hochschule für angewandte wissenschaften
FACHBEREICH ELEKTROTECHNIK UND INFORMATIK hamburg
university of applied sciences

Diplomarbeit

Alexander Babic

Entwicklung einer profilverarbeitenden ubiquitären Anwendung

Studiengang Softwaretechnik
Betreuender Prüfer: Prof. Dr. Gunter Klemke
Zweitgutachter: Prof. Dr. Kai von Luck
Abgegeben am 14. Februar 2003

BERLINER TOR 3 · D-20099 HAMBURG

Entwicklung einer profilverarbeitenden ubiquitären Anwendung

Stichworte

Ubiquitous Computing, Pervasive Computing, Bluetooth, mobile Dienste

Zusammenfassung

In Zukunft wird es immer mehr mobile Geräte mit integrierter Informationstechnologie geben, die drahtlos und gegebenenfalls spontan miteinander kommunizieren, wobei der Nutzer bzw. dessen Gerät zu jedem beliebigen Zeitpunkt und von jedem beliebigen Ort aus auf mobile Dienste und/oder Informationen zugreifen kann. *Ubiquitous/Pervasive Computing* erforscht diesen Bereich der Informationstechnologie, welcher allerdings noch in seinen Kinderschuhen steckt und dessen Möglichkeiten bei weitem noch nicht ausgeschöpft sind.

In dieser Diplomarbeit wird eine ubiquitäre bzw. pervasive Anwendung exemplarisch entwickelt, welche mittels eines PDAs auf Basis eines Profils arbeitet und mit der *Bluetooth-Technologie* umgesetzt wurde. Bei dieser Anwendung werden Profildaten spontan untereinander ausgetauscht und ausgewertet.

Development of a profile based ubiquitous application

Keywords

ubiquitous computing, pervasive computing, bluetooth, mobile services

Abstract

In future there will be more and more mobile devices with integrated informations-technology, which wireless, and if need be, communicate spontaneously with one another, whereby the user respectively whose device at any suitable time and from any suitable place can get access for mobile services and/or informations. *Ubiquitous/Pervasive Computing* research into the area of informations-technology, which certainly is still in its buds and all it possibilities are by far not been exhausted.

In this degree dissertation, an ubiquitous respectively pervasive application will be exemplary developed, of which a PDA means on a profile bases works and would be realized with the bluetooth-technology. By this application profile datas will be spontaneously replaced within one another, will be evaluated and the results will be interpreted.

Inhaltsverzeichnis

Kapitel 1

Einleitung

In naher Zukunft werden immer mehr mobile Geräte mit integrierter Informationstechnologie den Markt erobern. Diese Geräte werden drahtlos mit anderen mobilen und stationären Geräte kommunizieren, wobei angestrebt wird, dass das Gerät bzw. dessen Benutzer zu jedem beliebigen Zeitpunkt und von jedem beliebigen Ort aus auf mobile Dienste und/oder Informationen zugreifen kann. *Ubiquitous/Pervasive Computing* erforscht diesen Bereich der Informationstechnologie, welcher allerdings noch in den Kinderschuhen steckt und dessen Möglichkeiten bei weitem noch nicht ausgeschöpft sind.
Die Vision von *Ubiquitous/Pervasive Computing* geht sogar soweit, dass in ferner Zukunft informationstechnische Geräte in Alltagsgegenstände eingebettet sind, so dass sie als solche nicht mehr wahrgenommen werden. Ein Beispiel hierfür wird nachfolgend angegeben:

Intelligenter Regenschirm

Es wären intelligente Regenschirme denkbar, die über das Internet einen Wetterdienst abonniert haben. Bei einem regnerischen Wetter wird über einen Display an der Haustür eine Nachricht angezeigt, die darauf aufmerksam macht, dass es draussen regnet. Die hierfür benötigte Informations- und Funktechnologie ist im Regenschirm untergebracht und optisch nicht mehr wahrnehmbar. [Mattern]

Die oben angeführten mobilen Dienste[1] lassen sich in zahlreiche Anwendungsgebiete einteilen, wobei ein Teil der Anwendungen bereits heute schon real ist und ein anderer Teil in absehbarer Zukunft möglich sein wird. Einige Anwendungsgebiete werden nachfolgend vorgestellt, wobei es im Einzelnen zu Überschneidungen kommen kann:

Mobile Finanzdienstleistungen

Der Kunde wird in der Lage sein, seine Rechnungen mobil abzuwickeln (mobile payment). Dabei kann er vor Ort an der Kasse mit seinem Handy bezahlen. Über das Handy wird er Banküberweisungen tätigen und sein Kontostand abfragen können (mobile banking). Auch wird der Ankauf bzw. Verkauf von Aktien über das Handy möglich sein (mobile brokering).

[1] Viele mobile Dienste werden hier in Verbindung mit einem Handy erläutert. Handys sind als Plattform für die Nutzung von mobilen Dienste gut geeignet, da sie einen sehr hohen Verbreitungsgrad haben. Natürlich kommen auch andere mobile Geräte wie zum Beispiel PDA, Smartphone oder Laptop in Frage. Smartphone-Geräte bilden eine Kombination aus Handy- und PDA-Geräten.

Mobile Informationsdienstleistungen

Heute werden bereits eine Fülle von Informationsdienstleistungen angeboten. Dabei können Informationen abonniert und als SMS[2] versendet werden. Die Informationen werden im wesentlichen in den Bereichen Wetter, Nachrichten, Finanzen, Sport und Verkehr angeboten. In naher Zukunft werden ortsbezogene und stark personalisierte Informationsdienste angeboten. Im Rahmen dieser Diplomarbeit wird eine ubiquitäre Anwendung[3] entwickelt, welche die Eigenschaften eines ortsbezogenen und eines personalisierten Dienstes erfüllt.
Bei den ortsbezogenen Informationsdiensten verändert sich das Informationsangebot für den mobilen Benutzer mit der Änderung seiner Lokalität. Das folgende Anwendungsbeispiel soll dies verdeutlichen:

Ortsbezogener Informationsdienst

Kunden bewegen sich mit ihren mobilen Endgeräten in der Verkaufshalle des Supermarktes. Dem Kunden werden an unterschiedlichen Orten unterschiedliche Produktinformationen auf seinem mobilen Endgerät angezeigt. Befindet man sich mit seinem Einkaufswagen im Gang für Süsswaren, dann werden nur Informationen über Süsswarenprodukte auf dem Display des Gerätes angezeigt. Befindet man sich im Gang für Getränke, dann werden nur Informationen über Getränke angezeigt etc.. Die Anzeige der Produktinformationen kann dabei spontan und selbstständig geschehen, ohne dass der Kunde jedesmal die Produktinformation selbst über das Gerät anfordern muss.

In der heutigen Welt ist der einzelne Kunde einer hohen Informationsflut ausgesetzt. Personalisierte Informationsdienstleistungen ermöglichen dem Kunden, nur die für ihn relevanten Informationen auf seinem mobilen Gerät anzeigen zu lassen. Damit dies möglich ist, muss ein Kundenprofil auf dem mobilen Gerät vorhanden sein. Die Daten des Kundenprofils werden vom Kunden entweder selbst explizit angegeben oder implizit aus der Analyse des Kundenverhaltens gewonnen. Dabei muss der Kunde selbst festlegen können, welche Aspekte seines Profils für welchen mobilen Dienst verfügbar sind. Folgendes Anwendungsbeispiel zur Verdeutlichung:

Personalisierter Informationsdienst

Denkt man das oben angegebene Supermarkt-Beispiel (Beispiel für ortsbezogenen Informationsdienst) weiter, so könnte auf dem mobilen Endgerät ein persönliches Profil, welches alle möglichen Daten des Kunden enthält, angelegt werden. Dieses Profil ist so konfiguriert, dass der Informationsdienst nur auf Daten bezüglich den Produktinteressen über Funk Zugriff hat. Beinhaltet der Bereich Produktinteressen des Profils die einzige Information *nur Sonderangebote interessieren mich*, dann werden dem Kunden nur Sonderangebote auf seinem mobilen Gerät angezeigt.

[2] SMS steht für Short Messages Services und avancierte in den letzten Jahren zu einem der erfolgreichsten und beliebtesten mobilen Datendienste. Bei diesem Datendienst können Handy-Benutzer Kurztextmitteilungen senden und empfangen. Immerhin wurden im Jahr 2000 weltweit 200 Milliarden Nachrichten per SMS verschickt.

[3] Die im Rahmen dieser Diplomarbeit entwickelte ubiquitäre Anwendung wird im Abschnitt 1.1 auf Seite 4 vorgestellt.

Mobile Unterhaltung

Der mobile Benutzer wird die Möglichkeit haben, mobile Unterhaltung unterwegs nutzen zu können (mobile entertaiment). Einige der hier aufgelisteten Unterhaltungsdienste sind Zukunftsvisionen, andere sind heute schon bereits nutzbar:

- Streaming-Dienste für Musik und Videos
  Streaming bedeutet, dass der Benutzer sich während des Herunterladens einer Musikdatei bzw. Videodatei die Musik anhören bzw. das Video anschauen kann.
- Dienste für das Versenden von Musik, Bilder und Videos
- Infotainment-Dienste
  Infotainment umfasst u.a. SMS-Soaps[4], Horoskope, Meldungen über prominente Personen sowie aktuelle Sportmeldungen.
- SMS-Chat-Dienste
  Handy-Benutzer tauschen mit anderen Benutzern in einem virtuellen Raum Kurztextmitteilungen aus.
- Spiel-Dienste
  Der mobile Nutzer kann unterwegs mit entfernten Nutzern ein gemeinsames Spiel spielen, wie bei den Online-Spielen im Internet.

Mobiles Shopping

Der mobile Benutzer wird schätzungsweise in einigen Jahren Zugang zu den meisten Angeboten, welche sich rund um Einkaufen drehen und bereits im Internet etabliert sind, über sein mobiles Endgerät haben. Dabei umfasst das mobile Einkaufen (mobile shopping) u.a. den Kauf- und Verkauf von Waren, Tickets, Auktionshandel, Abrufen von Verbraucherinformationen und Werbung. Der mobile Kunde wird beim Einkauf im stationären Handel zudem die Möglichkeit haben, einen Preisvergleich anzufordern und kann gegebenenfalls einen neuen Rabatt aushandeln.

Mobiles Internet

In Zukunft wird es möglich sein, über diverse mobile Endgeräte zu jeden beliebigen Zeitpunkt und zu jedem beliebigen Ort Zugriff auf das Internet zu haben (mobile internet). Der mobile Benutzer wird damit über sein mobiles Endgerät auf alle Dienstleistungen, welche im Internet angeboten werden, zugreifen können. Auch in der Arbeitswelt können Mitarbeiter im Aussendienst auf das firmeninterne Intranet und auf das Internet zugreifen. Zum Beispiel könnte das Wartungspersonal über ihr mobiles Endgerät auf Dokumentationen über die zu wartenden Anlage zurückgreifen.
Heutzutage ist das mobile Internet schon beschränkt möglich. Probleme ergeben sich dadurch, dass die meisten entwickelten Standardwebseiten nicht für mobile Endgeräte mit kleinen Bildschirmen und kleiner Farbpalette zur Darstellung geeignet sind. Zudem haben mobile Endgeräte im Vergleich zum herkömmlichen Desktop-PC eine geringere Speicherkapazität und verfügen über eine geringere CPU-Leistung. Hinzu kommt, dass die

[4] SMS-Soaps sind kleine Geschichten aus dem *wirklichen* Leben, die regelmässig als Kurztextmitteilungen (SMS) empfangen werden.

Netzwerkverbindungen nur über eine geringere Übertragungsrate verfügen. Diese Fakten machen im Moment die Nutzung des Internets über mobile Endgeräte nur beschränkt möglich.

Der Bereich *Pervasive/Ubiquitous Computing* ist ein stark wachsender Milliardenmarkt. In Zukunft werden in diesem Bereich viele weitere innovative Anwendungen zu entdecken sein. Da die Anwendungen in diesem Bereich sehr zahlreich sind, ist es nicht möglich, alle Anwendungen in dieser Diplomarbeit anzugeben, deshalb wird hier auf die Literatur von [Burkhardt u. a. (2001)], [Britta Oertel (2001)] und [Amor (2002)] verwiesen. Welche mobile Dienste sich im Einzelnen auf dem Markt durchsetzen werden, hängt von folgenden entscheidenden Faktoren ab:

- Mobile Endgeräte, über welche die mobilen Dienste genutzt werden, müssen einfach und unkompliziert in ihrer Bedienung sein.
- Mobile Dienste müssen ebenfalls einfach und unkompliziert nutzbar sein.
- Mobile Dienstleistungen sollten eine einfache Tarifstruktur besitzen.
- Preise für mobile Endgeräte und mobile Dienstleistungen sollten in einem angemessenen Rahmen liegen.
- Mobile Dienstleistungen sollten für den Benutzer einen klar erkennbaren Nutzen haben.
- Datenschutzrechtliche Aspekte müssen von den Anbietern mobiler Dienstleistungen ausreichend berücksichtigt werden, da gerade bei den mobilen Endgeräten viele Daten über Funk gesendet werden. Es muss die Option zur Datenverschlüsselung bei Datenübertragungen via Funk gegeben sein. Die Benutzer müssen sich gegebenenfalls authentifizieren und Transaktionen müssen eventuell autorisiert werden.
- Da es viele Hersteller und noch mehr verschiedenartige pervasive Geräte mit unterschiedlichen Fähigkeiten gibt, müssen einheitliche Standards und Schnittstellen geschaffen werden.

1.1 Ziel und Motivation dieser Diplomarbeit

Das Ziel dieser Diplomarbeit ist es, eine profilverarbeitende, ubiquitäre Anwendung zu entwickeln, wobei ich mich bei der Art der Anwendung für eine Flirtmaschine entschieden habe. Über die *Flirtmaschine* sollen sich in Funkreichweite befindliche Profile spontan paarweise miteinander abgleichen und dabei auf Übereinstimmung geprüft werden. Hierbei haben die Teilnehmer zuvor über die *Flirtmaschine* die Möglichkeit, ein Profil zu erstellen, das sie hinsichtlich ihrer äusserlichen Erscheinung und Gewohnheiten charakterisiert, und welches die von ihnen eingegebenen Interessen und gewünschten Suchkriterien bei der Suche nach einem übereinstimmenden Profil berücksichtigt. Nähert sich ein Teilnehmer jetzt mit seiner mobilen Einheit einem anderen Teinehmer ausreichend, sollen die Geräte heimlich und spontan drahtlos ihre Flirtprofile austauschen. Sollte der Abgleich beider Flirtprofile einen bestimmten Übereinstimmungsgrad erzielen, wird jedem der Teilnehmer die Nachricht gemeldet, dass ein übereinstimmendes Profil gefunden worden ist.

Diese *Flirtmaschine* stellt einen Dienst zur Verfügung, welcher sich der Gruppe von personalisierten Diensten und ortbezogenen Diensten zuordnen lässt. Es ist ein personalisierter Dienst, da der Benutzer ein eigenes Profil über sich erstellt, das beim Abgleich mit den Profildaten eines anderen Benutzers übereinstimmen muss, damit ein Match gemeldet wird. Es ist ebenso ein ortsbezogener Dienst, da sich mit der Veränderung der Position des Benutzers im ubiquitären Raum auch die sichtbaren Profile ändern. Folgende zwei Szenarien wären für den Einsatz der Flirtmaschine vorstellbar:

Szenario Cafeteria

Zwei Personen befinden sich in einer Cafeteria und tragen jeweils ihre mobile Einheit bei sich, auf der die Flirtmaschine ausgeführt wird. Sobald beide Einheiten in Funkreichweite kommen, werden die Profile miteinander abgeglichen. Bei einer Übereinstimmung werden beide informiert und können sofort einen Kaffee miteinander trinken.

Szenario Stadt

Zwei Personen sind mit ihrer *Flirtmaschine* in der Stadt unterwegs. Sobald sie aneinander vorbeilaufen, bekommen beide im Falle einer Profilübereinstimmung eine dementsprechende Nachricht. Sie können sich jetzt ebenfalls sofort kennenlernen.

Die Motivation bei dieser Diplomarbeit liegt für mich vor allem im stark anwachsenden Zukunftsmarkt der *Ubiquitous/Pervasive Computing*. Der Bereich *Ubiquitous/Pervasive Computing* ist noch nicht ausgereift und steht am Anfang seiner Erforschung. Er stellt insbesondere unter dem Aspekt spontan kommunizierender Anwendungen auf mobilen Einheiten eine Neuheit dar, ein Faktum, das mich speziell sehr interessiert, wobei eine Kommunikationstechnologie wie die *Bluetooth-Technologie* der Schlüssel zur Umsetzung solcher ubiquitären Anwendungen ist. So wurde bei meiner Anwendung der *Flirtmaschine* auf diese *Bluetooth-Technologie* zurückgegriffen, auch aus dem persönlichen Interesse heraus, erste Erfahrungen und Fertigkeiten in diesem Bereich zu erwerben, um diese im späteren Berufsleben nutzen zu können.

1.2 Eingetragene Warenzeichen

Java ^{TM} ist ein eingetragenes Warenzeichen von Sun Microsystems.

Bluetooth ^{TM} ist ein eingetragenes Warenzeichen der schwedischen Firma Ericsson.

iPAQ ^{TM} und *Compaq* ^{TM} sind eingetragene Warenzeichen der Firma Compaq Computer Corporation.

Microsoft ^{TM} ist ein eingetragenes Warenzeichen der Firma Microsoft Corporation.

Kapitel 2

Grundlagen

Dieses Kapitel soll in die begrifflichen und in die technologischen Grundlagen einführen, die bei der Entwicklung der *Flirtmaschine* zum Einsatz kommen.

Im Abschnitt 2.1 werden die Begriffe *Pervasive Computing* und *Ubiquitous Computing* definiert. Auch der Unterschied zwischen diesen beiden Begriffe wird geklärt. Ausserdem wird eine Definition für *pervasive* und *ubiquitäre Geräte* angegeben.

Im Abschnitt 2.2 wird auf den Einsatz und die Motivation bei Erstellung von Kundenprofilen eingegangen. Es wird der Sinn eines einheitlichen Standards für Profile aufgezeigt. Schliesslich wird diskutiert, welche datenschutzrechtliche Aspekte beachtet werden müssen.

Im Abschnitt 2.3 gibt es eine kleine Exkursion über Verteilte Systeme, da die *Flirtmaschine* Eigenschaften eines Verteilten Systems erfüllt. Befindet sich ein mobiler Benutzer mit seinem bluetooth-basierten PDA, auf dem die *Flirtmaschine* ausgeführt wird, alleine im ubiquitären Raum, dann liegt noch kein Verteiltes System vor. Dies ist erst der Fall, wenn mindestens zwei oder mehrere in einem gemeinsamen ubiquitären Raum aufeinander treffen. Es wird darum als ein Verteiltes System bezeichnet, da hier ein gemeinsamer Zugriff auf eine Ressource, in diesem Fall auf das Profil eines Benutzers, von mehreren Benutzern erfolgt.

Im Abschnitt 2.4 wird das *Java*-Konzept für mobile Geräte beschrieben, da für die *Flirtmaschine* die Programmiersprache *Java* eingesetzt wurde. Im Abschnitt 2.5 wird die Bluetooth-Technologie näher vorgestellt. Mit dieser Bluetooth-Technologie wird die Umsetzung ubiquitärer Anwendungen möglich. Im Abschnitt 2.6 wird schließlich *JaxB* vorgestellt. *JaxB* ist eine reine Java-Technologie, um XML-Dokumente zu verarbeiten und wurde für die Datenhaltungskomponente im Abschnitt 4.5 eingesetzt.

2.1 Ubiquitous Computing / Pervasive Computing

2.1.1 Definition

Im Internet habe ich die folgende Definition von *Ubiquitous Computing* und *Pervasive Computing* gefunden:

> *Unter den beiden oft äquivalent gebrauchten Begriffen **Pervasive Computing** und **Ubiquitous Computing** wird die Allgegenwärtigkeit von Informationsverarbeitung und damit einhergehend der jederzeitige Zugriff auf Daten von beliebiger Stelle aus verstanden.*
> Zitat von Friedemann Mattern ([Mattern]; Seite 1)

Damit ist gemeint, dass Benutzer mit Hilfe mobiler Geräteeinheiten im Idealfall zu jedem Zeitpunkt und an jedem beliebigen Ort über Funk Zugriff auf diverse Dienste haben. Dabei werden die Dienste an unterschiedlichen Orten angeboten:

- Im Internet
  Zum Beispiel die Nutzung eines Emaildienstes mit einem internetfähigen PDA[1].
- Auf fremde mobile Geräteeinheiten
  Zum Beispiel das Herunterladen einer Visitenkarte von einem PDA eines anderen Benutzers.
- Auf eigenen mobilen Geräteeinheiten
  Zum Beispiel die Nutzung eines SMS-Dienstes über das eigene Handy .
- Auf einem Server
  Zum Beispiel das Einsehen einer Speisekarte durch Serveranbindung des Restaurants (Zukunftsvision).

Diese über Funk kommunizierenden mobilen Geräteeinheiten werden auch als *Pervasive Geräte* oder *Ubiquitäre Geräte* bezeichnet. Mit Bezug auf die Literatur von [Mattern] und [Burkhardt u. a. (2001)] kristallisiert sich für mich folgende Definition heraus:

Def: Pervasive/Ubiquitäre Geräte

Unter dem Begriff pervasive und ubiquitäre Geräte werden mobile Geräte mit integrierter Informationstechnologie verstanden, die jederzeit mit anderen mobilen Geräten (z.B: Handy, PDA, Laptop etc.) und stationären IT-Systemen mit Hilfe kabelloser Kommunikationstechnologien, hier sei als Beispiel Bluetooth[2] genannt, miteinander kommunizieren können. Dabei kann die Kommunikation unter den Geräten spontan erfolgen.

Quelle: [Burkhardt u. a. (2001)], [Mattern]

[1] PDA steht für Personal Digital Assistent. Dies ist ein mobiles Gerät mit PC-Funktionalitäten und wird oft auch als Organizer bezeichnet. Ein PDA kann mit einer Funktechnologie ausgestattet sein, um drahtlos mit anderen informationstechnischen Geräte kommunizieren zu können.

[2] Bluetooth ist eine Kurzfunktechnik. Eine Einführung in die Bluetooth-Technologie wird im Abschnitt 2.5 auf Seite 17 gegeben.

2.1.2 Der kleine Unterschied

Der Begriff *Ubiquitous Computing* wird oft mit dem Begriff *Pervasive Computing* synonym behandelt und wurde vor mehr als 10 Jahren von Mark Weiser geprägt:

> *The most profound technologies are those that disappear. They weave themselves into the fabric of everyday life until they are indistinguishable from it.*
> Zitat von Mark Weiser ([Weiser (1991)]; Seite 66)

Mark Weiser verstand unter dem Begriff *Ubiquitous Computing* die Einbettung der Informationstechnologie in ganz gewöhnliche Alltagsgegenstände. Dabei verschwindet die eingebettete Informationstechnologie in den Hintergrund. Das bedeutet, bei der Nutzung von ubiquitären Gegenständen wird die Informationstechnologie nicht mehr bewusst wahrgenommen. Diese Alltagsgegenstände sollen in der Lage sein, mit anderen Geräten oder Gegenständen in ihrer Umgebung drahtlos kommunizieren zu können. Ein Beispiel hierfür wäre der *Intelligente Regenschirm* von Seite 1.

Der kleine Unterschied [Mattern] zwischen *Ubiquitous Computing* und *Pervasive Computing* besteht darin, dass bei *Ubiquitous Computing* die informationsverarbeitende Technologie nicht wahrgenommen wird und erst in weiter Zukunft realiserbar ist, während bei *Pervasive Computing* die Nutzung von *electronic commerce-Szenarien* und Web-basierten Geschäftsprozessen im Vordergrund steht. Als Beispiel für *Pervasive Computing* wäre das *Mobile Shopping* von Seite 3 zu nennen.

Im weiteren Verlauf dieser Diplomarbeit werden diese beiden Begriffe als Synonyme behandelt.

Quelle: [Weiser (1991)], [Mattern]

2.2 Einsatz von Profilen

2.2.1 Motivation von Profilen

Drehen wir die Zeit um 40 bis 50 Jahre zurück, als noch viele kleine Tante Emma Läden existierten. Der Tante Emma Laden um die Ecke ist ein Supermarkt im Miniformat, in welchem der Besitzer des Ladens meist auch die Hauptarbeitskraft gewesen ist. Der Ladenbesitzer oder auch Händler besass damals relativ viele Stammkunden. Meist kannte er diese persönlich mit Namen und oft sogar auch deren Familien- und Einkommensverhältnisse, Bedürfnisse, Nöte sowie Sorgen und vieles mehr. Kurz gesagt, er hat sich ein Gesamtbild von jedem einzelnen Kunden gemacht, sprich, sich zu jedem seiner Kunden ein Kundenprofil in seinem Gedächtnis angelegt. Er wusste, bei welchen Kunden er die Rechnung anschreiben konnte und bei welchem er es lieber sein lassen sollte. Er konnte auf die Kunden individuell eingehen, sie beraten und ihnen entsprechend Angebote machen. Für ein kleines privates Gespräch mit dem Kunden fand er oft auch noch Zeit, so dass sich zwischen dem Kunden und dem Händler eine vertraute Beziehung enwickeltn konnte. Die Kunden waren zufrieden, da sie sich von ihrem Händler optimal betreut fühlten, was wiederum dem Händler die Stammkundschaft und den Umsatz sicherte.

Die Erstellung von Kundenprofilen, um dann auf den Kunden individuell einzugehen, wie es der Betreiber eines *Tante Emma Laden's* aus dem obigen Beispiel gemacht hat, nennt man heute im E-Commerce und Marketing-Bereich *Personalisierung*. Die Personalisierung kann dabei verschiedene Formen annehmen [Stolpmann (2000)]:

- Unterbreitung von individualisierten Angeboten (Dienstleistungen und Produkte).
- Persönliche Anrede des Kunden.
- Interaktive, individuelle Beratung und Support
- Individualisiertes Marketing
- Bereitstellung von indivdualisierten Internetinhalten

Das Ziel der *Personalisierung* ist es, eine stabile, vertraute und langfristige Beziehung zwischen dem Unternehmen und dem Kunden aufzubauen und natürlich Neukunden zu aquirieren. Durch die individuelle Betreuung bekommt der Kunde das Gefühl vermittelt, wichtig für das Unternehmen zu sein. Er fühlt sich exklusiv behandelt, wodurch sich seine Zufriedenheit erhöht, was letztendlich den Unternehmenserfolg steigert.

Heute haben viele Unternehmen erkannt, dass niedrige Preise allein nicht entscheidend sind, um Kunden an das Unternehmen langfristig zu binden und neue Kunden zu gewinnen. Sie bedienen sich daher der Personalisierung. Damit die Unternehmen Personalisierung betreiben können, müssen sie Profile über ihre Kunden erstellen. In Anlehnung an [Dorka (Juli 2001)], [Stolpmann (2000)] und an [Bullinger u. a. (2002)] schlussfolgere ich die nachstehende Definition:

Def: Kundenprofil

Unter einem Kundenprofil wird eine Ansammlung von Kundeninformationen verstanden, die direkt einem einzelnen Kunden zugeordnet werden können und die den Kunden hinsichtlich seines Verhaltens, seiner Person und seinen Präferenzen charakterisiert.

Um Kundenprofile erstellen zu können, müssen Kundeninformationen unter Beachtung datenschutzrechtlicher Gesetze gesammelt werden. Für das Gewinnen von Kundeninformationen gibt es unterschiedliche Methoden, welche sich grob in zwei Klassen aufteilen lassen:

- *Die explizite Infomationsgewinnung*
  Bei der expliziten Methode wird der Kunde direkt aufgefordert, seine Daten preis zu geben.
- *Die implizite Infomationsgewinnung*
  Bei der impliziten Methode versucht man neue Informationen über den Kunden zu gewinnen, indem man das Verhalten des Kunden untersucht. Dabei ist es dem Kunden nicht unbedingt bewusst, dass man über ihn Informationen sammelt.

Bei der *Flirtmaschine* geschieht die Kundeninformationsgewinnung explizit, der Benutzer kann seine Profildaten direkt über die GUI der *Flirtmaschine* eingeben. Da andere Formen der Informationsgwinnung bei der *Flirtmaschine* keine Rolle spielen, wird für weitere Informationen über die Kundeninformationsgewinnung auf die Literatur von [Bullinger u. a. (2002)], [Stolpmann (2000)] und [Dorka (Juli 2001)] verwiesen.

Im folgenden sollen noch 3 visionäre Beispiele für den Einsatz von Profilen mit dem besonderen Augenmerk auf die Welt des Ubiquitous Computing vorgestellt werden:

1. Anwendungsbeispiel

Ein Anwendungsbeispiel ist das bereits eingeführte Beispiel für den *Personalisierten Informationsdienst* von Seite 2 (Supermarkt-Beispiel; Einkaufsprofil).

2. Anwendungsbeispiel

Man befindet sich am Abend in der Stadt, möchte noch etwas unternehmen und schaltet sein PDA ein. Dieser verbindet sich über Funk mit einem lokalen Server. Dieser Server ermittelt die zum Profil des Benutzers passenden Veranstaltungstipps und sendet sie zur Ansicht an das PDA. Sollte sich also der Benutzer laut Profil für Jazz-Musik interessieren, dann werden alle lokalen Veranstaltungen, bei denen Jazz-Musik gespielt wird, auf dem PDA angezeigt.

3. Anwendungsbeispiel

Im Restaurant lässt man sich nicht wie gewohnt die Speisekarte bringen, sondern sieht ganz bequem über den PDA in seine individuelle Speisekarte ein. Der PDA verbindet sich mit dem lokalen Server des Restaurants. Der Server ermittelt nur die Liste der Gerichte, die dem Restaurantbesucher wirklich schmecken. Mag der Besucher partout keine Pilze, so werden keine Pilzgerichte angezeigt.

Beim Einsatz von Profilen muss nicht unbedingt der kommerzielle Aspekt vorhanden sein. Bei Interaktionen von ubiquitären Systemen können auch ausschliesslich private Personen beteiligt sein. Ein Beisiel dafür ist die im Rahmen dieser Diplomarbeit entwickelte ubiquitären Anwendung mit der Bezeichnung *Flirtmaschine*. Es wären viele weitere Visionen denkbar, die schon in naher Zukunft Realität sein könnten. Deshalb soll für weitere Visionen aus der Welt des *Ubiquitous Computing* und *Pervasive Computing* an dieser Stelle nochmals auf die Literatur von [Amor (2002)] und [Burkhardt u. a. (2001)] verwiesen werden.

Meine persönliche Vision von *Ubiquitous Computing* und *Pervasive Computing* unter Verwendung von Profilen geht soweit, dass jeder Benutzer auf seiner ubiquitären oder pervasiven Geräteeinheit nur ein einziges, umfangreiches und universelles Profil für alle möglichen Anwendunggebiete über sich anlegt. Dieses universelle Profil kann von anderen ubiquitären oder pervasiven Systemen unter Beachtung datenschutzrechtlicher Aspekte genutzt werden. Der Profileigner sollte dabei die Option haben, unterschiedliche Datenschutzregeln für unterschiedliche Teile seines Profils aufstellen zu können.

Beispiel

Der Profileigner legt fest, dass nur solche Profildaten, die seine Produktinteressen wiederspiegeln, für die Öffentlichkeit zugänglich sind. Der Supermarkt um die Ecke, wo der Profileigner stets einkaufen geht, hat also keinen Zugriff auf irgendwelche anderen Daten des Profils ausser auf die Produktinteressen des Kunden.

Diese Vision ist natürlich nicht möglich, wenn die Hersteller ubiquitärer Anwendungen eine proprietäre Darstellung für die Profildaten verwenden. Damit hätten ubiquitäre Anwendungen anderer Hersteller beim Zugriff auf solche proprietären Profildaten Schwierigkeiten, diese zu interpretieren. Um diese Vision umsetzen zu können, ist eine Standardisierung der Profile unausweichlich. Besonders erwähnenswert ist der *Customer Profile Exchange*-Standard, auf den im Abschnitt 2.2.2 eingegangen wird.

Quelle: [Stolpmann (2000)], [Dorka (Juli 2001)], [Bullinger u. a. (2002)]

2.2.2 Standard für Profile

Benutzen Unternehmen propritäre Profilspezifikationen, um ihre Kundenprofile zu erzeugen, bringt dies einen erheblichen Nachteil mit sich, welcher darin besteht, dass der Austausch von Kundenprofilen zwischen kooperierenden Unternehmen nicht so einfach möglich ist. Die Unternehmen müssen erst einen Abbildungsmechanismus, der die externen Profile auf ihre propritäre Profilspezifikation abbildet, entwickeln, um die externen Kundenprofile in ihren Systemen integrieren zu können. Die Integration externer Kundenprofile führt damit zu einem erhöhten Aufwand.

Um den Austausch von Kundenprofilen zwischen Unternehmen zu erleichtern, wird für die Darstellung ein Standard benötigt. Der *Customer Profile Exchange*-Standard (CPExchange) stellt ein solchen Standard dar. Der *CPExchange*-Standard wurde bei der Entwicklung der *Flirtmaschine* nicht eingesetzt, um den Umfang dieser Arbeit auf das geforderte Maß zu beschränken. Trotzdem hielt ich diesen Standard für erwähnenswert, da der Einsatz solch eines Standards sicherlich sinnvoll gewesen wäre, wenn man die Vision[3] von einem einzigen, umfangreichen und universellen Profil für die Entwicklung der *Flirtmaschine* umsetzen möchte.

[3] Die Vision vom universellen Profil findet man im Abschnitt 2.2.1 auf Seite 10.

IBM und mehr als 70 weitere namenhaften Unternehmen bilden zusammen ein Konsortium mit der Bezeichnung *Customer Exchange Profile Network* (CPExchange Network) und haben sich das Ziel gesetzt, die Spezifikation von Kundenprofilen im E-Business Bereich als offenen Standard zu etablieren. Ende Oktober 2000 wurde die *Customer Profile Exchange*-Spezifikation in der Version 1.0 [Bohrer und Holland (Oktober 2000)] von der *CPExchange Network* veröffentlicht. Dieser Standard soll den im E-Business tätigen Unternehmen den Austausch von Kundenprofilen unter Berücksichtigung der Datenschutzaspekte und die Wahrung der Privatssphäre erleichtern.

Die *CPExchange*-Spezifikation basiert auf das XML-Schema und der P3P[4]-Spezifikation. Mit dem in der *CPExchange*-Spezifikation definierten Datenmodell kann man Kundenprofile beschreiben. Die in XML-Format vorliegenden Profile werden über Standard-Internetprotokolle übertragen. Für unterschiedliche Datenteile eines Profils können unterschiedliche Regeln für den Datenschutz festgelegt werden.
Die Daten eines Profils können unter anderem folgende Daten umfassen: demografische Daten (z.B.: Geschlecht, Alter, Einkommen, Ausbildung), Adressen, Namen, Telefonnummer, Informationen über Interaktionen zwischen dem Kunden und dem Unternehmen und Präferenzen, die entweder explizit vom Profilinhaber angegeben worden sind oder aus dem Verhalten des Kunden geschlussfolgert worden sind. Für weitere Details der *CPExchange*-Spezifikation wird auf die Originalspezifikation [Bohrer und Holland (Oktober 2000)] verwiesen.

Quelle: [Bohrer und Holland (Oktober 2000)], [Dorka (Juli 2001)]

2.2.3 Datenschutzrechtliche Aspekte

Unternehmen müssen beim Sammeln von Kundeninformationen datenschutzrechtliche Aspekte berücksichtigen. Ohne Einwilligung des Kunden dürfen nur nicht-personenbezogene Kundeninformationen gesammelt werden, ansonsten wäre eine Einwilligung des Kunden notwendig. Die Unternehmen müssen den Kunden die Möglichkeit einräumen, selbst zu entscheiden, welche Daten sie unter welchen Bedingungen preisgeben wollen.

Die im Internet tätigen Unternehmen können ihre angewandten Datenschutzpraktiken mit dem P3P-Standard angeben. Der P3P-Standard (Platform for Privacy Preferences Project) ist ein Standard im Internet, mit dem die Unternehmen die Möglichkeit haben, eine genormte Datenschutzerklärung in XML-Format auf ihren Webseiten im Internet zu veröffentlichen. Die Besucher solcher Webseiten haben mit der Hilfe eines P3P-basierten Internetbrowser die Möglichkeit, sich die Datenschutzerklärung bequem über standardisierte Dialoge anzuschauen und können nun entscheiden, ob sie unter den angegebenen Bedingungen bereit sind, ihre persönlichen Daten anzugeben. Darüber hinaus können die Benutzer in einem P3P-fähigen Internetbrowser die Regeln, wie ihre persönlichen Daten geschützt werden sollen, aufstellen und den Internetbrowser automatisch entscheiden lassen, welche Webseiten akzeptabel oder inakzeptabel sind.

[4] Auf den P3P-Standard wird im Asbchnitt 2.2.3 auf Seite 12 eingegangen.

Für die zugrundeliegenden Profildaten der *Flirtmaschine* macht die Beachtung eines Datenschutzes meiner Meinung nach wenig Sinn, da die Benutzerdaten im Interesse des Benutzers öffentlich und unkompliziert für jeden zugänglich sein sollten, das heisst ohne ständige vorherige Abfrage der Sicherheitsaspekte. Deshalb, und um den Umfang dieser Diplomarbeit nicht zu übersteigen, wurde auf weitere Details der P3P-Spezifikation verzichtet. Trotzdem hielt ich den P3P-Standard für erwähnenswert, da zu einem die Beachtung von Datenschutz im allgemeinen für Profile wichtig sind und da zum anderen der *Customer Exchange Profile Standard* (siehe Abschnitt 2.2.2 auf Seite 11) auf der P3P-Spezifikation basiert. Weitere Informationen zu dem P3P-Standard findet man auch unter http://www.w3.org/P3P/ .

Quelle: [Langheinrich (2001)]

2.3 Verteilte Systeme

In der Literatur [Coulouris u. a. (2002)] habe ich folgende Definition über Verteilte Systeme gefunden.

Def: Verteilte Systeme

Bei einem Verteilten System arbeiten Hardware- und Softwarekomponenten, die sich auf vernetzten Computern befinden,zusammen. Sie kommunizieren nur über den Austausch von Nachrichten und koordinieren somit ihre Aktionen.

Die Motivation für Verteilte Systemen liegt darin, gemeinsame Ressourcen, bestehend aus Hardware oder Software, zu nutzen. Server können Resourcen verwalten, damit die Clients auf diese Resourcen zugreifen können. Beispiele für Ressourcen wären Drucker und Daten in einer Datenbank. Mehrere Benutzer können gemeinsam den Drucker im Netzwerk für ihre Druckdienste nutzen oder sie können gemeinsam auf dieselben Daten zugreifen.

Beispiel: Internet

Ein bekanntes Beispiel für ein Verteiltes System ist das Internet. Das Internet bildet sich aus dem weltweiten Zusammenschluss von einer sehr grossen Anzahl unterschiedlicher Arten von Computernetzwerken. Das Internet ermöglicht den Benutzern, Dienste, die im Internet von einem beliebigen Ort auf der Welt aus angeboten werden, in Anspruch zu nehmen. Dienste könnten u.a. Emaildienste oder das Herunterladen von Dateien sein. Dabei können die Art und die Anzahl der Dienste beliebig erweitert werden.

Beispiel: Pervasive/Ubiquitous Computing

Ubiquitäre oder pervasive Geräte haben die Möglichkeit, sich in Netzwerke, die sich in ihrer Umgebung befinden, über Funk einzubinden und können dort die zur Verfügung stehenden Dienste in Anspruch nehmen, um somit auf die angebotenen Ressourcen zu zugreifen. Ubiquitäre oder pervasive Geräte können natürlich auch über einen Netzwerkzugriffspunkt Zugang auf das weltweite Internet haben.

Weitere Informationen zum Thema *Verteilte System* sind in der Literatur von [Coulouris u. a. (2002)] zu finden.

Quelle: [Coulouris u. a. (2002)]

2.4 Java für pervasive Geräte

2.4.1 Das J2ME Framework

Vor kurzem hat die Firma *Sun Microsystems* damit begonnen, die plattformunabhängige Java-Technologie[5] auch für pervasive Geräte verfügbar zu machen. Pervasive Geräte besitzen recht unterschiedliche Fähigkeiten. Sie unterscheiden sich zum Beispiel in ihren Speicherressourcen, die meist knapp bemessen sind, in ihrer Rechengeschwindigkeit, Grafikfähigkeiten und Betriebssystemen. Die bisherig entwickelte Standardversion der *Java Virtuelle Maschine* für den Desktop-PC ist wegen der unterschiedlichen Charakteristika der pervasiven Geräte nicht auf die kleinen mobilen Geräte zu portieren. Die Portierung schlägt schon alleine deshalb fehl, da die Speicherresourcen der meisten mobilen Geräte lediglich im Größenbereich der Kilobytes liegen, die *Java Virtuelle Maschine* jedoch mehrere Megabytes benötigt.

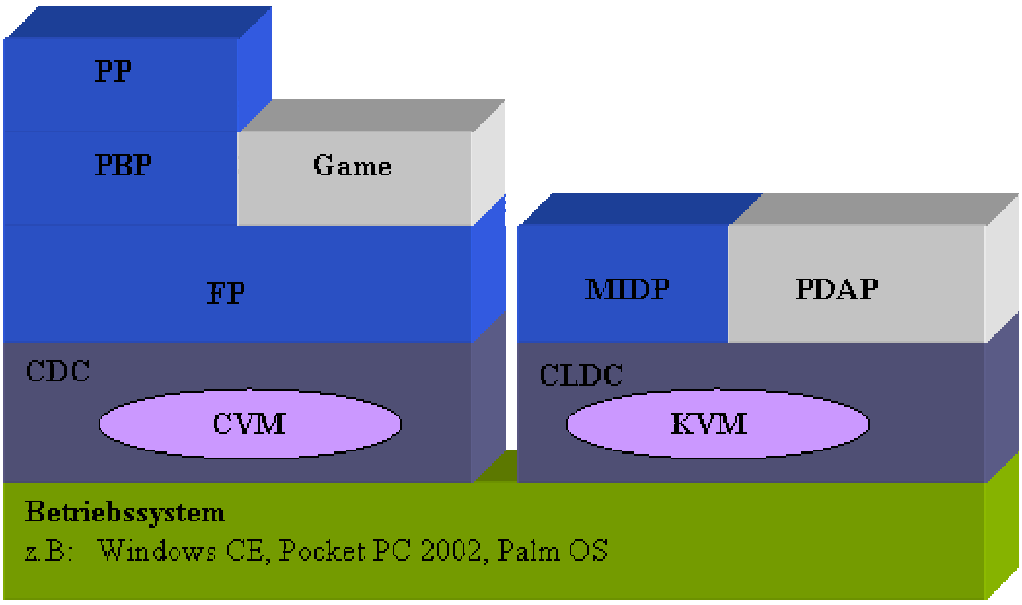

Abbildung 2.1: Das Schichtenmodell des J2ME-Frameworks

Um die Problematik, welche sich aus den unterschiedlichen Fähigkeiten der mobilen Geräte ergibt, in den Griff zu bekommen, entwickelte die Firma *Sun Microsystems* das *J2ME Framework* (Java Micro Edition; siehe Abbildung 2.1) für pervasive Geräte. Dieses Framework[6] setzt sich aus Konfigurationen, Profile und optionale API's zusammen.

[5] Das Konzept der Plattformunabhängigkeit wird mit einer virtuellen Maschine erreicht. Für weitere Informationen zu Java wird auf die Literatur von [Horstmann und Cornell (1999)] verwiesen.

[6] Ein Framework besteht aus einer Menge von zusammenarbeitenden Klassen, die einen wiederverwendbaren Entwurf für eine bestimmte Klasse von Software darstellen [Gamma u. a. (1996)]. Für weitere Informationen wird auch auf die Literatur von [balz2001]verwiesen.

Konfigurationen bestehen aus einer speziellen virtuellen Maschine, die eine minimale API[7] für einen ganz bestimmten Typ von pervasiven Gerät bereitstellt. Profile setzen sich aus einer Sammlung von Java-basierten API's zusammen, mit denen man die Konfigurationen oder andere Profile Schicht für Schicht erweitern kann. Zusätzlich zu den einzelnen Schichten kann man noch optionale API's hinzufügen. So hat man die Möglichkeit, einen individuellen API-Stack zusammenzustellen, welcher auf die Fähigkeiten eines bestimmten pervasiven Gerätes zugeschnitten ist.

2.4.2 Konfigurationen und Profile

In diesem Abschnitt sollen jetzt einige Konfigurationen und Profile aus der Abbildung 2.1 erläutert werden. Dabei sind die durch die hellfarbigen Blöcke dargestellten Profile noch nicht fertiggestellt (Game, PDAP).

Konfigurationen

Connected Device Configuration (CDC)
: CDC-Spezifikation basiert auf der klassischen *Java* Virtuelle Maschine-Spezifikation (J2SE). Sie beinhaltet den vollen Leistungsumfang einer Runtime Environment wie bei einem Desktop-System. Diese Konfiguration ist für mobile Geräte gedacht, die mindestens 2 Megabytes an Speicher zur Verfügung haben. Dieser Konfiguration liegt die *CVM (Connected Virtual Machine)* zu Grunde.

Connected Limited Device Configuration (CLDC)
: CDLC ist eine echte Teilmenge von CDC. Sie ist für mobile Geräte gedacht, die nur beschränkte Ressourcen zur Verfügung haben. Das Speichervolumen dieser Geräte sollte zwischen 128 und 512 KB betragen. Dieser Konfiguration liegt eine *KVM (Kilo Virtual Machine)* zu Grunde. *KVM* ist eine virtuelle Maschine, die für Geräte, deren Hauptspeichergrößen im Kilobyte-Bereich liegen, gedacht.

Profile

Mobile Information Device Profile (MIDP)
: Dieses Profil beinhaltet eine Ansammlung von API's, die speziell auf das Handy zugeschnitten sind.

Foundation Profile (FP)
: Dieses Profil erweitert *CDC* und dient als Basis für andere Profile. Es stellt fundamentale API's aus *J2SE*[8] zusammen, die Klassen und Interfaces aus *java.lang*, *java.security*, *java.util* usw. enthalten.

[7] API steht für Application Programming Interface und ist eine Sammlung von Objekten und Methoden mit klar definierten Schnittstellen und definiertem Aufgabenbereich zum Einsatz in eigenen Anwendungen.

[8] J2SE steht für Java Standard Edition und ist die Java-Plattform für den klassischen Desktop PC. Im Vergleich zu J2ME verfügt sie über einen umfangreicheren Satz an Programmierschnittstellen und virtuelle Maschinen.

Personal Profile (PP)

Das *Personal Profile* ist der Nachfolger des bisherigen *Personal Java 1.2a*-Standard und verhält sich zum Vorgänger voll kombatibel. Für die Entwicklung der *Flirtmaschine* kam jedoch die *Java Virtuelle Maschine* von Jeode, welche der *Personal Java 1.2* Spezifikation entspricht, zum Einsatz. Das bedeutet, dass das *Personal Profile* nicht für die Entwicklung der *Flirtmaschine* verwendet worden ist, da zum Entstehungszeitpunkt dieser Diplomarbeit noch keine geeignete Implementierung des *Personal Profile* aufzutreiben war. Stünde solch eine Implementierung zur Verfügung, wäre sie der *JVM Jeode* vorzuziehen, da das *Personal Profile* den Nachfolger von *Personal Java 1.2* bildet.

> Die API der *Personal* Java *Standard* enthält eine umfangreiche Klassenbibliothek und ist deshalb nur für pervasive Geräte, die einige Megabyte Speicherkapazität zur Verfügung haben, geeignet. Aktuell gibt es *Java Virtuelle Maschinen* auf dem Markt, die sich ausserhalb des *J2ME-Frameworks* befinden und der *Personal Java 1.2 Spezifikation* voll entsprechen. Die *Personal Java 1.2 Spezifikation* steht auf der Webseite *http://java.sun.com/products/personaljava/* zum Herunterladen zur Verfügung.

Personal Basis Profile (PBP)

Erweitert das *Foundation Profile* um weitere Klassen, die für die grafische Präsentation zuständig sind (unter anderem einige AWT-Klassen).

PDA Profile (PDAP)

Dieses Profil setzt auf die *CLDC Konfiguration* auf und soll eine Ansammlung von Programmierschnittstellen zur Verfügung stellen, die auf PDA-Geräte abgestimmt sind.

Der API-Stack kann auf den verschiedenen Ebenen noch durch weitere optionale API's erweitert werden. Es soll hier nur die Bluetooth[9]-API erwähnt werden, da sie als API für die Realisierung der Kommunikationskomponente (siehe Abschnitt 4.7 auf Seite 81) in Frage kommen würde. Die *Bluetooth-API* wird auf der Ebene der CLDC-Konfiguration hinzugefügt. Die *Bluetooth-API* stellt eine Programmierschnittstelle dar, mit der man auf die Bluetooth-Funktionen des Bluetooth-Moduls zugreifen kann. Für weitere Informationen über optionale API's soll hier auf das Internet [SunIntr] verwiesen werden.

Quelle: ([Burkhardt u. a. (2001)]; Seite 103 bis 106), [Dölfer (2002)], [SunIntr]

[9] Eine Einführung in Bluetooth wird im Abschnitt 2.5 auf der Seite 17 gegeben.

2.5 Bluetooth

Bluetooth ist eine Funktechnik für den Nahbereich, die drahtlose Kommunikation in einem Radius von 10 Metern zwischen diversen Geräten unterschiedlicher Hersteller ermöglicht. Bei den Geräten handelt es sich um mobile Geräte (z.B: Notebook, PDA, Handy), stationäre Geräte (z.B: Desktop-PC) oder andere kleinere Geräte-Einheiten (z.B: Drucker, Scanner, Fotoapparat...), wobei all diese Geräte entweder einen Bluetooth-Modul integriert haben müssen oder um ein Bluetooth-Modul erweitert sein müssen.

Dieser Abschnitt wurde unter Einbeziehung folgender Literatur geschrieben:

- [Daguhn (2001)], [Köpf (2001)], [Krauß (2002)], [Computers (Januar 2001)], [Bornträger (2001)], [Nathan (2001)], [Autoren (2001a)], [Autoren (2001b)], [Lind u. a. (2001)], [Olsson u. a. (2001)], [Sörensen u. a. (2001)], [Bisdikian u. a. (2001)], [Astarabadi u. a. (2001)], [Hedlund u. a. (2001)], [Farnsworth u. a. (2001)] und [Melin u. a. (2001)]

Die Dokumente [Autoren (2001a)] und [Autoren (2001b)] sind die original Bluetooth-Spezifikationen in der Version 1.1, die von der SIG (Bluetooth Special Interest Group) im Februar 2001 veröffentlicht wurden.

2.5.1 Historie

Die Entwicklung der Bluetooth-Technologie wurde 1994 von der schwedischen Firma Ericsson eingeleitet. Die Firmen Nokia, Toshiba, IBM und Intel gründeten mit Ericsson im Jahr 1998 ein Konsortium für drahtlose Übertragung. Dieses Konsortium trägt den Namen SIG und steht für Bluetooth Special Interest Group. Der SIG schlossen sich bis zum April 2000 ungefähr 2000 Firmen an. Die SIG hat das Ziel, die Bluetooth-Technologie als weltweiten Standard zu etablieren. Im Juli 1999 entwickelte die SIG eine für die Öffentlichkeit zugängliche Bluetooth-Spezifikation 1.0[10], welche die Bluetooth-Technologie beschreibt. Im Februar 2001 folgte die Veröffentlichung der Bluetooth-Spezifikation in der Version 1.1.

Quelle: [Nathan (2001)]

2.5.2 Systemarchitektur der Bluetooth-Hardware

Das Bluetooth-System setzt sich aus einer Funkeinheit, dem Link Controller und dem Link Manager zusammen. Da das Bluetooth-System in vielen Kleinstgeräten untergebracht werden soll, wurde bei der Entwicklung dieses Systems Wert darauf gelegt, dass es von seinen Abmessungen her möglichst gering ausfällt und energiesparend arbeitet. Die drei Komponenten des Bluetooth-Systems sollen jetzt kurz erläutert werden:

- **Funkeinheit**
  Die Funkeinheit besteht aus einem Sende- und Empfangsmodul und weiterer analoger Radioelektronik. Der Bluetooth-Funk verwendet ein ISM-Band[11], das in einem Frequenzbereich von 2,402 bis 2,484 GHz arbeitet. Bei der Funkübertragung

[10] Die Spezifikation ist unter der Internetadresse www.bluetooth.com erhältlich.

[11] ISM steht für Industrial Scientific Medical. Für dieses ISM-Band sind weltweit keine Lizenzen erforderlich.

von Signalen bei Bluetooth-Geräten wird das *Fast Frequency Hopping* Verfahren eingesetzt. Die Brutto-Übertragungsrate bei Bluetooth-Geräten beträgt 1 Mbit pro Sekunde, von denen netto 721 kBit pro Sekunde für Daten- und Sprachkommunikation genutzt werden können. Die Funkreichweite ist auf maximal 10 Meter begrenzt, ist aber auf 100 Meter erweiterbar.

- **Link Controller**
  Der Link Controller wird oft auch als Baseband bezeichnet. Die wesentlichen Aufgaben des Link Controllers sind:
  - Kontrolle und Steuerung des Kommunikationsaufbaus; Unterstützung von ACL und SCO-Verbindungen
  - Verwaltung der physikalischen Funkverbindung, Anwendung des Hopping Algorithmus
  - Verwaltung der logischen Verbindungen
  - Fehlerbehandlung
  - Datensicherheit, Verschlüsselung, Identifikation
  - Sprach und- Audiokommunikation
  - Packen und Entpacken der Datenpakete
- **Link Manager**
  Der Link Manager ist eine Firmware[12], die sich auf dem Bluetooth-Modul in einem Flash-Speicher oder auf einem ROM-Speicher befindet. Der Link Manager läuft auf einem im Bluetooth-Modul befindlichen Prozessor ab. Der Link Manager sorgt für den Verbindungsaufbau, die Sicherheit[13] und die Kommunikation mit dem Endgerät. Dabei geschieht die Kommunikation zweier Bluetooth-Geräte über das LMP (Link Manager Protocol; siehe Abbildung 2.2). Der Link Manager bedient sich der Funktionen der darunterliegenden Schicht, dem Link Controller, um seine Dienste auszuführen.

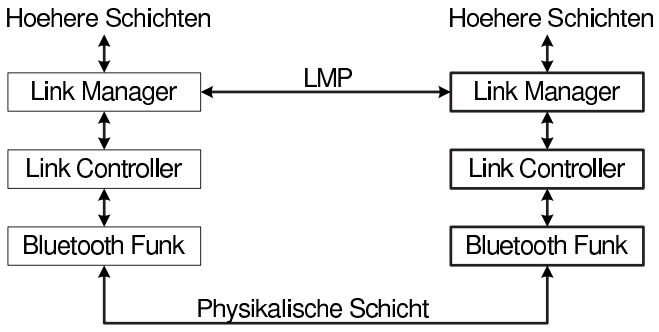

Abbildung 2.2: Die Kommunikation der Link Manager erfolgt über das LMP
Quelle: [Nathan (2001)], [Melin u. a. (2001)]

Quelle: [Nathan (2001)], [Computers (Januar 2001)], [Melin u. a. (2001)], [Daguhn (2001)]

[12] Unter Firmware wird eine Software verstanden, die in einem Rom-Modul eines Gerätes abgelegt ist und meist elementare Aufgaben ausführt.

[13] Sicherheit wird erreicht, indem Authentifizierungs- und Verschlüsselungsfunktionen angewendet werden. Mehr dazu auf Seite 21.

2.5.3 Verbindungsarten

In der Bluetooth-Spezifikation sind zwei Übertragungsarten definiert. Zu einem gibt es die asynchrone, verbindungslose (ACL;Asynchron Connectionless), zum anderen die synchrone, verbindungsorientierte (SCO;Synchronous Connection Orientied) Übertragung:

- Die asynchrone, verbindungslose Übertragung kann entweder über eine asymmetrische[14] oder symmetrische[15] paketvermittelte Punkt-zu-Mehrpunkt-Verbindung geschehen. ACL wird typischerweise nur für die Übertragung reiner Daten verwendet. Bei der Paketvermittlung werden in einem Pico-Netz die adressierten Datenpakete direkt an den Zielpunkt übertragen. Der Aufbau einer dauerhaften Verbindung findet hier nicht statt.

- Die synchrone, verbindungsorientierte Übertragung geschieht über eine symmetrische, leitungsvermittelte Punkt-zu-Punkt-Verbindung, die typischerweise für Sprache verwendet werden. Für die Übertragung von Sprache werden bis zu drei synchrone Kanäle benutzt. Jeder dieser Kanäle hat eine Übertragungsrate von 64 Kbps. Bei einer leitungsvermittelte Verbindung muss der Verbindungsaufbau explizit angefordert und aufgebaut werden. Die dauerhafte Verbindung bleibt auch dann erhalten, wenn keine Daten oder keine Sprache übertragen werden.

Quelle: [Nathan (2001)]

2.5.4 Adhoc-Netzwerke

Bluetooth-Geräte bilden Adhoc-Netzwerke. Adhoc-Netzwerke sind drahtlos, besitzen keine feste Infrastruktur und die Netzwerkbildung geschieht spontan. Die Topologie bei Adhoc-Netzwerken kann sich jederzeit verändern. Sobald sich ein Bluetooth-Gerät in Reichweite eines anderen Bluetooth-Gerätes aufhält, kann es zu dem anderen Gerät spontan eine Verbindung aufbauen. Es ist sogar möglich, dass einige Geräte als Router fungieren. Dadurch wird es für Geräte möglich, eine Nachricht an ein Zielgerät zu senden, obwohl das Zielgerät ausser Reichweite ist. Die Geräte, welche die Rolle des Routers übernehmen, leiten die Nachricht einfach weiter.

Quelle: [Nathan (2001)], [Bornträger (2001)]

[14] Asymmetrische Verbindung bedeutet, dass unterschiedliche Datenraten in der Sende- und Empfangsrichtung möglich sind.

[15] Symmetrische Verbindung bedeutet, das Sende- und Empfangsrichtung gleiche Datenraten haben.

2.5.5 Bluetooth Netztopologien

Bluetooth-Geräte können mit anderen Geräteeinheiten zu einem Pico-Netz verknüpft, mehrere Pico-Netze wiederum zu einem Scatternetz zusammengeschlossen werden. Folgende Abbildung veranschaulicht beide Netze:

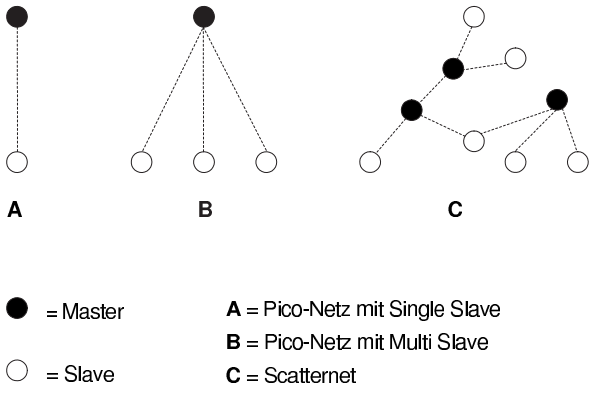

Abbildung 2.3: Mögliche Bluetooth-Netztopologien
Quelle: [Nathan (2001)], [Hedlund u. a. (2001)]

Ein Pico-Netz besteht aus genau einem Bluetooth-Gerät, das die Master-Rolle übernimmt, und aus bis zu 7 weiteren Bluetooth-Geräten, welche die Slave-Rolle übernehmen. Das Mastergerät gibt den Takt- und die Hop-Sequenz zur Synchronisation aller anderen Geräte im Pico-Netz an. Der Master leitet auch das Paging-Verfahren ein und stellt die Verbindung zu anderen Geräten her. Wie aus der obigen Abbildung erkennbar ist, sind normale Punkt-zu-Punkt-Verbindungen und Punkt-zu-Mehrpunkt-Verbindungen möglich.

Quelle: [Nathan (2001)], [Daguhn (2001)], [Köpf (2001)], [Computers (Januar 2001)], [Hedlund u. a. (2001)]

2.5.6 Fast Frequency Hopping Verfahren

Bei der Funkübertragung von Signalen bei Bluetooth-Geräten wird das *Fast Frequency Hopping* Verfahren eingesetzt (siehe folgende Abbildung 2.4):

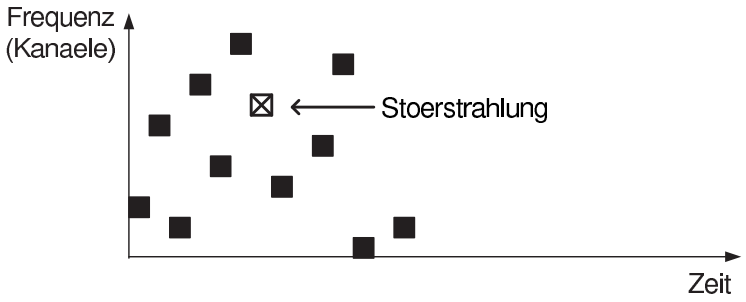

Abbildung 2.4: Das Fast Frequency Hopping Verfahren

Bei diesem Verfahren wird das Frequenzband in 79 Kanäle aufgeteilt. Bei der Übertragung von Funksignalen wird in der Sekunde 1600 mal der Übertragungskanal gewechselt (in Abbildung 2.4 dargestellt). Diese Übertragungsmethodik macht die Übertragung

abhörsicherer, da dieses Verfahren einen Algorithmus anwendet, der nur schwer mit technischen Aufwand zu dekodieren ist. Ausserdem wird mit dem Frequenzhopping eine hohe Unempfindlichkeit gegenüber Interferenzen mit externen Funksignalen erreicht.

Wird innerhalb einer Sekunde ein Kanal zu einem bestimmten Zeitpunkt bei der Übertragung eines Paketes durch Störstrahlungen blockiert, dann wird die Übertragung nur für $\frac{1}{1600}$ Sekunden unterbrochen. Damit werden immerhin noch 99,9% der Daten übertragen. Das nichtgesendete Paket wird dann zu einem späteren Zeitpunkt erneut gesendet werden.

Quelle: [Nathan (2001)], [Daguhn (2001)]

2.5.7 Sicherheit

Da Funksignale während der Übertragung leicht abgehört, manipuliert oder absichtlich gestört werden können, sind Sicherheitsmechanismen unabdingbar. Um die Daten- oder Sprachübertragung per Funk sicher zu machen, wurden in der Bluetooth-Spezifikation Sicherheitsmechanismen festgelegt, die für die Authentifikation und Verschlüsselung der zu übertragenden Funksignalen sorgen. Die Verschlüsselung der Daten macht die zu übertragenden Funksignale unkenntlich und die Authentifikation sorgt für den Beweis der Identität des Kommunikationspartners. Der Authentifizierungs-Prozess benutzt unter anderem die Bluetooth-Geräteadresse[16] des entfernten Gerätes. In der Bluetooth-Spezifikation sind die drei folgenden Sicherheitsmodi festgelegt worden:

- *Sicherheitsmodus 1*
  In diesem Modus werden keinerlei Sicherheitsfunktionen verwendet. Dieser Modus ist für die Übertragung von nicht-kritischen Daten oder Sprache vorgesehen.

- *Sicherheitsmodus 2*
  Dieser Modus bietet Sicherheit auf der Dienstebene an. Das bedeutet, dass die Sicherheitsmechanismen erst nach dem Verbindungsaufbau eingeleitet werden. Die Protokolle oder Anwendungen über der Verbindungsschicht sind selbst verantwortlich für die Verschlüsselung und die Authentifizierung. Beim Sicherheitsmodus 2 werden die Bluetooth-Geräte in drei Kategorien der Vertraulichkeit eingeteilt:

  1. In Vertrauenswürdige Geräte, bei denen das Pairing (siehe Seite 27) bereits geschehen ist.
  2. In Nicht-vertrauenswürdige Geräte, deren Geräteadressen zwar bekannt sind, bei denen aber noch kein Pairing durchgeführt worden ist.
  3. In Unbekannte Geräte.

  Ein vertrauenswürdiges Gerät hat uneingeschränkten Zugriff auf die Dienste, die wiederum in drei folgenden Kategorien eingeteilt sind:

  – Dienste, die eine Autorisation und Authentifizierung erfordern.
  – Dienste, die nur eine Authentifizierung erfordern.
  – Dienste, die weder Autorisation noch eine Authentifizierung erfordern.

[16] Die Bluetooth-Geräte werden durch eine eindeutige Geräteadresse, die der MAC-Adresse im Ethernet-Netzwerk ähnlich ist, identifiziert.

Die Autorisation legt fest, welche Dienste ein Kommunikationspartner in Anspruch nehmen darf. Für die Verwaltung und Durchsetzung dieser Regeln ist ein Sicherheitsmanager, der auf dem Bluetooth-Chip implementiert ist, verantwortlich.

- *Sicherheitsmodus 3*
  In diesem Modus wird die Sicherheit in der Verbindungsschicht geleistet. Die Sicherheitsmechanismen werden vor dem Verbindungsaufbau eingeleitet. Dieser Modus leistet die Authentifizierung und die Verschlüsselung auf der Verbindungsebene.

Quelle: [Nathan (2001)], [Krauß (2002)]

2.5.8 Der Bluetooth-Protokollstack

In verkabelten und drahtlosen Computernetzwerken kommunizieren alle Rechner über Netzwerkprotokolle. In einem Netzwerkprotokoll wird vereinbart, wie die Kommunikation zweier oder mehrerer Geräte untereinander im Netzwerk abläuft. Dabei werden Regeln für den Nachrichtenaustausch zwischen den beteiligten Parteien sowie das Format der Nachrichten festgelegt. Die Kommunikation findet dabei oft über mehrere Protokolle, einen sogenannten Protokollstack, statt. Weiterführende Informationen zu Protokollen findet man in der Literatur von [Tanenbaum (2000)].

Die Bluetooth-Spezifikation definiert für die Kommunikation der Geräte in einem PicoNetz Protokolle. Der Bluetooth-Protokollstack schaut folgendermassen aus:

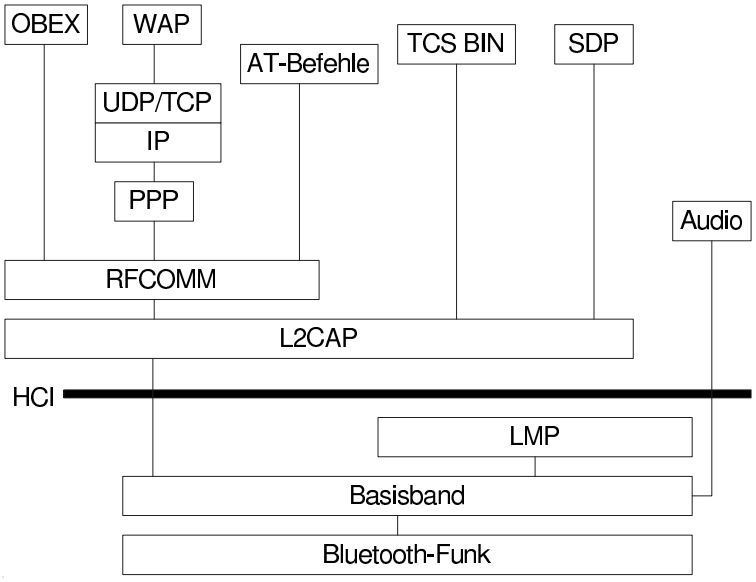

Abbildung 2.5: Der Bluetooth-Protokollstack
Quelle: [Nathan (2001)]

Anwendungen der Geräte müssen mit identischen Protokollstacks arbeiten, damit sie miteinander kommunizieren können. Dabei ist zu beachten, dass unterschiedliche Anwendungen je nach Bedarf auch unterschiedliche Protokollstacks verwenden können. In einem von einer Anwendung benutzten Protokollstack müssen nicht alle Protokolle vorkommen, sondern der Protokollstack kann individuell nach Bedarf für die Anwendung

zusammengestellt werden.

Der Blutooth-Protokollstack besteht aus Protokollen, die für Bluetooth spezifisch (z.B: L2CAP, LMP, usw.) sind und bereits existieren (z.B: TCP, IP). Dies hat für Anwendungen, die ein anderes physikalisches Übertragungsverfahren als Grundlage haben und ein allgemeines Protokoll, wie zum Beispiel TCP/IP, benutzen, einen wesentlichen Vorteil: Sie sind leichter auf Bluetooth-Geräte zu portieren, da die Bluetooth-Technologie das Protokoll TCP/IP unterstützt.

Die Protokolle Basisband, LMP und SDP bilden die Bluetooth-Kernprotokolle, die von den meisten Bluetooth-Geräten unterstützt werden müssen. Diese Protokolle liegen unter dem HCI (Host Controller Interface) aus Abbildung 2.5. HCI stellt eine Befehlschnittstelle zum Basisbandcontroller, zum Linkmanager und den Zugriff auf den Hardwarestatus und die Steuerregister dar. Im folgenden sollen die Protokolle aus Abbildung 2.5 kurz erläutert[17] werden:

- **Basisband, LMP (Link Manager Protocol) und Bluetooth-Funk**
  Die Aufgaben dieser Protokolle wurden bereits im Abschnitt 2.5.2 auf Seite 17 erläutert. Hier soll nochmal kurz angemerkt werden, dass das Basisband oft auch als Link Controller bezeichnet wird.

- **L2CAP (Logical Link Control and Adaption Protocol)**
  Dieses Protokoll hat folgende Aufgaben:

  – Segmentierung und Neuzusammensetzung von Paketen. Dabei können die gesendeten und empfangenden Pakete über L2CAP bis zu 64 KB lang sein. Für L2CAP sind nur ACL-Verbindungen, die mit Hilfe des LMP aufgebaut werden, definiert.

  – Unterstützung von Multiplexing[18] der höheren Ebene.

  – Überwachung der Verbindungsqualität zwischen den Teilnehmern

- **SDP (Service Discovery Protocol)**
  Die Aufgabe dieses Protokolls ist das Auffinden neuer Bluetooth-Geräte und der von ihnen angebotenen Dienste, die sich in Funkreichweite befinden. Unter Verwendung des SDP können die Geräteinformationen, Dienste und Leistungsmerkmale aller Dienste abgefragt werden.

- **RFCOMM (Radio Frequeny Communication)**
  RFCOMM ist ein Kabelersatzprotokoll, dessen wesentliche Aufgabe darin besteht, serielle Schnittstellen (RS-232) zu emulieren.

- **TCS BIN (Telephony Control Specification Binary)**
  Dieses bitorientiertes Telefonieprotokoll regelt die Rufsignalisierungen zum Aufbau von Sprache und Daten zwischen Bluetooth-Geräten (Besonders im Bereich der Mobiltelefonie und schnurlosen Headsets interessant).

[17] Weitere Erläuterungen zu den Protokollen sind auch im Abschnitt 2.5.9 auf Seite 25 zu finden.

[18] Unter Multiplexing versteht man die Aufteilung einer physischen Verbindung in mehrere logische Kanäle.

- **OBEX (Object Exchange Protocol)**
  Mit diesem Protokoll ist es möglich, unkompliziert Objekte zwischen Bluetooth-Geräten auszutauschen.

- **Audio**
  Mit Hilfe der Audioschicht werden Audiodaten über SCO-Verbindungen, die das Baseband zur Verfügung stellt, gesendet.

Die Protokolle *PPP* (Point-to-Point Protocol), *TCP* (Transmission Control Protocol), *UDP* (User Datagram Protocol), *IP* (Internet Protocol), *WAP* (Wireless Application Protocol) und *AT-Befehle* sind keine bluetooth-spezifischen Protokolle und sind im Rahmen für diese Diplomarbeit auch nicht vom grossen Interesse. Deshalb erlaube ich mir an dieser Stelle, für diese Protokolle auf die Literatur von [Tanenbaum (2000)] zur verweisen.

Quelle: [Nathan (2001)], [Autoren (2001a)], [Tanenbaum (2000)], [Daguhn (2001)], [Köpf (2001)]

2.5.9 Bluetooth Profile

Die Bluetooth SIG hat eine Reihe von Anwendungsmodellen definiert, die jeweils genau einem Anwendungsprofil zugeordnet sind. Die in der Abbildung 2.6 dargestellten Profile werden durch Protokolle und Eigenschaften definiert. Die Eigenschaften werden durch Nachrichten und Prozeduren der Protokolle implementiert. Dabei sind einige Eigenschaften entweder optional, verbindlich oder unverbindlich. Diese Profile gewährleisten die Interoperabilität zwischen Bluetooth-Geräte unterschiedlicher Hersteller. Beim Zustandekommen einer Verbindung zwischen zwei Bluetooth-Geräten werden erstmal die Profile der Geräte untereinander ausgetauscht, um herauszufinden, welche Dienste sie für den jeweiligen anderen Partner zur Verfügung stellen können.

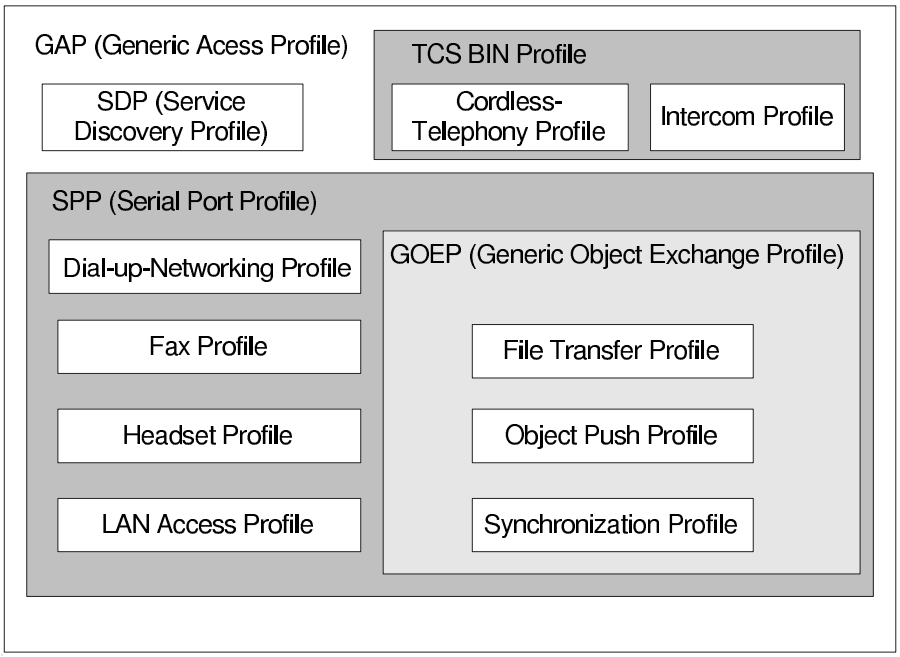

Abbildung 2.6: Die Bluetooth Profile
Quelle: [Nathan (2001)], [Autoren (2001b)]

Es gibt zwei Gruppen von Profilen:

- *Anwendungsprofile*
  Die Anwendungsprofile definieren die erforderlichen Protokolle und Eigenschaften, welche für die Implementierung der dazugehörigen Anwendungsmodelle nötig sind.

- *Allgemeine Profile*
  Die vier allgemeinen Profile sind GAP, SDAP, SPP und GOEP. Sie dienen als Grundlage für die Anwendungsprofile.

Im Folgenden werden die Profile im Hinblick für die Aufgabenstellung dieser Diplomarbeit je nach Bedarf mehr oder weniger ausführlich beschrieben:

GAP (Generic Access Profile)

Das *Generic Access Profile* aus Abbildung 2.5.9 beschreibt allgemeine Prozeduren zur Erkennung von Bluetooth-Geräten sowie Prozeduren für das Verbindungsmanagement. In diesem Profil kommen die Protokolle LC (Link Control) und LMP (Link Management Protocol) zum Einsatz. Ausserdem beschreibt GAP noch die Sicherheitsprozeduren und das Verhalten der Geräte im Standby- und Connecting-Status. Alle GAP-fähigen Bluetoothgeräte müssen, wenn sie auf der GAP-Ebene zusammenarbeiten möchten, die folgenden Parameter unterstützen:

- **Gerätename**
  Der Bluetooth-Gerätenamen kann bis zu 248 byte betragen.
- **PIN (Personal Identification Number)**
  Die PIN, auch Bluetooth-Sicherheitscode genannt, wird bei der erstmaligen Verbindung mit einem entfernten Gerät benötigt, um eine vertrauenswürdige Beziehung aufzubauen (siehe auch unter Pairing auf Seite 27).
- **Geräteklasse**
  Wenn ein Gerät ein entferntes Gerät erkennt, dann wird dieser Parameter *Geräteklasse* vom entfernten Gerät übertagen. Dieser Parameter gibt Auskunft über die Art bzw. Klasse des entfernten Gerätes und die von ihm unterstützten Dienste.
- **Erkennungsmodi**[19]
  Bluetooth-Geräte können sich in zwei folgende Modi befinden:
  - *discoverable mode*
    In diesem Modus sind sie für andere Bluetooth-Geräte sichtbar. Befindet sich ein Bluetoothgerät im Modus *discoverable mode*, dann werden sie in den *INQUIRY-scan*-Status versetzt. Geräte im *INQUIRY-scan*-Status suchen im Funkraum nach eingeleitenden Anfragen von anderen Geräten, um sie beantworten zu können.
  - *non-discoverable-mode*
    In diesem Modus sind sie für andere Bluetooth-Geräte nicht sichtbar. Befindet sich ein Bluetooth-Gerät im Modus *non-discoverable mode*, dann wird es niemals in den *INQUIRY-response*-Status versetzt. Geräte, die sich nicht im *INQUIRY-response*-Status befinden, beantworten keine Anfragen anderer Geräte.
- **Verbindungsmodi**
  Bluetooth-Geräte können sich entweder im *verbindungsfähigen (connectable mode)* oder sich im *nicht-verbindungsfähigen Modus (non-connectable mode)* befinden. Im verbindungsfähigen Modus werden die Geräte periodisch in den *Page-scan*-Status versetzt, im nicht-verbindungsfähigen Modus hingegen nicht. Die sich im

[19] Der *discoverable mode* wird normalerweise noch in zwei weitere Modi unterteilt (*general mode und limited mode*). In der Literatur von ([Nathan (2001)]; Seite206) erfährt man mehr über diese Modi.

Page-scan-Status befindlichen Geräte horchen periodisch auf Pagenachrichten[20], welche von entfernten Geräten gesendet worden sind. Eine Pagenachricht ist ein sogenannter *Device Access Code (DAC)*, der bis zu 72 Bit lang ist. Dieser *DAC* ist aus einem Teil der Bluetooth-Geräteadresse (128 Bit) eines der horchenden Geräte aus dem Funkbereich abgeleitet worden. Somit reagiert nur ein ausgewähltes Gerät auf den DAC. Nachdem es diesen DAC empfängt, beginnen beide Geräte sich auf eine Hopsequenz zu einigen und wechseln dann in den Verbindungsstatus über. Das Gerät, das die Page-Prozedur eingeleitet hat, übernimmt automatisch die Master-Rolle dieser Verbindung, das von ihm ausgewählte Gerät nimmt die Slave-Rolle an.

- **Paarungsmodi**
  Bluetooth-Geräte können sich entweder im *paarbaren Modus* (pairable mode) oder sich im *nicht-paarbaren Modus* (non-pairable mode) befinden. Im *paarbaren Modus* wird ein eingeleiteter Verbindungsaufbau (Pairing) von einem entfernten Gerät zugelassen, im *nicht-paarbaren Modus* nicht.
  Mit Pairing ist ein Initialisierungsverfahren gemeint, mit dem man eine vertrauenswürdige Beziehung zwischen zwei Geräten aufbauen kann. Die Initialisierungsphase wird immer bei der erstmaligen Kommunikation zweier Geräte eingeleitet. Dabei wird ein gemeinsamer Verbindungscode, der für die Authentifizierung eines entfernten Gerätes benötigt wird, erzeugt. Für zukünftige Authentifizierungen wird der Verbindungscode von beiden Geräten gespeichert. Bei der Erzeugung eines Verbindungscodes wird eine PIN (Personal Identifiaction Number) verwendet. Diese PIN, auch Bluetooth-Sicherheitscode genannt, wird bei der erstmaligen Verbindung zweier Geräte durch den Benutzer eingegeben.

- **Sicherheitsmodi**
  Dieser Parameter gibt an, im welchen Sicherheitsmodus sich das Bluetooth-Gerät befindet. Es gibt drei Sicherheitsmodi. Mehr zu den Sicherheitsmodi steht im Abschnitt 2.5.7 ab Seite 21.

Nachdem wir nun die Parameter behandelt haben, werden jetzt die Prozeduren, die im Leerlaufmodus[21] von allen GAP-fähigen Geräte eingeleitet werden können, erläutert:

- **Anfrage (inquiry)**
  Geräte, die eine Anfrage initiieren, werden mit der Bluetooth-Geräteadresse, der Taktfrequenz, der Geräteklasse und dem Page-scan-Modus allgemein erkennbarer Geräte versorgt. Das initiierende Gerät sendet dabei Anfragen mit einem sogenannten IAC[22] (Inquiry Accesss Code) aus. Andere Geräte, die sich in Funkreichweite des initiierenden Gerätes und sich im *discoverable mode* befinden, sind so eingestellt, dass sie nach Anfragen mit einem *IAC*-Code suchen.

- **Namenserkennung (name discovery)**
  Diese Prozedur beschafft sich den Namen anderer verbindungsfähiger Geräte.

[20] Ein Gerät, das Pagenachrichten an ein entferntes Gerät sendet, um eine Verbindung aufzubauen, wird auch als *Paging* bezeichnet.

[21] Geräte, die sich im Leerlaufmodus befinden, unterhalten keine eingerichteten Verbindungen zu anderen Geräten.

[22] Die Anfragen können mit drei unterschiedlichen Anfragecodes durchgeführt werden. Bei den Anfragecodes handelt es sich hierbei um den GIAC (General Inquiry Accesss Code), LIAC (Limited Inquiry Access Code) und den DIAC (Dedicated Inquiry Access Code). Mehr dazu findet man in der Literatur von [Nathan (2001)] und [Autoren (2001a)].

- **Geräteerkennung (device discovery)**
  Diese Prozedur führt eine Anfrage (inquiry) durch. Anschliessend wird bei einigen oder bei allen Geräten, welche auf die Anfrage reagieren, eine Namenserkennung durchgeführt.

- **Verbindungsaufbau (Bonding)**
  Beim Verbindungsaufbau geht es darum, eine vertrauenswürdige Beziehung zwischen zwei Geräten aufzubauen. Während der Bonding-Prozedur wird ein gemeinsamer Verbindungscode (bond) erzeugt und für zukünftige Authentifizierungen von beiden Geräten gespeichert (Pairing). Es gibt zwei Arten des Verbindungsaufbaus:

  – *Bestimmtes Bonding* erzeugt nur einen gemeinsamen Verbindungscode und wird auf beiden Seiten abgespeichert.

  – *Allgemeines Bonding* ist Teil der üblichen Prozeduren zur Einrichtung des Kanals und der Verbindung.

- **Einrichtungsprozeduren**
  Es gibt drei Prozeduren, um Verbindungen einzurichten:

  1. **Die Einrichtung einer physikalischen Verbindung (link establishment)**
     Die Einrichtung einer physikalischen Verbindung zwischen Blutooth-Geräten erfolgt über die Protokolle LMP (Link Manager Protocol) und LCP (Link Controller Protocol). Das *Paging*-Verfahren, eine LCP-Prozedur, ist wesentlich am physikalischen Verbindungsaufbau beteiligt. Das *Paging*-Verfahren wird auf Seite 27 erläutert. Ein komplettes Beispiel für die Einrichtung einer physikalischen Verbindung wird auf der Seite 85 angegeben.
  2. **Die Einrichtung einer logischen Verbindung (channel establishment)**
     Sobald eine physikalische Verbindung aufgebaut ist, können eine oder mehrere logische Verbindungen über das Protokoll *L2CAP* eingerichtet werden. Ein Beispiel hierfür findet man ebenfalls auf der Seite 85.
  3. **Die Einrichtung einer Verbindung auf Anwendungsebene (connection establishment)**
     Sobald eine logische Verbindung besteht, können auf diese ein oder mehrere Verbindungen auf der Anwendungsebene über die Prozedur *Establish DLC* des *RFCOMM*-Protokolls erstellt. werden. Der Aufbau einer Verbindung auf Anwendungsebene mit der Prozedur *Establish DLC* wird auf der Seite 30 beschrieben.

Quelle: [Lind u. a. (2001)], [Nathan (2001)]

SPP (Serial Port Profile)

Dieses Profil baut auf dem *Generic Accesss Profile* auf und dient, wie man aus der Abbildung 2.6 auf Seite 25 erkennt, als Grundlage für weitere Profile. Dieses Profil emuliert über das RFCOMM-Protokoll eine serielle Kabelverbindung, damit Anwendungen, die eigentlich eine Kabelverbindung benötigen, auch über Bluetooth-Funk kommunizieren können. Genauer gesagt, wird hier die serielle Schnittstelle (RS-232) emuliert. Das *RFCOMM*-Protokoll unterstützt bis zu 60 simultan eröffnete serielle Verbindungen (Ports), jedoch ist die gleichzeitig genutzte Anzahl von seriellen Verbindungen auf Bluetooth-Geräten implementationsabhängig. Der Protokollstack für das *Serial Port Profile* ist in der folgenden Abbildung dargestellt:

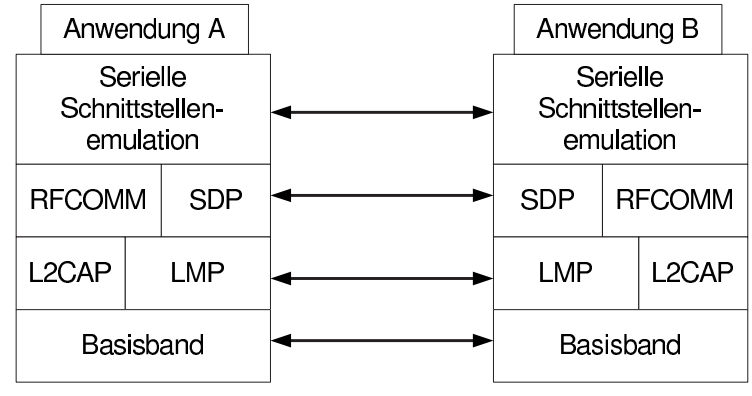

Abbildung 2.7: Der von SPP verwendete Protokollstack
Quelle: [Nathan (2001)], [Sörensen u. a. (2001)]

In der Bluetooth-Spezifikation werden die erforderlichen Prozeduren, welche für die Erstellung einer virtuellen Kabelverbindung zwischen zwei Bluetooth-Geräten verantwortlich sind, beschrieben. Bei den Prozeduren handelt es sich unter anderen, um die beiden folgenden Prozeduren:

- *Establish link and set up virtual serial connection*
  Diese Prozedur wird vom lokalen Gerät (Initiator) eingeleitet und beschreibt, welche Schritte erforderlich sind, um eine emulierte serielle Verbindung mit einem entfernten Gerät einzurichten.

  1. Mit Hilfe von *SDP*-Transaktionen[23] muss die *RFCOMM-Server-Kanalnummer* der Anwendung des entfernten Gerätes ermittelt werden. Auf der Seite 87 wird ein Beispiel angegeben, wie diese *RFCOMM-Server-Kanalnummer* ermittelt wird.
  2. Optional kann die Authentifikation des entfernten Gerätes und die Aktivierung der Verschlüsselung gefordert werden.
  3. Es muss eine *L2CAP*-Verbindung angefordert werden.
  4. Die Einleitung einer *RFCOMM*-Sitzung muss auf den eingerichtenen *L2CAP*-Kanal erfolgen. Dies geschieht mit der Prozedur *Initialize RFCOMM session* des *RFCOMM*-Protokolls.

[23] SDP steht für das Service Discovery Protocol. Für weitere Informationen hierzu wird auf die Seite 33 verwiesen.

5. Es muss eine neuen Datenverbindung mit Hilfe der obengenannten Server-Kanalnummer über die *RFCOMM*-Sitzung eingerichtet werden. Dabei erfolgt die Einrichtung der Datenverbindung über die Prozedur *Establish DLC (Data Link Connnection)* des *RFCOMM*-Protokolls.

- *Accept link and establish virtual serial connection*
  Diese Prozedur wird vom entfernten Gerät (Acceptor) ausgeführt und beschreibt, an welchen erforderlichen Schritte sich das entfernte Gerät beteiligen muss.

  1. Falls erforderlich, muss das entfernte Gerät die angeforderte Authentifizierung durchführen und die Verschlüsselung aktivieren.
  2. Das entfernte Gerät muss den *L2CAP*-Verbindungswunsch stattgeben.
  3. Das entfernte Gerät muss die angeforderte *RFCOMM*-Sitzung zulassen.
  4. Das entfernte Gerät muss die angeforderte Datenverbindung über die *RFCOMM*-Sitzung zulassen.

Quelle: [Sörensen u. a. (2001)], [Nathan (2001)]

GOEP (Generic Object Exchange Profile)

Dieses Profil ermöglicht den Austausch von Datenobjekte zwischen Bluetooth-Geräten. GOEP stellt drei Eigenschaften zur Verfügung:

- *Establishing an Object Exchange session*
  Diese Eigenschaft wird benutzt, um eine *OBEX*-Sitzung zwischen dem Client und dem Server zu eröffnen. Sobald eine Sitzung eröffnet worden ist, können Objekte zwischen Client und dem Server mit den Eigenschaften *Push* und *Pull* transferiert werden. Voraussetzung für die Eröffnung einer Sitzung ist das Bestehen einer bereits vorhandenen *RFCOMM* Verbindung.

- *Push*
  Die *Push*-Eigenschaft ermöglicht die Übertragung von Datenobjekten vom Client zum Server.

- *Pull*
  Die *Pull*-Eigenschaft ermöglicht die Übertragung von Datenobjekten vom Server zum Client.

Dieses Profil dient als Grundlage für das *File Transfer Profile*, *Object Push Profile* und *Synchronization Profile*. Der Protokollstack, den *GOEP* verwendet, sieht folgendermassen aus:

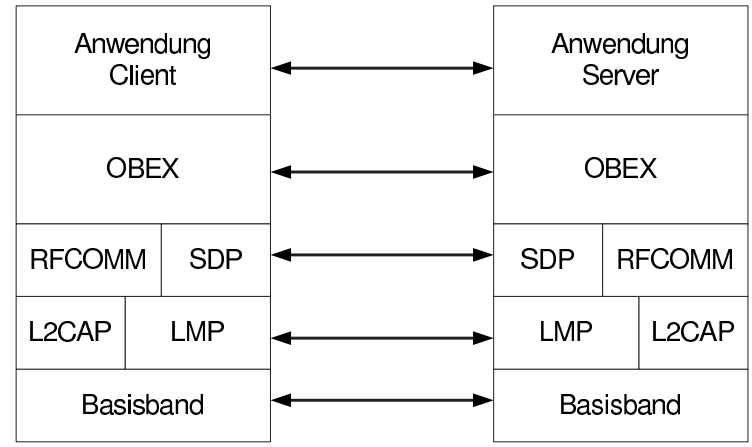

Abbildung 2.8: Der von GOEP verwendete Protokollstack
Quelle: [Nathan (2001)], [Olsson u. a. (2001)]

Die eben genannten Eigenschaften werden mit den Operationen des Protokolls *OBEX* implementiert. Es handelt sich hierbei um folgende *OBEX*-Operationen:

- *Connect*
  Mit der Operation *Connect* kann man eine Session zwischen dem Client und dem Server eröffnen. Mit dieser Operation wird die Eigenschaft *Establishing an Object Exchange session* implementiert.

- *Disconnect*
  Mit dieser Operation wird eine Session zwischen dem Client und dem Server beendet.

- *Put*
  Mit dieser Operation wird ein Datenobjekt vom Client zum Server transferiert und implementiert die oben genannte Eigenschaft *Push*. Die Übertragung eines Datenobjektes kann über mehrere *OBEX*-Paketen geschehen (gilt ebenfalls für die nachfolgende *Get*-Operation).

- *Get*
  Mit dieser Operation wird ein Datenobjekt vom Server zum Client transferiert und implementiert die oben genannte Eigenschaft *Pull*.

- *Abort*
  Mit dieser Operation kann der Client einen eingeleiteten Objektransfer, der sich aus der Übertragung mehrerer Pakete zusammensetzt, vorzeitig abbrechen. Die gesendete Nachricht *Abort* an den Server kann den Grund für den Abbruch der Übertragung enthalten.

- *SetPath*
  Mit der Operation *SetPath* kann die Clientseite auf der Serverseite ein aktuelles Verzeichnis einstellen. In dieses Verzeichnis lassen sich die Objekte mittels der Operation *Put* vom Client zum Server und mittels der Operation *Get* vom Server zum Client übertragen. Darüberhinaus ermöglicht die Operation *SetPath* das Anlegen neuer Ordner, das Löschen von Objekten (Dateien, Ordner) und die Navigation in den Ordnern. Die Navigation in den freigegebenen Odnern, erfolgt dabei schrittweise. Auch ermöglicht *SetPath*-Operation das Setzen des Rootverzeichnisses des freigegebenen Ordners in einem Schritt.

Quelle: [Olsson u. a. (2001)], [Nathan (2001)]

SDAP (Service Discovery Application Profile)

SDAP ist ein Anwendungsprofil. Es definiert Protokolle und Eigenschaften, mit dessen Hilfe sich die registrierten Dienste auf entfernten Geräten auskundschaften lassen. Dabei sind lediglich nur die Dienstinformationen eines entfernten Gerätes abrufbar, die Nutzung der Dienste ist allerdings nicht über SDAP möglich. Das SDAP verwendet in seinem Protokollstack (siehe Abbildung 2.9 auf Seite 33) unter anderem das SDP (Service Discovery Protocol), das für die Erkennung von Diensten auf entfernten Geräten sorgt. Das SDP (Service Discovery Protocol) bietet die Möglichkeit, Anfragen zu stellen, die nach ganz bestimmten verfügbaren Diensten suchen. Bei diesen Dienstanfragen kann man explizit angeben, welchem Typ und welchen Eigenschaften der Dienst entsprechen muss.
Zum anderen hat man die Möglichkeit, Anfragen zu stellen, die allgemein verfügbare Dienste suchen. Diese Dienstanfragen sind natürlich erst nach dem Verbindungsaufbau zum entfernten Gerät durchführbar.

Die Informationen über die Dienste sind in Form von Dienstdatensätzen in der Dienstedatenbank aus Abbildung 2.9 abgelegt. Dies ist die einzige Information, welche die Bluetoothspezifikation über diese Dienstedatenbank verliert. Eine Zugriffsschnittstelle zur Datenbank wird in der Bluetoothspezifikation nicht beschrieben. Nur auf Umweg über die SDP-Transaktionen (siehe Seite 35) wird der lesende Zugriff auf die Dienstdatensätze der Datenbank ermöglicht.

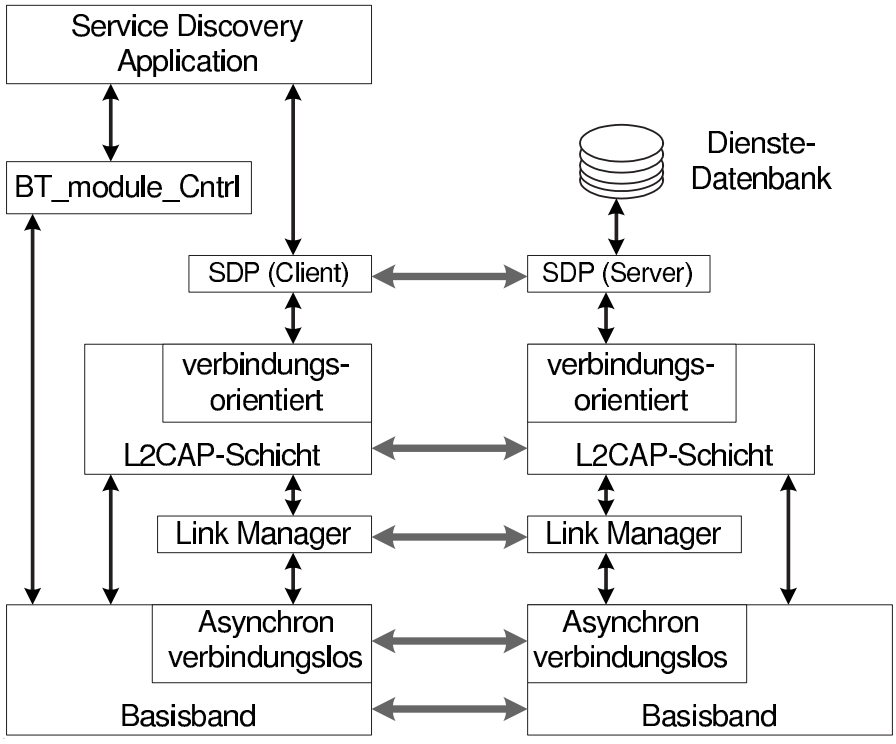

Abbildung 2.9: Der Protokollstack für SDP (Service Discovery Profile)
Quelle: [Nathan (2001)], [Bisdikian u. a. (2001)]

Das SDAP definiert die zwei folgenden Rollen, welche die Bluetooth-Geräte einnehmen können:

- *Lokales Gerät*
  Ein lokales Gerät leitet die Prozedur für die Diensterkennung mit einer Anfrage (Request) ein und muss mindestens den Client-Teil der SDP-Architektur (Service Discovery Profile) (siehe Abbildung 2.9) enthalten. Das lokale Gerät enthält die Applikation für die Diensterkennung (Service Discovery Application).

- *Entferntes Gerät*
  Ein entferntes Gerät ist irgendein Gerät, das dadurch am Prozess der Diensterkennung teilnimmt, indem es eine Dienstanfrage des lokalen Gerätes mit einer Antwortnachricht (Response) beantwortet. Es muss mindestens den Server-Teil der SDP-Architektur (Service Discovery Profile) enthalten. Der Server-Teil der SDP-Architektur beinhaltet eine Datenbank mit Dienstdatensätzen. Stellt ein lokales Gerät eine Dienstanfrage, greift der Server-Teil der SDP-Architektur, also das entfernte Gerät, auf diese Datenbank zu, um eine Antwort zu erzeugen.

Dabei ist es möglich, dass ein Bluetooth-Gerät entweder beide der eben genannten Rollen gleichzeitig oder nur eine der beiden einnimmt.

Die Informationen eines Dienstes werden durch einen Datensatz in der Datenbank repräsentiert. Ein Dienstdatensatz[24] besteht aus einer Liste von Dienstattributen. Jedes Dienstattribut besteht aus einer 16-bit AttributID[25] und einem Wert. Die AttributID gibt an, um welchen Typ Attribut es sich handelt. Neben den einfachen Datentypen können die Werte auch komplexe Datenstrukturen beinhalten. Universelle Attribute sind solche Attribute, welche für die Beschreibung aller Datensätze eingesetzt werden können. Die zwei universellen Attribute *Service RecordHandle* und *ServiceClassIDList* sind jedoch verbindlich, die restlichen universellen Attribute hingegen optional. Im Folgenden werden einige universelle Attibute vorgestellt:

- *ServiceRecordHandle* $\Longrightarrow$ AttributID: 0x0000
  Ein *ServiceRecordHandle* ist ein 32-bit Wert, der jeden Dienstdatensatz aus der Dienstedatenbank eindeutig identifiziert.

- *ServiceClassIDList* $\Longrightarrow$ AttributID:0x0001
  Dieses Attribut besteht aus einer Liste von UUID's[26] und muss mindestens einen Eintrag enthalten, wobei jede UUID eine Dienstklasse angibt. Die Liste gibt also an, welchen Dienstklassen der Dienst angehört. Dabei werden die Dienstklassen oder UUID's in der Liste nach dem fallenden Grad ihrer Spezifikation aufgelistet. Eine Dienstklasse wird durch Attribute beschrieben, die alle Instanzen dieser Dienstklasse gemeinsam haben. Die Dienstklassen sind in weitere Unterklassen unterteilbar. Eine Unterklasse beinhaltet alle Attribute ihrer Oberklasse und hat mindestens noch ein zusätzliches Attribut.

[24] Ein Beispiel für Einträge von Dienstdatensätze in die Dienstedatenbank wird für das *File Transfer Profile* auf Seite 41 gegeben.

[25] Die zugewiesenen AttributeID's kann man sich im Internet unter [btassnum (2002)] anschauen.

[26] Eine UUID (Universally Unique Identifier) identifiziert eindeutig eine Dienstklasse (ClassID) oder einen Dienst. Zugewiesene UUID's kann man sich im Internet unter [btassnum (2002)] anschauen. Die UUID ist ein 128-bit Wert, der auch als 16 oder 32-bit dargestellt werden kann.

Beispiel

Ein Druckdienst, der das *postscript*-Format unterstützt und in Farbe drucken kann, könnte durch die in dieser Reihenfolge angegebenen Dienstklasseneinträge in der Dienstklassenliste beschrieben werden: ([Autoren (2001a)];Seite344):

- ColorPostScriptPrinterServiceClassID
- PostScriptPrinterServiceClassID
- PrinterServiceClassID

- *ProtocolDescriptorList* $\Longrightarrow$ AttributID: 0x0004
  Dieses Attribut besteht aus einer Liste, die sich aus ein oder mehreren Protokollstacks zusammensetzt. In der Liste werden nur die Protokollstacks angegeben, die für die Nutzung eines Dienstes notwendig sind. Ein einzelner Protokollstack besteht aus einer Liste von Protokollen. Dabei werden zuerst die niederen Protokolle und dann die höheren Protokolle aufgelistet. Ein Protokoll wird durch einen *ProtocolDescriptor* repräsentiert. Dieser *ProtocolDescriptor* besteht aus einer Liste. Diese Liste beginnt mit einer Protokoll-UUID und es können optional noch weitere protokollspezifische Parameter folgen.

- *ServiceName* $\Longrightarrow$ AttributID Offset: 0x0000
  Dieser Attribut repräsentiert den Namen des Dienstes. Diese AttributID wird mit Hilfe eines Offsets berechnet. Für Details hierzu wird auf die Bluetooth-Spezifikation verwiesen [Autoren (2001a)].

- *BluetoothProfileDescriptorList* $\Longrightarrow$ AttributID: 0x0009
  Dieses Attribut besteht aus einer Liste von Profildeskriptoren (*ProfileDescriptor*). Ein *ProfileDescriptor* beinhaltet Informationen über ein Profil und besteht aus einer Liste mit zwei Elementen. Das erste Element besteht aus der Profil-UUID und das zweite Element gibt eine Profilversionsnummer an.

Eine SDP-Transaktion besteht aus einer *Request*- und eine *Response*-Nachricht. Eine Nachricht, die von einer Protokollinstanz zur anderen gesendet wird, bezeichnet man auch als PDU[27]. Im Bezug auf die Diensterkennung gibt es verschiedene PDU-Typen[28], die nachfolgend beschrieben sind:

- *SDP_ErrorResponse*
  Diese PDU wird gesendet, wenn Fehler in einer SDP-Transaktion auftreten. Ein Fehlercode wird mitgeliefert, der den Fehler beschreibt.

- *SDP_ServiceSearchRequest*
  Diese PDU stellt eine Dienstanfrage dar, mit der ein lokales Gerät die Dienste eines entfernten Gerätes auskundschaften kann. Unter anderem beinhaltet diese PDU

[27] Die formale Spezifikation von PDU's (Protocol Data Unit) findet man in der Bluetoothspezifikation [Autoren (2001a)].

[28] Ein ausführliches Beispiel über SDP-Transaktionen findet man in der Bluetoothspezifikation [Autoren (2001a)] im Anhang *Appendix A - Background Information* (Seite 378 bis 392).

einen Parameter *ServiceSearchPattern*, der aus einer Liste von UUID's besteht. Eine UUID repräsentiert einen Dienst oder eine Dienstklasse (Mehr zu UUID siehe Seite 34).

- *SDP_ServiceSearchResponse*
  Diese PDU stellt eine Dienstantwort dar, die das entfernte Gerät nach dem Erhalt einer Dienstanfrage an das lokale Gerät senden kann. Unter anderem beinhaltet diese PDU einen Parameter *ServiceRecordHandleList*, der aus einer Liste von *Handle's* besteht. Ein *Handle* (siehe Seite 34) eines Dienstdatensatzes wird nur in die Antwortliste aufgenommen, wenn alle UUID's aus dem *ServiceSearchPattern* in den Attributwerten des Datensatzes vorkommen. Ein weiterer Parameter *CurrentServiceRecordCount* gibt die Anzahl der *Handle's* in der Antwortliste an.

- *SDP_ServiceAttributeRequest*
  Mit dieser PDU kann ein lokales Gerät die Attributwerte eines bestimmten Dienstdatensatzes eines entfernten Gerätes anfordern. Unter anderem enthält diese PDU die Parameter *ServiceRecordHandle* und *AttributeIDList*. Der Parameter *ServiceRecordHandle* identifiziert den Datensatz, der Parameter *AttributeIDList* beinhaltet eine Liste, die sich aus einzelnen AttributID's oder
  AttributID-Bereichsangaben zusammensetzt.

- *SDP_ServiceAttributeResponse*
  Mit dieser PDU antwortet ein entferntes Gerät auf
  eine Dienstattributanfrage (*SDP_ServiceAttributeRequest*) eines lokalen Gerätes. Unter anderem enthält diese PDU einen Parameter *AttributeList*, welcher eine Liste aus Attributpaaren darstellt. Die Attributpaare setzen sich aus einer AttributID und einem Attributwert zusammen.

- *SDP_ServiceSearchAttributeRequest*
  Mit dieser PDU kann ein loklales Geräte eine Dienstattributanfrage in einem Schritt durchführen. Diese PDU stellt also eine Kombination der beiden PDU-Typen *SDP_ServiceSearchRequest* und *SDP_ServiceAttributeRequest* dar.

- *SDP_ServiceSearchAttributeResponse*
  Mit dieser PDU kann ein entferntes Gerät eine Dienstattributanfrage eines lokalen Gerätes mit einer einzigen PDU beantworten. Diese PDU stellt also eine Kombination der beiden PDU-Typen *SDP_ServiceSearchResponse* und
  SDP_ServiceAttributeResponse dar.

Über die Komponente *BT_ module_ Cntrl*[29] kann die Applikation für die Diensterkennung (Service Discovery Application) das Bluetooth Modul anweisen, zu welchem Zeitpunkt es die Suchoperation einleitet. Um eine Anfrage durchführen zu können, müssen die entfernten Geräte sich im Modus *discoverable mode* und sich im Modus *connectable mode* befinden. Das lokale Gerät und das entfernte Geräte müssen eine Verbindung zueinander aufgebaut haben. Bei der Ausführung einer Anfrage können die Prozesse für die Authentifizierung und Verschlüsselung optional eingesetzt werden und das lokale Gerät kann entweder die Master- oder Slaverolle einnehmen.

[29] Die Komponente *BT_ module_ Cntrl* könnte ein Teil einer Bluetooth-Stack Implementation sein, auf den die Applikation für die Diensterkennung aufsetzt. Oder es könnte auch eine low-level Implementation sein, die direkt als Funktion in der Applikation für die Diensterkennung implementiert ist. Da nirgendwo festgelegt worden ist, wie die Implementation der *BT_ module_ Cntrl*-Komponente auszuschauen hat, wird sie als logische Komponente separiert in der Abbildung 2.9 aufgeführt.

Die Applikation für die Diensterkennung aus Abbildung 2.9 sollte mindestens mit den folgenden abstrakten Dienstprimitiven[30] ausgestattet sein:

- *serviceBrowse(List[RemDev], List[RemDevRelation], List[browseGroup], getRemDevName, stopRule)*
  Diese Dienstprimitive führt eine Dienstsuche auf den in der Liste *List[RemDev]* angegebenen entfernten Geräten durch. Dabei beschränkt sich die Dienstsuche nur auf solche Dienste, die zu der in der Liste genannten Dienstgruppen *List[browseGroup]* [31] gehören. Die Dienstsuche kann noch zusätzlich weiter durch die Liste *List[RemDevRerlation]* eingeschränkt werden. Diese Liste enthält Parameter, welche die Beziehung zwischen den Geräten beschreiben. Zum Beispiel kann man die Dienstanfrage auf bereits verbundene und vertrauenswürdige Geräte beschränken. Ist der Parameter *getRemDev* auf den Wert *yes* gesetzt, dann werden die Geräte, die dieser Dienstanfrage entsprechen, zurückgegeben. Die Dienstsuche hält solange an bis die Abbruchbedingung *stopRule* erfüllt worden ist.

- *serviceSearch(List[RemDev], List[RemDevRelation], List[searchPattern, attributeList], getRemDevName, stopRule)*
  Diese Dienstprimitive führt eine Dienstsuche auf den in der Liste *List[RemDev]* angegebenen entfernten Geräten durch. Dabei beschränkt sich die Dienstsuche auf solche Dienste, die mit den Diensten aus der Liste *List[searchPattern, attributeList]* übereinstimmen. Jeder einzelne Dienst in der Liste wird durch die UUID's im *SearchPattern* und den Attributen in der Attributliste *attributeList* charakterisiert. Die Dienstsuche kann zusätzlich durch die Liste *List[RemDevRerlation]* eingeschränkt werden. Diese Liste enthält Parameter, welche die Beziehung zwischen den Geräten beschreiben. Zum Beispiel kann man die Dienstanfrage auf bereits verbundene und vertrauenswürdige Geräte beschränken. Ist der Parameter *getRemDev* auf den Wert *yes* gesetzt, dann werden die Geräte, die dieser Dienstanfrage entsprechen, zurückgegeben. Die Dienstsuche hält solange an, bis die Abbruchbedingung erfüllt worden ist.

- *enumerateRemDev(List[classOFDevice], stopRule)*
  Die Dienstprimitive *enumerateRemDev(Parameterliste)* führt mit Unterstützung der Komponente *BT _module _Cntr* aus Abbildung 2.9 einen Suchprozess nach neuen Geräten in Funkreichweite durch und liefert als Ergebnis eine Liste der in Funkreichweite befindlichen Geräte. Das Ergebnis dieser Dienstprimitive kann von anderen Dienstprimitiven (z.B: serviceSearch(...)) verwertet werden. Mit dem Parameter *classOfDevice* dieser Dienstprimitive kann man nach Geräten ganz bestimmter Klassen suchen. Diese Dienstprimitive arbeitet solange bis die Abbruchbedingung (Parameter *stopRule*) erfüllt worden ist.

- *terminatePrimitive (primitiveHandle, returnResults)*
  Diese Dienstprimitive beendet eine Dienstprimitive und liefert ein Teilergebnis. Dabei enthält der Parameter *primitiveHandle* die ID der aufgerufenen Dienstprimitve, der Parameter *returnResults* das Teilergebnis.

[30] Abstrakte Dienstprimitive sind abstrakte Operationen, die in ihrer Syntax und Semantik nicht exakt definiert und plattformabhänig (Hardware, Betriebssystem etc.) sind.

[31] Die Dienste können mit Hilfe der Attributen *BrowseGroupList* und *GroupId* und mit Hilfe der Dienstklassen *BrowseGroupDescriptorServiceClassID* und *PublicBrowseGroup* in Gruppen und Untergruppen hierarchisch eingeteilt werden. Für ausführliche Informationen wird auf die Bluetoothspezifikation [Autoren (2001a)] verwiesen.

Quelle: [Bisdikian u. a. (2001)], [Nathan (2001)], [Farnsworth u. a. (2001)]

FTP (File Transfer Profile)

Das *File Transfer Profile* ist ein Anwendungsprofil und ermöglicht die bidirektionale Übertragung, die Manipulation und die Navigation von Dateiobjekten zwischen Bluetooth-Geräten, die *FTP* unterstützen. Bei den Dateiobjekten kann es sich um einzelne Dateien oder um Ordner mit kompletten Inhalt handeln. Wie aus Abbildung 2.6 auf Seite 25 erkennbar ist, baut *FTP* auf das *GOEP* auf. Der Protokollstack für *FTP* schaut folgendermassen aus:

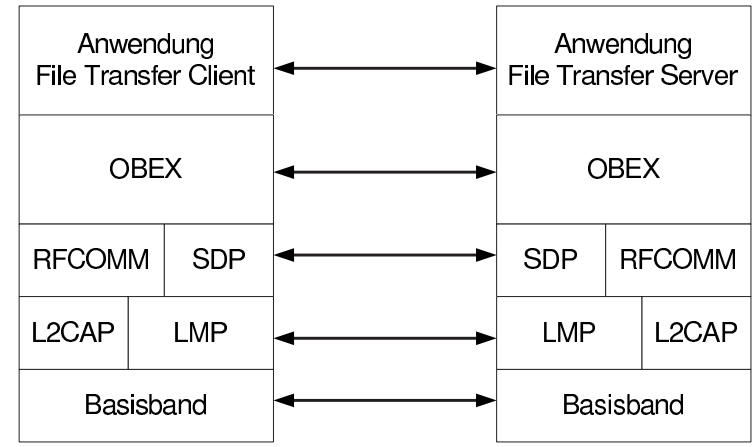

Abbildung 2.10: Der von FTP verwendete Protokollstack
Quelle: [Nathan (2001)], [Astarabadi u. a. (2001)]

Folgende Funktionen wären für die Anwendung *File Transfer Client* aus Abbildung 2.10 vorstellbar:

- **Select Server**
  Mit dieser Funktion kann ein Client einen Server aus einer Liste von entfernten Servern, die das *File Transfer Profile* unterstützen, auswählen. Mit dem ausgewählten Server wird eine Verbindung erstellt und anschliessend können FTP-Operationen ausgeführt werden. Der Benutzer operiert dabei auf ein vom FTP-Server freigegebenes Basisverzeichnis. Hierbei ist zu beachten, dass FTP-Operationen nur dann auf einem Server ausgeführt werden können, wenn der Server sich im *File Transfer Mode* und im *Discoverable Mode* befindet.

- **Navigate Folders**
  Mit dieser Funktion kann man durch Ordnerstrukturen und Dateien eines entfernten Gerätes navigieren.

- **Pull Object**
  Mit dieser Funktion kann eine Datei oder ein Ordner mit dem kompletten Inhalt vom Server zum Client transferiert werden.

- **Push Object**
  Mit dieser Funktion kann eine Datei oder ein Ordner mit dem kompletten Inhalt vom Client zum Server transferiert werden.

- **Delete**
  Mit dieser Funktion können Ordner mit dem kompletten Inhalt oder einzelne Dateien auf dem Server gelöscht werden.

- **Create Folder**
  Mit dieser Funktion können auf dem Server neue Ordner angelegt werden.

Die *Server File Transfer Anwendung* aus Abbildung 2.10 bearbeitet die Clientanfragen und trägt die Dienste für FTP in der Diensterkennungsdatenbank (SDDB, Service Discovery Database) ein.

Das *File Transfer Profile* wird durch die drei Eigenschaften *Folder Browsing*, *Object Transfer* und *Object Manipulation* beschrieben:

- *Folder Browsing*
  Die Funktion *Navigate Folders* wird der Eigenschaft *Folder Browsing* zugeordnet. Die Eigenschaft *Folder Browsing* ist für die Client- und Serverseite verbindlich vorgeschrieben. Diese Eigenschaft ermöglicht die Navigation in einem freigegebenen Ordnerverzeichnis auf einem entfernten Server. Dabei können für verschiedene Benutzer zur Navigation unterschiedliche Publikordner vom Server zur Navigation freigegeben werden.

- *Object Transfer*
  Die Funktionen *Pull Object* und *Push Object* werden der Eigenschaft *Object Transfer* zugeordnet. Diese Eigenschaft wird in den zwei Eigenschaften *File Transfer* und *Folder Transfer* unterteilt:

  – Die Eigenschaft *File Transfer* ist für die Client- und Serverseite verbindlich vorgeschrieben und ermöglicht die bidirektionale Übertragung von Dateiobjekten zwischen dem Server und dem Client.

  – Die Eigenschaft *Folder Transfer* ist für die Client- und Serverseite optional. Unterstützt der Server diese Eigenschaft nicht und ein Client will aber diese Eigenschaft nutzen, dann muss der Server in der Lage sein, mit einem entsprechenden Fehlercode zu antworten. Diese Eigenschaft ermöglicht die bidirektionale Übertragung von Ordnern zwischen dem Server und dem Client. Dabei kann bei der Übertragung von Ordnern der komplette Inhalt der Ordner, sprich alle Dateiobjekte und der komplette Inhalt aller Unterordner, transferiert werden.

- *Object Manipulation*
  Die Funktionen *Delete* und *Create Folder* sind der Eigenschaft *Object Manipulation* zuzuordnen. Diese Eigenschaft ist ebenfalls für die Client- und Serverseite optional, wobei auch hier ein Server, der diese Eigenschaft nicht unterstützt, mit einem entsprechenden Fehlercode dem Client antworten muss, falls dieser Client diese Eigenschaft nutzen möchte. Diese Eigenschaft ermöglicht zu einem die Erzeugung neuer Dateien- und Ordnerobjekte auf entfernten Servergeräten und zum anderen die Löschung von Dateien- und Ordnerobjekten auf entfernten Servergeräten.

Diese Eigenschaften werden mit den *OBEX*-Operationen *Connect*, *Disconnect*, *Put*, *Get*, *Abort* und *SetPath* implementiert. Diese Operationen wurden bereits im Abschnitt 2.5.9 auf Seite 31 vorgestellt.

In der Abbildung 2.11 sind die erforderlichen Einträge der Dienstdatensätze in die Dienstedatenbank für das *File Transfer Profile* dargestellt.

Item	Definition	Type/ Size:	Value	AttrID	Status	Default Value
Service Class ID List				@	V	
Service Class #0		UUID	OBEX-File Transfer		V	
Protocol Descriptor List				@	V	
Protocol ID #0		UUID	L2CAP		V	
Protocol ID #1		UUID	RFCOMM		V	
Param #0	CHANNEL	Uint8	Varies		V	
Protocol ID #2		UUID	OBEX		V	
Service Name	Displayable Text Name	String	Varies	@	O	"OBEX File Transfer"
Bluetooth Profile Descriptor List				@	O	
Profile ID #0	Supported Profile	UUID	OBEX-File Transfer			OBEX File Transfer
Param #0	Profile Version	Uint16	0x100			0x100

Abbildung 2.11: Einträge in die Dienstedatenbank für das FTP
V = verbindlich O = Optional @ = [btassnum (2002)]
Quelle: [Astarabadi u. a. (2001)]

Quelle: [Nathan (2001)], [Astarabadi u. a. (2001)]

2.6 JaxB

JaxB (*Java*Architecture for XML Binding) ist ein rein javabasiertes System von Sun und bietet eine komfortable Möglichkeit, um eine bidirektionale Abbildung mit Hilfe eines sogenannten *Schema Compiler's* zwischen XML-Dokumenten[32] und Java-Objekten durchzuführen. Hier werden nur die wesentlichen Aspekte, die im Rahmen dieser Diplomarbeit liegen, angesprochen. Für eine ausführliche Beschreibung von *JaxB* wird auf das Internet [SunJaxbDoc] verwiesen. Dort stehen die *JaxB* -API, Tools, Spezifiaktionen und Dokumente zum Download zur Verfügung.

2.6.1 Der Schema Compiler

Wie in der Abbildung 2.12 dargestellt, generiert der Schema Compiler auf der Grundlage eines XML-Schemas und eines XML-Binding-Schema einen Satz Java-Klassen. Das XML-Schema, das durch eine *.dtd-Datei (Document Type Definition) repräsentiert wird, spezifiziert die Struktur der XML-Daten. Das XML-Binding-Schema, das durch eine

[32] Informationen zu XML findet man auf der Webseite von World-Wide-Web-Konsortium unter der Internetadresse *http://www.w3c.org/*.

*.xjs-Datei repräsentiert wird, legt fest, wie das XML-Schema auf Java-Klassen abgebildet wird.

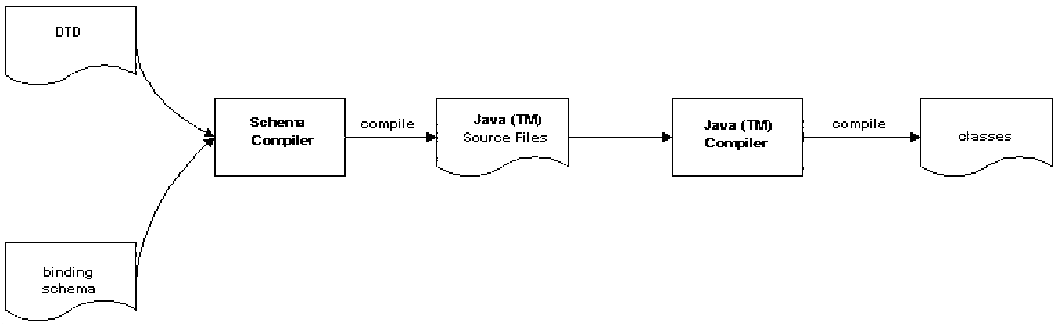

Abbildung 2.12: Erzeugung der Java-Klassen mit dem Schema Compiler
Quelle: [Microsystems (2001)]

Der Programmierer oder Entwickler muss sich nicht mit der Realisierung der komplexen Parsemethoden auseinandersetzen, da die vom Schema Compiler generierten Klassen mit *marshal()*-Methoden und *unmarshal*-Methoden ausgestattet sind. Diese Methoden kapseln die komplexen Parseaufgaben.

Eine Anwendung kann ein baumförmig strukuriertes Java-Ojekt aus den generierten Klassen erzeugen und sie mit der *marshal()*-Methode in eine XML-Datei transformieren. Auch kann man aus einer XML-Datei mit der *unmarshal()*-Methode ein neues baumförmig strukturiertes Java-Objekt generieren. Über*get*-Methoden kann man die Attribute des Java-Objektes auslesen und über *set*-Methoden kann das Java-Objekt manipuliert werden. Diese Vorgänge sind in der folgenden Abbildung 2.13 dargestellt:

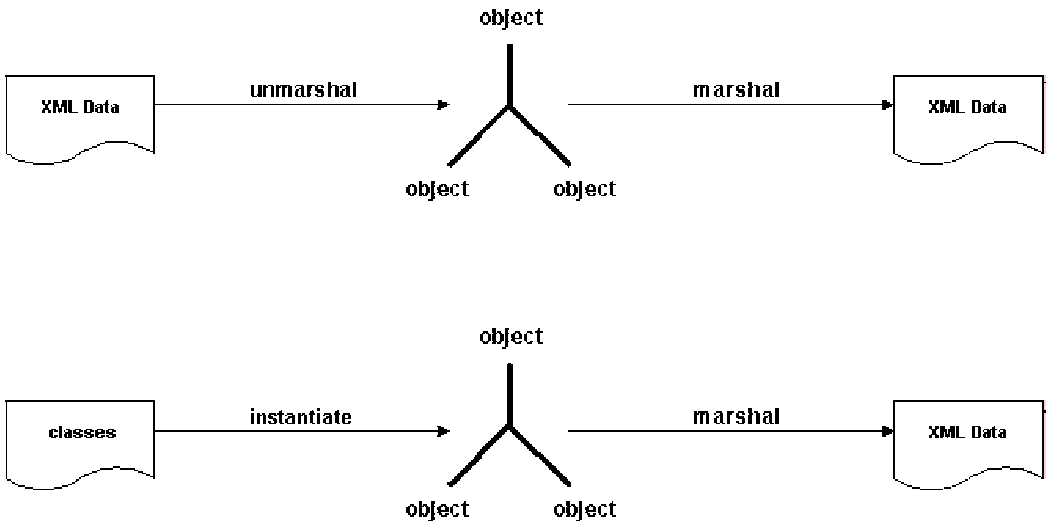

Abbildung 2.13: Der Transformationsprozess
Quelle: [Microsystems (2001)]

Quelle: [Microsystems (2001)], [Lin (2002)]

2.6.2 Ein kleines JaxB Beispiel

An dieser Stelle wird ein *JaxB* -Beispiel [Lin (2002)] beschrieben. Dieses Beispiel bezieht sich nicht direkt auf die *Flirtmaschine*, soll aber eine Einführung demonstrieren, wie man mit *JaxB* arbeitet. In diesem Beispiel sollen Produktinformationen in einer XML-Datei festgehalten werden. Eine XML-Datei könnte folgendermassen aussehen:

```
1   <item_list>
2      <item_info>
3         <name>Pen</name>
4         <price>1.99</price>
5      </item_info>
6   </item_list>
```

Abbildung 2.14: Die Datei item.xml

Wie aus der Abbildung 2.14 ersichtlich, werden zu jedem Produkt der Name und der Preis festgehalten. Zwischen den Tags können entweder Werte oder andere Tags enthalten sein.

Die Strukur der XML-Daten aus Abbildung 2.14 wird durch eine DTD-Datei beschrieben. Die Schreibweise, um die Strukur der XML-Daten in einer DTD zu beschreiben, ist an die EBNF[33]-Form angelehnt. Für die Datei *item.xml* aus Abbildung 2.14 sieht die DTD folgendermassen aus:

```
1   <!ELEMENT item_list (item_info*)>
2   <!ELEMENT item_info (name,price)>
3   <!ELEMENT name (# PCDATA)>
4   <!ELEMENT price (# PCDATA)>
```

Abbildung 2.15: Die Datei item.dtd

Mit dem *<!ELEMENT Name Ausdruck>*-Konstrukt werden die Tags in einem XML-Dokument definiert. Der *Name* spezifiziert den Namen des Tags, der *Ausdruck* gibt den Inhalt eines Tags an. Der Ausdruck kann aus Tags oder aus einer Zeichenkette bestehen. Die Zeichenkette wird durch *# PCDATA*[34] symbolisiert. Die Struktur der XML-Daten kann man sich baumförmig vorstellen, d.h., die Elemente können aus Kindelementen bestehen und diese Kindelemente können sich wiederum aus weiteren Kindelementen zusammensetzen. Das Element, welches keinem anderen Element untergeordnet ist, bildet das Wurzelelement. Die inneren Knoten sind einfache Elemente. Die Blätter des Baumes entsprechen Zeichenketten (*# PCDATA*).

[33] EBNF steht für *Extended Backus Naur Form*. Es ist eine Notation, mit dessen Hilfe man die Syntax einer Grammatik beschreiben kann.

[34] PCData steht für *Parse Character Data* und kann aus einer Kombination beliebiger Zeichen oder aus einer leeren Zeichenkette bestehen (Nicht alle Sonderzeichen sind erlaubt!).

Erläuterung der Datei *item.dtd* aus Abbildung 2.15:

- *Zeile 1*
  Mit dem Sternsymbol über dem *item_info*-Tag wird erreicht, dass das *item_list*-Tag beliebig viele *item_info*-Tags beinhalten darf. Das *item_list*-Tag bildet das Wurzelelement.

- *Zeile 2*
  Das *item_info*-Tag besteht aus den Tags *name* und *price*. Dabei müssen die Tags in der XML-Datei genau in derselben Reihenfolge erscheinen wie die Tag-Namen in der geschlossenen runden Klammer.

- *Zeile 3*
  Das *name*-Tag kann einen beliebigen Wert beinhalten. Dieser Wert kann aus einer Kombination von beliebigen Zeichen bestehen.

- *Zeile 4*
  Analog zu Zeile 3.

Als nächstes muss ein XML-Binding-Schema erstellt werden. Ein XML-Binding-Schema wird durch eine XJS-Datei repräsentiert und legt fest, wie die Elemente eines XML-Schema auf Java-Klassen abgebildet werden. Es soll hierbei angemerkt werden, dass nicht für jedes Element aus der DTD-Datei eine Bindungsanweisung, welche die Abbildung auf dem Java-Code beschreibt, in der XJS-Datei existieren muss. Für Elemente, die zwar in der DTD-Datei aber nicht in der XJS-Datei vorkommen, wird eine Standardabbildung benutzt. Für das obige Beispiel sieht die XJS-Datei folgendermassen aus:

```
01   <xml-java-binding-schema>
02      <options package=″profiling.xml″/>
03      <element name=″price″ type=″value″ convert=″int″ />
04      <element name=″name″ type=″value″ />
05      <element name=″item_list″ type=″class″ root=″true″ >
06         <content>
07            <element-ref name=″item_info″/>
08         </content>
09      </element>
10      <element name=″item_info″ type=″class″ >
11         <content>
12            <element-ref name=″name″/>
13            <element-ref name=″price″/>
14         </content>
15      </element>
16   </xml-java-binding-schema>
```

Abbildung 2.16: Die Datei item.xjs

Erläuterung der Datei *item.xjs* aus Abbildung 2.16:

- *Zeile 1*
  Das XML-Binding-Schema muss immer in dem *xml-java-binding-schema*-Tag eingehüllt sein.

- *Zeile 2*
  Das *package*-Attribut legt fest, welchen Packagenamen die Java-Klassen erhalten.

- *Zeile 3*
  Mit dem *element*-Tag wird beschrieben, wie ein bestimmter Tag an den Java-Code gebunden wird. Das *name*-Attribut aus der XJS-Datei muss mit dem Namen des Tags aus der DTD übereinstimmen. Das *type*-Attribut hat in diesem Fall den Wert *value* angenommen, damit wird das *price*-Tag aus der DTD an das Attribut einer Java-Klasse gebunden. Fehlt das *convert*-Attribut, dann werden Tags standardmäßig an String-Attribute einer Java-Klasse gebunden. Ansonsten, wie in diesem Fall, hat *convert* den Wert *int* angenommen und somit das *price*-Tag als Integer-Attribut an eine Java-Klasse gebunden. Zeile 3 erzeugt folgenden Java-Code (fett gedruckt dargestellt):

```
.   public class ItemInfo extends MarshallableObject
.      implements Element
.   {
.       ...
.      private int  _Price;
.      public int getPrice();
.      public void setPrice(int  _Price);
.   }
```

- *Zeile 4*
  Die Zeile 4 verhält sich analog zu der Erläuterung aus Zeile 3 und generiert folgenden Code (fett gedruckt dargestellt):

```
.   public class ItemInfo extends MarshallableObject
.      implements Element
.   {
.       ...
.      private String  _Name;
.      public String getName();
.      public void setName(String  _Name);
.   }
```

- *Zeile 5*
  Für das *item_list*-Tag wird eine eigene Klasse generiert, da das *type*-Attribut den Wert *class* angenommen hat. Diese Klasse ist ein Wurzelelement, da das *root*-Attribut den Wert *true* angenommen hat. Zeile 5 erzeugt folgenden Code (fett gedruckt dargestellt):

```
.   public class ItemList extends MarshallableRootElement
.      implements RootElement
.
.    ...
```

- *Zeile 6 bis 8*
  Das *Content*-Tag beschreibt, aus welchen Tags sich das *item_list*-Tag zusammensetzt, d.h., es beinhaltet die Kindelemente. Die Bindung der Kindelemente an ihren Elternelementen im Java-Code wird mit *element-ref*-Tag erreicht. Zeile 7 erzeugt folgenden Java-Code in der Klasse *ItemList* (fett gedruckt dargestellt):

```
.   public class ItemList
.      extends MarshallableRootElement
.      implements RootElement
.   {
.       ...
.      private List _ItemInfo;
.      public List getItemInfo();
.      public void deleteItemInfo();
.      public void emptyItemInfo();
.   }
```

  Die Klasse *List* verwaltet hier Instanzen von der Klasse *ItemInfo*. Da vorher in der DTD aus der Abbildung 2.15 mit dem Sternsymbol über das *item_info*-Tag festgelegt wurde, dass das *item_list*-Tag mehrere oder keine *item_info*-Tags beinhalten kann, referenziert die Klasse *ItemList* eine Instanz der Klasse *List*.

  Würde man das Sternsymbol weglassen, dann muss das *item_list*-Tag genau ein *item_info*-Tag enthalten, so würde hier folgender Code in der Klasse *ItemList* generiert werden (fett gedruck dargestellt):

```
.   public class ItemList
.      extends MarshallableRootElement
.      implements RootElement
.   {
.      private ItemInfo _ItemInfo;
.      public ItemInfo getItemInfo();
.      public void setItemInfo(ItemInfo _ItemInfo);
.   }
```

Die Java-Klassen, die vom *Schema Compiler* generiert werden, benutzen Klassen, erweitern Klassen und implementieren Schnittstellen aus dem *Binding Framework*. Das Jar-File **jaxb-rt-1.0-ea.jar** beinhaltet die Klassen und Interfaces für das *Binding Framework*. Das Jar-File **jaxb-xjc-1.0-ea.jar** beinhaltet die Klassen und Interfaces für den *Schema*

Compiler. Um aus der item.dtd und item.xjs die Java-Klassen zu erzeugen, muss man nur noch folgenden Kommandozeilen-Aufruf machen:

```
java -jar ”jaxb-xjc-1.0-ea.jar” item.dtd item.xjs
```

Ein Code-Beispiel für den Aufruf der *unmarshall*-Methode (siehe Seite 42):

```
01    File file = new File(”item.xml”);
02    ItemList il = new ItemList();
03    try { FileInputStream fin = new FileInputStream(file);
04          il = il.unmarshal(fin); }
05    finally { fin.close(); }
```

Abbildung 2.17: Der Unmarshall Aufruf

Erläuterung des Codes aus Abbildung 2.17:

- *Zeile 1*
  Quellpfadangabe der XML-Datei wird an den *File*-Konstruktor übergeben.
- *Zeile 4*
  Die *unmarshall*-Methode transformiert die Werte aus der XML-Datei *item.xml* in eine Javainstanz der Klasse *ItemList* und liefert diese Javainstanz als Rückgabewert, die der *il*-Variablen zugewiesen wird. Auf diese Werte kann man mit *get*- und *set*-Methoden zugreifen und manipulieren.

Ein Code-Beispiel für den Aufruf der *marshall*-Methode (siehe Seite 42):

```
01    File outFile = new File(”item.xml”);
02    try {
03       FileOutputStream fout=new FileOutputStream(outFile);
04       il. marshal(fout); }
05    finally { fout.close(); }
```

Abbildung 2.18: Der Marshall Aufruf

Erläuterung des Codes aus Abbildung 2.18:

- *Zeile 1*
  Zielpfadangabe der XML-Datei wird an den *File*-Konstruktor übergeben.
- *Zeile 4*
  Die *marshall*-Methode transformiert das *il*-Objekt in die XML-Datei *item.xml*.

Es ist also möglich, mit Hilfe der *unmarshall*-Methode eine Instanz der Klasse *ItemList* aus einer XML-Datei zu erzeugen oder, wenn keine XML-Datei vorhanden ist, eine Instanz der Klasse *ItemList* mit der *new*-Methode. Die Werte des erzeugten Java-Objektes können jetzt mit *set*-Methoden manipuliert werden. Anschließend kann man das Java-Objekt wieder mit der *marshall*-Methode in eine XML-Datei schreiben.

Quelle: [Lin (2002)]

Kapitel 3

Analyse

In diesem Kapitel soll eine Lösung für die im Abschnitt 1.1 vorgestellte *Flirtmaschine*, welche ein verteiltes System darstellt, mit Hilfe objektorientierter Konzepte modelliert[1] werden. Die Modellierung der Lösung hier in der Analyse umfasst dynamische und statische Aspekte und ist frei von implementierungstechnischen Details. Dieses Kapitel geht auf die nachfolgenden Punkte ein:

- **Abschnitt 3.1**
  Festlegung der Anforderungen an die Anwendung *Flirtmaschine*.
- **Abschnitt 3.2**
  Ermittlung und Spezifikation der Anwendungsfälle der *Flirtmaschine* auf Basis der Anforderungen.
- **Abschnitt 3.3**
  Modellierung der Klassen für die Interaktionskomponente.
- **Abschnitt 3.4**
  Modellierung der Klassen für die Kommunikationskomponente.
- **Abschnitt 3.5**
  Modellierung der Klassen für die Verarbeitungskomponente.
- **Abschnitt 3.6**
  Modellierung der Sequenzdiagramme.

3.1 Anforderungen

Im Abschnitt 1.1 auf Seite 4 wurde die *Flirtmaschine* und die beiden Beipielszenarien Stadt und Cafeteria bereits vorgestellt. In diesem Abschnitt werden die Anforderungen der *Flirtmaschine* im Detail festgehalten. Folgende Anforderungen werden an die *Flirtmaschine* gestellt:

1. Erstellen von Profilen
   Es soll möglich sein, ein eigenes Profil zu erstellen, deren Daten über eine GUI (Graphic User Interface; grafische Benutzeroberfläche) gesetzt werden. Dabei soll

[1] Für die Modellierung wurde in dieser Diplomarbeit UML eingesetzt. UML steht für *Unified Modeling Languages* und hat sich als Standardnotation in der Objektorientierung durchgesetzt. Mehr Informationen zur UML findet man in der Literatur von [Balzert (2001)], [Oestereich (1998)] und [Kahlbrandt (1998)].

die Eingabemöglichkeit möglichst so gestaltet werden, dass man die Attribute nur mit gültigen Werten setzen kann. Am besten wird die Auswahl der Werte über eine Liste zur Verfügung gestellt. Die Profildaten sind in 3 Gruppen unterteilt:

- Charakterisierende Attribute hinsichtlich der Person
  Name, Match-Wert, Alter, Gewicht, Grösse, Geschlecht, Augenfarbe, Haarfarbe, Figur, Figurtyp, Rauchen, Alkohol, Kinder, Kinder zuhause, Beruf, Kinderwunsch, Beschreibung
- Charakterisierende Attribute hinsichtlich der Präferenzen
  Party, Reisen, Sport, Tanzen, Disko, Essen gehen, Rad fahren, Segeln, Camping, Kunst, Computer, Konzerte, Filme, Joggen,Tennis, Theater, Musical, Einkaufen gehen, Spazieren gehen, Schreiben, Lesen, Kino, Mode
- Suchkriterien, die mit einem verglichenem Profil konform sind, um eine Übereinstimmung zu erreichen
  Geschlecht, Freundschaft,Partnerschaft, Altersbereich, Grössenbereich, Augenfarbe, Rauchen, Alkohol, Figur, Figurtyp, Haarfarbe, mit Kinder, Übereinstimmungswert

2. Aktivieren oder Deaktivieren der *Flirtmaschine* im Funkraum

   - Im aktivierten Zustand soll die *Flirtmaschine* den Funkraum nach anderen Profilen absuchen.
   - Im deaktivierten Zustand soll die *Flirtmaschine* den Funkraum nicht absuchen.

3. Vergleichsmethode des eigenen Profils mit einem im Funkraum gefundenen Profil

   - Die Vergleichsmethode soll einen Übereinstimmungsgrad der beiden Profilen in % ermitteln, dessen Wert auf beiden Seiten identisch ist.
   - Dabei soll es möglich sein, dass der Benutzer einen eigenen Übereinstimmungsgrad vorgeben kann (unter 1c als *Übereinstimmungswert* bezeichnet). Der Treffer wird den beiden Benutzern erst dann gemeldet, wenn der ermittelte Übereinstimmungsgrad (unter 1a als *Match-Wert bezeichnet*) grösser oder gleich dem höchsten der beiden vom Benutzer jeweils vorgegebenen Übereinstimmungsgrade ist.
   - Wurde kein eigener Übereinstimmungsgrad eingestellt, soll der Standardwert 50% gelten.
   - Wurde ein übereinstimmendes Profil gefunden , ist der Benutzer zu benachrichtigen. Dies soll durch ein akkustisches Signal unterstrichen werden. Nachdem die Nachricht bestätigt wurde, soll das Profil geladen und angezeigt werden.

4. Speichern, Laden on Profilen
   Das Laden und Speichern von Profilen muss möglich sein. Beim Laden des Profils soll zuerst das Bild des Profils erscheinen, die Profildaten kann man im Nachhinein betrachten.

5. Anzeige
   Die Profildaten und das dazugehörige Bild sollen über eine GUI angezeigt werden, wobei jede Profildatengruppe aus den 3 Profildatengruppen (siehe Punkt 1) sowie das Bild separat auf dem Display erscheinen.

6. Die *Flirtmaschine* muss mit der Situation umgehen können, wenn sie mehr als 1 Profil im Suchraum findet.

3.2 Anwendungsfälle

In diesem Abschnitt werden die Anwendungsfälle aus den Anforderungen bestimmt und durch Szenarien spezifiziert. In allen Szenarien gibt es nur einen einzigen Akteur und dies ist der mobile Benutzer, der mit der Flirtmaschine in Interaktion tritt.

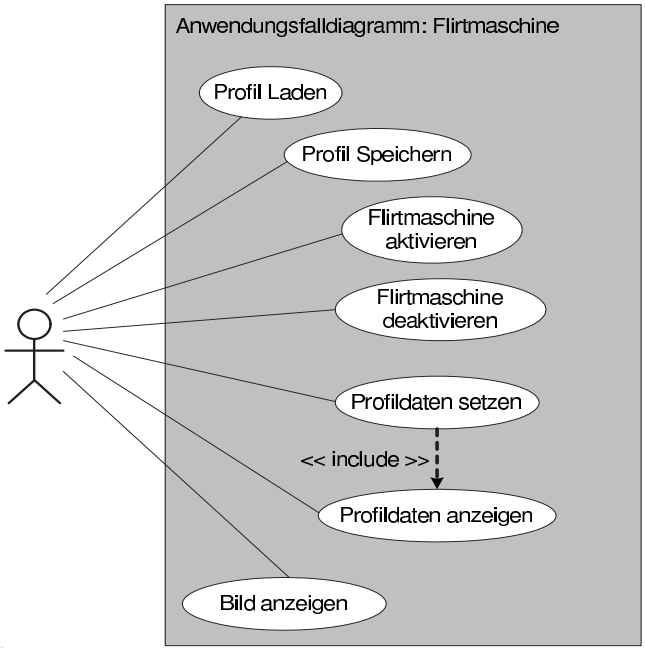

Abbildung 3.1: Anwendungsfalldiagramm: Flirtmaschine

Wie aus der obigen Abbildung 3.1 ersichtlich, gibt es insgesamt 7 Anwendungsfälle, die in den folgenden Szenarien beschrieben werden.

1. **Szenario: Profildaten anzeigen**
   Es zeigt die gesamten Attribute einer der drei Profildatengruppen auf dem Display separat an. Die Profildaten beziehen sich auf das aktuell geladene Profil. In der Menüleiste gibt es ein Menü *Ansicht*, das u.a. die Menüpunkte *über mich*, *meine Interessen* und *ich suche* beinhaltet.

   (a) Wird der Menüpunkt *über mich* selektiert, dann werden die charakterisierenden Attribute bezüglich der Person auf dem Display angezeigt:

   Name, Match-Wert, Alter, Gewicht, Grösse, Geschlecht, Augenfarbe, Haarfarbe, Figur, Figurtyp, Rauchen, Alkohol, Kinder, Kinder zuhause, Beruf, Kinderwunsch, Beschreibung

(b) Wird der Menüpunkt *meine Interessen* selektiert, dann werden die charakterisierenden Attribute hinsichtlich der Präferenzen auf dem Display angezeigt:

> *Party, Reisen, Sport, Tanzen, Disko, Essen gehen, Rad fahren, Segeln, Camping, Kunst, Computer, Konzerte, Filme, Joggen, Tennis, Theater, Musical, Einkaufen gehen, Spazieren gehen, Schreiben, Lesen, Kino, Mode*

(c) Wird der Menüpunkt *ich suche* selektiert, dann werden die Suchkriterien auf dem Display angezeigt, deren Kriterien mit den Profildaten eines anderen Profils konform sein müssen, damit ein Match zustande kommen kann. Folgende Suchkriterien gibt es:

> *Geschlecht, Freundschaft, Partnerschaft, Altersbereich, Grössenbereich, Augenfarbe, Rauchen, Alkohol, Figur, Figurtyp, Haarfarbe, mit Kinder, Übereinstimmungswert*

2. **Szenario: Profildaten setzen**
   Der Benutzer kann die einzelnen Attribute jeder Profildatengruppe setzen. Hierfür holt er sich lediglich die gewünschte Profildatengruppe zur Ansicht (siehe Anwendungsfall *Profildaten anzeigen*).

3. **Szenario: Bild anzeigen**
   In der Menüleiste gibt es ein Menü *Ansicht*, das den Menüpunkt *mein Bild* beinhaltet. Wird dieser Menüpunkt vom Benutzer angewählt, zeigt das Display nur das Bild zum Profil an. Ist kein Bild vorhanden, erscheint ein leeres Bild.

4. **Szenario: Profil laden**

   (a) In der Menüleiste gibt es ein Menü *File*, das den Menüpunkt *Profil laden* enthält. Wird dieser Menüpunkt vom Benutzer angewählt, erscheint ein Dialogfenster.

   (b) Anschließend kann man jetzt die gewünschte Profildatei angeben.

   (c) Entweder kann man den Dialog jetzt abbrechen, das Dialogfenster schließt sich und es ist nichts passiert, oder man bestätigt den Dialog und es wird versucht, das angegebene Profil zu laden. Ist es nicht vorhanden, passiert ebenfalls nichts. Beim erfolgreichen Laden werden die Profildaten gesetzt und das Bild des Profils erscheint auf dem Display.

5. **Szenario: Profil speichern**

   (a) Der Benutzer muss das Profil entweder selbst vorher setzen (s. Anwendungsfall *Profil setzen*) oder es vorher laden (s. Anwendungsfall *Profil Laden*). Wird weder ein Profil gesetzt noch geladen, verwendet die *Flirtmaschine* ein Profil mit voreingestellten Werten für den Speichervorgang.

   (b) In der Menüleiste gibt es ein Menü *File*, das den Menüpunkt *Profil speichern* enthält. Wird dieser Menüpunkt angewählt, erscheint ein Dialogfenster.

   (c) In diesem Dialogfenster kann man den Zielort und den Namen der Profildatei angeben.

(d) Entweder kann man den Dialog jetzt abbrechen, das Dialogfenster schließt sich und es passiert nichts, oder man bestätigt den Dialog und das Profil wird am angegebenen Zielort gespeichert. Beim erfolgreichen Speichervorgang wird eine Profildatei am angegebenen Zielort angelegt, ansonsten geschieht nichts.

6. **Szenario: Flirtmaschine aktivieren**

   (a) Betätigt der Benutzer den Button *Flirtmaschine aktivieren*, beginnt die *Flirtmaschine* kontinuierlich im Funkraum nach neuen Profilen zu suchen.

   (b) Wird ein Profil gefunden, dann gleicht die *Flirtmaschine* ab, ob das gefundene Profil mit dem Profil des Benutzers übereinstimmt.

   - Im Fall einer Übereinstimmung der Profile wird der Benutzer in Form eines akkustischen Signals darauf aufmerksam gemacht. Es erscheint ein Dialogfenster mit den Informationen über das gefundene Profil. Die Informationen geben den Namen des gefundenen Profils und die Höhe des Übereinstimmungsgrades an. Bestätigt der Benutzer das Dialogfenster, wird das übereinstimmende Profil geladen (s. Anwendungsfall *Profil laden*).
   - Im Fall einer Nichtübereinstimmung der Profile geschieht nichts.

   (c) Ist die *Flirtmaschine* immer noch aktiv, wird der Suchraum weiterhin nach neuen Profilen abgesucht. Gehe zurück nach Schritt 6.b.

7. **Szenario: Flirtmaschine deaktivieren**
   Betätigt der Benutzer den Button *Flirtmaschine deaktivieren*, bricht die *Flirtmaschine* ihre Suche im Funkraum ab.

3.3 Klassen der Interaktionskomponente

In diesem Abschnitt wird auf die Klassen eingegangen, die für die Interaktion mit dem Benutzer verantwortlich sind. Diese Klassen stellen die grafische Benutzeroberfläche für den Benutzer dar und leiten alle Aktionen des Benutzers an die Verarbeitungskomponente weiter (siehe Abschnitt 3.5).

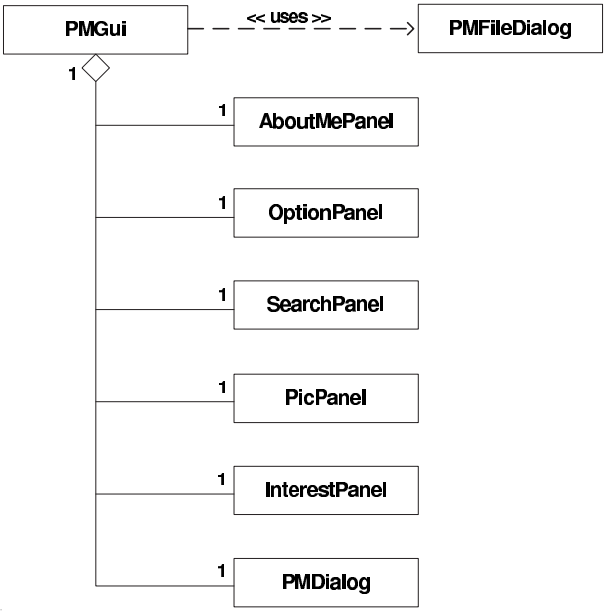

Abbildung 3.2: Die Klassen der Interaktionskomponente

Die in der Abbildung 3.3 dargestellte Klasse *PMGUI* stellt die grafische Benutzeroberfläche der Flirtmaschine dar.

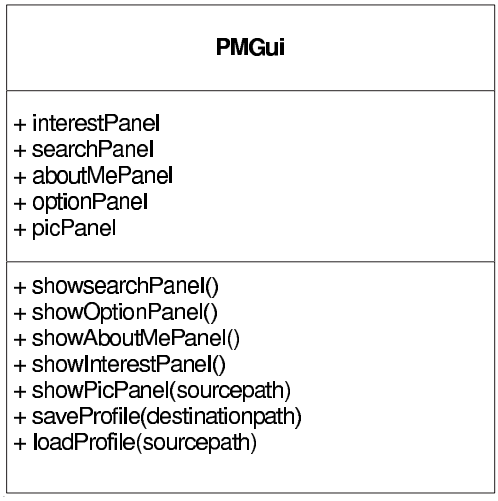

Abbildung 3.3: Die Klasse PMGUI

- *showSearchPanel()*
  Diese Methode zeigt die Suchkriterien für den gewünschten Partner an. Für die Visualisierung dieser Suchkriterien ist die Klasse *SearchPanel* verantwortlich.

- *showOptionPanel()*
  Diese Methode zeigt die Buttons *deactivate* und *activate* an, mit denen der Benutzer die *Flirtmaschine* deaktivieren bzw. aktivieren kann.

- *showAboutMePanel()*
  Diese Methode zeigt die personenbeschreibenden Attribute des aktuell geladenen Profils an. Für die Visualisierung dieser Attribute ist die Klasse *AboutMePanel* verantwortlich.

- *showInterestPanel()*
  Diese Methode zeigt die Präferenzen der Person an. Für die Visualisierung dieser Attribute ist die Klasse *InterestPanel* verantwortlich.

- *showPicPanel(sourcepath)*
  Diese Methode zeigt das Bild der Person an, wobei der *sourcepath*-Parameter den Quellort angibt. Für die Visualisierung des Bildes ist die Klasse *PicPanel* verantwortlich.

- *loadProfile(sourcepath)*
  Diese Methode lädt die Profildaten und setzt sie in der GUI, wobei der *sourcepath*-Parameter den Quellort des Profils angibt.

- *saveProfile(destinationpath)*
  Diese Methode entnimmt die Profildaten aus der GUI und speichert sie, wobei der *destinationpath*-Parameter den Speicherort des Profils angibt.

Die in der Abbildung 3.4 dargestellte Klasse *OptionPanel* beinhaltet die Buttons *deactivate* und *activate*.

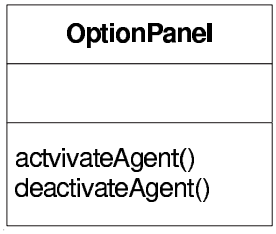

Abbildung 3.4: Die Klasse OptionPanel

- *activateFlirtengine()*
  Diese Methode aktiviert die Flirtmaschine, die daraufhin im Funkraum mit der kontinuierlichen Suche nach neuen Profilen beginnt.

- *deactivateFlirtengine()*
  Diese Methode deaktiviert die Flirtmaschine, die daraufhin die Suche abbricht.

Die in der Abbildung 3.2 dargestellten Klassen *SearchPanel*, *InterestPanel*, *AboutMePanel* und *PicPanel* sind für die Visualisierung der Profildaten verantwortlich. Auf die einzelnen Methoden und Attribute dieser Klassen soll hier nicht weiter eingegangen werden, da sie stark von der gewählten Programmiersprache abhängen und nicht zur konzeptionellen Lösung des Kernproblems beitragen. Das Kernproblem besteht darin, eine Lösung für die Kommunikationskomponente, Verarbeitungskomponente und Datenhaltungskomponente zu finden. Die Details zur Realisierung der Interaktionskomponente ist dem beigefügten Quellcode dieser Diplomarbeit zu entnehmen (siehe Anhang A auf Seite 101).

Die in der Abbildung 3.2 dargestellte Klasse *PMDialog* sorgt dafür, dass bei Übereinstimmung zweier Profile die entsprechenden Informationen angezeigt werden. Diese beinhalten den Datei- und Benutzernamen des Profils und den Match-Wert (Übereinstimmungsgrad) beider Profile.

Die in der Abbildung 3.5 dargestellte Klasse *PMFileDialog* stellt dem Benutzer die geforderten Speicher- und Ladefunktionalitäten bezüglich der Profile zur Verfügung.

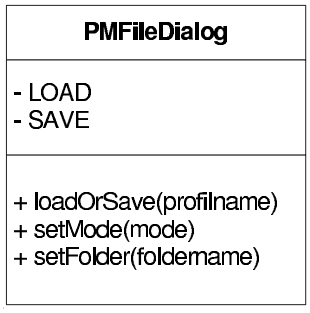

Abbildung 3.5: Die Klasse FileDialog

- *loadOrSave(profilname)*
  Diese Methode lädt oder speichert das Profil, je nachdem welcher Modus eingestellt worden ist, wobei der *profilname*-Parameter den Namen des geladenen bzw. gespeicherten Profils angibt.

- *setMode(mode)*
  Der übergebende Parameter *mode* der *set*-Methode kann als Wert entweder die LOAD- oder SAVE-Konstante annehmen. Wird die *LOAD*-Konstante der *set*-Methode übergeben, dann befindet sich der Filedialog im Lademodus und ein Profil kann geladen werden. Bei der *SAVE*-Konstante der *set*-Methode hingegen befindet sich der Filedialog im Speichermodus und ein Profil kann gespeichert werden.

- *setFolder(foldername)*
  Der *foldername*-Parameter dieser Methode legt das Verzeichnis (Ordner) fest, dass den genauen Ziel- bzw. Quellort des zu speichernden bzw. zu ladenden Profils angibt.

3.4 Klassen der Kommunikationskomponente

In diesem Abschnitt wird auf die Klassen, die für die Kommunikation verantwortlich sind, eingegangen. Die Kommunikationskomponente ermöglicht, dass zwei Geräte mittels Funk ihre Flirtprofile austauschen können. Mit einer Instanz der Klasse *PMClient* ist es möglich, Anfragen nach dem auf den Servern jeweils befindlichen Flirtprofil zu stellen. Diese Anfragen werden von einer Instanz der Klasse *PMServer* beantwortet. Dabei nimmt die *Flirtmaschine* des jeweiligen Gerätes die Client- und die Server-Rolle zugleich ein.

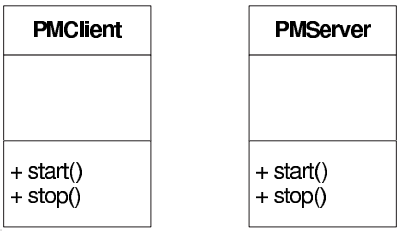

Abbildung 3.6: Die Klassen der Kommunikationskomponente

Die in der Abbildung 3.6 dargestellte Klasse *PMServer* beinhaltet die zwei folgenden Methoden:

- *start()*
  Diese Methode aktiviert den Server, welcher daraufhin anfragende Clients mit dem lokalen Flirtprofil bedient.

- *stop()*
  Diese Methode deaktiviert den Server, welcher nun keine Anfragen mehr entgegen nimmt.

Die in der Abbildung 3.6 dargestellte Klasse *PMClient* beinhaltet die zwei folgenden Methoden:

- *start()*
  Diese Methode aktiviert den Client, der daraufhin in der Funkumgebung nach neuen Geräten sucht, welche die *Flirtmaschine* unterstützen. Hauptaufgabe dieser Methode ist es, die Flirtprofile per Funk anzufordern.

- *stop()*
  Diese Methode deaktiviert den Client, der nun nicht mehr nach neuen Geräten bzw. Profilen sucht.

3.5 Klassen der Verarbeitungskomponente

In diesem Abschnitt wird die in der Abbildung 3.7 dargestellte Klasse *ProfileManager*, welche die einzige Klasse der Verarbeitungskomponente bildet, vorgestellt. Die Klasse *ProfileManager* ist für die eigentliche Umsetzung der Anwendungsfälle und der dazugehörigen Szenarien aus dem Abschnitt 3.2 verantwortlich. Nachfolgend werden die Methoden und Attribute dieser Klasse beschrieben:

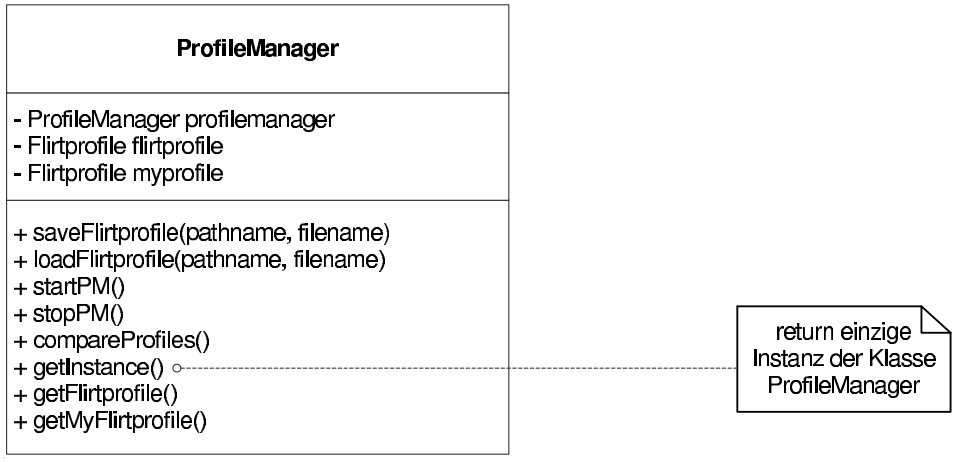

Abbildung 3.7: Die Klasse Profilemanager

- *saveFlirtprofile (pathname, filename)*
  Diese Methode speichert das Flirtprofil unter dem angegebenen Zielort ab, wobei die Parameter *pathname* und *filename* den Zielort repräsentieren.

- *loadFlirtprofile (pathname,filename)*
  Diese Methode lädt das Flirtprofil unter denm angegebenen Quellort, wobei die Parameter *pathname* und *filename* den Quellort repräsentieren.

- *startPM()*
  Diese Methode leitet den Suchvorgang im Funkraum nach neuen Profilen ein.

- *stopPM()*
  Diese Methode beendet den Suchvorgang.

- *compareProfiles()*
  Diese Methode gleicht das gefundene Profil mit dem eigenem Profil ab und liefert als Rückgabewert einen boolschen Wert. Ist der boolsche Wert *wahr*, dann stimmen die Profile miteinander überein, ansonsten nicht .

- *getFlirtprofile()*
  Diese Methode liefert als Rückgabewert das aktuell geladene Profil.

- *getMyFlirtprofile()*
  Diese Methode liefert als Rückgabewert das eigene Profil.

- *getInstance()*
  Diese Methode liefert als Rückgabewert die einzige Instanz der Klasse *Profilemanager* und soll auf der Basis des *Singleton Patterns*[2] realisiert werden.

  Das **Singleton Pattern** stellt sicher, dass es von einer Klasse nur ein einziges Objekt gibt und stellt einen globalen Zugriffspunkt darauf bereit. Wie aus der Abbildung 3.7 erkennbar, besitzt die Klasse *ProfileManager* eine Klassenvariable *profilemanager*, welche auf die einzige Instanz der Klasse *ProfileManager* zeigt. Auf diese Instanz kann man nur über die Klassenmethode *getInstance()* zugreifen. Diese Klassenmethode liefert die einzige Instanz der Klasse *ProfileManager* als Rückgabewert und stellt dabei sicher, dass wirklich nur eine Instanz dieser Klasse existiert.

[2] Singleton Pattern ist ein Entwurfsmuster. In der Softwareentwicklung beschreiben Entwurfsmuster eine allgemeine Lösung für immer wiederkehrende Problemsituationen. Für weitere Erläuterungen soll an dieser Stelle auf die Literatur von [Gamma u. a. (1996)] verwiesen werden.

3.6 Sequenzdiagramme

In diesem Abschnitt werden die beiden Anwendungsfälle *Flirtmaschine aktivieren* und *Flirtmaschine deaktivieren* als Sequenzdiagramme dargestellt.

Aktivieren der Flirtmaschine

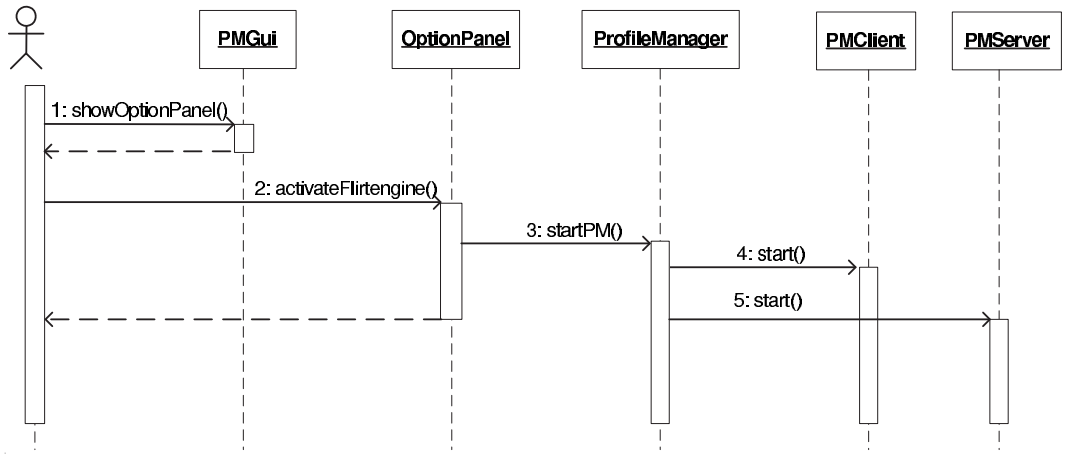

Abbildung 3.8: Die Methode activateFlirtengine()

Das Diagramm in Abbildung 3.8 beschreibt die Vorgänge, die beim Aktivieren der Flirtmaschine ablaufen:

1. *showOptionPanel()*
   Das Aufrufen dieser Methode durch den Benutzer bewirkt die Ansicht der *activate*- und *deactivate*-Buttons auf dem Display.

2. *activateFlirtengine()*
   Das Betätigen des activate-Buttons durch den Benutzer aktiviert die Flirtmaschine. Der Aktvierungsprozess wird mit der Methode *startPM()* an die einzige Instanz der Klasse *ProfileManager* weiter delegiert.

3. *startPM()*
   Der interne Aufruf dieser Methode leitet die Suche nach neuen Profilen im Funkraum ein und aktiviert den Server. Was sie im Detail leistet, wird im Abschnitt 4.6 auf der Seite 73 beschrieben.

4. *start()*
   Der interne Aufruf dieser Methode aktiviert den Client, der nun in der Funkumgebung nach neuen Profilen sucht.

5. *start()*
   Der interne Aufruf aktiviert den Server, der nun anfragende Clients mit dem eigenen Profil bedient.

Deaktivieren der Flirtmaschine

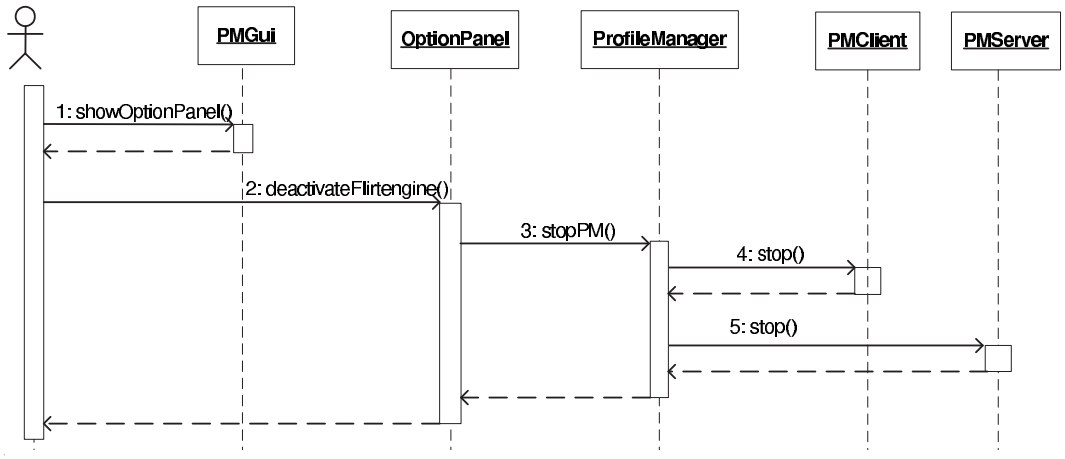

Abbildung 3.9: Die Methode deactivateFlirtengine()

Das Diagramm in der Abbildung 3.9 beschreibt die Vorgänge, die beim Deaktivieren der Flirtmaschine ablaufen:

1. *showOptionPanel()*
   Das Aufrufen dieser Methode durch den Benutzer bewirkt die Ansicht der *activate*- und *deactivate*-Buttons auf dem Display.

2. *deactivateFlirtengine()*
   Das Betätigen des *deactivate*-Buttons durch den Benutzer deaktiviert die Flirtmaschine. Der Deaktivierungsprozess wird mit der Methode *stopPM()* an die einzige Instanz der Klasse *ProfileManager* weiter delegiert.

3. *stopPM()*
   Der interne Aufruf beendet den Suchvorgang im Funkraum nach neuen Profilen und deaktiviert den Server. Was diese Methode im Detail leistet, wird im Abschnitt 4.6 auf der Seite 73 beschrieben.

4. *stop()*
   Der interne Aufruf deaktiviert den Client, welcher die Suche nach neuen Profilen abbricht.

5. *stop()*
   Der interne Aufruf deaktiviert den Server, welcher nun keine Anfragen von Clients mehr entgegen nimmt.

Kapitel 4

Design

In diesem Kapitel wird erläutert, welche Technologien, Programmiersprachen und Algorithmen für die Realisierung der *Flirtmaschine* zum Einsatz gekommen sind. Zudem wird die Architektur der *Flirtmaschine* beschrieben und das Klassenmodell aus der Analyse wird hier insbesondere um programmierspezifische Klassen und Methoden erweitert. Bereits bestehende Methoden werden weiterhin konkretisiert hinsichtlich ihrer Arbeitsweise und den eingesetzten Algorithmen. Auch wird auf die verwendete Zielplattform eingegangen und im letzten Abschnitt wird ein Konzept für die kommunikation vorgestellt. Neu hinzugekommen ist die Datenhaltungskomponente, welche für die Persistenz der Profildaten verantwortlich ist.

4.1 Zielplattform

Abbildung 4.1: Der Compaq iPAQ H3870

Als Zielplattform für die Entwicklung der *Flirtmaschine* wurde mir vom Labor ein PDA vom Typ *Compaq iPAQ H3870* zur Verfügung gestellt. Dieser PDA ist vom Werk mit 64 MB RAM, 32 MB ROM, Strongarm SA1110 Prozessor, integrierter Bluetooth-Schnittstelle, JVM Jeode[1] und mit dem Betriebssystem Microsoft Pocket PC 2002 ausgestattet.

[1] Jeode ist eine Java Virtuelle Maschine (JVM) von der Firma Insignia (www.insignia.com). Jeode entspricht dem Personal Java 1.2 Spezifikation. Mehr Informationen zu Java findet man im Abschnitt 2.4 ab Seite 14.

4.2 Programmiersprache

In diesem Abschnitt wird erläutert, warum und welche Programmiersprache eingesetzt worden ist.
Für die Interaktionskomponente, Verarbeitungskomponente und die Datenhaltungskomponente wurde *Personal Java 1.2* verwendet. *Personal Java 1.2* ist speziell für mobile Geräte entwickelt worden, die über reichlich Speicherressourcen verfügen. Auf der Zielplattform wird die *Java Virtuelle Maschine Jeode* verwendet, die der *Personal Java 1.2* Spezifikation entspricht. Die Programmiersprache *Java* wurde deshalb verwendet, da der geschriebene Code plattformunabhängig ist und somit auf jeder Rechnerplattform ausführbar ist, die eine *Java Virtuelle Maschine* (JVM) ausführt. *JaxB* ist ein weiterer Grund für mich gewesen, *Java* einzusetzen. *JaxB* ist ein Framework, mit dem man bequem seine eigene tagbasierte Dokumentensprache entwickeln kann. Die *JaxB* -Technologie ist bei der Implementierung der Datenhaltungskomponente (siehe Seite 68) eingesetzt worden.

Zum Zeitpunkt, als diese Diplomarbeit entstand, musste ich leider feststellen, dass es noch nicht möglich ist, die Bluetooth-Schnittstelle unter der Programmiersprache *Java* anzusprechen. Eine Bluetooth-API [2] für die CLDC-Konfiguration ist schon auf dem kommerziellen Weg erhältlich, aber nicht wie von mir benötigt für die CDC-Konfiguration. Ich denke, dass in naher Zukunft geeignete Java-API's, welche die Bluetooth-Technologie unterstützen, auf dem Markt angeboten werden. Da im Moment also keine Bluetooth-API in Java zur Verfügung steht, musste von mir eine alternative Lösung gefunden werden.
Eine Alternative wäre eine hybride Lösung. Ein Teil der *Flirtmaschine* könnte in Java und der andere Teil in der Programmiersprache *C++* implementiert werden. Die Kommunikationskomponente benötigt den Zugriff auf den Bluetooth-Funk und könnte mit der Programmiersprache *C++* implementiert werden. Die restlichen Komponenten der *Flirtmaschine*, wie bereits weiter oben in diesem Abschnitt erwähnt, sind in *Java* implementiert worden. Nachteil der *C++*-Implementation wäre, dass sie nicht so ohne weiteres auf eine andere Zielplattform portierbar ist. Die *C++*-Implementation muss also für eine andere Zielplattform angepasst oder gar neu entworfen werden.

Auf dem Markt werden *C++*-Bibliotheken[3] speziell für den *Compaq Ipaq H3870*-PDA, welcher die Bluetechnologie unterstützt, angeboten. Aufgrund der relativ hohen Lizenzkosten für diese Programmierbibliotheken haben wir, ich und mein betreuender Professor Dr. Klemke, entschieden, auf diese Bibliotheken zu verzichten. Stattdessen soll für die Kommunikationskomponente ein konzeptioneller Lösungsansatz entworfen werden (siehe Seite 81).

[2] Die Bluetooth-API für die CLDC-Konfiguration kann man zum Beispiel bei Motorola auf dem kommerziellen Weg erhalten. Informationen findet man unter der Internetadresse *http://www.motorola.com/java*.

[3] Die Bluetooth-Bibliotheken werden von der Firma Widcomm (*www.widcomm.com*) und der Firma High Point Software (*www.high-point.com*) angeboten.

4.3 Systemarchitektur

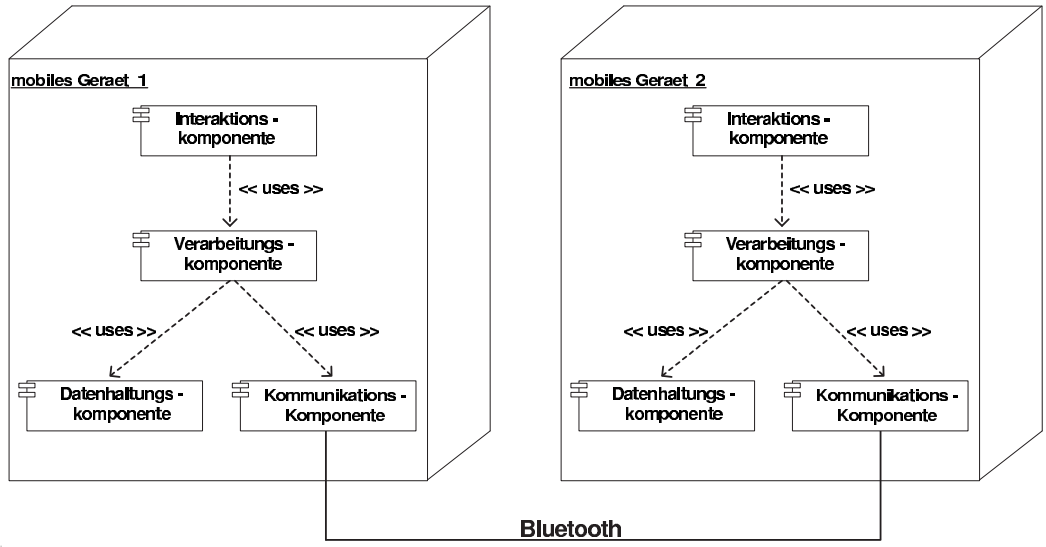

Abbildung 4.2: Die Systemarchitektur der Flirtmaschine

Wie aus der Abbildung 4.3 ersichtlich ist, besteht die *Flirtmaschine* aus 4 Komponenten, dessen Aufgaben wie folgt grob beschrieben werden:

- **Interaktionskomponente**
  Wie bereits im Abschnitt 3.3 erwähnt, nimmt die Interaktionskomponente die Aktionen der Benutzer entgegen und leitet sie zur Ausführung an die Verarbeitungskomponente weiter. Die Interaktionskomponente stellt die Benutzeroberfläche dar.

- **Verarbeitungskomponente**
  Die Aufgabe der Verarbeitungskomponente wurde auch schon im Abschnitt 3.5 vorgestellt. Sie sorgt für die eigentliche Umsetzung aller Aktionen, die von der Interaktionskomponente kommen.

- **Datenhaltungskomponente**
  Die Datenhaltungskomponente ist für die Persistenz der Profilaten verantwortlich und stellt Operationen für den lesenden, den schreibenden und den manipulierenden Zugriff zur Verfügung.

- **Kommunikationskomponente**
  Die vorgestellte Kommunikationskomponente aus Abschnitt 3.4 sorgt für die Übertragung der Profile via Funk von einem mobilen Gerät zu einem anderen.

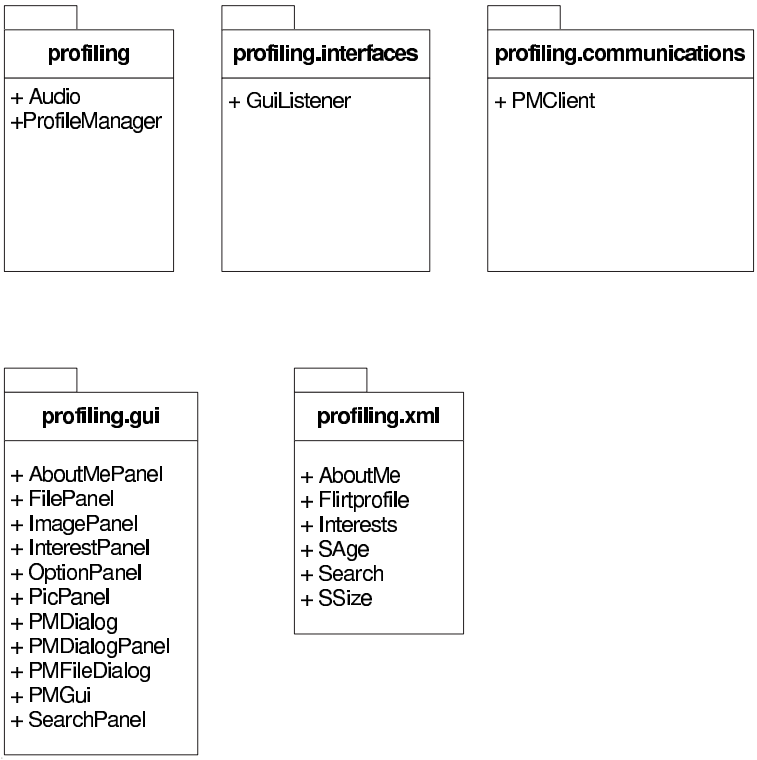

Abbildung 4.3: Die Pakete der Flirtmaschine

In der obigen Abbildung 4.3 sind die Pakete der *Flirtmaschine* dargestellt. Die Interaktionskomponente setzt sich aus den folgenden Klassen und Schnittstellen zusammen:

- Alle Klassen aus dem Paket *profiling.gui*
- Interface *GuiListener* aus dem Paket *profiling.interfaces*
- Klasse *Audio* aus dem Paket *profiling*

Die Verarbeitungskomponente wird durch die Klasse *Profilemanager* aus dem Paket *profiling*, die Kommunikationskomponente[4] wird durch die Klasse *PMClient* aus dem Paket *profiling.communications* und die Datenhaltungskomponente wird durch alle Klassen aus dem Paket *profiling.xml* repräsentiert.

[4]Die Klasse *PMServer* aus der Analyse, wird nicht benötigt. Der Grund dafür wird im Abschnitt 4.7.1 gegeben.

4.4 Interaktionskomponente

4.4.1 Das erweiterte Klassenmodell

In diesem Abschnitt wird das Klassenmodell für die Interaktionskomponente aus der Analyse (siehe Seite 53) um programmiersprachenspezifische Klassen aus *Java* erweitert. Das aus der Analyse erweiterte Klassenmodell der Interkationskomponente ist in der Abbildung 4.4 dargestellt.

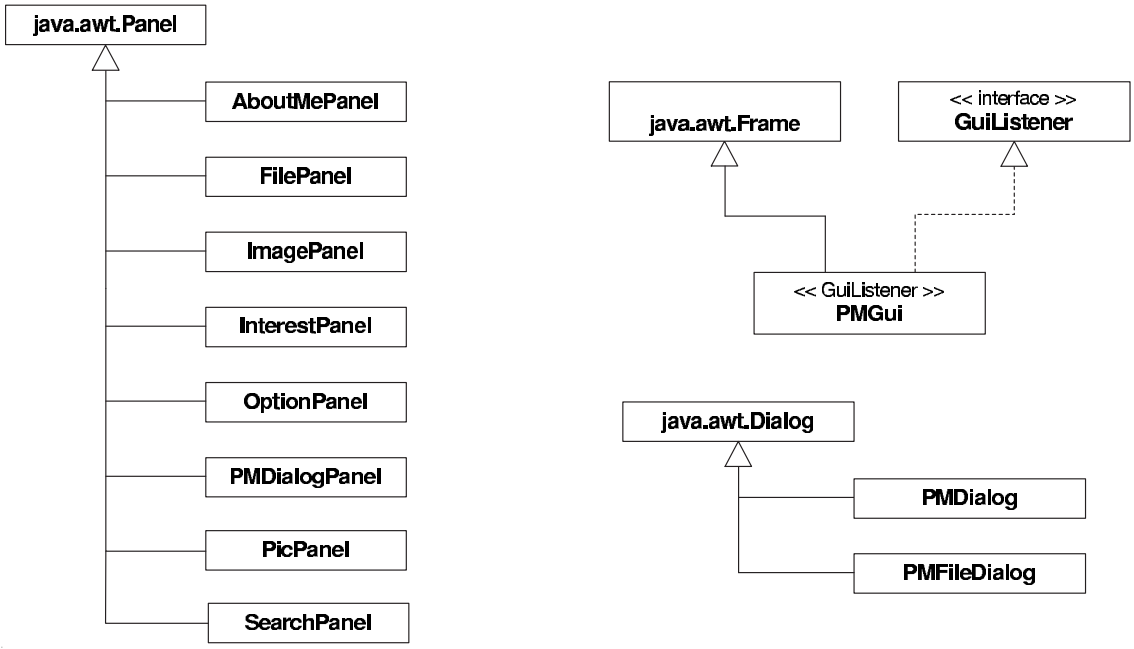

Abbildung 4.4: Die Klassen der Interaktionskomponente

Wie im Abschnitt 4.2 erwähnt, wurde hauptsächlich die Programmiersprache *Java* für die *Flirtmaschine* eingesetzt. Für die Klassen der Interaktionskomponente wurde die AWT[5]-Klassenbibliothek verwendet, da sie 100% kompatibel zu *Personal Java 1.2* ist. Die Klassen sind bereits in der Analyse im Abschnitt 3.3 ausreichend besprochen worden. In dieser Diplomarbeit soll nicht auf die kompletten Details der Klassen aus der Interaktionskomponente eingegangen werden, da der Schwerpunkt dieser Arbeit nicht bei der Erstellung einer grafischen Benutzeroberfläche liegt (siehe unter Kernproblem auf Seite 55).

Wie man aus der Abbildung 4.4 leicht erkennt, erweitert die *PMGui*-Klasse die *Frame*-Klasse, alle *Dialog*-Klassen (*PMDialog*, *PMFileDialog*) erweitern die *Dialog*-Klassen und alle *Panel*-Klassen (*AboutMePanel*, *FilePanel* ...) erweitern die *Panel*-Klasse. Mit der Klasse *Frame* kann man einen Windows-Fensterrahmen erzeugen. In so einem Rahmen lassen sich diverse GUI-Objekte platzieren. Ein *Panel*-Objekt stellt eine Fläche dar, auf der man zeichnen und ebenfalls andere GUI-Objekte platzieren kann, dabei hat man die Möglichkeit, ein *Panel*-Objekt in mehrere andere *Panel*-Objekte in beliebiger Tiefe zu schachteln. Ein *Dialog*-Objekt erzeugt ein Dialogfenster.

[5] AWT steht für Abstract Window Toolkit. AWT ist eine Klassenbibliothek und beinhaltet Klassen, mit deren Hilfe man leicht Elemente für die grafische Benutzeroberfläche erzeugen kann. Für weitere Informationen zur Programmiersprache *Java* und grundlegende Konzepte wird auf die Literatur von [Horstmann und Cornell (1999)] verwiesen.

4.4.2 Die Aggregatstruktur der Interaktionskomponente

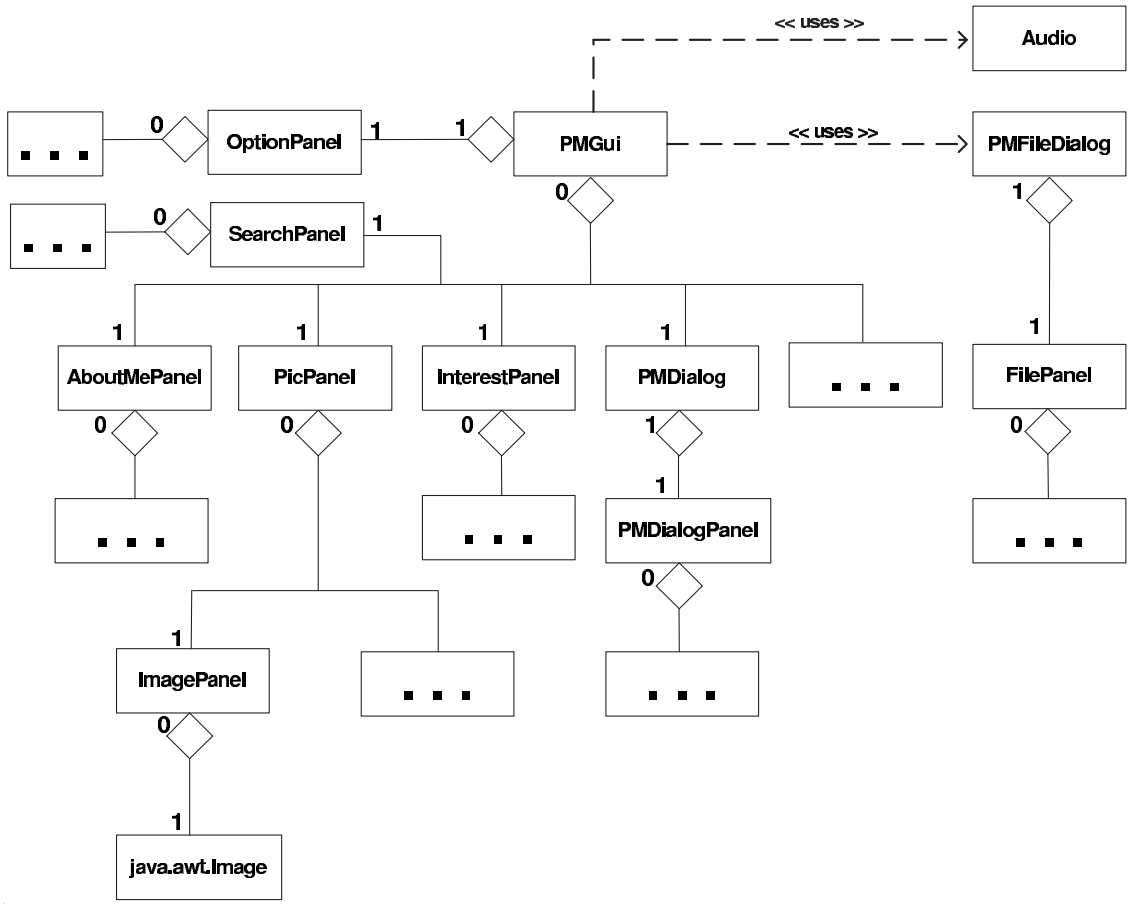

Abbildung 4.5: Aggregatstruktur der Interaktionskomponente

Die Aggregatstruktur der Interaktionskomponente ist in der obigen Abbildung 4.5 dargestellt. Dabei symbolisieren die mit 3 Punkten markierten Klassen mindestens eine oder mehrere javaspezifische Klassen aus der AWT-Klassenbibliothek, welche Bestandteile der Aggregatklassen sind. Zum Beispiel besteht ein Objekt der Klasse InterestPanel aus mehreren *Checkbox*-Elementen. Wie bereits erwähnt, wird auf solche Details nicht eingegangen. Detaillierte Informationen zur Realisierung der Interaktionsklassen sind direkt aus dem Quellcode entnehmbar (siehe Anhang A auf Seite 101).

4.4.3 Unabhängigkeit der Verarbeitungskomponente von der Interaktionskomponente

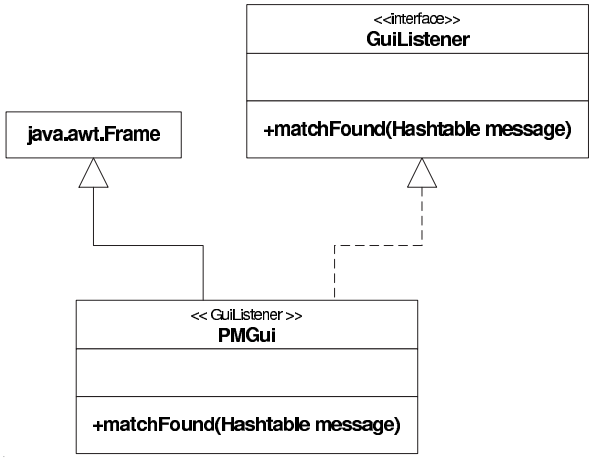

Abbildung 4.6: Die Klasse GuiListener

Wie in der Abbildung 4.6 dargestellt wird, erbt die Klasse *PMGui* von dem Interface *GuiListener*. Alle GuiListener-Objekte haben die Möglichkeit, sich mit der *addGuiListener(GuiListener gui_ listener)*-Methode der Klasse *ProfileManager* bei der Verarbeitungskomponente zu registrieren (siehe Seite 70). Sobald die Verarbeitungskomponente ein übereinstimmendes Profil findet, wird dieses Ereignis an alle registrierte *GuiListener*-Objekte gemeldet. Die Meldung dieses Ereignisses geschieht, indem die *matchFound(Hashtable message)*-Methode der *GuiListener*-Objekte aufgerufen wird, welches dieses Ereignis entsprechend behandelt. Durch dieses Verfahren wird erreicht, dass die Verarbeitungskomponente unabhängig von der Interaktionskomponente ist. Somit ist es möglich, die GUI jederzeit gegen eine neue GUI, die nur das Interface *GuiListener* implementieren muss, auszutauschen.

Was die *matchFound(Hashtable message)*-Methode leisten soll, wird durch das folgende Aktivitätsdiagramm (Abbildung 4.7) dargestellt:

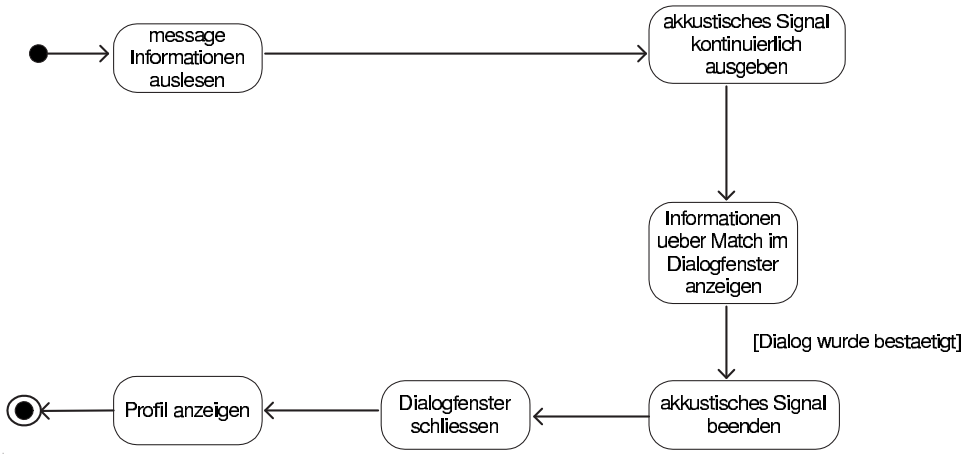

Abbildung 4.7: Aktivitätsdiagramm: Methode *matchFound(Hashtable message)*

Bei den Informationen, die mit dem *message*-Parameter übergeben werden, handelt es sich einerseits um den Profilnamen und den Übereinstimmungsgrad der Profile, welche

im Dialogfenster angezeigt werden, andererseits handelt es sich um technische Informationen wie der Dateiname des Profils und die Angabe des Zielpfades, an dem die Profile persistent gemacht werden.

4.5 Datenhaltungskomponente

Die Klassen der Datenhaltungskomponente sind für die Persistenz der Daten verantwortlich, sie stellen Methoden für den lesenden, schreibenden und manipulierenden Zugriff auf die Profildaten zur Verfügung. Die Klassen der Datenhaltungskomponente wurden mit der javabasierte *JaxB* -Technologie, welche eine bidirektionale Abbildung zwischen Java-Objekten und XML-Dokumenten ermöglicht, generiert. Für die Erzeugung dieser Klassen ist ein *Schema Compiler* verantwortlich, welcher auf der Basis einer *DTD* und *XJS*-Datei arbeitet. In der nachfolgenden Abbildung 4.8 sind alle Klassen aufgezeigt, welche vom *Schema Compiler* erstellt worden sind, wobei die *String*-Klassen eine Ausnahme bilden und vom Generierungsprozeß ausgeschlossen gewesen sind. Die DTD-Datei definiert die Struktur einer XML-Datei und die XJS-Datei legt fest, wie die XML-Struktur auf *Java*-Klassen abgebildet wird. Auf die Implementierung der *XJS*-Datei und der *DTD*-Datei wird hier nicht eingegangen, sie sind aber auf der beigefügten CD dieser Diplomarbeit enthalten.

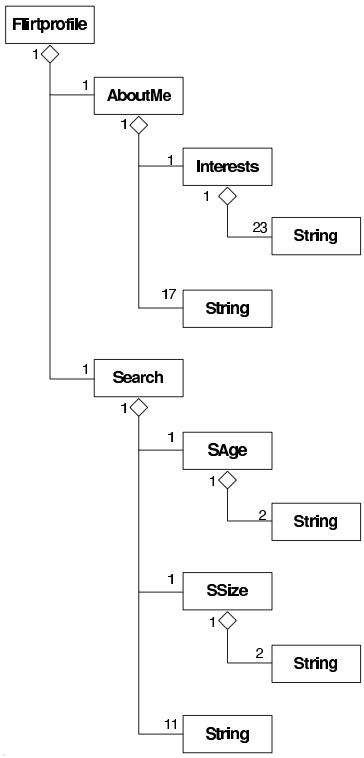

Abbildung 4.8: Aggregatstruktur der Klassen aus der Datenhaltungskomponente

Die in der obigen Abbildung 4.8 abgebildete Aggregatstruktur ist baumartig organisiert. Es gibt ein Wurzelobjekt, innere Objekte und es gibt Blätter, die *String*-Objekte darstellen. Die *String*-Objekte repräsentieren die eigentlichen Daten (Dabei müssen die Blätter nicht immer unbedingt *String*-Objekte darstellen. Die Objekte können vom unterschiedlichen Datentyp sein).

Eine Javainstanz der Klasse *Flirtprofile* repräsentiert die Profildaten eines Benutzers und besteht aus zwei Java-Objekten, nämlich dem *Aboutme*-Objekt und dem Search-Objekt. Das *Aboutme*-Objekt besteht aus 17 *String*-Objekten, welche die charakterisierenden Attribute der Person beschreiben, und aus einem *Interests*-Objekt, das sich wiederum aus 23 *String*-Objekten zusammensetzt und die Präferenzen des Benutzers beschreibt.
Das *Search*-Objekt besteht aus einem *SAge*-Objekt, einem *SSize*-Objekt und aus 11 *String*-Objekten, welche die Suchkriterien repräsentieren. Das *SAge*-Objekt setzt sich aus 2 *String*-Objekten, die einen Altersbereich angeben, zusammen. Das *SSize*-Objekt besteht ebenfalls aus 2 String-Objekten, die einen Grössenbereich in cm angeben. Für die Spezifikation der Profildaten wird auf den Abschnitt 1 auf Seite 50 verwiesen. Auf alle diese Objekte mit Ausnahme der elementaren Datentypen (z.B: int, float, usw.) kann man per *get*-Methoden zugreifen und sie per *set*-Methoden manipulieren.

Die Vererbungsstruktur der vom *Schema Compiler* erzeugten Klassen ist in der Abbildung 4.9 dargestellt. Wie man sieht, sind alle erzeugten Klassen entweder direkt oder indirekt von der Klasse *MarshallableObject* abgeleitet. Dies bedeutet, dass auf alle Objekte dieser Klassen der *unmarshall*- und der *marshall*-Prozess anwendbar ist.

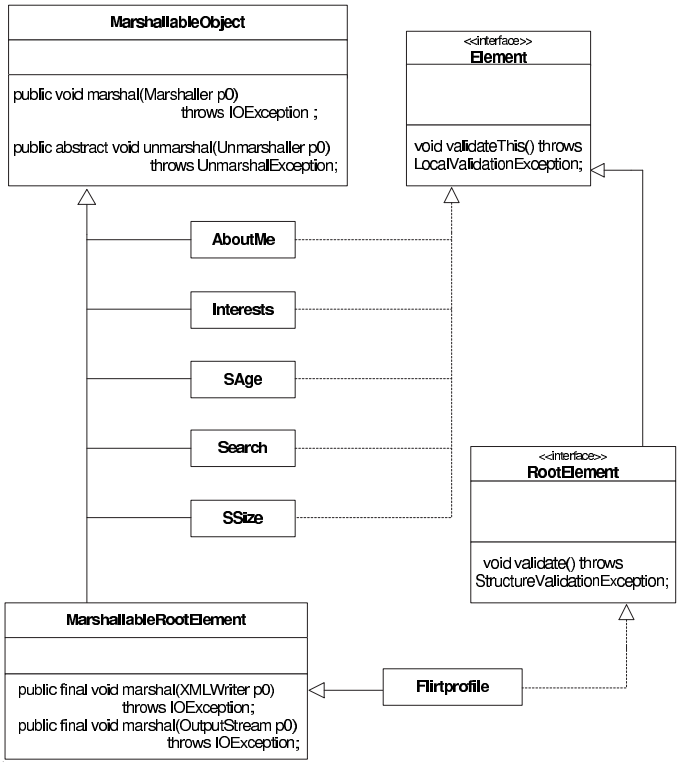

Abbildung 4.9: Vererbungsstruktur der Klassen aus der Datenhaltungskomponente

Ausserdem implementieren alle Klassen die Methode *validateThis()* aus dem Interface *Element*. Mit dieser Methode wird überprüft, ob die *Java*-Objekt-Repräsentation der *XML*-Daten mit der Definiton des XML-Schemas auf lokaler Ebene konform ist.
Die von der Klasse *Flirtprofile* implementierte *validate()*-Methode aus dem Interface *RootElement* überprüft hingegen, ob die *Java*-Objekt-Repräsentation mit der Definiton des XML-Schemas auf globaler Ebene konform ist.

4.6 Verarbeitungskomponente

Wie bereits im Abschnitt 3.5 ab Seite 57 erwähnt wurde, bildet die Klasse *ProfileManager* die einzige Klasse der Verarbeitungskomponente und wird in diesem Abschnitt um weitere Methoden und Attribute erweitert. In diesem Abschnitt soll nun die Implementierungskonzepte der wichtigsten Methoden der Klasse *ProfileManager* beschrieben werden. Die kompletten Details dieser Klasse sind direkt aus dem Quellcode der beigefügten CD dieser Diplomarbeit entnehmbar.

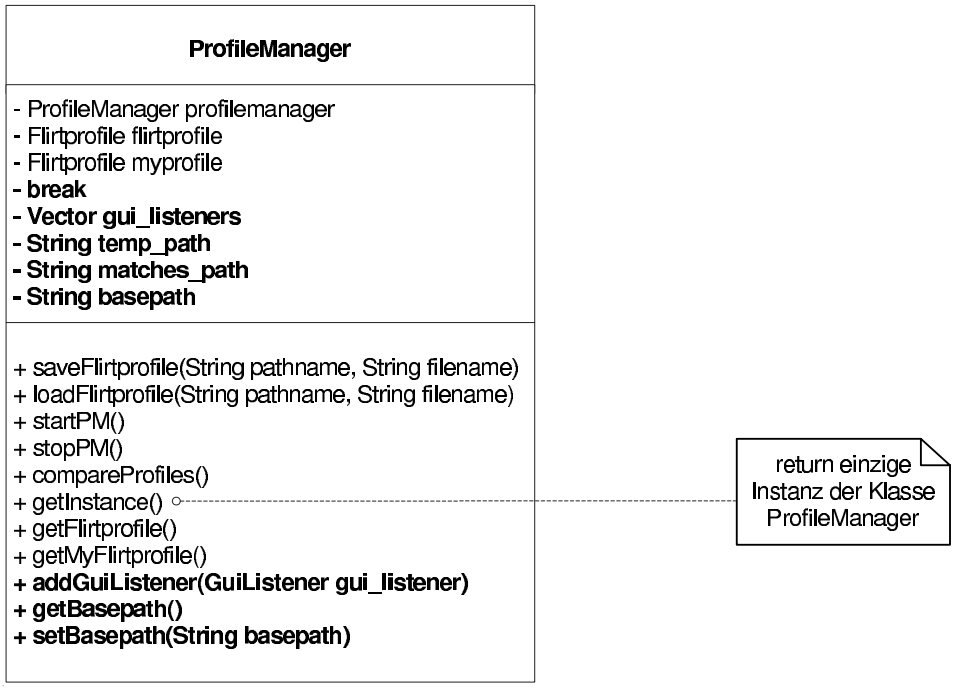

Abbildung 4.10: Die Klasse ProfileManager

Die fettgedruckten Methoden und Attribute der Klasse *ProfileManager* aus der Abbildung 4.10 repräsentieren die wichtigsten Erweiterungen, welche im Folgenden näher beschrieben werden:

addGuiListener(GuiListener gui_ listener)

Mit dieser Methode können ein oder mehrere Listener-Objekte[6], die das Interface *GuiListener* implementieren, registriert werden. Die GuiListener-Objekte werden von der Instanzvariable *gui_ listeners*, die vom Datentyp *Vector* ist, verwaltet. Sobald ein übereinstimmendes Profil gefunden wird, muss jedes registrierte GuiListener-Objekt über dieses Ereignis informiert werden, indem die Methode *matchFound(message)* der *GuiListener*-Objekte aufgerufen wird. Falls bei der Verarbeitungskomponente kein einziges *GuiListener*-Objekt registriert ist, dann wird einfach auch kein Objekt benachrichtigt. Mit diesem

[6]Zwar besteht die Möglichkeit, mehrere *GuiListener*-Objekte simultan zu registrieren, jedoch sollte es vermieden werden, da keine Änderungen im Datenmodell der Verarbeitungskomponente an die *GuiListener*-Objekte weitergegeben werden und es somit zu Inkonsistenzen führen kann. Es wird lediglich nur das Ereignis über ein neu gefundenes, übereinstimmendes Profil an alle *GuiListener*-Objekte mitgeteilt. Bei der Implementierung der *Flirtmaschine* wird nur ein einziges *GUIListener*-Objekt bei der Verarbeitungskomponente registriert. Besser wäre es gewesen, wenn man nur die Möglichkeit hätte, nur ein einziges *GUIListener*-Objekt bei der Verarbeitungskomponente zu registieren, um Verwirrungen zu vermeiden.

Verfahren wird erreicht, dass die Verarbeitungskomponente unabhängig von der Interaktionskomponente ist (siehe dazu die Erläuterung der *matchFound(Hashtable message)* auf Seite 67).

setBasepath(String basepath)

Diese Methode setzt das Basisverzeichnis. In der GUI gibt es unter dem Menüpunkt *Einstellungen* einen Button mit einem 3-Punkte Symbol. Durch einen Klick auf diesen Button wird die *setBasepath()*-Methode der Klasse *OptionPanel* aufgerufen. Diese Methode sorgt dafür, dass ein Dateibrowser geöffnet wird. Dort muss man die Datei *basepath.txt* auswählen, damit das Basisverzeichnis gesetzt wird. Sobald sich der Dateibrowser schliesst, wird die *setBasepath(String basepath)*-Methode der Klasse *ProfileManager* aufgerufen.

Die *Verarbeitungskomponente* speichert die Profile, die aus einer XML-Datei *profilname.xml* und einem Bild *pic.jpg* bestehen, in einem Ordner, das denselben Namen der XML-Datei trägt, ab. Wie sich der Inhalt dieses Ordners aufbaut, ist in der folgenden Abbildung 4.11 dargestellt:

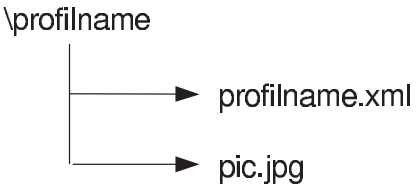

Abbildung 4.11: Inhalt des Profilordners

Die Orte, an denen die Profile gespeichert werden, liegen relativ zum Basisverzeichnis. Das Basisverzeichnis ist immer der Ordner *profilematcher*. Die Struktur des Ordners *profilematcher* ist in der folgenden Abbildung 4.12 dargestellt:

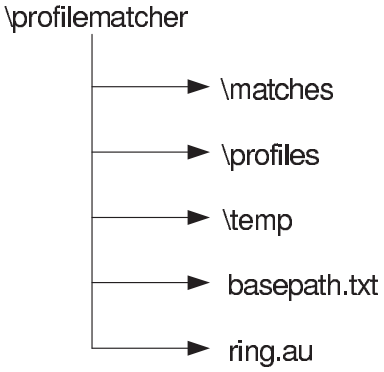

Abbildung 4.12: Die Struktur des Ordners *profilematcher*

Das Fileschema aus Abbildung 4.12 wird im Folgenden erläutert:

- *ring.au*
  Diese Datei repräsentiert das akkustische Signal.
- Ordner *temp*
  Dieser Ordner stellt das Arbeitsverzeichnis für die Verarbeitungskomponente dar. Hier speichert die Kommunikationskomponente im aktivierten Zustand alle Profile, die es in der Funkumgebung findet. Sobald die *Flirtmaschine* aktiviert ist, prüft die Verarbeitungskomponente kontinuierlich den *temp*-Ordner

auf neue Profile. Unabhängig davon ob das Profil mit dem eigenem übereinstimmt, wird das Profil aus dem *temp*-Ordner nach der Überprüfung immer gelöscht. Der *temp*-Ordner wird durch die Instanzvariable *temp_ path* repräsentiert.

- Ordner *matches*
  Dieser Ordner beinhaltet alle übereinstimmende Profile. Sobald die Verarbeitungskomponente aus dem *temp*-Ordner ein übereinstimmendes Profil findet, wird das Profil in den *matches*-Ordner umkopiert. Der *matches*-Ordner wird durch die Instanzvariable *matches_ path* repräsentiert.
- Ordner *profiles*
  Dieser Ordner ist zum Speichern diverser Profile vorgesehen. Ausserdem muss dieser Ordner das eigene Profil mit dem Namen *myprofile* beinhalten, damit die Verarbeitungskomponente auf das eigene Profil zugreifen kann.

getBasepath()

Diese Methode liefert als Rückgabewert das Basisverzeichnis.

startPM()

Die Methode *startPM()* wird durch das folgende Aktivitätsdiagramm in der Abbildung 4.13 beschrieben. Die Methode *startPM()* muss in einem eigenen Thread ablaufen, damit sie jederzeit mit der Methode *stopPM()* unterbrochen werden kann.

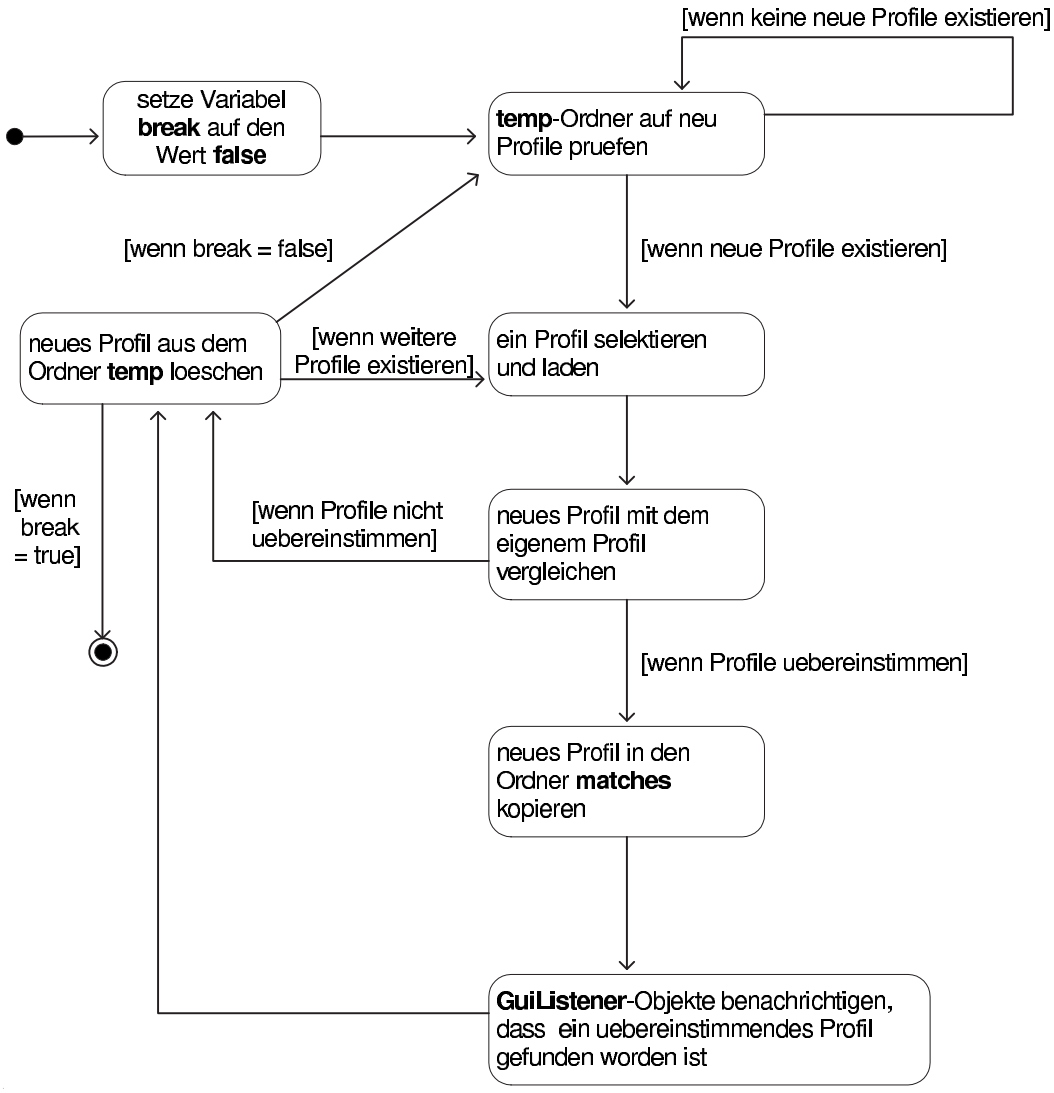

Abbildung 4.13: Aktivitätsdiagramm der Methode startPM()

stopPM()

Diese Methode *stopPM()* unterbricht die Methode *startPM()*, indem sie die Instanzvariable *break* auf den Wert *true* setzt.

saveFlirtprofile (String Pathname, String filename)

Hier wird nicht nur die Methode *saveFlirtprofile (String Pathname, String filename)*, die in der Abbildung 4.14 mit der Nummer 10 gekennzeichnet ist, erläutert, sondern es soll auch die Schnittstelle für das Speichern von Profilen zwischen den Komponenten für die Verarbeitung, Interaktion und Datenhaltung deutlich gemacht werden. Damit das Sequenzdiagramm einigermassen übersichtlich und verständlich bleibt, sind nur die wesentlichen Aspekte für den Speichervorgang dargestellt worden. Diese Methoden wurden mit ihren Signaturen später in den Quellcode übernommen. Für die Details wird auf den Quellcode der beigefügten CD dieser Diplomarbeit verwiesen.

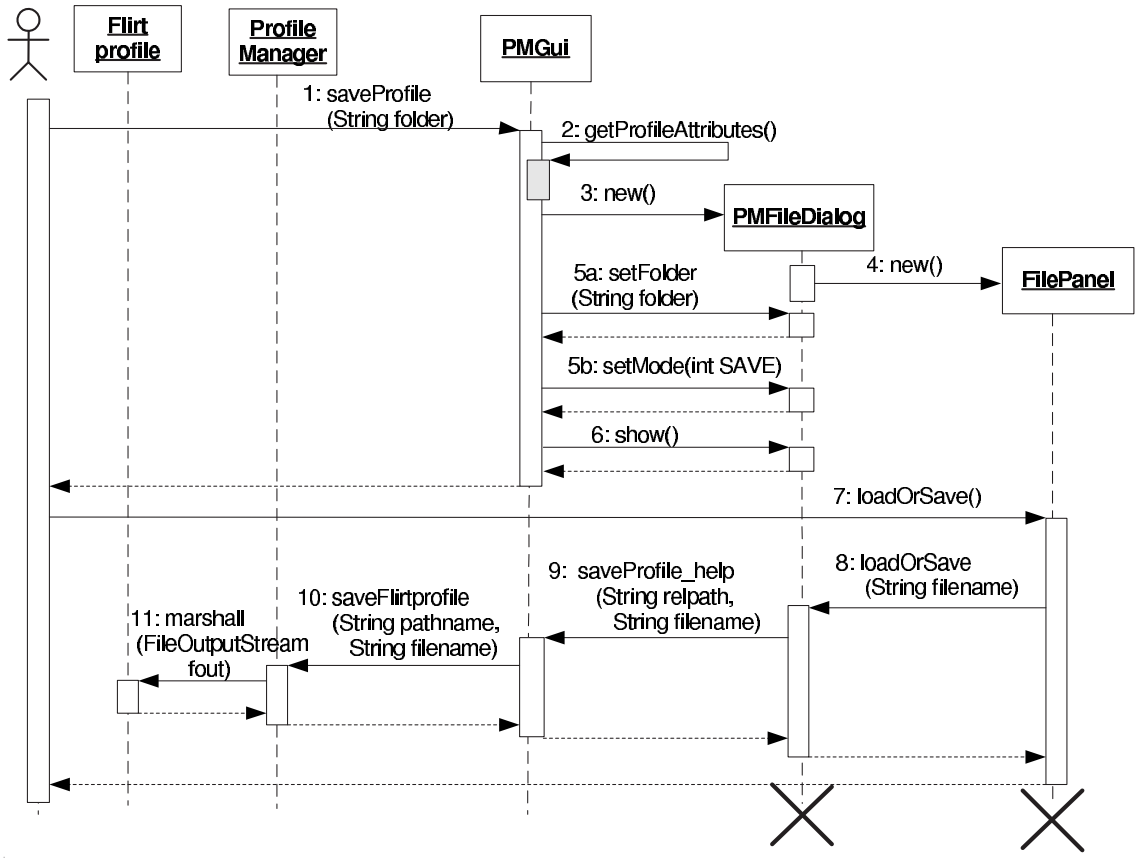

Abbildung 4.14: Sequenzdiagramm: Speichervorgang von Profilen

- 1: *saveProfile (String folder)*
  Der Benutzer leitet das Speichern eines Profils ein, indem er im Menü *File* entweder den Menüpunkt *Profil speichern* oder *Matches speichern* anwählt. Der Parameter *folder* gibt den Speicherort des Profils an und nimmt je nachdem, welchen Menüpunkt man gewählt hat, entweder den Wert *'matches'* oder *'profiles'* an.

- 2: *getProfileAttributes()*
  Die Methode *getProfileAttributes()* erzeugt eine leere Instanz[7] der Klasse *Flirtprofile* und holt sich die Profildaten aus der GUI, die über *set*-Methoden in die Instanz übertragen werden.

[7]Diese Instanz repräsentiert ein komplettes Profil. Die Klasse *Flirtprofile* gehört zur Datenhaltungskomponente. Mehr Informationen dazu findet man im Abschnitt 4.5 auf Seite 68.

- 3+4+5a+5b+6: *new(), new(), setFolder(String folder) setMode (SAVE), show()*
  Diese Methoden erzeugen ein Dateiendialogfeld zum Speichern von Profilen. Der Benutzer kann über diesen Dialog dem zu speichernden Profil einen Dateinamen zuweisen und abspeichern. Die *SAVE*-Konstante ist dafür verantwortlich, dass ein Dialog für das Speichern von Profilen erzeugt wird. Der Parameter *folder* gibt an, in welchem Ordner das Profil abgespeichert wird. Dieser Parameter ist vom Typ *String* und kann entweder den Wert *'matches'* oder *'folder'* annehmen.

- 7: *loadOrSave()*
  Klickt der Benutzer den Button *Speichern* des Dialogfeldes an, dann wird die Methode *loadOrSave()* abgefeuert. Diese Methode delegiert durch den Methodenaufruf *loadOrSave (String filename)* den Speichervorgang an die Klasse *PMFileDialog* weiter.

- 8: *loadOrSave (String filename)*
  Diese Methode delegiert durch den Methodenaufruf *saveProfile_help (String relpath, String filename)* den Lade- oder Speichervorgang an die Klasse *PMGui*. Da in diesem Fall der Speichermodus eingestellt ist, wird das Profil gespeichert. Der Parameter *filename* der Methode *loadOrSave(...)* gibt den Namen des Profilordners und der Profildatei (ohne Dateiendung) an. Aus der Instanzvariable *folder* und aus dem Parameter *filename* wird der Parameter *relpath* zusammengesetzt. Der Parameter *filename* der Methode *saveProfile_help(...)* wird um die Dateiendung *'*.xml'* erweitert.

- 9: *saveProfile _help (String relpath, String filename)*
  Der Parameter *relpath* wird benutzt, um den kompletten Verzeichnispfad zu konstruieren. Diese Methode delegiert den Speichervorgang durch den Methodenaufruf *saveFlirtprofile (String pathname, String filename)* an die Klasse *ProfileManager* weiter, wobei der Parameter *pathname* dem kompletten Verzeichnispfad entspricht. In dieses Verzeichnis wird das Profil gespeichert.

- 10: *saveFlirtprofile (String pathname, String filename)*
  Diese Methode ruft den Marshall-Prozess auf.

- 11: *marshall(FileOutputStream fout)*
  Diese Methode führt den Marshall-Prozess durch. Der Marshall-Prozess sorgt dafür, dass die Java-Instanz *Flirtprofile* (siehe Nummer 2) in eine XML-Datei geschrieben wird.

loadFlirtprofile (String Pathname, String filename)

An dieser Stelle wird die Methode *loadFlirtprofile (String Pathname, String filename)*, die in der Abbildung 4.15 mit der Nummer 10 gekennzeichnet ist, erläutert sowie die Schnittstelle für das Laden von Profilen zwischen den Komponenten für die Verarbeitung, Interaktion und Datenhaltung deutlich gemacht. Damit das Sequenzdiagramm einigermassen übersichtlich und verständlich bleibt, sind nur die wesentlichen Aspekte für den Ladevorgang dargestellt worden. Diese Methoden wurden mit ihren Signaturen später in den Quellcode auf der beigefügten CD dieser Diplomarbeit übernommen.

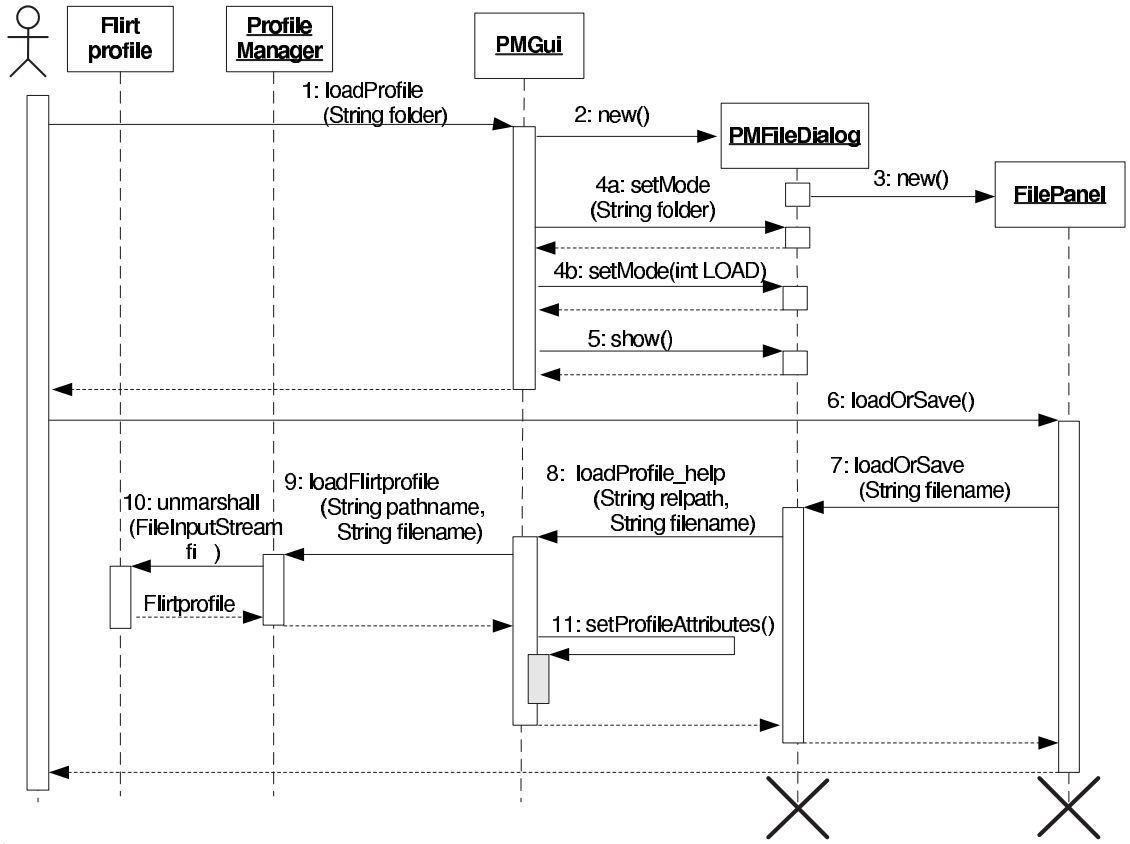

Abbildung 4.15: Sequenzdiagramm: Ladevorgang von Profilen

- 1: *loadProfile (String folder)*
  Der Benutzer leitet das Laden eines Profils ein, indem er im Menü *File* entweder den Menüpunkt *Profil laden* oder *Matches laden* anwählt. Der Parameter *folder* gibt den Ladeort des Profils an und nimmt je nachdem welchen Menüpunkt man gewählt hat, entweder den Wert *'matches'* oder *'profiles'* an.

- 2+3+4a+4b+5: *new(), new(), setFolder(String folder), setMode (LOAD), show()*
  Diese Methoden erzeugen ein Dateiendialogfeld zum Laden von Profilen. Der Benutzer kann über diesen Dialog den Namen des Profils angeben, das geladen werden soll. Die *LOAD*-Konstante ist dafür verantwortlich, dass ein Dialog für das Laden von Profilen erzeugt wird. Der Parameter *folder* gibt an, von welchem Ort das Profil geladen wird. Dieser Parameter ist vom Typ *String* und kann entweder den Wert *'matches'* oder *'folder'* annehmen.

- 6: *loadOrSave()*
  Klickt der Benutzer den Button *Laden* des Dialogfeldes an, dann wird die Methode *loadOrSave()* abgefeuert. Diese Methode delegiert durch den Methodenaufruf *loadOrSave (String filename)* den Ladevorgang an die Klasse *PMFileDialog* weiter.

- 7: *loadOrSave (String filename)*
  Diese Methode delegiert durch den Methodenaufruf *loadProfile_help (String relpath, String filename)* den Lade- oder Speichervorgang an die Klasse *PMGui* weiter. Da in diesem Fall der Lademodus eingestellt ist, wird das Profil geladen. Der Parameter *filename* der Methode *loadOrSave(. . .)* gibt den Namen des Profilordners und der Profildatei (ohne Dateiendung) an. Aus der Instanzvariable *folder* und aus dem Parameter *filename* wird der Parameter *relpath* zusammengesetzt. Der Parameter *filename* der Methode *loadProfile_help()* wird um die Dateiendung *'*.xml'* erweitert.

- 8: *loadProfile _help (String relpath, String filename)*
  Der Parameter *relpath* wird benutzt, um den kompletten Verzeichnispfad zu konstruieren. Diese Methode delegiert den Ladevorgang durch den Methodenaufruf *loadFlirtprofile (String pathname, String filename)* an die Klasse *ProfileManager* weiter, wobei der Parameter *pathname* dem kompletten Verzeichnispfad entspricht. Aus diesem Verzeichnis wird das Profil geladen.

- 9: *loadFlirtprofile (String pathname, String filename)*
  Diese Methode ruft den Unmarshall-Prozess auf.

- 10: *unmarshall(FileInputStream fi)*
  Diese Methode führt den Unmarshall-Prozess durch. Der Unmarshall-Prozess erzeugt aus der angegebenen Profildatei, die eine XML-Datei ist, eine Javainstanz der Klasse *Flirtprofile*.

- 11: *setProfileAttributes()*
  Die Methode *setProfileAttributes()* liest die Werte der Javainstanz der Klasse *Flirtprofile* über *get*-Methoden aus und setzt die Profildaten entsprechend in der GUI.

compareProfiles()

Diese Methode vergleicht das eigene Profil mit dem externen Profil und ermittelt einen Übereinstimmungsgrad. Um so höher dieser Übereinstimmungsgrad ist, desto ähnlicher sind die Profile zueinander. Falls beide Profile eine ausreichende Übereinstimmung erzeugen, liefert diese Methode den boolschen Wert *true*, ansonsten *false*. Die Vergleichsmethode arbeitet folgendermassen:

- Die Suchkriterien des einen Profils müssen mit den Profildaten des anderen Profils exakt übereinstimmen, ansonsten liefert die Vergleichsmethode *false*.
- Die *compareProfiles()*-Methode bildet aus den gewählten Interessen des eigenen Profils eine Menge *A* und aus den gewählten Interessen des externen Profils eine Menge *B*. Diese beide Mengen vereint bilden die Menge *C*, die allen gewählten Interessen beider Profile entspricht. Es wird eine Anzahl *n* von Interessen ermittelt, die sowohl in der Menge *A* als auch in der Menge *B* vorkommen. Der Übereinstimmungsgrad *u* beider Profile wird jetzt folgendermassen ermittelt:

$$n = |A \cap B| \tag{4.1}$$

$$u = \frac{n}{(|A| \cup |B|)} = \frac{n}{|C|} \tag{4.2}$$

  Beide Profile besitzen jeweils die Information *Übereinstimmungswert*[8] im Bereich der Suchkriterien. Der Übereinstimmungsgrad, auch als Match-Wert bezeichnet, muss das Maximum der beiden Übereinstimmungswerte erreichen, damit die Vergleichsmethode *compareProfiles()* positiv verläuft, also den Wert *true* liefert. Der **Vorteil** dieses Verfahren ist, dass der berechnete Übereinstimmungsgrad (Match-Wert) auf beiden Seiten gleich ist. Das heisst, es gibt keine Widersprüchlichkeiten in den Treffermeldungen. Dies erspart jeweils einen zusätzlichen Nachrichtenaustausch zwischen den beiden Parteien.

Beispiel

Man stelle sich vor, dass die Anzahl *n* nicht angibt, wieviele Elemente die Menge *A* und *B* gemeinsam haben, sondern dass die Anzahl *n* angibt, wie oft die Elemente aus *A* bzw. aus *B* in der Menge *C* vorkommen. Dann könnte es dazu kommen, dass auf beiden Seiten ein unterschiedlicher Übereinstimmungsgrad *u* berechnet wird, da auf beide Seiten ein unterschiedlicher *n*-Wert möglich wäre . Damit sind paradoxe Treffermeldungen möglich, d.h., eine Seite meldet einen Treffer während die andere Seite es nicht tut. Um diese Widersprüchlichkeit zu vermeiden, müssten jeweils beide Seiten der anderen ihr Vergleichsergebnis mitteilen und erst wenn beide Ergebnisse positiv sind, dürfen beide Seiten einen Treffer melden. Dies erfordert jeweils einen zusätzlichen Nachrichtenaustausch zwischen den beiden Parteien.

[8] Über das Setzen des Übereinstimmungswertes in den Suchkriterien kann der Benutzer vorgeben, ab welchem erreichten Match-Wert ein Profil als Treffer gewertet wird. Der Match-Wert ist der für beide Profile berechnete Übereinstimmungsgrad (siehe unter dem Punkt 1a auf Seite 50).

Ein **Nachteil** dieses Verfahrens ist, dass bei nicht exakter Übereinstimmung der eigenen Suchkriterien mit den Profildaten des externen Profils die Vergleichsmethode rigoros den Wert *false* liefert, d.h., kleine Abweichungen werden nicht geduldet.

Beispiel

Ein Flirtpartner sucht jemanden im Grössenbereich von 170 cm und 180 cm. Ist der andere Partner aber tatsächlich nur 169 cm groß, dann wird dieser Partner nicht als Treffer gemeldet, obwohl das Grössenkriterium doch so gut wie erfüllt worden ist.

Dieser Nachteil kann mit der Fuzzy-Logik [Heinsohn und Socher-Ambrosius (1999)] beseitigt werden. Ich habe es leider nicht mehr geschafft, die Fuzzy-Logik in der Implementierung umzusetzen, da ich sonst Schwierigkeiten bekommen hätte, meine Diplomarbeit termingrecht fertigzustellen. Stattdessen soll an dieser Stelle die Fuzzy-Logik anhand eines konkreten Beispiels erläutert werden.

Bei der Fuzzy Logik werden Elemente einer Menge X auf einem Wert zwischen 0 und 1 abgebildet. Dieser Wert gibt den Grad r der Zugehörigkeit der Elemente zu einem vagen Begriff F an. So eine Abbildung, auch Zugehörigkeitsfunktion genannt, sieht folgendermassen aus:

- $\mu_F : X \longrightarrow [0,1]$
- $\mu_F(x) = r$
  - r = 1, x $\in$ X wird zu 100% dem Begriff F zugeordnet.
  - r = 0, x $\in$ X wird nicht dem Begriff F zugeordnet.
  - 0 < r < 1, x $\in$ X wird mit dem Grad r dem Begriff F zugeordnet.

Die Fuzzy-Logik kann beim Vergleich von Attributen zweier Profile eingesetzt werden. Es soll zur Verdeutlichung ein konkretes Beispiel angeführt werden, hierbei wählen wir das Attribut *Körpergrösse*.

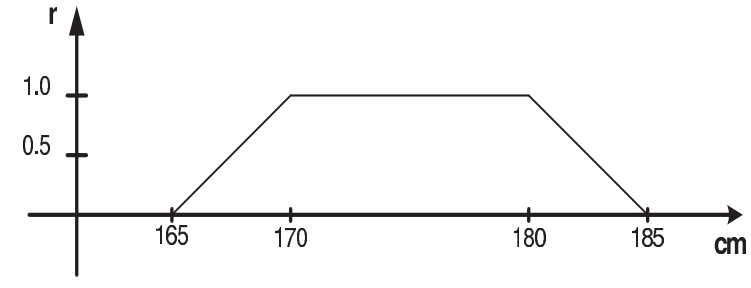

Abbildung 4.16: Zugehörigkeitsfunktion des Begriffes Übereinstimmung

1. Der Wertebereich des Attributes *Körpergrösse* wird von der Menge X repräsentiert. Die Metrik für die Elemente der Menge X wird auf cm festgelegt.

   X = {100,101,102, . . . ,220 }

2. Für den vagen Begriff $F = \textit{Übereinstimmung}$ wird folgende Zugehörigkeitsfunktion bestimmt (Grafische Darstellung in Abbildung 4.16):

$$\mu_{\textit{Übereinstimmung}}(x) = \begin{cases} 0 & \text{mit} \quad 0 \leq x < 165 \\ \frac{1}{5}x - 33 & \text{mit } 165 \leq x < 170 \\ 1 & \text{mit } 170 \leq x < 180 \\ -\frac{1}{5}x + 37 & \text{mit } 180 \leq x < 185 \\ 0 & \text{mit } 185 \leq x \leq 220 \end{cases}$$

   - $r = 1$, die Körpergrösse stimmt überein.
   - $r = 0$, die Körpergrösse stimmt nicht überein.
   - $0 < r < 1$, die Körpergrösse stimmt mit dem Grad r überein. Dabei soll hier festgelegt werden, dass bei $0.5 \leq r \leq 1$ eine Übereinstimmung vorliegt.

3. Es wird angenommen, dass jemand einen Partner sucht, der im Grössenbereich von 170 cm bis 180 cm liegt. Laut der oben angegebenen Zugehörigkeitsfunktion liegt eine Übereinstimmung im Grössenbereich zwischen 168 und 182 cm vor, da stets $r \geq 0.5$ ist. Das heisst, das Suchkriterium für die Körpergrösse des eigenen Profils muss nicht exakt mit dem Attribut *Körpergrösse* des externen Profils übereinstimmen. Mit anderen Worten, kleine Abweichungen sind jetzt erlaubt.

4.7 Kommunikationskomponente

4.7.1 Das Konzept auf der Protokollebene

In diesem Abschnitt wird ein Konzept für die Kommunikation auf der Protokollebene vorgestellt. Die Idee bei diesem Konzept ist, einen Ordner auf dem Servergerät für alle Clientgeräte freizugeben, damit diese über Funk auf die Profildateien des Servergerätes zugreifen können. Der oberste Punkt der Ordnerhierarchie ist das Rootverzeichnis. In dieses Rootverzeichnis wird der Ordner *profilematcher*[9] mit seiner Ordner- und Filestruktur abgelegt. Im Nachfolgenden wird in der Abbildung 4.17 der Protokollstack für die *Flirtmaschine* dargestellt, und es werden die Klassen der Kommunikationskomponente und deren Methoden konzeptionell erläutert:

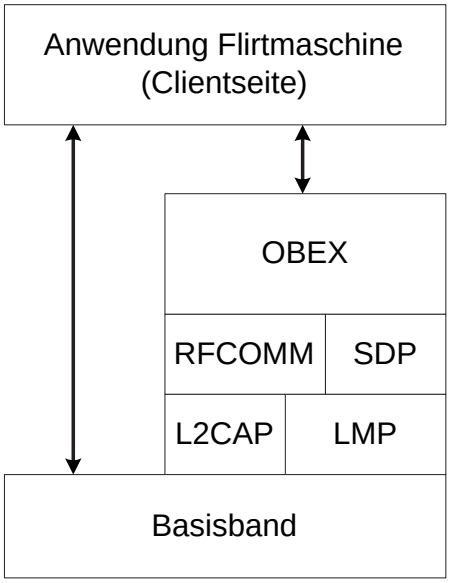

Abbildung 4.17: Der Protokollstack für die *Flirtmaschine*

PMServer

Die im Abschnitt 3.4 vorgestellte Klasse *PMServer* wird nicht mehr benötigt, da die Bluetooth-Geräte, die das *File Transfer Profile* unterstützen, selbst einen eigenen Server für den Transfer von Dateien implementieren.

PMClient

Die Klasse *PMClient* fordert die Flirtprofile von entfernten Geräten an. Geben diese der Anforderung statt, werden die Flirtprofile über Funk vom Server zum Client übertragen. Diese Klasse beinhaltet die zwei Methoden:

- *stop()*
  Diese Methode unterbricht den durch die Methode *start()* ausgeführten Suchvorgang, indem es die Abbruchbedingung erfüllt. Diese Abbruchbedingung könnte zum Beispiel aus einer boolschen Variable *break* bestehen, die für das Abbrechen

[9] Der Ordner *profilematcher* stellt das Basisverzeichnis dar (mehr dazu siehe auch auf Seite 71).

der Methode *start()* von der Methode *stop()* auf den Wert *true* gesetzt werden muss. Damit die Methode *stop()* den Fokus der Programmsteuerung erhalten kann, muss die Methode *start()* in einem eigenen Thread ablaufen. Ausserdem wird noch das Attribut *Running* aus der Datei *status.xml*[10] auf den Wert *no* gesetzt, damit die *Flirtmaschine* als inaktiv gekennzeichnet ist. Es wäre empfehlenswert, die Methode *stop()* in der Methode *stopPM()* der Klasse *ProfilManager* unterzubringen. Sobald keine Profile mehr über Funk übertragen werden, macht es nämlich keinen Sinn mehr, das Verzeichnis *temp*[11] auf neu eingegangene Profile zu prüfen.

- *start()*
  Diese Methode sucht kontinuierlich in der Funkumgebung nach neuen Geräten. Falls diese Geräte eine aktivierte Flirtmaschine ausführen, wird das Flirtprofil über Funk angefordert und in den Ordner *temp* für die weitere Verarbeitung gespeichert. Es wäre ratsam, die Methode *start()* als erste Aktion in der Methode *startPM* der Klasse *ProfileManager* unterzubringen, denn es macht Sinn, sobald die Profildateien über Funk übertragen werden, das Verzeichnis *temp* auf neu eingegangene Profile zu prüfen. Das in der Abbildung 4.18 dargestellte Aktivitätsdiagramm veranschaulicht die einzelnen Vorgänge in der Methode *start()*.

[10] Die Datei *status.xml* beinhaltet die Information, ob eine Flirtmaschine aktiv oder inaktiv ist. Eine Erläuterung zu der Datei *status.xml* findet man in der Aktivität 1 auf Seite 84.

[11] Alle über Funk übertragenen Profile werden erstmals zur Verarbeitung in den Ordner *temp* abgelegt. Für weitere Informationen zu dem Ordner *temp* wird auf die Seite 71 verwiesen.

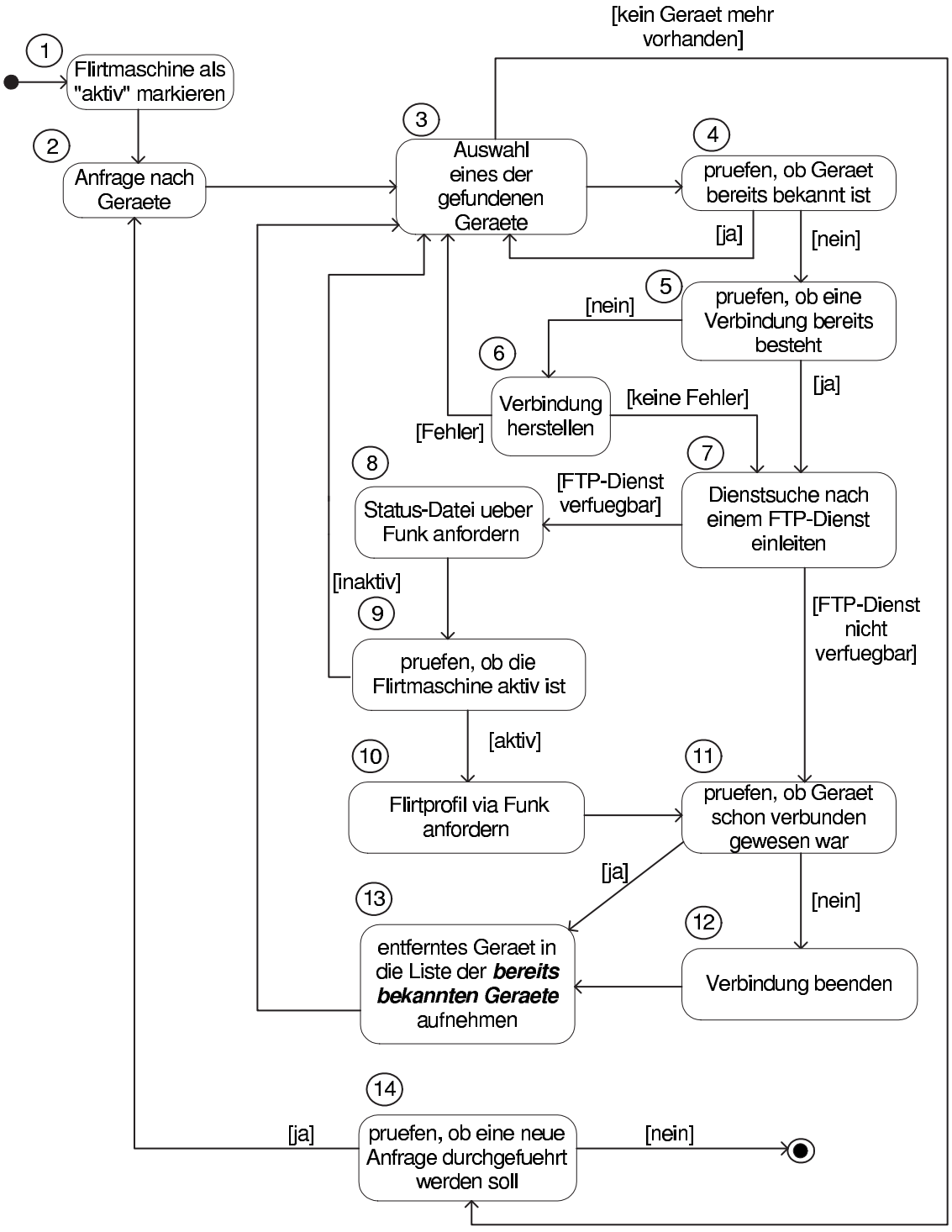

Abbildung 4.18: Die Methode *start()* der Klasse *PMCLient*

Das Aktivitätsdiagramm aus der Abbildung 4.18 wird an dieser Stelle im Einzelnen erläutert:

- *Aktivität 1*
  Diese Aktivität kennzeichnet die *Flirtmaschine* als aktiv (siehe Seite 52). Die *Flirtmaschine* wird erst dann als aktiv markiert, wenn sich das Gerät im Modus *discoverable mode* und im Modus *connectable mode* befindet. Der Benutzer selbst versetzt das Gerät in diese Modi. Man könnte die Information, ob eine *Flirtmaschine* aktiv oder inaktiv ist, in der nachfolgenden Datei *status.xml* unterbringen:

```
1   <status>
2      <running>yes</running>
3   </status>
```

Abbildung 4.19: Die Datei *status.xml*

  Das *status*-Tag enthält momentan nur das *running*-Tag. Beinhaltet das *running*-Tag den Wert *yes*, dann ist die *Flirtmaschine* aktiv, beim Wert *no* hingegen inaktiv. Für spätere eventuelle Erweiterungen können im *status*-Tag noch weitere Tags angegeben werden, welche Statusinformationen beschreiben. Die Datei *status.xml* könnte man direkt unter dem Ordner *profilematcher* anlegen. Anschliessend wird zur Aktivität 2 übergegangen.

- *Aktivität 2*

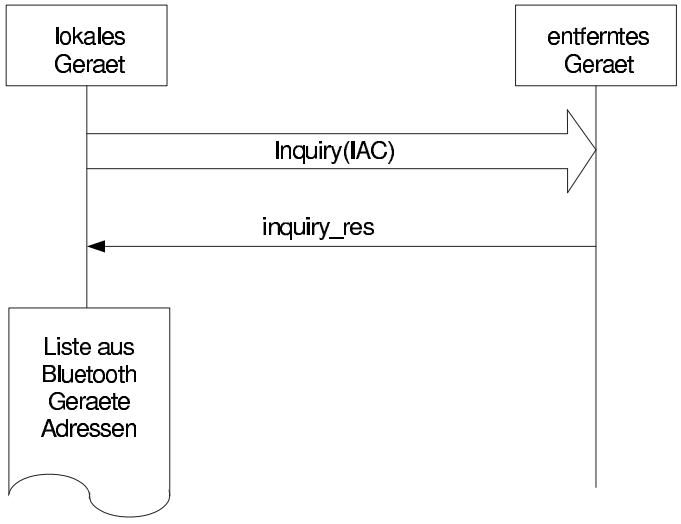

Abbildung 4.20: Die Inquiry Prozedur

  Diese Aktivität initiiert eine Geräteanfrage, indem sie die Inquiry-Prozedur der *Link Controller*-Schicht aufruft. Die Inquiry-Prozedur sendet einen *IAC*-Signal (Inquiry Acess Code; maximal 72 Bit) aus. Alle entfernten Geräte, die sich im Modus *discoverable mode* befinden, suchen im Funkraum nach diesem IAC-Signal. Sobald sie ein IAC-Signal finden, antworten sie mit einer Responsenachricht *inquiry_res*. Diese Responsenachricht beinhaltet unter anderem die Bluetooth-Geräteadresse, die für das Paging-Verfahren benötigt wird. Es wird mit der Aktivität 3 fortgefahren.

- *Aktivität 3*
  Falls die Anfrage keine Geräte in der Umgebung ermitteln konnte oder alle gefundenen Geräte bereits abgearbeitet worden sind, dann wird zur Aktivität 14 gewechselt. Konnte die Anfrage aus der Umgebung jedoch Geräte ermitteln, dann wird ein aus der Anfrage gefundenes Gerät ausgewählt und bei der Aktivität 4 fortgesetzt.

- *Aktivität 4*
  Die Bluetooth-Adressen aller Geräte, die seit der ersten Aktivierung der *Flirtmaschine* gefunden worden sind, werden in einer dynamischen Liste registriert (Aktivität 13). Diese Liste wird hier als *Liste der bereits bekannten Geräte* bezeichnet und verhindert, dass Geräte mehrfach geprüft werden. Sie könnte in einer Instanzvariable der Klasse *PMClient* festgehalten werden. Die Aktivität 4 prüft nun anhand dieser Liste ob die jeweilige Bluetooth-Geräteadresse in ihr bereits enthalten ist. Bei positivem Ergebnis wären weitere Schritte überflüssig und der Suchvorgang setzt zur Aktivität 3 zurück. Ist die Adresse nicht in dieser Liste enthalten, wird mit Aktivität 5 fortgesetzt.

- *Aktivität 5*
  Diese Aktivität überprüft, ob zum ausgewählten Gerät bereits eine logische Verbindung besteht. Ist dies der Fall, wird zur Aktivität 7 gewechselt. Besteht keine logische Verbindung, wird zur Aktivität 6 übergegangen.

- *Aktivität 6*
  Diese Aktivität ist verantwortlich für den Aufbau einer logischen Verbindung zum entfernten Gerät. Die Abbildung 4.21 veranschaulicht die Schritte, die für diesen Aufbau nötig sind:

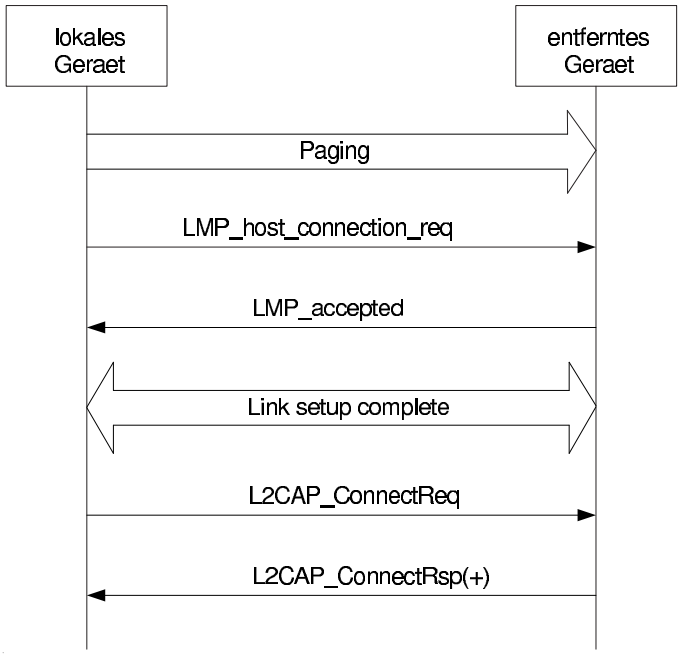

Abbildung 4.21: Physikalischer und logischer Verbindungsaufbau

Im ersten Schritt muss ein *Paging Verfahren*[12] eingeleitet werden, um eine physikalische Verbindung zum entfernten Gerät aufbauen zu können. Nach der erfolg-

[12]Das *Paging Verfahren* wird auf Seite 27 erläutert.

reichen Durchführung eines *Paging Verfahrens* besteht zunächst nur eine physikalische Leitung, über die eine Verbindungsanforderung *LMP_host_connection_req* an das entfernte Gerät gesendet werden muss. Ist dieses mit dem Verbindungswunsch einverstanden, wird die PDU *LMP_accepted* empfangen, ansonsten erhält man die PDU *LMP_not_accepted*. Sobald beide Geräte keine weiteren Prozeduren für die physikalische Verbindungseinrichtung erwarten, senden sie jeweils die Nachricht *LMP_setup_complete*. Jetzt ist der physikalische Verbindungsaufbau komplett und erfolgreich durchgeführt worden. Nachdem die physikalische Verbindung steht, muss nun eine logische Verbindung aufgebaut werden. Es muss die Nachricht *L2CAP_ConnectReq* dem entfernten Gerät gesendet werden, um einen Verbindungswunsch auf logischer Ebene anzukündigen.
Ist das entfernte Gerät mit diesem Verbindungswunsch einverstanden, wird die positive Nachricht *L2CAP_ConnectRsp(+)* empfangen, ansonsten die negative *L2CAP_ConnectRsp(-)*. Besteht bereits eine physikalische Verbindung, müssen nur die letzten beiden Schritte durchgeführt werden. Die Paging-Prozedur wird von der *Link Controller*-Schicht zur Verfügung gestellt. Die *LMP*-Nachrichten werden über die *Link Manager*-Schicht gesendet bzw. empfangen, die *L2CAP*-Nachrichten über die *L2CAP*-Schicht. Sollte wegen auftretender Fehler kein Verbindungsaufbau zustandekommen, wird bei der Aktivität 3 fortgefahren, ansonsten bei der Aktivität 7.

- *Aktivität 7*

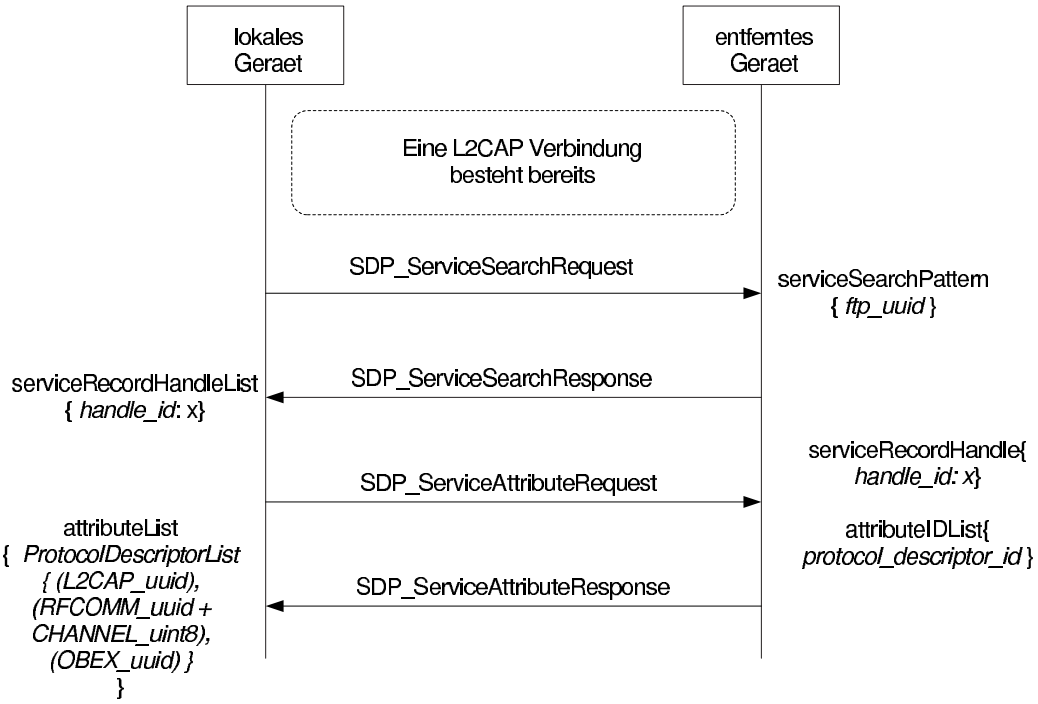

Abbildung 4.22: SDP Transaktionen

Diese Aktivität führt eine Dienstsuche durch und prüft, ob ein *FTP*-Dienst auf dem verbundenen Gerät existiert. Falls ein *FTP*-Dienst unterstützt wird, wird die *RFCOMM-Server-Kanalnummer*, über den die *FTP*-Daten gesendet werden, ermittelt. Die obige Abbildung 4.22 veranschaulicht die notwendigen *SDP*-Transaktionen hierfür.
Zuerst muss eine Dienstanfrage gestellt werden, indem man eine PDU *ServiceSearchRequest* an das verbundene Gerät sendet. Die PDU *ServiceSearchRequest* beinhaltet ein Parameter *serviceSearchPattern*, der die UUID *ftp_uuid* für den *FTP*-Dienst enthält. Die Serverseite prüft für jeden Datensatz, ob in dessen Attribut *ServiceClassIDList* die UUID *ftp_uuid* enthalten ist. Ist die UUID *ftp_uuid* in einem Dienstdatensatz enthalten, wird der Handle dieses Datensatzes im Parameter *serviceRecordHandleList* aufgenommen. Hier in diesem Beispiel hat der aufgenommene Handle den Wert *x*. Die Serverseite verpackt den Parameter *serviceRecordHandleList* in eine Antwort-PDU *SDP_ServiceSearchResponse* und sendet sie dem Client, welcher sie empfängt und auswertet. Enthält die Antwort keinen Handle-Eintrag, wird auf dem verbundenen Gerät kein *FTP*-Dienst unterstützt, die Dienstsuche abgebrochen und bei der Aktivität 11 fortgefahren. Sollte jedoch ein Handle-Eintrag vorhanden sein, wird der *FTP*-Dienst unterstützt und anschliessend eine Attributanfrage für den Datensatz mit dem Handle *x* durchgeführt. Dazu muss eine PDU *SDP_ServiceAttributeRequest*, die als Parameter den Handle *x* und eine AttributID-Liste enthält, an den Server gesendet werden. Die AttributID-Liste beinhaltet in diesem Fall nur die ID für das Attribut *ProtocolDescriptorList*, für welches man den dazugehörigen Attributwert erhalten möchte. Der Server wertet diese At-

tributanfrage aus und erzeugt eine Antwortnachricht *SDP_ServiceAttributeResponse*, welche an den Client gesendet wird. Diese Antwortnachricht enthält unter anderem den Parameter *AttributeList*, welcher den Attributwert für das Attribut *ProtocolDescriptorList* kapselt. Diese Protokolldeskriptorliste, welche den notwendigen Protokollstack für den *FTP*-Dienst beschreibt, besteht aus mehreren Protokolldeskriptoren. Der *RFCOMM*-Protokolldeskriptor enthält neben der *RFCOMM_uuid* noch den protokollspezifischen Wert *RFCOMM Server Channel*, welcher die Nummer des Kanals angibt, über den die *FTP*-Daten gesendet werden. Anschliessend tritt die Aktivität 8 in Aktion.

- *Aktivität 8*

Diese Aktivität überträgt über Funk die Datei *status.xml* vom entfernten zum lokalen Gerät. Dieser Vorgang ist in der nachfolgenden Abbildung 4.23 dargestellt:

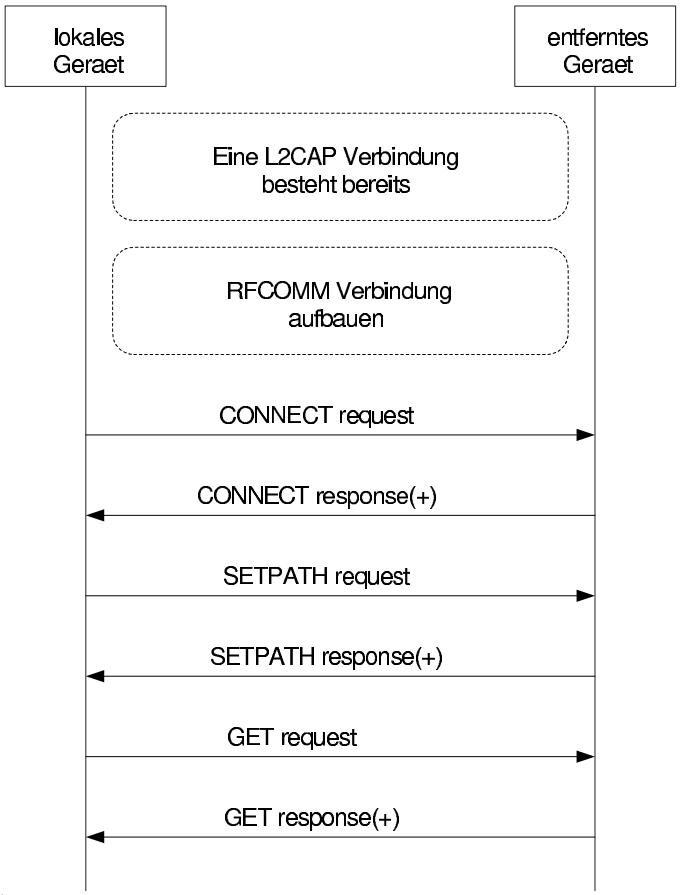

Abbildung 4.23: Übertragung der Datei *status.xml* vom Server zum Client

Zunächst muss eine RFCOMM-Verbindung mit den RFCOMM-Prozeduren hergestellt werden. Damit eine RFCOMM-Verbindung aufgebaut werden kann, muss zuerst eine RFCOMM-Sitzung mit der Prozedur *initialize RFCOMM Session* des *RFCOMM*-Protokolls eröffnet werden. Anschliessend wird ein Verbindungskanal mit der Prozedur *Establish DLC* und mit der aus der Aktivität 7 ermittelten *RFCOMM-Server-Kanalnummer* aufgebaut. Steht die RFCOMM-Verbindung, eröffnet man eine OBEX-Sitzung mit der OBEX-Transaktion *CONNECT*. Im nächsten Schritt muss der Client mit den zwei OBEX-Transaktionen *SETPATH* das Verzeichnis auf

dem Server so einstellen, dass es auf das Basisverzeichnis zeigt (siehe Seite 71). Eine *SETPATH*-Transaktion ist erforderlich, um das Rootverzeichnis einzustellen, eine weitere, um das Basisverzeichnis zu setzen. Zur Vereinfachung wird nur eine dieser *SETPATH*-Transaktionen in der Abbildung 4.23 dargestellt. Nun wird mit einer *GET*-Transaktionen die Datei status.xml vom Server zum Client übertragen. Dabei stellt die Nachricht *GET request* die Objektanfrage dar und die Nachricht *GET response* enthält das angeforderte Objekt. Die Response-Nachrichten aus Abbildung 4.23 sind mit einem Plusssymbol markiert. Dies bedeutet, dass die dazugehörige Anfrage mit Erfolg durchgeführt worden ist. Nach erfolgreicher [13] Funkübertragung der Status-Datei wird zur Aktivität 9 gewechselt.

- *Aktivität 9*
  Diese Aktivität prüft, ob die entfernte *Flirtmaschine* aktiv ist. Um dieses festzustellen, muss der Wert des Tags *Running* ausgelesen werden. Beim Wert *yes* ist die entfernte *Flirtmaschine* aktiv, bei *no* inaktiv. Bei dem Wert *no* müssen alle Verbindungen, die nicht vor dem Aufruf der Methode *start()* mit dem verbundenen Gerät bestanden haben, mit den entsprechenden Operationen der beteiligten Protokolle abgebaut werden. Anschliessend geht es bei der Aktivität 3 weiter. Bei dem Wert *yes* wird bei der Aktivität 10 fortgefahren.
  Die Aktivität 9 stellt sicher, dass ein lokales Gerät nur dann ein *Flirtprofil* anfordern kann, wenn das entfernte Gerät selbst Zugang zum *Flirtprofil* des lokalen Gerätes hat. Damit wird erreicht, dass im Falle einer Übereinstimmung beide Benutzer benachrichtigt werden.

- *Aktivität 10*

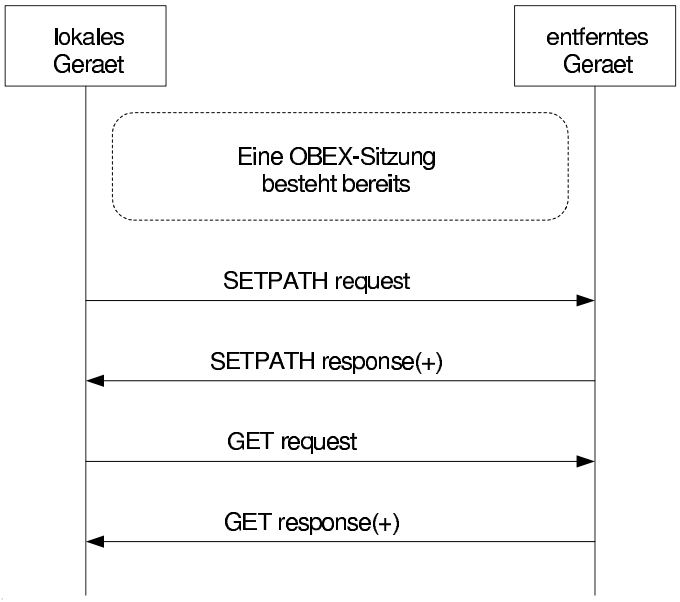

Abbildung 4.24: Übertragung des Profils vom Server zum Client

[13] Es wird hier unterstellt, dass die Funkübertragung der Datei *status.xml* stets reibungslos funktioniert. Ansonsten müsste im Falle einer fehlerhaften Übertragung oder im Falle einer nicht vorhandenen Status-Datei in die Aktivität 3 gewechselt werden. Ist die Datei *status.xml* nicht vorhanden, wird davon ausgegangen, dass die Flirtmaschine nicht unterstützt wird.

Diese Aktivität ist für die Übertragung des Flirtprofils vom Server zum Client per Funk zuständig (siehe Abbildung 4.24). Damit Dateien per Funk übertragen werden können, ist eine *OBEX*-Sitzung erforderlich, welche bereits in der Aktivität 8 eröffnet wurde. Der Client setzt nun mit einer Folge von OBEX-Transaktionen *SETPATH* das Verzeichnis so, dass es auf das Verzeichnis *myprofile* zeigt. Dies wird erreicht, indem man schrittweise über die Ordner *profilematcher*, *profiles* und *myprofile* navigiert. Anschliessend werden durch 2 *GET-Transaktionen* jeweils die 2 Dateien *myprofile.xml* und *pic.jpg* vom Server zum Client übertragen. Zur Vereinfachung wurde hier ebenfalls nur eine *SETPATH*- sowie nur eine *GET*-Transaktion in der Abbildung 4.24 dargestellt. Nach erfolgreicher[14] Funkübertragung des Profils wird zur Aktivität 11 gewechselt.

- *Aktivität 11*
  Diese Aktivität prüft, welche der Verbindungen zum entfernten Gerät bereits existiert haben, bevor die Methode *start()* aufgerufen worden ist. Falls nicht alle Verbindungen existiert haben, wird bei der Aktivität 12 angeknüpft. Sollten jedoch alle jetzigen Verbindungen bereits existiert haben, wird bei der Aktivität 13 fortgefahren.

- *Aktivität 12*
  Diese Aktivität beendet mit den entsprechenden Operationen der Protokolle jede einzelne Verbindung, die vorher nicht mit dem entfernten Gerät existiert hat. Das heisst, es wird der alte Verbindungszustand, welcher vor dem Aufruf der Methode *start()* bestand, mit dem entfernten Gerät hergestellt. Anschliessend wird bei der Aktivität 13 forgesetzt.

- *Aktivität 13*
  Diese Aktivität nimmt die Bluetooth-Geräteadresse des entfernten Gerätes in die Liste der bereits bekannten Geräte auf. Damit stellt diese Aktivität sicher (in Kooperation mit der Aktivität 4), dass keine Bluetooth-Geräte in der Umgebung mehrfach abgehandelt werden. Es wird bei der Aktivität 3 weiter gemacht.

- *Aktivität 14*
  Diese Aktivität prüft, ob die Abbruchbedingung erfüllt worden ist. Ist die Abbruchbedingung nicht erfüllt, dann wird zur Aktivität 2 gewechselt. Falls die Abbruchbedingung erfüllt ist, wird die Methode *start()* der Klasse *PMClient* beendet. Nur die Methode *stop()* der Klasse *PMClient* kann die Methode *start()* beenden (siehe Seite 81).

[14] Es wird hier unterstellt, dass die Übertragung des Profils über Funk stets reibungslos funktioniert. Ansonsten müsste im Falle einer fehlerhaften Übertragung zur Aktivität 3 gewechselt werden.

4.7.2 Das Konzept auf der Profilebene

Das Konzept für die Kommunikation auf der Protokollebene, welches im vorherigen Abschnitt 4.7.1 vorgestellt wurde, veranschaulicht die detaillierten Vorgänge, die beim Austausch der Flirtprofile zwischen den Bluetooth-Geräten durchgeführt werden. Allerdings wird der Anwendungsentwickler wohl nicht soweit in die Details gehen müssen, da die Programmierbibliotheken auf der Ebene der Bluetooth-Profile angesiedelt sind. Deshalb wird in diesem Abschnitt noch ein Konzept auf der Profilebene erläutert, welches weniger aufwändig als das vorherige Konzept ist, da es sich auf einem höheren Abstraktionsniveau befindet.

Für das Konzept auf der Profilebene muss das bereits in der Abbildung 4.18 (siehe Seite 83) vorgestellte Aktivitätsdiagramm für die Methode *start()* der Klasse *PMClient* nur ein wenig geändert werden. Die Aktivitäten, welche eine Änderung bedürfen, werden nachfolgend beschrieben:

- *Aktivität 2*
  Diese Aktivität initiiert eine Geräteanfrage mit dem Aufrufen der *Inquiry*-Prozedur des *GAP*-Profils. Nach der Geräteanfrage wird zur Aktivität 3 übergegangen.

- *Aktivität 5*
  Diese Aktivität überprüft, ob zum ausgewählten Gerät bereits eine *RFCOMM*-Verbindung besteht. Ist dies der Fall, wird zur Aktivität 8 gewechselt. Besteht keine *RFCOMM*-Verbindung, wird zur Aktivität 6 übergegangen.

- *Aktivität 6*
  Diese Aktivität stellt eine *RFCOMM*-Verbindung her, indem die Prozedur *Establish link and set up virtual serial connection* des *SPP*-Profils aufgerufen wird. Diese Prozedur führt SDP-Transaktionen durch, um die *RFCOMM-Server-Kanalnummer* zu erhalten, richtet einen *L2CAP*-Kanal ein und eröffnet eine *RFCOMM*-Verbindung. Dies bedeutet, dass man sich um die internen Schritte der Prozedur *Establish link and set up virtual serial connection* nicht selbst kümmern muss. Bei einem erfolgreichen Verbindungsaufbau wird bei der Aktivität 8 angeknüpft, sonst bei einem nicht erfolgreichen Verbindungsuafbau wird zur Aktivität 3 zurückgesetzt.

- *Aktivität 7*
  Diese Aktivität fällt weg, da die benötigte *RFCOMM-Server-Kanalnummer* von der Aktivität 6 bereits ermittelt worden ist.

- *Aktivität 8*
  Diese Aktivität überträgt die Datei *status.xml* vom entfernten zum lokalen Gerät. Für die Übertragung muss eine *OBEX*-Sitzung mit der Prozedur *Establishing an Object Exchange session* des *GOEP*-Profils eröffnet werden. Ist dies geschehen, kann mit der der Übertragung der Datei *status.xml* mit der *PULL*-Eigenschaft des Profils *GOEP* durchgeführt werden. Mit Sicherheit wird es dabei möglich sein, einen Parameter, welcher den Verzeichnispfad und den Namen der zu übertragenen Datei angibt, der *PULL*-Eigenschaft zu übergeben. Dies hängt davon ab, wie die Eigenschaften der Profile in den Programmierbibliotheken implementiert sind. Die *PULL*-Eigenschaft erspart einem die mehrfache Ausführung der *SETPATH*-Transaktionen, welche das Verzeichnis für die Dateiübertragung festlegen. Das Verzeichnis, indem sich die Datei *status.xml* befindet, lässt sich auch mit der Eigen-

schaft *Folder Browsing* des *FTP*-Profils einstellen. Im Anschluss wird zur Aktivität 9 gewechselt.

- *Aktivität 10*
  Diese Aktivität überträgt die Profildateien *myprofile.xml* und *pic.jpg* ebenfalls mit der *PULL*-Eigenschaft des *GOEP*-Profils. Eine *OBEX*-Sitzung muss hier nicht eröffnet werden, da dies bereits in der Aktivität 8 geschehen ist. Auch hier fällt die mehrfache Ausführung der *SETPATH*-Transaktionen weg und erleichtert somit, wie bei der Aktivität 8, die Übertragung von Dateien. Anschliessend wird bei der Aktivität 11 fortgefahren.

Kapitel 5

Realisierung

Dieses Kapitel verschafft einen kurzen Überblick über den Status Quo der Realisierung. Anschliessend werden noch einige Screenshots der *Flirtmaschine* vorgestellt.

5.1 Status Quo

Fast alle Pakete und die darin enthaltenen Klassen aus der Abbildung 4.3 auf der Seite 64 sind implementiert worden. Der Quellcode liegt als Anhang auf der beigefügten CD dieser Diplomarbeit bei.

Die einzige Ausnahme bildet die Klasse *PMClient* aus dem Paket *communications*. Die Klasse *PMClient* ist zwar im Quellcode vorhanden, aber sie beinhaltet keinerlei Implementierung. Die Klasse *PMClient* ist dafür verantwortlich, dass die Flirtmaschinen auf verschiedenen Geräten über die Bluetooth-Technologie drahtlos miteinander kommunizieren können. Nur leider, wie bereits im Abschnitt 4.2 auf Seite 62 erwähnt, habe ich keine günstige Bluetooth-Programmierbibliothek auf dem Markt finden können. Aus diesem Grund haben wir, ich und mein betreuender Professor, entschieden, die Kommunikation nur rein konzeptionell zu lösen.

Für den Abgleich zweier Profile und die Bewertung, ob zwei Profile miteinander übereinstimmen, ist die Methode *compareProfiles()* zuständig (siehe Seite 78). Die Implementierung dieser Methode nutzt nur einen trivialen Algorithmus, um die Profile miteinander abzugleichen. Dabei müssen die Suchkriterien des eigenen Profils mit den Profildaten des entfernten Gerätes exakt übereinstimmen, damit eine Übereinstimmung (Match) auf beiden Seiten gemeldet wird. Eine nicht exakte Übereinstimmung würde demnach niemals zu einem Match führen. Dies ist natürlich ein Nachteil, der mit der Fuzzy-Logik behoben werden kann. Würde der Vergleichsalgorithmus auf Basis der Fuzzy-Logik arbeiten, dann wären kleine Abweichungen erlaubt. Die Fuzzy-Logik wurde zwar nicht implementiert, aber sie wird dafür auf Seite 79 erläutert.

5.2 Screenshots

Dieser Abschnitt stellt einige Screenshots der *Flirtmaschine* vor. Dabei werden auf die Menüs *Einstellungen*, *Profil* und *File* eingegangen. Da das Menü *Hilfe* nur eine Informationsanzeige beinhaltet, wird auf die dazugehörige Screenshot verzichtet. Die Menüpunkte beziehen sich auf die Anwendungsfälle aus der Analyse (siehe Abschnitt 3.2; Seite 50).

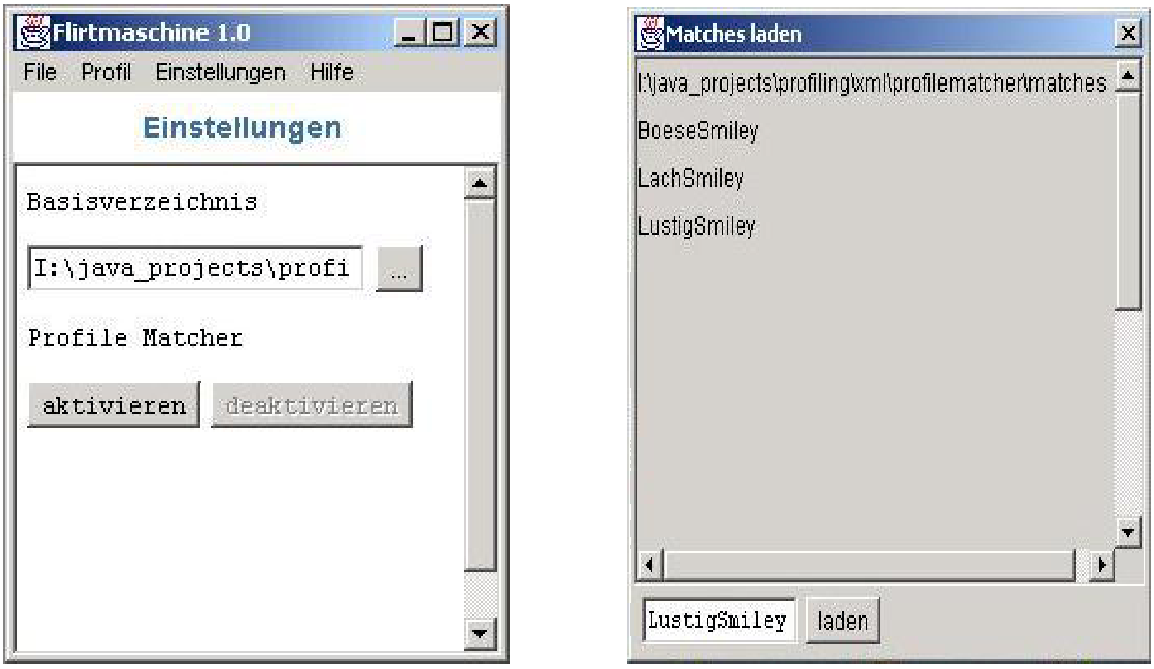

Abbildung 5.1: Menüpunkte Einstellungen & Matches laden

Im Menü *Einstellungen* gibt es einen Menüpunkt *Einstellungen*. Wird dieser Menüpunkt selektiert, dann erscheint die in Abbildung 5.1 links dargestellte Fläche. Mit dem *Drei-punkte* Button kann man das Basisverzeichnis setzen. Mit dem Button *aktivieren* wird die Flirtmaschine aktiviert, mit dem Button *deaktivieren* deaktiviert. Beim Aktivieren der *Flirtmaschine* wird die Methode *start()* der Klasse *PMClient* und die Methode *startPM()* der Klasse *ProfileManager* aufgerufen. Beim Deaktivieren werden diese Methoden gestoppt.

Im Menü *File* gibt es einen Menüpunkt *Matches laden*. Wird dieser Menüpunkt ausgewählt, dann erscheint ein zusätzliches Fenster (Abbildung 5.1;rechte Seite). Dort kann man den Namen eines Profils, das geladen werden soll, in einem Textfeld eingeben. In diesem Fall kann man eines der Profile, die sich im Ordner *matches* befinden, laden. Alle weiteren Menüpunkte im Menü *File* (*Matches speichern*, *Profil laden*, *Profil speichern*) funktionieren analog dazu.

Abbildung 5.2: Menüpunkte mein Bild & über mich

Im Menü *Profil* gibt es die Menüpunkte *mein Bild*[1], *meine Interessen*, *ich suche* und *über mich*. Ist der Menüpunkt *mein Bild* angewählt, wird das Bild des aktuell geladenen Profils angezeigt. Ist der Menüpunkt *über mich* selektiert, werden die charakterisierenden Profildaten hinsichtlich der Person angezeigt. Über die Fläche *über mich* kann man auch die Profildaten setzen und über das Menü *File* können die Änderungen gespeichert werden.

[1]Der Name *mein Bild* ist irreführend. Man könnte denken, dass damit stets das Bild des eigenen Profils angezeigt wird. Jedoch bezieht sich das angezeigte Bild stets auf das aktuell geladene Profil. Ebenso verhält es sich mit den Profildaten, die unter den Menüpunkten *meine Interessen*, *ich suche* und *über mich* angezeigt werden.

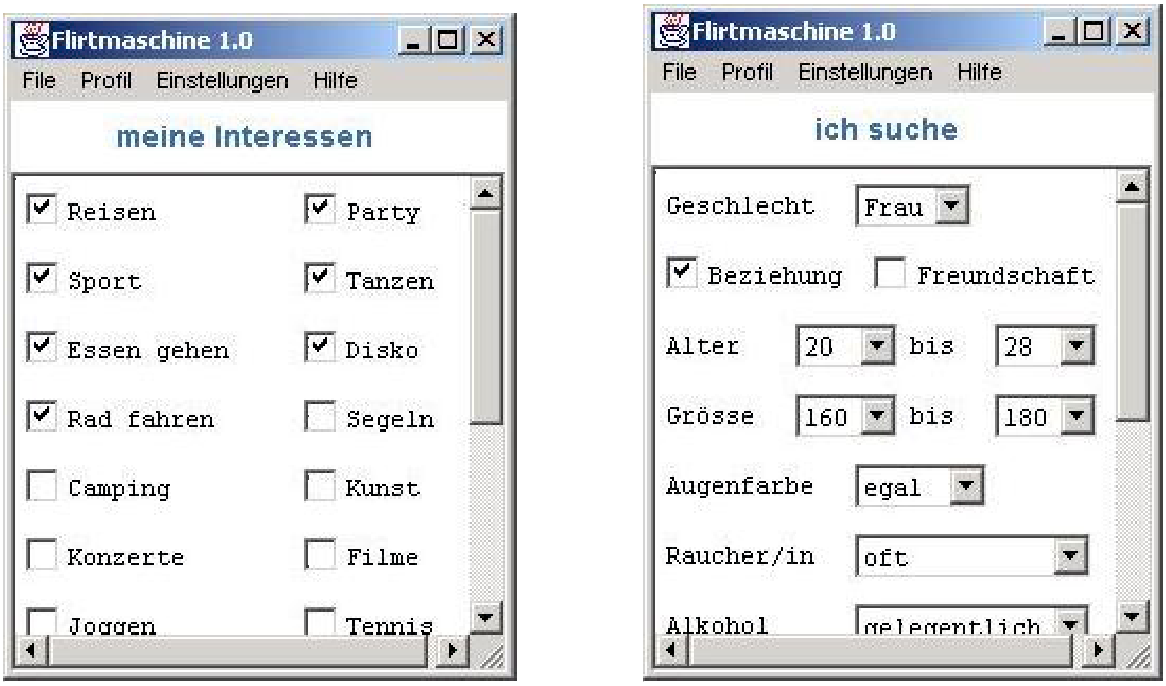

Abbildung 5.3: Menüpunkte meine Interessen & ich suche

Über den Menüpunkt *meine Interessen* kann man sich die Präferenzen der Person anzeigen lassen, unter dem Menüpunkt *ich suche* deren Suchkriterien. Diese Suchkriterien müssen mit den Profildaten eines anderen Profils exakt konform sein, damit es zu einem Match führt. Die Interessen und die Suchkriterien können über die Displays gesetzt und die Änderungen über das Menü *File* gespeichert werden.

Kapitel 6

Resümee und Ausblick

Abschliessend wird in diesem Kapitel die Diplomarbeit zusammengefasst, das Ziel bewertet und ein Ausblick über mögliche Weiterentwicklungen speziell der Flirtmaschine und im Allgemeinen gegeben.

6.1 Zusammenfassung

In der *Einleitung* dieser Diplomarbeit wurde zunächst einmal in das Thema *Pervasive*- bzw. *Ubiquitous Computing* eingeführt und deren Anwendungsfelder beschrieben. Weiterhin wurden am Ende des einleitenden Kapitels das Ziel und die Motivation dieser Diplomarbeit erläutert, wobei das Ziel darin bestand, die profilverarbeitende ubiquitäre Anwendung *Flirtmaschine* zu entwickeln.

Im Anschluss wurden im Kapitel *Grundlagen* die Begrifflichkeiten und die Technologien erklärt, welche bei der Entwicklung der *Flirtmaschine* eine essentielle Rolle gespielt haben. Hierbei wurde die Motivation und der Einsatz von Profilen diskutiert. Unter anderem wurden auch die Technologien *Bluetooth* und *JaxB* vorgestellt, da auf diese beiden Technologien für die Entwicklung der *Flirtmaschine* zurückgegriffen worden ist.

Im darauffolgenden Kapitel *Analyse* wurden die Anforderungen an die *Flirtmaschine* bestimmt. Auf Basis dieser Anforderungen wurden die Anwendungsfälle ermittelt und spezifiziert, desweiteren ist die *Flirtmaschine* durch Klassenmodelle und Sequenzdiagramme beschrieben worden.

Das vierte Kapitel *Design* erläutert, welche Technologien, Programmiersprachen und Algorithmen für die Realisierung der *Flirtmaschine* zum Einsatz gekommen sind. Zudem wurde die Architektur der Flirtmaschine beschrieben und das Klassenmodell aus der Analyse um programmierspezifische Klassen und Methoden erweitert. Bereits bestehende Methoden wurden weiterhin hinsichtlich ihrer Arbeitsweise und den eingesetzten Algorithmen konkretisiert. Auch wurde die eingesetzte Zielplattform festgelegt und im letzten Abschnitt ein Konzept für die Kommunikation vorgestellt.

Das fünfte Kapitel *Realisierung* klärt zunächst auf, welche Teile des Klassemodells aus der Analyse und dem Design tatsächlich realisiert und welche rein konzeptionell entwickelt wurden. Im Anschluss daran wurden einige Screenshots der *Flirtmaschine* präsentiert.

6.2 Fazit

Das Ziel dieser Diplomarbeit ist die exemplarische Entwicklung der profilverarbeitenden ubiquitären Anwendung *Flirtmaschine* gewesen. Die Interaktionskomponente, die Datenhaltungskomponente und die Verarbeitungskomponente der *Flirtmaschine* konnten von mir erfolgreich umgesetzt werden. Die Kommunikationskomponente ist die einzige Komponente, welche nur rein konzeptionell umgesetzt wurde, der Grund hierfür wird am Ende dieses Abschnittes dargelegt.

Meiner Meinung nach macht der Einsatz der *Flirtmaschine* nur dann Sinn, wenn sie von vielen Benutzern angewendet wird, damit möglichst viele *Flirtprofile* miteinander abgeglichen werden können. Am besten geschieht die Nutzung der *Flirtmaschine* innerhalb eines vorher festgelegten Ortes und Zeitraums, damit genügend *Flirtmaschinen* aufeinander treffen. Folgendes Beispiel zur Veranschaulichung:

Beispiel

Veranstaltungen finden stets an einem bestimmten Ort und zu einer bestimmten Zeit statt. Nun könnte man sich vorstellen, dass Gäste ihre Bluetooth-Geräte zur Veranstaltung mitbringen. Die Gäste haben entweder bereits Zuhause aus dem Internet die *Flirtmaschine* heruntergeladen oder können es nun vor Ort auf der Veranstaltung von einem bereitgestellten Bluetooth-Server aus tun. Jetzt müssen die Gäste nur noch ein persönliches Flirtprofil erstellen und anschliessend ihre *Flirtmaschine* aktivieren. Der Flirt mit den anderen Gästen kann beginnen.

Zu bedenken ist jedoch, dass Bluetooth-Geräte im allgemeinen noch nicht so stark verbreitet und somit der Einsatz der *Flirtmaschine* zurzeit nur beschränkt möglich ist. Ein weiteres Problem bei der Ausführung der *Flirtmaschine* auf der mobilen Bluetooth-Einheit stellt die Energieversorgung dar. Spontan kommunizierende Anwendungen wie die *Flirtmaschine* haben den erheblichen Nachteil, immer in periodischen Zeitabständen ihre Funkumgebung überwachen und auswerten zu müssen, damit sie einwandfrei ihre Aufgaben erfüllen können. Dies bringt einen erhöhten Stromverbrauch mit sich und kann zu Engpässen in der Stromversorgung führen.

Nichtsdestotrotz kann diese Diplomarbeit hilfreich für Projekte oder andere Diplomarbeiten sein, die sich ebenfalls mit der Entwicklung einer ubiquitären Anwendung beschäftigen, welche insbesondere auf Basis von ortsbezogenen und/oder auf personalisierten Daten[1] arbeitet, da diese Diplomarbeit einen guten Einstieg in die hierfür notwendigen Technologien, speziell sei hier die Bluetooth-Technologie genannt, bietet.

Ich bin mit dem Ablauf, der Vorgehensweise und mit dem Ergebnis dieser Diplomarbeit sehr zufrieden. Bei der Vorgehensweise halte ich es für besonders wichtig, im Vorfeld grundlegende konzeptionelle Überlegungen anzustellen, um die Machbarkeit der Aufgabe zu prüfen. So habe ich feststellen müssen, dass Programmierschnittstellen für die Bluetooth-Technologie für den *Compaq Ipaq*-PDA unter dem Betriebssystem *Pocket PC*

[1] Was genau unter ortsbezogene und personalisierte Daten zu verstehen ist, wird im Abschnitt *Mobile Informationsdienstleistung* auf Seite 2 erläutert.

2002 rar sind. Da die wenigen existierenden Programmierbibliotheken relativ hohe Lizenzkosten[2] aufweisen, wurde für die Lösung der Kommunikationskomponente der konzeptionelle Weg eingeschlagen. Hätte man jedoch eine der Bluetooth-Programmierbibliotheken eingesetzt, dann stünde der technischen Umsetzung der *Firtmaschine* nichts mehr im Wege.

6.3 Ausblick

Ich denke, dass in einigen Jahren die nachteiligen Einschränkungen aus dem vorherigen Abschnitt 6.2 für den Einsatz der *Flirtmaschine* minimiert werden können.

Laut Schätzungen der *Cahner In-Stat Group* wird bis zum Jahre 2005 die Anzahl der drahtlosen Bluetooth-Geräte weltweit auf mehr als 670 Millionen ansteigen [Nathan (2001)]. Der Bluetooth-Chip wird dabei in Mobiltelefonen, Notebooks, Computern und in anderen elektronischen Geräten implementiert sein. Damit wird wohl mit der Zeit die Einschränkung - *Bluetooth-Geräte sind in der Anzahl noch nicht so stark verbreitet* - schwinden.

Auch bin ich zuversichtlich, dass man das Problem mit der Energieversorgung der Geräte in naher Zukunft in den Griff bekommt. Ich vermute, dass sich die Technologien für die Energieversorgung stetig verbessern werden. Es wäre denkbar, neben den typischen Akkugeräten auch Solarenergie oder umgewandelte Bewegungsenergie für die Geräte zu nutzen. Die Energiekapazitäten für Akkugeräte werden vermutlich in den nächsten Jahren noch weiter gesteigert.

Für den Abgleich zweier Profile wäre es besser, einen Algorithmus einzusetzen, welcher auf der Basis der Fuzzy-Logik[3] arbeitet. Die Fuzzy-Logik hat den Vorteil, dass sie kleine Abweichungen bei dem Abgleich der Suchkriterien des eigenen Profils mit den Profildaten des entfernten Gerätes zulässt.

Meine Vision geht dahin, dass in Zukunft die Benutzer auf ihren mobilen Endgeräten ein universelles Profil über sich ablegen und Anbieter von mobilen Diensten legitimiert sind, nur auf bestimmte Aspekte des universellen Profils zuzugreifen. Deshalb ist es besser für die Repräsentation der Profildaten den *Customer Profile Exchange*-Standard[4] zu verwenden. Bei diesem Standard hat der Benutzer die Möglichkeit, für unterschiedliche Aspekte der Profildaten unterschiedliche Zugriffsregeln für die Anbieter zu definieren.
Für die Anbieter von mobilen Diensten hätte es den Vorteil, dass sie bequem mit kooperierenden Firmen Kundenprofile austauschen können. Der Benutzer hat den Vorteil, dass ihm die Erstellung mehrfacher Profile bei der Anmeldung zur Nutzung von mobilen Diensten erspart bleibt.

[2] Die Firma *www.high-point.com* stellt ab Januar 2003 eine Bluetooth-Programmierbibliothek in C++ für den *Compaq Ipaq*-PDA auf ihrer Webseite kostenfrei zum Herunterladen zur Verfügung. Diese Version hat allerdings den Nachteil, beim ersten Zugriff auf den Bluetooth-Stack eine Werbenachricht anzuzeigen. Aufgrund der zu späten Veröffentlichung dieser Bibliotheken konnte ich die Realisierung der Kommunikation der *Flirtmaschine* mit diesen Bibliotheken nicht mehr in meinem Zeitplan unterbringen.

[3] Die Fuzzy-Logik wurde auf Seite 79 erläutert.

[4] Der *Customer Profile Exchange*-Standard wurde im Abschnitt 2.2.2 auf Seite 2.2.2 vorgestellt.

Ich bin gespannt, inwiefern sich der Markt für den Bereich *Pervasive/Ubiquitous Computing* in Zukunft weiterentwickeln wird. Ich denke, dass dieser Markt ein grosses Potential hat und die Anzahl der innovativen mobilen Dienstleistungen/Anwendungen weiter zunehmen wird. Welche dieser Dienstleistungen/Anwendungen sich auf dem Markt behaupten werden, hängt hauptsächlich vom Preis, erkennbaren Nutzen und den geleisteten Datenschutz ab. Auch denke ich, dass ganz besonders ortsbezogene und personalisierte mobile Dienstleistungen stärker zunehmen werden, damit Unternehmen auf die Kunden individuell eingehen und sich somit Wettbewerbsvorteile sichern können.

Anhang A

Inhalt der CD-ROM

Dieser Diplomarbeit ist eine CD, welche beim Prof. Dr. Gunter Klemke hinterlegt worden ist, beigefügt. Unter dem Rootverzeichnis dieser CD befinden sich folgende Ordner und Dateien:

- Ordner *flirtmaschine_java_project*
  Dieser Ordner beinhaltet das Java-Projekt für die *Flirtmaschine*. In diesem Projekt findet sich der gesamte Java-Quellcode der *Flirtmaschine* und die *JaxB*-Klassen wieder. Für die Implementierung der *Flirtmaschine* ist die Java Entwicklungsumgebung *Code Warrior Wireless Studio 7.0 Professional Edition* von der Firma Metrowerks eingesetzt worden. Die Datei *profilematcher.mcp*, welche sich im Ordner *flirtmaschine_java_project* befindet, repräsentiert die Projektdatei.

- Ordner *profilematcher*
  Dieser Ordner verwaltet die *Flirtprofile*. Die Struktur und Inhalt dieses Ordners ist auf der Seite 71 beschrieben.

- Ordner *schema_compiler*
  Dieser Ordner beinhaltet wiederum die folgenden Dateiobejekte:

  – Ordner *lib*
    Dieser Ordner enthält die *jar*-Dateien *jaxb-rt-1.0-ea* und *jaxb-xjc-1.0-ea*. Die erstere beinhaltet die Klassen für das *JaxB*-Framework. Die zweitere beinhaltet die Klassen des Schema Compilers.

  – Ordner *profiling*
    Alle Dateien die sich im Ordner *profiling* befinden, sind vom Schema Compiler automatisch generiert.

  – build.bat
    Diese Stapelverarbeitungsdatei enthält den Aufruf für den Schema Compiler.

  – myprofile.dtd
    Diese Datei repräsentiert das XML-Schema, welches die die Struktur der XML-Daten für das Flirtprofil spezifiziert.

  – myprofile.xjs
    Diese Datei repräsentiert das XML-Binding-Schema und beschreibt, wie das XML-Schema des Flirtprofils auf Java-Klassen abgebildet wird.

- Datei *diplomarbeit.pdf*
  Diese Datei beinhaltet diese Diplomarbeit als PDF-Dokument.

- Datei *pm_start.lnk*
  Diese Datei beinhaltet den Kommandozeilen-Aufruf, um die *Flirtmaschine* auf dem PDA über die Java-Virtuelle-Maschine *Jeode* zu starten.

Literaturverzeichnis

[SunIntr] *Introduction to Wireless Java[tm] Technology*. USA, California : Sun Microsystems Inc.. – URL http://wireless.java.sun.com/getstart/. – Zugriffsdatum: 25. Januar 2003

[SunJaxbDoc] *JaxB-Webseite von Sun Microsystems Inc.* USA, California : Sun Microsystems Inc.. – URL http://java.sun.com/xml/jaxb/index.html. – Zugriffsdatum: 25. Januar 2003

[btassnum 2002] *Bluetooth Assigned Numbers*. Bluetooth SIG, 2002. – URL http://www.bluetooth.org/assigned-numbers.htm. – Zugriffsdatum: 23.12.2002

[Amor 2002] AMOR, Daniel: *Das Handy gegen Zahnschmerzen und andere Geschäftsmodelle für die Dienstleister von morgen*. Germany - Bonn : Galileo Press, 2002. – ISBN 3-89842-173-2

[Astarabadi u. a. 2001] ASTARABADI, Shaun u. a.: *Specification of the Bluetooth System - File Transfer Profile (Seite 366-396)*. Bluetooth Special Interest Group (SIG), 2001. – URL www.bluetooth.com/pdf/Bluetooth_11_Profiles_Book.pdf. – Zugriffsdatum: 26.11.2002

[Autoren 2001a] AUTOREN, Diverse: *Specification of the Bluetooth System - Core*. Bluetooth Special Interest Group (SIG), 2001. – URL www.bluetooth.com/pdf/Bluetooth_11_Specifications_Book.pdf. – Zugriffsdatum: 26.11.2002

[Autoren 2001b] AUTOREN, Diverse: *Specification of the Bluetooth System - Profiles*. Bluetooth Special Interest Group (SIG), 2001. – URL www.bluetooth.com/pdf/Bluetooth_11_Profiles_Book.pdf. – Zugriffsdatum: 26.11.2002

[Balzert 2001] BALZERT, Helmut: *Lehrbuch der Software Technik; Software-Entwicklung*. 2. Auflage. Spektrum Akademischer Verlag, 2001. – URL www.spektrum-verlag.com. – ISBN 3-8274-0480-0

[Bisdikian u. a. 2001] BISDIKIAN, Chatschik u. a.: *Specification of the Bluetooth System - Service Discovery Application Profile (Seite 64-98)*. Bluetooth Special Interest Group (SIG), 2001. – URL www.bluetooth.com/pdf/Bluetooth_11_Profiles_Book.pdf. – Zugriffsdatum: 26.11.2002

[Bohrer und Holland Oktober 2000] BOHRER, Kathy ; HOLLAND, Bobby: *Customer Profile Exchange (CPExchange) Specification Version 1.0*. Customer Profile Exchange Network, Oktober 2000. – URL http://www.cpexchange.org/standard/cpexchangev1_0f.pdf;http://www.cpexchange.org. – Zugriffsdatum: 25.01.2003

[Bornträger 2001] BORNTRÄGER, Christian: *Seminararbeit - Adhoc Netzwerke*. Ilmenau : Technische Universität Ilmenau, 2001. – URL http://ikmcip1.e-technik.tu-ilmenau.de/~webkn/STUD-DIPLOM/STUDREFERAT/ad-hoc-netze/HS-SS01-AdHocNetze.html. – Zugriffsdatum: 25.01.2003

[Britta Oertel 2001] BRITTA OERTEL, Lothar B.: *Entwicklung und zukünftige Bedeutung mobiler Multimediadienste*. Berlin : Institut für Zukunftsstudien und Technologiebewertung, 2001. – URL www.izt.de/mmd. – ISBN 3-929173-49-2

[Bullinger u. a. 2002] BULLINGER ; BAUMANN ; FRÖSCHLE ; MACK ; TRUNZER ; WALTERT: *Business Communities - Professionelles Beziehungsmanagement von Kunden, Mitarbeitern und B2B-Partnern im Internet*. Bonn/Deutschland : Galileo Press, 2002. – ISBN 3-89842-121-X

[Burkhardt u. a. 2001] BURKHARDT, Jochen ; HENN, Horst ; HEPPER, Stefan u. a.: *Pervasive Computing - Technologie und Architektur mobiler Internetanwendungen*. Addison-Wesley, 2001. – URL www.addison-wesley.de. – ISBN 3-8273-1729-0

[Computers Januar 2001] COMPUTERS, Fujitsu S.: *White Paper - Bluetooth Technologie der drahtlosen Kommunikation der Zukunft*. München : Fujitsu Siemens Computers, Januar 2001. – URL http://www.fujitsu-siemens.de/rl/produkte/wireless/download/lifebook_whitepaper_bluetooth_technologie_neu.pdf. – Zugriffsdatum: 25.01.2003

[Coulouris u. a. 2002] COULOURIS, George ; DOLLIMORE, Jean ; KINDBER, Tim: *Verteilte System - Konzepte und Design*. 3. überarbeitete Auflage. Addison Wesley, 2002. – URL www.addison-wesley.de. – ISBN 3-82737022-1

[Daguhn 2001] DAGUHN, Walter: *Herstellerunabhängige Informationen über die Bluetooth Technologie und deren wirtschaftlichen Aspekte*. Mönchengladbach : rfi mobile technologies AG, 2001. – URL http://www.rfi.de/downloads/blue.pdf. – Zugriffsdatum: 25.01.2003

[Dölfer 2002] DÖLFER, Jörg: *Programmierung von mobilen Endgeräten mit der J2ME*. Sigs Datacom, 2002. – URL http://www.sigs.de/publications/os/2001/06/dslfer_OS_06_01.pdf. – Zugriffsdatum: 25.01.2003

[Dorka Juli 2001] DORKA, Matthias: *Diplomarbeit - Ein Referenzmodell für Profiling bei e-Business-Websites*. Dortmund : Universität Dortmund, Juli 2001. – URL http://ls10-www.cs.uni-dortmund.de/LS10/Pages/vupartner/Publications/DAs2001/da_Matthias_Dorka.pdf. – Zugriffsdatum: 25.01.2003

[Farnsworth u. a. 2001] FARNSWORTH, Dale u. a.: *Specification of the Bluetooth System - Service Discovery Protocol (Seite 332-392)*. Bluetooth Special Interest Group (SIG), 2001. – URL www.bluetooth.com/pdf/Bluetooth_11_Specifications_Book.pdf. – Zugriffsdatum: 26.11.2002

[Gamma u. a. 1996] GAMMA, Erich ; HELM, Richard ; JOHNSON, Ralph ; VLISSIDES, John: *Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software*. 3., unveränderter Nachdruck. Addison-Wesley, 1996. – URL www.addison-wesley.de. – ISBN 3-89319-950-0

[Hedlund u. a. 2001] HEDLUND, Henrik u. a.: *Specification of the Bluetooth System - Baseband Specification (Seite 34-182)*. Bluetooth Special Interest Group (SIG), 2001. – URL www.bluetooth.com/pdf/Bluetooth_11_Specifications_Book.pdf. – Zugriffsdatum: 26.11.2002

[Heinsohn und Socher-Ambrosius 1999] HEINSOHN, Jochen ; SOCHER-AMBROSIUS, Rolf: *Wissensverarbeitung - Eine Einführung*. Spektrum Akademischer Verlag, 1999. – ISBN 3-8274-0308-1

[Horstmann und Cornell 1999] HORSTMANN, Gay S. ; CORNELL, Gary: *Java Band 1 - Grundlagen*. Prentice Hall, 1999. – ISBN 3-8272-9565-3

[Kahlbrandt 1998] KAHLBRANDT, Bernd: *Software-Engineering*. Springer Verlag, 1998. – URL www.springer.de. – ISBN 3-540-63309-X

[Köpf 2001] KÖPF, Boris: *Seminararbeit - Bluetooth Eine Einführung*. Konstanz : Universität Konstanz, 2001. – URL http://www.softwareresearch.net/projects/ArchitecturesForMobility/courses/special_topics/Boris_Koepf-Bluetooth.pdf. – Zugriffsdatum: 26.11.2002

[Krauß 2002] KRAUSS, Christoph: *Seminararbeit - Bluetooth Sicherheit*. Darmstadt : Technische Universität Darmstadt, 2002. – URL http://www.ito.tu-darmstadt.de/edu/sem-iuk-w01/Bluetooth.pdf. – Zugriffsdatum: 25.01.2003

[Langheinrich 2001] LANGHEINRICH, Marc: *P3P - Ein neuer Standard für Datenschutz im Internet*. URL http://www.inf.ethz.ch/vs/publ/papers/p3p-digma.pdf. – Zugriffsdatum: 25.01.2003, 2001

[Lin 2002] LIN, Tai-Wei: *JAXB: A Primer*. USA, California : Sun Microsystems Inc., 2002. – URL http://developer.java.sun.com/developer/technicalArticles/xml/jaxb/. – Zugriffsdatum: 25. Januar 2003

[Lind u. a. 2001] LIND, Patric u. a.: *Specification of the Bluetooth System - Generic Access Profile (Seite 14-62)*. Bluetooth Special Interest Group (SIG), 2001. – URL www.bluetooth.com/pdf/Bluetooth_11_Profiles_Book.pdf. – Zugriffsdatum: 26.11.2002

[Mattern] MATTERN, Friedemann: *Pervasive Computing - Ubiquitous Computing*. Schweiz : Eidgenössische Technische Hochschule Zürich. – URL http://www.inf.ethz.ch/vs/publ/papers/UbiPvCSchlagwort.pdf. – Zugriffsdatum: 25.01.2003

[Melin u. a. 2001] MELIN, Tobias u. a.: *Specification of the Bluetooth System - Link Manager Protocol (Seite 184-252)*. Bluetooth Special Interest Group (SIG), 2001. – URL www.bluetooth.com/pdf/Bluetooth_11_Specifications_Book.pdf. – Zugriffsdatum: 26.11.2002

[Microsystems 2001] MICROSYSTEMS, Sun: *The Java[TM] Architecture for XML Binding Users Guide - Early Access Draft*. USA, California : Sun Microsystems Inc., 2001. – URL http://java.sun.com/xml/jaxb/jaxb-docs.pdf. – Zugriffsdatum: 25.01.2003

[Nathan 2001] NATHAN, Müller: *Bluetooth - Die Referenz für den neuen Bluetooth-Standard*. 1. Auflage. Bonn : MITP-Verlag GmbH, 2001. – ISBN 3-8266-0738-4

[Oestereich 1998] OESTEREICH, Bernd: *Objektorientierte Softwareentwicklung - Analyse und Design mit der Unified Modeling Language*. Oldenbourg, 1998. – ISBN 3-486-24787-5

[Olsson u. a. 2001] OLSSON, Patrik u. a.: *Specification of the Bluetooth System - Generic Object Exchange Profile (Seite 310-338)*. Bluetooth Special Interest Group (SIG), 2001. – URL `www.bluetooth.com/pdf/Bluetooth_11_Profiles_Book.pdf`. – Zugriffsdatum: 26.11.2002

[Sörensen u. a. 2001] SÖRENSEN, Johan u. a.: *Specification of the Bluetooth System - Serial Port Profile (Seite 172-196)*. Bluetooth Special Interest Group (SIG), 2001. – URL `www.bluetooth.com/pdf/Bluetooth_11_Profiles_Book.pdf`. – Zugriffsdatum: 26.11.2002

[Stolpmann 2000] STOLPMANN, Markus: *Online - Marketingmix Kunden finden, Kunden binden im E-Business*. Bonn/Deutschland : Galileo Press, 2000. – ISBN 3-934358-10-1

[Tanenbaum 2000] TANENBAUM, Adrew S.: *Computernetzwerke*. 3. revidierte Auflage. Pearson Studium, 2000. – ISBN 3-8273-7011-6

[Weiser 1991] WEISER, Mark: The Computer for the 21st Century. In: *Scientific American* (1991), September, S. 66–75

Abbildungsverzeichnis

Index

Eidesstattliche Versicherung

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §24(5) ohne fremde Hilfe selbständig verfasst und nur die angegebenen Hilfsmittel benutzt habe.

Ort, Datum	Unterschrift des Studenten