d/evs/budtab2.htm [Stand: 10.02.2004]

Internetquelle Symbian-Active Objects:
URL: http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/cpp/Base/InterProcessCommunication/AsynchronousServicesGuide/AsynchronousServicesGuide3/LifeCycleActiveObjects.guide.html#AsynchronousServicesGuide3%2elife%2dcycle
[Stand: 17.03.2004]

Internetquelle Symbian-Anteilübernahme:
URL: http://de.internet.com/dynamic/print.html?id=2026252 [Stand: 11.02.2004]

Internetquelle Symbian-Desiginig For UIQ:
URL: http://www.symbian.com/developer/techlib/papers/uiq/symbian_designing_for_UIQ.pdf
[Stand: 20.03.2004]

Internetquelle Symbian-Deskriptoren:
URL: http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/cpp/Base/BuffersAndStrings/Descriptors/DescriptorsGuide1/DescriptorsBasics.guide.html#DescriptorsGuide1%2ebasics [Stand: 16.02.2004]

Internetquelle Symbian-Familien:
URL: http://www.symbian.com/technology/devices.html [Stand: 12.02.2004]

Internetquelle Symbian-Mobiltelefone:
URL: http://www.symbian.com/phones/index.html [Stand: 10.02.2004]

Internetquelle Symbian-OS Version 7.0:
URL: http://www.symbian.com/technology/symbos-v7s-det.html [Stand: 11.02.2004]

Internetquelle Symbian-SDK Nokia:
URL: http://www.forum.nokia.com/main/0,6566,034-4,00.html [Stand: 20.03.2004]

Internetquelle Symbian-SDK SonyEricsson:
URL: http://developer.sonyericsson.com/site/global/docstools/symbian/p_symbian.jsp
[Stand: 20.03.2004]

Internetquelle Symbian-SMIL-Parser:
URL: http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/cpp/Messaging/MMS/smil.html [Stand: 17.03.2004]

Internetquelle Symbian-Teilhaber:
URL: http://www.symbian.com/about/ownership.html [Stand: 10.02.2004]

Internetquelle Symbian-Telefon-Herstellung:
URL: http://www.symbian.com/technology/create-symb-OS-phones.html [Stand: 12.02.2004]

Internetquelle Symbian-UIQ:
URL: http://www.symbian.com/developer/techlib/papers/Qapps/ianh.html#ianh.ianh-008
[Stand: 17.03.2004]

Internetquelle Symbian-Unterstützte Audio-Formate:
URL: http://www.symbian.com/technology/symbos-v7x-det.html#t5 [Stand: 26.02.2004]

Internetquelle Symbian-Vibration-API:
URL: http://www.newlc.com/article.php3?id_article=105 [Stand: 26.02.2004]

Internetquelle Symbian-View Architecture:
URL: http://www.symbian.com/developer/techlib/v70docs/SDL_v7.0/doc_source/DevGuides/cpp/ApplicationFramework/ViewArchitectureOverview.guide.html#ApplicationFrameworkOverview%2eViewArchitectureOverview%2emain [Stand: 20.03.2004]

Internetquelle Symbian-White Papers:
URL: http://www.symbian.com/technology/whitepapers.html [Stand: 13.02.2004]

Internetquelle UUIDs:
URL: https://www.bluetooth.org/foundry/assignnumb/document/assigned_numbers
[Stand: 18.02.2004]

Internetquelle Windows Mobile OS:
URL: http://www.silicon.com/networks/mobile/0,39024665,39117463,00.htm [Stand: 11.02.2004]

8.6 CD

Inhalt der beigelegten CD:

1_Diplomarbeit:
Diese Diplomarbeit im PDF-Format als Druckversion (300dpi) und Bildschirmversion (72dpi).

2_Stadtinfo:

- Quellcode der programmierten Anwendungen auf P800 und MBT.
- Installationsdatei der Stadtinfo-Anwendung für ein SonyEricsson P800/P900-Mobiltelefon.
- Erstellte Testdateien (XML-Dateien, Textdateien und Audiodateien in unterschiedlichen Formaten).

3_Dokumentation:
Alle PDF-Dokumente gemäß Bibliographie, sowie zusätzliche PDF-Dokumente.

4_Video:
Das Video zeigt den Prototypen im Einsatz mit simuliertem Fahrkartenautomat, Anzeige der MBT-Ausgaben in einem MS-DOS-Fenster und Benutzung des P800.