OBEX Object Passing über Bluetooth

Hochschule Furtwangen · 31. März 2006

Wintersemester 2005/2006
Jens Frey · jens.frey@coffeecrew.org
Matrikelnummer · 216401
Fakultät · Informatik
Studiengang · Computer Networking

Diplomarbeit

Referent · Prof. Dr. Harald Gläser
Koreferent · Prof. Dr. Wolfgang Bauer

Eidesstattliche Erklärung

Ich erkläre hiermit an Eides statt, dass ich die vorliegende Diplomarbeit selbstständig und ohne unzulässige fremde Hilfe angefertigt habe. Die verwendeten Quellen und Hilfsmittel sind vollständig zitiert.

Furtwangen, 31. März 2006

Jens Frey

Abstrakt

Die vorliegende Arbeit beschreibt das Design sowie die Implementierung des OBEX Object Passing (OOP) Mechanismus, welcher dazu entwickelt wurde um Java Objekte über eine Bluetooth Funkstrecke von einem Bluetooth und Java fähigen Endgerät auf ein zweites zu übertragen.

Inhaltsverzeichnis

KAPITEL 1

Einleitung

Die Bluetooth Technologie erfreut sich in letzter Zeit immer grösserer Beliebtheit, wird doch durch sie das lästige Kabel endlich obsolet. Im Bereich der Applikationsentwicklung fehlt dem Programmierer jedoch die Möglichkeit Java Objekte zwischen zwei Endgeräten auszutauschen. Diese Lücke wird durch die vorliegende Arbeit geschlossen.

1.1 Aufbau der Arbeit

Die Arbeit ist wie folgt aufgeteilt:

Kapitel 1 beschreibt die Zielsetzung, technische Randbedingungen, sowie denkbare Anwendungsszenarien, und Bezugsquellen der vorliegenden Arbeit.

Kapitel 2 gibt einen kurzen Überblick über die zur Realisierung der Bibliothek eingesetzten Technologien.

Kapitel 3 beschreibt das Design der entwickelten Bibliothek, sowie deren Funktionsweise.

Kapitel 4 beschreibt den konkreten Einsatz der Bibliothek anhand eines einfachen Beispiels, und vermittelt die Grundlagen der J2ME Applikationsentwicklung.

Kapitel 5 fasst die Arbeit zusammen und gibt einen Ausblick auf Weiterentwicklungsmöglichkeiten der Bibliothek.

Anhang A demonstriert, wie innerhalb der eingesetzten Netbeans IDE ein J2ME Projekt erstellt werden kann, das die entwickelte Bibliothek einsetzt. Ferner wird beschrieben wie die Beispielapplikation in Betrieb genommen werden kann.

1.2 Zielsetzung der Arbeit

Ziel der Arbeit ist es eine Bibliothek zu erstellen, mit deren Hilfe sich Java Objekte über ein Bluetooth Funknetzwerk übertragen lassen. Ein solcher Mechanismus findet Anwendung, wenn man gezwungen ist, Objekte über ein Netzwerk transportieren zu müssen, wie es in grösseren Implementierungen der Fall ist. Dies ist bspw. dann notwendig, wenn Daten aus einer Datenbank an ein bestimmes Endgerät oder einen Serverprozess auf einer physisch differenten Maschine zu senden sind.

Die Anwendungsentwicklung – im Bereich der Datenübertragung – wird mit der entwickelten Bibliothek merklich vereinfacht. Der Programmierer wird lediglich angehalten, in seinen Entitätsobjekten eine vorgegebene Schnittstelle zu implementieren. Sofern die Schnittstelle implementiert wurde, ist die Bibliothek in der Lage, das gegebene Objekt selbstständig zu übertragen.

Diese Arbeit bildet *keinen* Serialisierungsmechanismus nach, auch wenn es auf Grund der Schnittstellendefinition danach aussehen mag. Die Implementierung der Serialisierung muss vom Programmierer selbst durchgeführt werden.

1.3 Anwendungsszenarien

Die Anwendungsszenarien der Bibliothek sind vielfältig. Sie kann prinzipiell bei jeder Art von Objektdatenübertragung eingesetzt werden.

Eine konkrete Anwendung wäre bspw. eine Applikation, die mit einem Kühlschrank interagiert. Die Software des Kühlschranks verwaltet einen „Einkaufszettel“ mit Hilfe von Produktobjekten. Diese Objekte können mit Hilfe der Bibliothek leicht auf ein mobiles Endgerät übertragen werden, das im Geschäft als „Einkaufszettel“ dient.

Eine andere Anwendungsmöglichkeit wäre z. B. die Übertragung von Punktobjekten, die von einer Landkartenapplikation ausgewertet werden.

1.4 Technische Randbedingungen

Der Einsatz von Bluetooth sowie der *Java 2 Platform, Micro Edition* waren technische Randbedingungen der Arbeit. Um eine maximale Transparenz und Kosteneffizienz zu erreichen, sind, zur Realisierung des Projekts, bestehende Technologien aus dem Open Source Bereich eingesetzt worden.

1.5 Bezugsquellen

Die vorliegende Arbeit, sowie deren zugehöriger Quellcode kann unter der URL http://oop.coffeecrew.org/ bezogen werden.

An dieser Stelle soll speziell auf die verfügbare JavaDoc hingewiesen werden, die direkt unter der URL http://oop.coffeecrew.org/doc/ erreicht werden kann. Sie ist zusätzlich mit Beispielen versehen worden, um die Einarbeitungszeit des Entwicklers zu verkürzen.

Sämtliche Beispielprogramme sind unter der angegebenen Adresse verfügbar. Die Webseite ist zusätzlich auf der beigelegten CD-ROM offline verfügbar gemacht.

KAPITEL 2

Technische Grundlagen

In diesem Kapitel werden die zur Umsetzung, bzw. zum Verständnis der vorliegenden Arbeit, notwendigen Technologien bzw. Spezifikationen kurz angesprochen. Ist der Leser mit den folgenden Technologien bereits vertraut, kann dieses Kapitel problemlos übersprungen werden. Die Technologien werden nicht in vollem Umfang diskutiert, lediglich der jeweils relevante Teil wird herausgegriffen und kurz zusammengefasst bzw. gegeneinander abgegrenzt. Ausgehend davon, dass die zukünftige Entwicklung im Bereich mobiler Endgeräte nicht stagniert, wird speziell nur auf die zum Zeitpunkt der Drucklegung aktuelle Version der entsprechenden Spezifikation eingegangen. Die einzige Ausnahme stellt die Bluetooth-Spezifikation dar, die lediglich in der Version 1.2 vorgestellt wird, da die Mehrzahl der aktuell verfügbaren Endgeräte lediglich diese Version unterstützt. Die aktuelle Version (zum Zeitpunkt der Drucklegung v2.0) kann unter `https://www.bluetooth.org/spec/` bezogen werden. Die ältere Version 1.1 der Bluetooth-Spezifikation, auf die der JSR-82 aufsetzt, kann unter der selben URL angefordert werden.

Die angesprochenen Technologien und Spezifikationen umfassen dabei:

- Connected Limited Device Configuration (Version 1.0 siehe [T+00], Version 1.1 siehe [T+03])
- Mobile Information Device Profile (Version 1.0 siehe [Van00], Version 2.0 siehe [VW02])
- Bluetooth (Spezifikation v1.2 siehe [Blu03])
- Java APIs for Bluetooth Wireless Technology (JSR-82) [M+05]

2.1 Bluetooth

Die im Folgenden beschriebenen Grundlagen beziehen sich auf die Bluetooth-Spezifikation in der Version 1.2. Die Änderungen im Vergleich zur Version 1.1 beziehen sich hauptsächlich auf die Reduktion der Interferenzen mit anderen Funktechnologien und sind deshalb minimal. Die Version 1.1 hätte von der Bluetooth SIG speziell angefordert werden müssen, was somit vermieden werden konnte, da die Version 1.2 zum Zeitpunkt der Drucklegung ohne Registrierung verfügbar war.

Die Bluetooth Funktechnologie ist ein Kurzstreckenfunkverfahren, das primär dazu entwickelt wurde mobile Endgeräte kabellos zu verbinden. Das Bluetooth Grundsystem besteht aus Basisband, Sende-/Empfangseinheit und dem zugehörigen Protokollstapel[1]. Das Kurzstreckenfunkverfahren wird im weltweit lizenzfreien 2.4 GHz ISM[2] Band betrieben. Um die Komplexität der Sende- und Empfangseinheit gering zu halten und somit die Kosten zu minimieren, wird als Modulationsverfahren dass *Gaussian Frequency Shift Keying (GFSK)*[3] eingesetzt (Vgl. : [Blu03, Architecture, S. 13; PDF, S. 89]).

Die Reichweite des Bluetooth-Kurzstreckenfunks ist, wie bei jeder Funktechnologie, abhängig von der Sendeleistung des Geräts. Die Sendeleistung – und somit auch die Reichweite – des Funks wird in drei Klassen eingeteilt:

KLASSIFIKATION	REICHWEITE	SENDELEISTUNG
Klasse 1	100 m	100 mW
Klasse 2	40 m	2.5 mW
Klasse 3	10 m	1 mW

Tabelle 2.1: Reichweiten der einzelnen Bluetooth Klassen (Vgl. : [Wik06b])

Die Datenübertragungsraten sind abhängig von der Spezifikation bzw. der Bluetooth-Version des jeweiligen Produkts. Die Übertragungsrate hat sich während der ersten drei Versionen nicht gesteigert, sie wurde erst mit der Bluetooth-Spezifikation Version 2.0 erhöht. Ein Überblick über die Übertragungsraten, sowie die grossen Änderungen/Probleme, der jeweilgen Bluetooth-Versionen ist in [Tab. : 2.2, S. 7] dargestellt.

Der Datentransport innerhalb der Bluetooth-Architektur sowie sämtliche Betriebsmodi folgen dem selben generischen Ansatz. Dieser generische Ansatz ist mit Hilfe einer Schichtenarchitektur realisiert, die in [Abb. : 2.1, S. 7] dargestellt, und mit dem OSI-Schichtenmodell vergleichbar ist.

[1] Engl. : baseband, transceiver, protocolstack
[2] *Industrial, Scientific, and Medical Band* [Wik06c]
[3] Illustration: [Bec05, Folie 9]

BLUETOOTH-VERSION	MAXIMALE DATEN-ÜBERTRAGUNGS-RATE	ÄNDERUNG/PROBLEM
1.0 und 1.0B	723.2 Kbit/s	Enthält Sicherheitsprobleme
1.1	723.2 Kbit/s	Indikator für die Signalstärke hinzugefügt *Received Signal Strength Indicator* (RSSI)
1.2	723.2 Kbit/s	*Adaptive Frequency-Hopping spread spectrum* (AFH) eingeführt; reduziert Interferenzen mit anderen Funktechnologien (z. B. WLAN)
2.0	2.1 Mbit/s	Etwa dreifache Datenübertragungsgeschwindigkeit durch *Enhanced Data Rate* (EDR)

Tabelle 2.2: Datenübertragungsraten von Bluetooth (Vgl. : [Wik06b, Versionen])

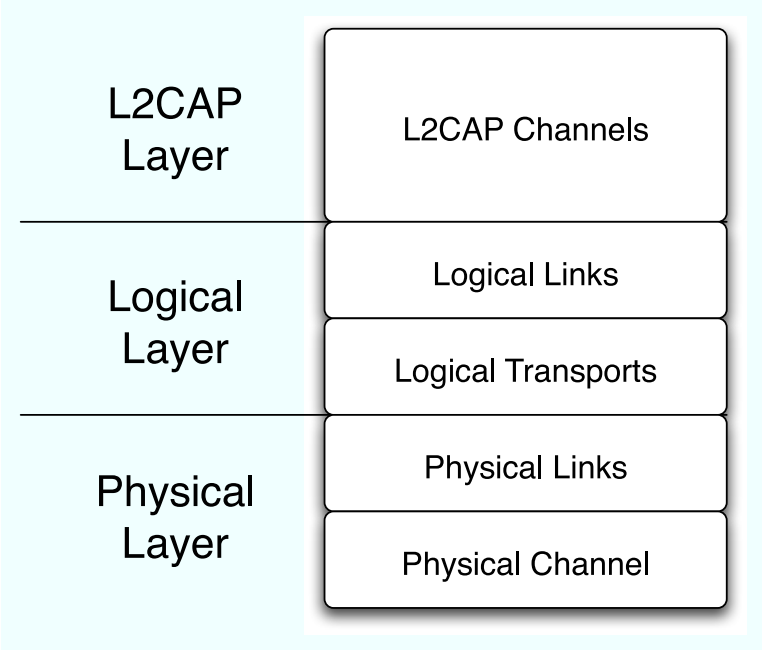
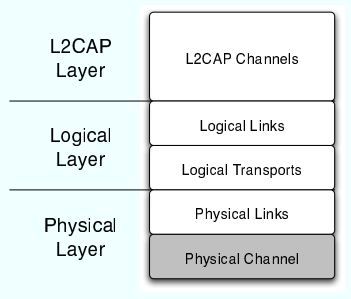


Abbildung 2.1: Allgemeine Datentransport Architektur von Bluetooth (Vgl. : [Blu03, Architecture, S.25; PDF, S. 101])

2.1.1 Physical Layer

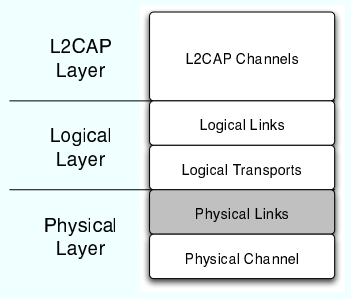
Der Physical Layer bestimmt die Art und Weise wie die Daten übertragen werden, bzw. wie die Übertragung der Daten stattzufinden hat. Er spezifiziert bspw. was notwendig ist um Kollisionen zu vermeiden, oder welches Verfahren verwendet wird um parallele Operationen zu unterstützen.

2.1.1.1 Physical Channel



Auf der untersten Ebene dieser Architektur befindet sich der *Physical Channel*. Er wird von zwei Bluetooth Geräten zur Kommunikation genutzt. Um miteinander kommunizieren zu können, müssen beide Sende- und Empfangseinheiten in Reichweite sein und auf die selbe Funkfrequenz eingestellt werden. Um unbeabsichtigte Kollisionen zu vermeiden wird der Kommunikation ein Zugriffscode angefügt. Dies verhindert Kollisionen, würden mehrere Geräte auf die selbe Frequenz eingestellt werden. In der Spezifikation sind vier physische Kanäle definiert, von denen jeder für einen bestimmten Anwendungszweck definiert und optimiert wurde. Um parallele Operationen zu unterstützen, wird das *Time Division Multiplexing*[4] Verfahren verwendet. Durch diesen Mechanismus wird es ermöglicht, dass das Gerät während einer bestehenden Kommunikation auffindbar und zugänglich ist. Die Spezifikation nimmt weiterhin an, dass das Gerät lediglich in der Lage ist, sich ausschliesslich mit einem Übertragungskanal zu verbinden (Vgl.: [Blu03, Architecture, S. 32; PDF, S. 108]).

2.1.1.2 Physical Links



Der *Physical Link* repräsentiert die Basisbandverbindung[5] zwischen den Endgeräten. Eine solche physische Verbindung ist immer mit genau einem physischen Kanal verknüpft, obwohl ein physischer Kanal mehr als eine physische Verbindung unterstützen kann. Die physische Verbindung ist innerhalb eines Bluetooth-Systems lediglich ein virtuelles Konzept. Innerhalb eines Bluetooth-Pakets existiert kein Feld, welches es ermöglichen würde die physische Verbindung direkt zu identifizieren. Die physische Verbindung kann stattdessen über den logischen Transportkanal[6] identifiziert werden. Physische Verbindungen

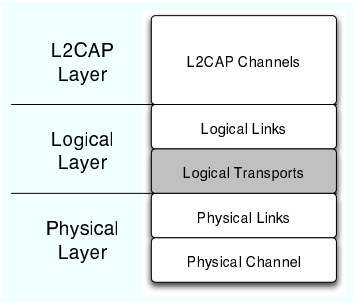
[4] Siehe [Wik06d]
[5] Frequenzbereich des Nutzsignals (siehe [Wik06a])
[6] engl.: *Logical Transport*, siehe [Kapitel 2.1.2.1, S. 9]

haben üblicherweise gemeinsame Eigenschaften, die auf alle logischen Transportkanäle – die zu dieser Verbindung gehören – angewandt werden. Beispiele solcher Eigenschaften sind die Verschlüsselung und Leistungssteuerung *(Power Control)* des Geräts. Soll eine Übertragung über mehrere physische Verbindungen hinweg ausgelöst werden *(Broadcast)*, werden die Sendeparameter entsprechend angepasst, sodass auf allen physischen Verbindungen gesendet werden kann (Vgl. : [Blu03, Architecture, S. 37; PDF: S. 113]).

2.1.2 Logical Layer

Der *Logical Layer* definiert, wie die logischen Transportverbindungen physisch zugeordnet werden.

2.1.2.1 Logical Transports



Zwischen *Master* und *Slave* – *Master* und *Slave* ist das Analogon zum Client-/Server-Modell bei Protokollen wie z. B. FTP – können verschiedene Typen von logischen Transportverbindungen (*Logical Transports*) hergestellt werden. Dabei ist es jedem Gerät möglich, den *Master* oder *Slave* Status einzunehmen; der Status kann sogar während der bestehenden Verbindung gewechselt werden. Logische Transportverbindungen werden von aktiven physischen Verbindungen getragen und sind in der Lage, verschiedene Arten logischer Verbindungen zu beinhalten (Vgl. : [Blu03, Architectur, S. 39f; PDF, S. 115f]).

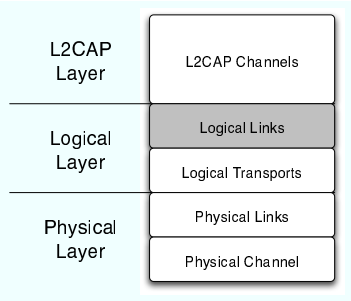
In der Spezifikation sind fünf logische Transportverbindungen zwischen *Master* und *Slave* definiert:

- *Synchronous Connection-Oriented (SCO) logical transport*
- *Extended Synchronous Connection-Oriented (eSCO) logical transport*
- *Asynchronous Connection-Oriented (ACL) logical transport*
- *Active Slave Broadcast (ASB) logical transport*
- *Parked Slave Broadcast (PSB) logical transport*

Die synchronen Transportverbindungen stellen hierbei Punkt-zu-Punkt-Verbindungen von einem *Master* zu einem *Slave* dar. Sie unterstützen typischerweise zeitabhängige Informationen wie z. B. Sprache. Das Master-Gerät regelt hierbei die Verbindung indem es reservierte *Slots* in regelmäßigen Abständen benutzt. Die ACL Verbindungen stellen ebenfalls eine

Punkt-zu-Punkt-Verbindung zwischen einem *Master* und einem *Slave* dar. Der *Master* kann die ACL Verbindung zu einem beliebigen *Slave* – inklusive der *Slaves* die bereits eine bestehende synchrone Verbindung haben – herstellen, indem er die *Slots* benutzt, die nicht für die synchrone Verbindung reserviert sind. Eine genaue Beschreibung der einzelnen Transportverbindungen ist unter [Blu03, Baseband Specification, S. 85ff; PDF, S.243ff] zu finden.

2.1.2.2 Logical Links



Um die verschiedenen Anforderungen einer Applikation bezüglich dem Datentransport abzudecken, ist eine Vielzahl logischer Verbindungen (*Logical Links*) verfügbar. Jede logische Verbindung wird mit einer logischen Transportverbindung assoziiert, die über verschiedene Charakteristika verfügt. Diese beinhalten z. B. Flusskontrolle, Sequenznummerierung sowie die Ablaufkoordination *(Scheduling)*. Innerhalb einer logischen Transportverbindung kann die logische Verbindung anhand des *Logical Link Identifiers (LLID)* erkannt werden, der im Header eines Basisbandpakets zu finden ist (Vgl. : [Blu03, Architectur, S. 46f; PDF, S. 122f]).

Die folgenden fünf logischen Verbindungen sind in der Spezifikation definiert (Vgl. : [Blu03, Baseband Specification, S. 95ff; PDF, S. 253ff])

Link Control (LC) Diese logische Verbindung trägt Kontrollinformationen der unteren Ebenen, wie z. B. *Acknowledgement/Repeat Request (ARQ)*, Flusskontrolle und Nutzlastcharakterisierungen.

ACL Control (ACL-C) Die ACL-C Verbindung trägt Kontrollinformationen, die zwischen den *Link Managern (LM)*[7] der Endgeräte ausgetauscht werden müssen. Der Austausch dieser Informationen geschieht über das *Link Manager Protokoll (LMP)*.

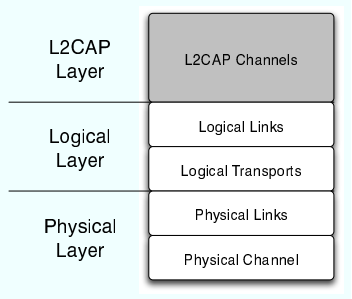
User Asynchronous/Isochronous (ACL-U) Die ACL-U Verbindung trägt asynchrone, sowie isochrone L2CAP Benutzerdaten. Diese Nachrichten können in einem oder mehreren Paketen übermittelt werden. Bei fragmentierten Daten wird im Header der LLID auf den entsprechenden Wert gesetzt.

User Synchronous (SCO-S) Die SCO-S Verbindung transportiert synchrone Benutzerdaten.

User Extended Synchronous (eSCO-S) Die eSCO-S Verbindung transportiert ebenfalls synchrone Benutzerdaten.

[7] Der LM ist verantwortlich für die Erzeugung, Modifikation und Freigabe logischer Verbindungen [Blu03, Architecture, S. 23; PDF, S. 99].

2.1.3 L2CAP Layer



Das *Logical Link Control and Adaption Protocol (L2CAP)* unterstützt Multiplexing zu Protokollen höherer Ebenen, Paketsegmentierung und -reassemblierung, sowie das Übermitteln von *Quality of Service*[8] Informationen. Applikationen und Service-Protokolle kommunizieren mit dem L2CAP-Layer über ein sogenanntes *Channel-Orientated Interface*, um Verbindungen zu anderen Geräten herzustellen. Die entsprechenden Endpunkte werden über einen *Channel Identifier (CID)* gekennzeichnet, dessen Wert vom L2CAP-Layer zugewiesen wird. Das Hauptaugenmerk des L2CAP-Layers liegt aber auf dessen Multiplexing-Fähigkeit. Er ist dafür zuständig, dass die über das *Channel Interface* ankommenden Daten (*Service Data Units [SDUs]*) auf die ACL-U Verbindungen verteilt werden. Ist ein HCI[9] vorhanden, hat der L2CAP-Layer ebenfalls dafür Sorge zu tragen, dass die Daten entprechend der Puffergrösse des Basisbands segmentiert werden (Vgl. : [Blu03, Architectur, S. 48; PDF, S. 124]).

2.1.4 Bluetooth Sicherheit

Die Bluetooth-Spezifikation beschreibt das Sicherheitssystem, das bereits auf der Verbindungsebene *(Link Layer)* angewendet wird, sehr ausführlich. Verschlüsselung, Authentifizierung sowie Schlüsselerzeugungsschemata und die Erzeugung von Zufallszahlen sind spezifiziert. Die Authentifizierungs- und Verschlüsselungsalgorithmen müssen in jedem Gerät eine äquivalente Implementierung aufweisen. Um die Sicherheit auf der Verbindungsebene zu realisieren, werden die vier in [Tab. : 2.3, S. 11] aufgeführten Entitäten verwendet.

ENTITÄT	GRÖSSE
Bluetooth Device Address (BD_ADDR)	48 Bit
Privater Schlüssel des Benutzers (Authentifizierung)	128 Bit
Privater Schlüssel des Benutzers (Verschlüsselung)	8-128 Bit
Pseudozufallszahl (RAND)	128 Bit

Tabelle 2.3: In Authentifizierung und Verschlüsselung involvierte Entitäten (Vgl. : [Blu03, Security Specification, S. 749; PDF, S. 907 Tabelle 1.1])

Die Authentifizierung beider Geräte erfolgt in zwei Schritten, das heisst es wird eine gegenseitige Authentifizierung beider Geräte vorgenommen. In direktem Anschluss der Authenti-

[8] Kurz: QoS, bezeichnet ein Verfahren, das einem (Paket-)Dienst eine Mindestbandbreite zusichert.
[9] Das *Host Controller Interface (HCI)* befindet sich zwischen logischem und L2CAP-Layer und stellt eine einheitliche Schnittstelle zum Bluetooth-Controller dar.

fizierung des Geräts A bei Gerät B wird die Authentifizierung in entgegengesetzter Richtung vorgenommen[10] (Vgl. : [Blu03, Security Specification, S. 758; PDF, S. 914]).

Benutzerdaten können mittels des angebotenen Verschlüsselungsalgorithmus geschützt werden. Es werden jedoch lediglich die Nutzdaten des zu sendenden Pakets verschlüsselt. Der Header des Pakets bleibt unangetastet. Der Algorithmus implementiert eine Stromchiffrierung mit Hilfe der SAFER+ Methode, die frei erhältlich ist. Sie ist eine verbesserte Version des SAFER[11] Algorithmus [Blu03, Security Specification, S. 777ff; PDF, S. 935ff].

2.1.5 Generic Access Profile (GAP)

Sinn und Zweck des *Generic Access Profiles (GAP)* ist es, Definitionen, Empfehlungen sowie allgemeine Anforderungen in Bezug auf verschiedene Betriebsmodi und Zugriffsprozeduren zu beschreiben, die von Transport- und Applikationsprofilen verwendet werden sollten. Es wird weiterhin beschrieben, wie sich die Geräte im Ruhe- und im Verbindungsaufbauzustand verhalten sollten. Speziell wird der Fokus hierbei auf die Gerätesuche, Erzeugung der Verbindung sowie die Sicherheitsprozeduren gelegt. Das Schichtenmodell des GAP folgt dem in [Abb. : 2.2, S. 12] dargestellten Aufbau [Blu03, Generic Access Profile, S. 179ff; PDF: S. 1127ff].

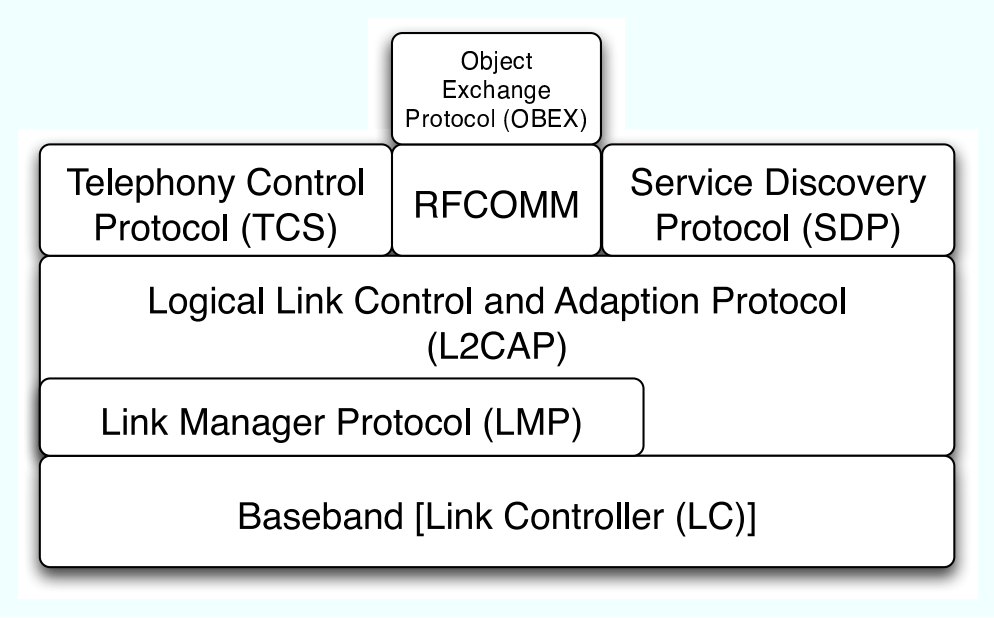


Abbildung 2.2: GAP Schichtenmodell (Vgl.: Abbildung 2.1: Profile stack covered by this profile, S. 181 [Blu03])

Die Spezifikation des GAP beschäftigt sich hauptsächlich damit, zu beschreiben, welchem Zweck die unteren Schichten des Bluetooth Protokollstapels dienen (LC und LMP). Um Si-

[10] Sogenanntes *Challenge-Response* Verfahren (siehe [FS06, S. 56])
[11] Siehe [Sch96, S. 392ff]

cherheitsbezogene Alternativen zu diskutieren, wurden ebenfalls höhere Ebenen mit einbezogen (L2CAP, RFCOMM und OBEX) [Blu03, Generic Access Profile, S. 181; PDF: S. 1129].

In Bezug auf eine Suchanfrage *(inquiry)* kann sich ein Bluetooth-Gerät in drei verschiedenen Zuständen befinden. Die möglichen Zustände sind in [Tab. : 2.4, S. 13] abgedruckt.

AUFFINDBARKEITSZUSTAND	BESCHREIBUNG
Non-discoverable mode	Ist das Gerät in diesem Zustand, so kann es von einer Suchanfrage nicht gefunden werden. Ein Bluetooth-Gerät in diesem Zustand wird als "Stilles Gerät" *(silent device)* bezeichnet.
Limited discoverable mode (LIAC)	Das Gerät ist für einen beschränkten Zeitraum oder bis zum Eintritt eines bestimmten Ereignisses auffindbar. Es sollte allerdings nicht länger als eine vordefinierte Zeit in diesem Zustand bleiben *(timeout)*. Ist ein Gerät in diesem Zustand, antwortet es auf Suchanfragen die den *Limited Dedicated Inquiry Access Code (LIAC)* verwenden.
General discoverable mode (GIAC)	Das Gerät befindet sich in einem dauerhaft auffindbaren Zustand. Ist ein Gerät in diesem Zustand, antwortet es auf Suchanfragen die den *General/Unlimited Inquiry Access Code (GIAC)* verwenden.

Tabelle 2.4: Auffindbarkeitszustände eines Bluetooth-Geräts [M+05, Generic Access Profile, S. 189ff; PDF, S.1137ff]

Die spezifizierten Werte der Zugriffskennzeichner LIAC und GIAC sind auf der Bluetooth Homepage[12] zu finden [Blu06].

2.1.6 Bluetooth Implementierungen

Um Bluetooth auf einem Gerät verwenden zu können ist es notwendig, dass eine – zur Hardware passende – Implementierung eines Bluetooth-Stacks bereitgestellt wird. Für die mobilen Endgeräte ist in den meisten Fällen eine Bluetooth Implementierung des jeweiligen Herstellers auf dem Gerät vorhanden. Um allerdings auf einem Desktop-System Bluetooth-Unterstützung zu erhalten, muss i. d. R. ein entsprechender Stack nachträglich installiert werden.

[12] Siehe https://www.bluetooth.org/

2.2 Java 2 Micro Edition (J2ME)

Die *Java 2 Platform, Micro Edition (J2ME)* ist eine Plattform, die speziell auf die Bedürfnisse von *Embedded Devices* zugeschnitten ist. Die J2ME Architektur definiert Konfigurationen, Profile sowie optionale Pakete, mit deren Hilfe es möglich ist, Java Laufzeitumgebungen zu entwickeln, die eine möglichst breite Palette an Endgeräten des jeweiligen Zielmarktes abdecken. Konfigurationen bestehen aus der Kombination einer virtuellen Maschine (VM) sowie einem zugehörigen minimalen Set an Klassenbibliotheken. Um eine komplette Laufzeitumgebung für domänenspezifische Endgeräte zu schaffen, muss die Konfiguration um ein Set zusätzlicher Klassenbibliotheken erweitert werden, die in einem Profil oder optionalen Paket definiert sind. Die J2ME Plattform lässt sich, wie in [Abb. : 2.3, S. 14] dargestellt, im Java Umfeld einordnen (Vgl. : [Sun06]).

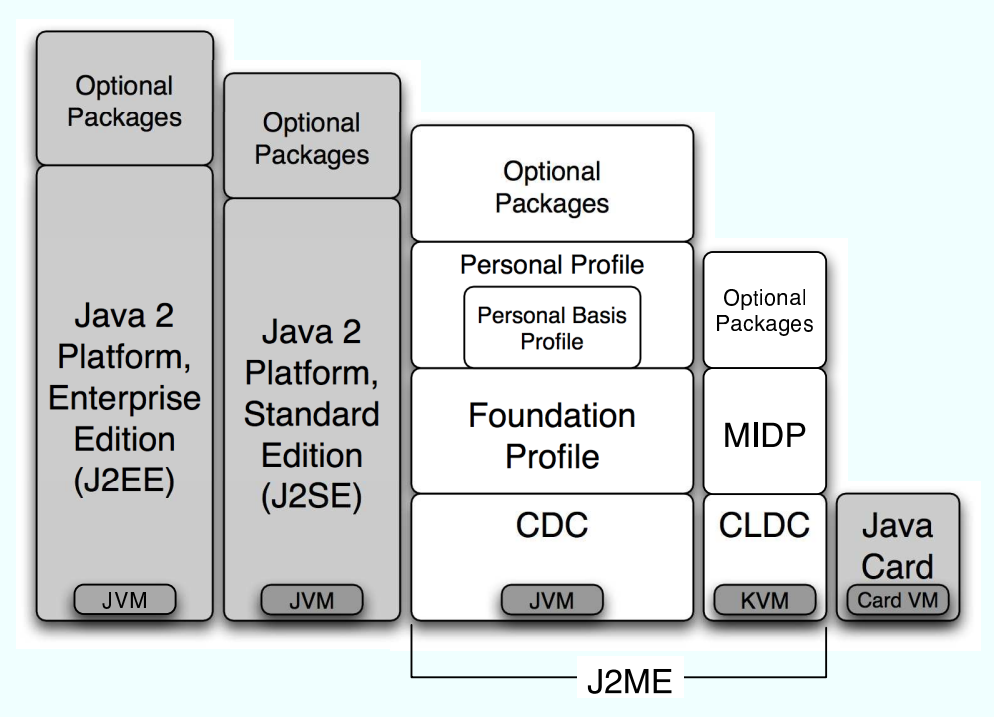


Abbildung 2.3: Einordnung von J2ME im Java Umfeld (Vgl. : [Sun06])

Die J2ME Architektur folgt also prinzipiell dem in [Abb. : 2.4, S. 15] dargestellten Aufbau. Die einzelnen Schichten der Architektur (CLDC [Kapitel 2.3, S. 15] und MIDP [Kapitel 2.4, S. 18]) werden im Folgenden genauer beschrieben.

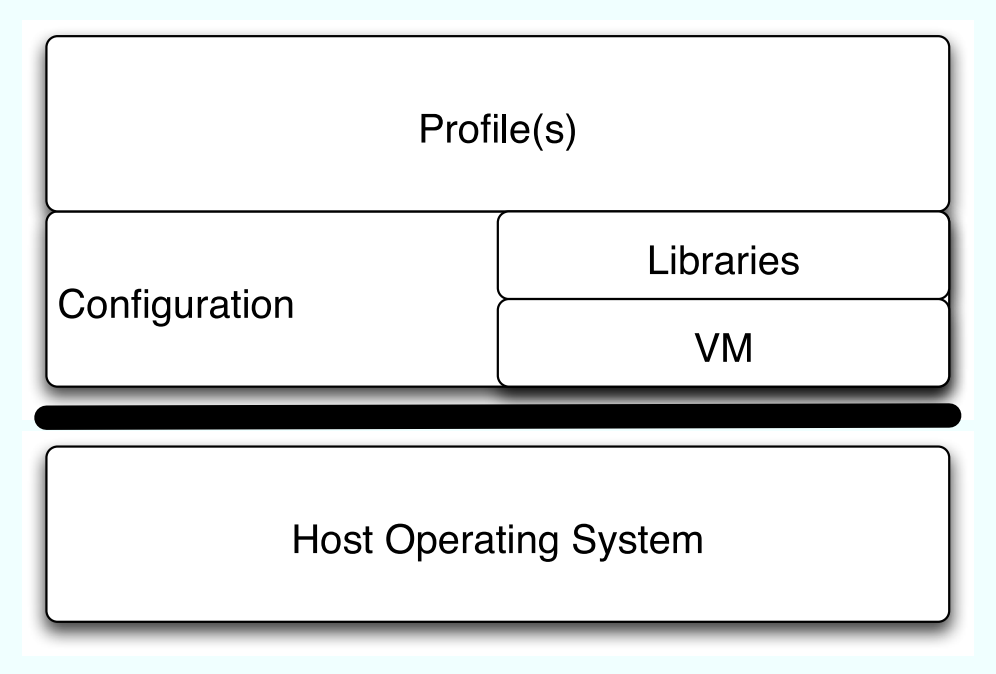
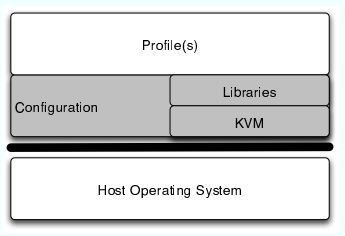


Abbildung 2.4: J2ME Architektur (Vgl.: [T+03, Abb. 1, S. 28])

2.3 Connected Limited Device Configuration (CLDC)



Die *Connected Limited Device Configuration (CLDC)* stellt, zusammen mit dem *Mobile Information Device Profile (MIDP)*[13], die Grundlage der J2ME dar. Wie nahezu alle Standards rund um Java, ist die CLDC ebenfalls Ergebnis eines *Java Specification Request (JSR)*. Die CLDC 1.0 wird als JSR-30 [T+00], die Version 1.1 als JSR-139 [T+03] geführt[14]. Die CLDC 1.1 ist nicht als komplette Neuerung, sondern lediglich als Erweiterung der CLDC 1.0 zu sehen. Das bedeutet, dass die CLDC 1.1, mit ein paar wenigen Ausnahmen wie z. B. der Unterstützung für Gleitpunktoperationen, abwärtskompatibel zur CLDC 1.0 ist. Die CLDC Spezifikation hat zum Ziel, eine Entwicklungsplattform für netzwerkfähige, jedoch in ihren Ressourcen limitierte, portable, Geräte zu standardisieren. Solche Geräte sind bspw. ein Mobiltelefon oder ein Personal Digital Assistant (PDA). Eine Konfiguration der J2ME Plattform spezifiziert hierbei die Basismenge der Java Programmiersprache sowie die Funktionalität der zugehörigen VM. *Sun Microsystems, Inc.* bietet eine Referenzimplementierung einer solchen virtuellen Maschine an, die in der Spezifikation als *K Virtual Machine (KVM)* bezeichnet wird, aber mittlerweile unter der Bezeichnung *CLDC HotSpot Implementation Virtual Machine*[15] bekannt ist, die die 8-10 fache Geschwindigkeit der KVM erreicht. Die CLDC dient als Basis für

[13] Siehe [Kapitel 2.4, S. 18]
[14] Alle JSRs sind abrufbar unter: http://www.jcp.org/en/home/index
[15] Siehe http://java.sun.com/products/cldc/

ein oder mehrere Profile, die ein dem Gerät entsprechendes API anbieten (Vgl.: [T+03, S. xi]).

Da die CLDC darauf ausgerichtet wurde, eine möglichst grosse Anzahl verschiedener Geräte zu unterstützen, besitzt sie, ausser der Mindestanforderung an den Speicher, keine weiteren Anforderungen bzgl. der zu Grunde liegenden Hardware. Typischerweise besitzen die angesprochenen Geräte jedoch limitierte Ressourcen:

- 160 Kilobyte nichtflüchtiger Speicher
- 32 Kilobyte flüchtiger Speicher
- 16 Bit oder 32 Bit Prozessor
- Möglichkeit, eine Verbindung zu einem Netzwerk herzustellen

In keinem Fall ist die CLDC mit der *Connected Device Configuration (CDC)*[16] zu verwechseln, die einen erheblich grösseren Umfang als die CLDC bietet. Innerhalb der CDC ist z. B. der Mechanismus der Serialisierung und Deserialisierung von Objekten implementiert, was in der CLDC nicht der Fall ist. Die CDC ist für portable High-End-Geräte entwickelt worden.

Die Implementierung der CLDC kann keine Kenntnis eines Dateisystems auf dem Endgerät voraussetzen. Viele, in ihren Ressourcen eingeschränkte Geräte, verfügen nicht über die Kenntnis eines Dateisystems oder Ähnlichem, welches es ermöglicht dynamisch bezogene Daten persistent auf dem Gerät zu speichern. Die Implementierung muss lediglich in der Lage sein, die Applikation zu laden und sie direkt nach ihrer Ausführung wieder zu verwerfen. Verfügt das Gerät allerdings über einen Persistenzmechanismus, ist es Aufgabe des Betriebssystems, einen Mechanismus zur Verwaltung der Applikationen zur Verfügung zu stellen. Auf Grund der variablen Bandbreite verfügbarer Geräte, ist ein solcher Mechanismus allerdings stark abhängig vom Gerät selbst und wird aufgrunddessen nicht von der Spezifikation behandelt (Vgl.: [T+03, S. 29]).

2.3.1 Generic Connection Framework (GCF)

In einer einheitlichen, erweiterbaren Art und Weise bietet das *Generic Connection Framework (GCF)* die Möglichkeit, auf Ein- und Ausgabeoperationen sowie Netzwerkressourcen zuzugreifen. Anstatt auf eine Vielzahl verschiedener Abstraktionsmechanismen für jeden Kommunikationstyp zurückzugreifen, wird eine Menge zugehöriger Abstraktionen auf Ebene der Applikationsprogrammierung benutzt. Dieser generische Mechanismus besitzt folgende Form:

```
Connector.open("<protocol>:<address>;<parameters>");
```

[16] Siehe [C+05]; Einordnung siehe [Abb.: 2.3, S. 14]

Das GCF, das in der CLDC spezifiziert wird, definiert allerdings kein zwingend zu unterstützendes Netzwerkprotokoll. Sie fordert ebensowenig eine Implementierung bereits bestehender Protokolle. Sie bietet jedoch ein erweiterbares Rahmenwerk, das von J2ME Profilen, wie z. B. dem MIDP[17] angepasst werden kann. Die tatsächliche Implementierung der entsprechenden Protokolle erfolgt also auf Ebene des Profils (Vgl.: [T+03, S. 56ff]).

2.3.2 CLDC Sicherheit

Das von der J2SE bekannte, sehr mächtige, Sicherheitsmodell konnte aufgrund des Code-Umfangs nicht übernommen werden, da es die spezifizierten Speichergrenzen der CLDC bei weitem überschreitet. Das Sicherheitsmodell ist deshalb in drei verschiedene Ebenen eingeteilt.

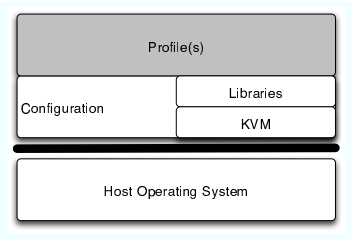
Low-level security Der Begriff *low-level security* ist auch unter der Bezeichnung *Virtual machine security* bekannt. *Low-level security* stellt sicher, dass Applikationen, die in der VM ausgeführt werden, die Semantik der Java Programmiersprache einhalten. Sie stellt weiterhin sicher, dass jedweder fehlerhafte oder schädliche Programm-Code nicht in der Lage ist, das Gerät zu beschädigen oder es zum Absturz zu bringen (Vgl. : [T+03, S. 30]).

Application-level security *Application-level security* bedeutet, dass die Applikation nur auf Ressourcen, Bibliotheken und andere Komponenten zugreifen kann, die von der Java Laufzeitumgebung und dem Gerät genehmigt worden sind. Realisiert wird das durch ein sogenanntes Sandbox-Modell. Es muss gewährleisten, dass Java Applikationen nicht aus ihrer Sandbox ausbrechen können. Das CLDC Sandbox-Modell verbietet ebenfalls ein herunterladen beliebiger neuer Bibliotheken, die nicht zum Umfang der Java Bibliotheken der CLDC oder des Herstellers gehören. Das Ausführen nativen Codes ist ebenfalls verboten (Vgl. : [T+03, S. 31]).

End-to-end security Mit *end-to-end security* bezeichnet man ein Modell, das gewährleistet, dass die über ein Netzwerk gesendeten Daten – während ihres Transports durch diese Netze – geschützt werden. Um dies zu gewährleisten ist ein Verschlüsselungs- oder ein äquivalenter anderer Schutzmechanismus notwendig. Aufgrund des breiten Spektrums verfügbarer Endgeräte kann allerdings keine einheitliche Lösung des Problems geschaffen werden. Die Spezifikation deklariert die *end-to-end security* diesbezüglich als implementierungsabhängig (Vgl. : [T+03, S. 32]).

[17] Siehe [Kapitel 2.4, S. 18]

2.4 Mobile Information Device Profile (MIDP)



Das *Mobile Information Device Profile (MIDP)* spezifiziert ein Profil, welches auf die Fähigkeiten eines kleinen mobilen Endgeräts – einem sogenannten *Mobile Information Device (MID)* – abgestimmt ist. Das MIDP 2.0 ist das Ergebnis des JSR-118, basiert auf dem MIDP 1.0 (JSR-37) und ist zu diesem abwärtskompatibel, sodass MIDlets[18], die für das MIDP 1.0 geschrieben wurden, ebenfalls auf dem MIDP 2.0 lauffähig sind[19]. Das MIDP spezifiziert eine auf der CLDC[20] aufgesetzte Schicht. MIDP 2.0 setzt mindestens die Version 1.0 der CLDC voraus, jedoch wird angenommen, dass die meisten MIDP 2.0 Implementierungen auf der CLDC 1.1 aufsetzen werden. Da MIDs typischerweise eine sehr breite Spanne an Ressourcen zur Verfügung stellen, ist die Anzahl der Programmierschnittstellen auf das Wesentliche reduziert worden, um eine möglichst hohe Portabilität zu gewährleisten. Aus diesem Grund sind einige Funktionsbereiche abgegrenzt worden. Die abgegrenzten Bereiche sind die systemnahen Programmierschnittstellen *(System-level APIs)* sowie die Sicherheit der virtuellen Maschine *(Low-level security)* (Vgl.: [VW02, S. 5f]).

Die spezifizierten Mindestanforderungen an das MID sind als zusätzliche Anforderungen zur CLDC zu betrachten. Die Anforderungen sind unterteilt in Hard- und Softwareanforderungen. Die für die vorliegende Arbeit relevanten Spezifikationspunkte sind das Netzwerk sowie der spezifizierte Persistenzmechanismus:

Netzwerk Die Netzwerkhardware muss eine Zwei-Wege-Funkverbindung (mit limitierter Bandbreite) zur Verfügung stellen. Um das Netzwerk API zu unterstützen, muss der Software lesender und schreibender Zugriff auf die Funkverbindung gewährt werden.

Persistenzmechanismus Ein Mechanismus, um auf den nichtflüchtigen Speicher zu schreiben und von ihm zu lesen, um die Anforderungen des *Record Management Systems*[21] abzudecken.

In der Spezifikation sind weitere Anforderungen an des Gerät definiert, die weitere Punkte wie Anzeige, Eingabemechanismen, Speicheranforderungen und Audioeigenschaften festlegen. Es wird weiterhin festgelegt, wie die Software-Landschaft des Geräts auszusehen hat. Dies umfasst z. B. einen minimalistischen Kernel sowie einen Mechanismus, der den Lebenszyklus der Applikation auf dem Gerät verwaltet (Vgl.: [VW02, S. 7ff]).

[18] Programme, die auf einem MID ausgeführt werden können.
[19] JSR-118: [VW02]; JSR-37 [Van00]
[20] Siehe [Kapitel 2.3, S. 15]
[21] Siehe [Kapitel 2.4.1, S. 19]

2.4.1 Record Management System (RMS)

Das *Record Management System (RMS)* ist ein Mechanismus, der es MIDlets ermöglicht, Daten persistent auf dem Gerät abzulegen. Der Mechanismus ist wie eine einfache datensatzorientierte Datenbank modelliert (Vgl.: [VW02, S. 463]).

2.4.1.1 Record Store

Ein *Record Store* ist eine Ansammlung an Datensätzen, die, über mehrere Aufrufe der Applikation hinweg, persistent erhalten bleiben. Die zu Grunde liegende Plattform ist für die Integrität der Datensätze verantwortlich. Sie ist auch für das Erzeugen des *Record Stores* verantwortlich, der an einem der Plattform spezifischen Ort liegt. Dieser Speicherort darf dem MIDlet, aus Sicherheitsgründen, nicht bekanntgegeben werden. Der Name des *Records Stores* wird allerdings auf Ebene des MIDlets festgelegt. Dieser Name muss innerhalb eines MIDlets eindeutig sein. Das RMS-API definiert keinen Locking-Mechanismus. Schreiboperationen auf das Datenbanksystem werden automatisch synchronisiert, was bedeutet, dass das API *Thread-Safe*[22] ist (Vgl.: [VW02, S. 463f]).

2.4.1.2 Records

Records sind als Byte-Arrays implementiert. Der Entwickler kann also `DataInputStream` und `DataOutputStream` sowie `ByteArrayInputStream` und `ByteArrayOutputStream` verwenden, um Daten in einen Datensatz zu schreiben bzw. von ihm zu lesen. Jeder *Record* kann innerhalb eines *Record Stores* anhand seiner eindeutigen Identifikationsnummer (ID) zugeordnet werden. Das bedeutet; wird ein Datensatz eingefügt, erhält er die ID „n“. Der nächste eingefügte Datensatz erhält die ID „n+1“ (Vgl.: [VW02, S. 464]).

Dieses Konzept der Enumeration wird konsequent verfolgt. Wird z. B. ein Datensatz gelöscht, wird die ID nicht wiederverwendet. Das bedeutet; sind drei Datensätze mit den IDs »1«, »2« und »3« eingefügt worden und wird der Datensatz mit der ID »2« gelöscht, erhält der nächste eingefügte Datensatz die ID »4« und nicht wie evtl. angenommen werden könnte, die ID »2«.

2.4.2 MIDP Sicherheit

Da sich das MIDP auf die Bereitstellung eines APIs für die Applikationsentwicklung bezieht, wurden die *System-level* APIs nicht weiter spezifiziert. Das heisst sie unterliegen der korrekten Implementierung und den Sicherheitsanforderungen des Herstellers. Es werden seitens

[22] Parallele Aufrufe einer Funktion auf einen Datensatz/-block führen nicht zu Inkonsistenzen dessen.

der Spezifikation explizit keine weiteren *Low-level* Sicherheitsmechanismen definiert (Vgl.: [VW02, S. 6]).

Da das MIDP jedoch auf der CLDC aufsetzt, erbt es die bereits in der CLDC vorhandenen Sicherheitsmechanismen. Das *System-level* API wird prinzipiell von der *Application-level security*, die *Low-level* Sicherheit von der gleichnamigen Sicherheit in der CLDC abgedeckt[23].

Das MIDP 1.0 zwingt jedes MIDlet dazu, innerhalb einer Sandbox ausgeführt zu werden, die jeglichen Zugriff auf sensitive Schnittstellen oder Funktionen des Geräts unterbindet. Dieses Konzept wird vom MIDP 2.0 ebenfalls für jedes nicht vertrauenswürdige *(untrusted)* MIDlet verlangt. Jede MIDP 2.0 Implementierung ist verpflichtet, Unterstützung für *untrusted* MIDlets anzubieten. Mit dem MIDP 2.0 ist das Konzept vertrauter *(trusted)* Applikationen eingeführt worden, das den Zugriff auf sensitive Schnittstellen erlaubt. Nicht vertrauenswürdige MIDlets müssen innerhalb der vom MIDP 1.0 spezifizierten Sandbox ausgeführt werden. Findet beim Ausführen eines solchen MIDlets ein Zugriff auf nicht vertrauenswürdige Schnittstellen statt, muss der Zugriff entweder verworfen, oder, nach einer explizit angeforderten Genehmigung des Benutzers, erlaubt werden. Das *trusted* Modell definiert im Gegensatz dazu drei verschiedene Interaktionsmuster, die als *User Permission Modes* bezeichnet werden. Sie ermöglichen es dem Benutzer, den Zugriff auf ein bestimmtes API zu erlauben oder zu verwehren (Vgl.: [VW02, S. 23ff]).

Die verschiedenen Zugriffsmodi, die vom Benutzer festgelegt werden können, sowie deren Gültigkeitszeitraum sind in [Tab.: 2.5, S. 20] festgehalten.

USER PERMISSION	GÜLTIGKEITSZEITRAUM
blanket	Gültig für jede Ausführung der MIDlet-Applikation bis sie deinstalliert oder die Rechte vom Benutzer geändert werden.
session	Gültig vom Start eines MIDlets, bis zu dessen Termination. Der "session" Modus ist verpflichtet, den Benutzer beim oder vor dem ersten Aufruf einer geschützten Funktion einen Dialog anzuzeigen, in dem der Benutzer entweder den Zugriff gestattet oder ablehnt. Bei einem erneuten Start des MIDlets muss diese Abfrage erneut erfolgen.
oneshot	Eine Interaktion mit dem Benutzer ist bei jedem Zugriff auf ein geschütztes API notwendig.

Tabelle 2.5: Gültigkeitszeitraum der Rechte eines MIDlets (Vgl.: [VW02, S. 26])

[23] Siehe [Kapitel 2.3.2, S. 17]

2.5 Java API for Bluetooth (JSR-82)

Um die Bluetooth Funktionalität von einem MIDlet aus ansprechen zu können, ist ein entsprechendes API notwendig. Der JSR-82[24] definiert ein solches API sowie dessen Architektur, um Drittherstellern die Möglichkeit zu geben, Programme zu entwickeln, die eine Bluetooth Funkverbindung zur gegenseitigen Kommunikation nutzen. Das mit dem JSR-82 entwickelte API, das auch unter dem Akronym JABWT *(Java API for Bluetooth Wireless Technology)* bekannt ist, baut auf der Grundlage der CLDC [Kapitel 2.3, S. 15] auf und basiert auf der Bluetooth Spezifikation in der Version 1.1[25]. Es ist als optionale Erweiterung eines MIDP [Kapitel 2.4, S. 18] zu sehen (Vgl. : [M+05, S. 11]).

Mit der Spezifikation wurde beabsichtigt, eine Grundlage zu schaffen, mit der es ermöglicht wird, weitere Profile zu entwickeln. Dazu werden APIs für das L2CAP und OBEX[26] Protokoll angeboten, auf deren Basis zukünftige Profile entwickelt werden können. Das API bietet folgende Dienste an:

1. Registrierung eines Dienstes
2. Gerät- und Dienstsuche
3. Möglichkeit, L2CAP und OBEX Verbindungen herzustellen.
4. Sichere Ausführung dieser Aktivitäten.

Da nicht alle Profile und Ebenen der Bluetooth Spezifikation übernommen werden können, beschränkt sich der Umfang des APIs lediglich auf Datendienste. Sprachdienste können nicht verarbeitet bzw. angesteuert werden (Vgl. : [M+05, S. 14]).

2.5.1 Anforderungen

Die vom JSR-82 definierten Anforderungen sind als Zusätze zu den bereits bestehenden Anforderungen der CLDC zu betrachten. Sie umfassen Anforderungen an die Spezifikation selbst, wie z. B. eine alleinige Abhängigkeit zu den CLDC Bibliotheken. Auf die Spezifikationsanforderungen soll jedoch nicht näher eingegangen werden. Es sind vielmehr die Anforderungen an das Gerät sowie die des zu Grunde liegenden Bluetooth-Systems von Interesse.

Hardware-Anforderungen Das JSR-82 API ist entwickelt worden, um auf Geräte mit folgenden Hardware-Charakteristika eingesetzt werden zu können.

[24] Siehe http://www.jcp.org/en/jsr/detail?id=82
[25] Version 1.1 kann unter https://www.bluetooth.org/spec/ angefordert werden.
[26] Siehe [Kapitel 2.5.8.2, S. 27]

- Mindestens 512 Kilobyte Speicher für die Java Plattform (ROM/Flash und RAM). Der für die Applikationen notwendige Speicher ist im Speicherkontingent nicht enthalten und muss additiv bereitgestellt werden.
- Bluetooth Kommunikations-Hardware, mit zugehörigem Stack und Funkmodul.
- Implementierung der J2ME CLDC APIs oder einem Superset derer, wie z. B. der CDC.

Bluetooth-Anforderungen Die Anforderungen an das zu Grunde liegende Bluetooth-System sind folgende:

- Das System muss das *Bluetooth Qualification Program* für mindestens das *Generic Access Profile*, das *Service Discovery Application Profile* und das *Serial Port Profile* durchlaufen haben.
- Die folgenden Schichten müssen, wie in der Bluetooth Spezifikation (Version 1.1) unterstützt werden und die Implementierung muss Zugriff auf diese haben.
  - *Service Discovery Protocol (SDP)*
  - RFCOMM[27] (Typ 1 Unterstützung)
  - *Logical Link Control and Adaption Protocol (L2CAP)*
- Eine vom System bereitgestellte Entität, das *Bluetooth Control Center (BCC)*, welches es einem Benutzer oder *Original Equipment Manufacturer (OEM)* erlaubt, spezifische Bluetooth-Parameter zu konfigurieren.

Die Unterstützung für das *OBject EXchange (OBEX)* Protokoll kann entweder bereits vom zu Grunde liegenden Bluetooth-System, oder durch Implementierung des JSR-82 APIs bereitgestellt werden [M+05, S. 14f].

2.5.2 Paketierung

In der Spezifikation sind zwei Pakete definiert:

- `javax.bluetooth`
- `javax.obex`

Auf Grund der Tatsache, dass das OBEX API unabhängig ist von Bluetooth, wird es in einem separaten Paket ausgeliefert. Eine CLDC ist somit in der Lage, entweder nur eines der beiden, oder beide auszuliefern. Das Paket `javax.bluetooth` beinhaltet die Bluetooth Basisbibliotheken, wohingegen das `javax.obex` Paket die Implementierung des OBEX APIs beinhaltet. Aus diesem Grund werden auch zwei verschiedene *Technology Compatibility Kits (TCK)* zur Verfügung gestellt, mit Hilfe derer die Implementierung gegenüber der Spezifikation getestet werden kann [M+05, S. 21].

[27] Siehe [Kapitel 2.5.8.1, S. 27]

2.5.3 Bluetooth Control Center

Bei Geräten die das JSR-82 API implementieren, kann es u. U. möglich sein, dass mehrere Applikationen zeitgleich ausgeführt werden. Die Notwendigkeit eines *Bluetooth Control Centers (BCC)* ergibt sich aus dem Wunsch, zu verhindern, dass eine Applikation eine andere nachteilig beeinflusst. Es ist als zentrale Anlaufstelle des lokalen Bluetooth-Geräts zu sehen. Die Implementierung des BCC kann entweder als native Applikation bzw. als Applikation mit separatem API oder als fixe Gruppe von Eigenschaften, die nicht vom Benutzer geändert werden können, umgesetzt werden. Das BCC ist nicht als Klasse oder Schnittstelle innerhalb des JSR-82 APIs spezifiziert, stellt jedoch eine wichtige Rolle für die Sicherheitsarchitektur des JSR-82 dar. Das BCC muss der JSR-82 Implementierung folgende Funktionen zur Verfügung stellen:

- Basissicherheitseinstellungen des Geräts, inklusive der in der Bluetooth-Spezifikation definierten Sicherheitsmodi.
- Liste der dem Gerät bereits bekannten Bluetooth-Geräte, die sich nicht zwingend in Reichweite befinden müssen.
- Liste der vom Gerät bereits als vertrauenswürdig eingestuften Bluetooth-Geräte, die sich nicht zwingend in Reichweite befinden müssen.

Keine dieser Informationen darf von einer Applikation aus geändert werden können. Die einzige Applikation, die berechtigt ist dies zu tun, ist das BCC. Das BCC kann weitere Eigenschaften des Geräts zugänglich machen, wie z. B. die Einstellung des Gerätenamens *(friendly name)* oder die Möglichkeit, das Gerät auf die Werkseinstellungen zurückzusetzen [M+05, S. 22].

2.5.4 Eigenschaften des Geräts

Da sich je nach Einsatzzweck die Konfiguration der Bluetooth-Geräte unterscheidet, besteht die Notwendigkeit, bestimmte Eigenschaften des Geräts abzufragen. Das API definiert eine Reihe von Eigenschaften, die durch einen Aufruf der Funktion `LocalDevice.getProperty()` abgefragt werden können. Die Werte, die abgefragt werden können, sind in [Tab. : 2.6, S. 24] dargestellt. Soll ein solcher Wert abgefragt werden, ist zu beachten, dass die Zeichenketten *case sensitive*[28] sind. Ist die Eigenschaft nicht definiert, oder unbekannt, muss `null` zurückgegeben werden. Sämtliche Eigenschaften, die durch `LocalDevice.getProperty()` abgefragt werden können, müssen ebenfalls durch die von der CLDC bereitgestellte Funktion `System.getProperty()` zugänglich sein [M+05, S. 23f].

[28] Gross-/Kleinschreibung ist zu beachten.

EIGENSCHAFT	BESCHREIBUNG
bluetooth.api.version obex.api.version	Version des *Java APIs for Bluetooth wireless technology* die unterstützt wird. Für die Version 1.1 wird entsprechend "1.1" zurückgeliefert.
bluetooth.l2cap.receiveMTU.max	*Maximum Transfer Unit* (MTU) in Empfangsrichtung, die von L2CAP unterstützt wird. Der zurückgelieferte Wert ist dezimal in einem `java.lang.String` kodiert.
bluetooth.connected.devices.max	Maximale Anzahl verbundener Geräte.
bluetooth.connected.inquiry	Kann eine Gerätesuche durchgeführt werden, während das Gerät bereits verbunden ist?
bluetooth.connected.page	Kann das lokale Gerät eine Verbindung zu einem entfernten Gerät herstellen, wenn das lokale Gerät bereits verbunden ist?
bluetooth.connected.inquiry.scan	Kann das lokale Gerät auf eine Suchanfrage antworten, während es mit einem anderen Gerät verbunden ist?
bluetooth.connected.page.scan	Kann das lokale Gerät eine Verbindung von einem anderen Gerät annehmen, während es bereits mit einem anderen Gerät verbunden ist?
bluetooth.master.switch	Ist der *master/slave* Wechsel erlaubt?
bluetooth.sd.trans.max	Maximale Anzahl zeitgleicher Dienstsuchen *(Service Discovery)*
bluetooth.sd.attr.retrievable.max	Maximale Anzahl der *Service*-Attribute die mit einem *Service Record* empfangen werden können.

Tabelle 2.6: Übersicht der Eigenschaften des Geräts (Vgl. [M+05, S. 24, Tabelle 3-2 Device Properties])

2.5.5 Client-/Server Modell

Ein Bluetooth-Dienst (Service) ist eine Server-Applikation, die einem Client eine bestimmte Funktionalität ermöglicht. Ein Service zum Drucken über Bluetooth wäre bspw. eine solche Applikation. Entwickler, die mit Bluetooth arbeiten, können ihre eigenen Server-Applikationen schreiben, die sie anderen (Clients) zur Verfügung stellen. Dies wird realisiert, indem der *Service Discovery Database (SDDB)* des lokalen Bluetooth-Geräts ein sogenannter *Service Record* hinzugefügt wird, der den angebotenen Dienst beschreibt. Nachdem der Service registriert wurde, wartet der Server auf einen Client, der die Verbindung zu dem gewünschten Service initiiert [M+05, S. 24].

2.5.6 Discovery

Da die meisten mit Bluetooth ausgestatteten Geräte tragbar sind, wird ein Mechanismus benötigt, der es ermöglicht, dass sich zwei oder mehr Geräte finden können. Dieser Prozess wird als *Discovery* bezeichnet. Es existieren hierbei zwei Discovery-Arten; Device- und Service-Discovery [M+05, S. 27].

2.5.6.1 Device Discovery

Geräte können von einer Applikation mit Hilfe der Funktion `startInquiry()`, die nicht blockierend implementiert ist, oder mit ihrem blockierenden Äquivalent `retrieveDevices()` gefunden werden. Die `retrieveDevices()` Methode ist jedoch nicht in der Lage, “neue” Geräte zu finden, sondern liefert lediglich eine Liste der dem Gerät bereits bekannten Geräte. Das sind Geräte, die bspw. in einer vorhergehenden Suche bereits gefunden wurden oder Geräte die als *Pre-known* klassifiziert sind. Als *Pre-known* Geräte bezeichnet man Geräte, die im BCC definiert wurden. Die `retrieveDevices()` Methode führt keine aktive Suche durch, stellt aber die Möglichkeit dar, eine Liste von Geräten zu erhalten, die sich im näheren Umfeld befinden könnten. Wird die `startInquiry()` Methode verwendet, muss zusätzlich ein Listener implementiert werden, der benachrichtigt wird sobald ein neues Gerät gefunden wurde. Der zu implementierende Listener ist im Interface `javax.bluetooth.DiscoveryListener` definiert. Die Klasse `javax.bluetooth.DiscoveryAgent` bietet die entsprechenden Methoden zur Geräte- und Dienstsuche an [M+05, S. 28f].

2.5.6.2 Service Discovery

Nachdem ein Gerät mittels einer *Device Discovery* ausfindig gemacht werden konnte, ist es i. d. R. von Interesse herauszufinden, ob auf dem gefundenen Gerät der gewünschte Dienst

vorhanden ist. Die Suche nach einem solchen Dienst wird als *Service Discovery* bezeichnet. Eine lokale Service-Suche wird vom JSR-82 API allerdings nicht unterstützt. Die in eine Service-Suche involvierten Klassen sind Folgende [M+05, S. 30f]:

- `javax.bluetooth.UUID`
- `javax.bluetooth.DataElement`
- `javax.bluetooth.ServiceRecord`
- `javax.bluetooth.DiscoveryAgent`
- `javax.bluetooth.DiscoveryListener`

In der Spezifikation ist zu jeder der aufgelisteten Klassen Beispielcode zu finden, der die Anwendung der Klassen verdeutlicht.

2.5.7 Generic Access Profile (GAP)

Im Kapitel 7 der JSR-82 Spezifikation sind die Klassen aufgeführt, die dazu dienen, Eigenschaften des Geräts zu manipulieren, die Teil des GAPs[29] sind. Die Standardmechanismen zur Manipulation des lokalen Geräts befinden sich in der Klasse `javax.bluetooth.LocalDevice`, die Methoden um Informationen von einem entfernten Gerät zu erhalten befinden sich in der Klasse `javax.bluetooth.RemoteDevice`. Die Klasse `javax.bluetooth.DeviceClass` definiert Werte, mit Hilfe derer es möglich ist den Gerätetyp sowie den Typ des Angebotenen Dienstes zu identifizieren [M+05, S. 50f].

2.5.8 Kommunikation

Um den Dienst eines anderen Bluetooth-Geräts in Anspruch nehmen zu können, müssen beide Geräte in der Lage sein miteinander zu kommunizieren. Dies erfordert die Kenntnis eines gemeinsamen Kommunikationsprotokolls, sodass verschiedene Applikationen miteinander kommunizieren können. Der JSR-82 stellt APIs zur Verfügung, mit deren Hilfe man Verbindungen zu RFCOMM, L2CAP oder OBEX Diensten herstellen kann. Das *Generic Connection Framework (GCF)*[30] der CLDC stellt dabei den Basismechanismus der Protokollimplementierung dar (Vgl. : [M+05, S. 63]).

[29] Siehe [Kapitel 2.1.5, S. 12]
[30] Siehe [Kapitel 2.3.1, S. 16]

2.5.8.1 Serial Port Profile

Das RFCOMM Protokoll emuliert mehrere RS-232 Ports zwischen zwei Bluetooth Endgeräten. Die Bluetooth-Adressen der beiden Endgeräte identifizieren hierbei die RFCOMM Sitzung. Zwischen zwei Geräten darf immer nur eine Sitzung bestehen, die jedoch über mehr als eine Verbindung verfügen darf. Die Anzahl gleichzeitiger Verbindungen ist hierbei abhängig von der jeweiligen Implementierung. Eine Applikation, die einen Dienst auf Basis des *Serial Port Profiles (SPP)* anbietet, wird dabei als SPP Server, eine die zu einem solchen eine Verbindung initiiert, als SPP Client bezeichnet. Nachdem eine Verbindung hergestellt ist, können Daten in beiden Richtungen ausgetauscht werden. Die Aushandlung der Verbindungsparameter sowie der Flusskontrolle muss hierbei automatisch zwischen den an der Kommunikation teilnehmenden Endgeräten von der SPP Implementierung gehandhabt werden (Vgl.: [M+05, S. 64ff]).

Innerhalb eines Code-Fragments sind SPP Verbindungen (Client- oder Serverseitig) am String `"btspp://"` innerhalb von `Connector.open()` zu erkennen. Stellt die Applikation einen Dienst zur Verfügung, ist `"btspp://localhost"` zu Beginn des Strings enthalten.

URLs, die einen Verbindungsaufbau zu einem entfernten Gerät ermöglichen, sind nach folgendem Schema aufgebaut:

`{scheme}:[{target}{params}]`

Um RFCOMM zu benutzen ist das Schema (`{scheme}`), das für Client und Server verwendet wird, `btspp`. Die beiden anderen Platzhalter (`{target}` und `{params}`) unterscheiden sich, je nachdem, ob die Applikation einen Client oder Server darstellt. Alle gültigen Parameter, die in einem RFCOMM, L2CAP oder OBEX über RFCOMM Verbindungskennzeichner vorkommen können, sind in [Tab.: 2.7, S. 28] aufgelistet.

2.5.8.2 Object Exchange Protocol (OBEX)

Das OBEX Protokoll wurde ursprünglich von der *Infrared Data Association (IrDA)*[31] entwickelt, um Objekte auf einen Client oder Server zu "schieben" *(push)* oder sie von diesem zu "ziehen" *(pull)*. Um Objekte zu transferieren, etabliert OBEX zunächst eine Sitzung. Eine OBEX Sitzung beginnt mit einer CONNECT Anforderung und endet mit einer DISCONNECT Anforderung. Während die Sitzung etabliert ist, kann der Client Objekte vom Server holen (GET) oder Objekte am Server ablegen (PUT). Diese Objekte können Dateien, Visitenkarten (vCards[32]) oder Byte-Arrays darstellen. Indem der Client das SETPATH Kommando sendet, kann er den Server dazu veranlassen, sein aktuelles Verzeichnis zu wechseln (Vgl.: [M+05, S. 93]).

[31] Siehe http://www.irda.org/
[32] Datenformat für eine elektronische Visitenkarte

NAME	BESCHREIBUNG	ZULÄSSIGE WERTE	CLIENT ODER SERVER
master	Gibt an, ob das Gerät master der Verbindung sein muss.	`true`, `false`	Beide
authenticate	Gibt an, ob das entfernte Gerät authentifiziert werden muss bevor eine Verbindung hergestellt werden kann.	`true`, `false`	Beide
encrypt	Gibt an, ob die Verbindung verschlüsselt werden muss.	`true`, `false`	Beide
authorize	Gibt an, ob alle Verbindungen zu diesem Gerät authorisiert werden müssen um den Dienst zu nutzen.	`true`, `false`	Server
name	Der Name des angebotenen Dienstes, der in der SDDB gespeichert wird.	Jeder gültige String	Server

Tabelle 2.7: Gültige Parameter für RFCOMM Verbindungskennzeichner (Vgl. : [BJJ04, S. 59, Tabelle 4.1 Valid Parameters for RFCOMM Connection Strings])

OBEX skaliert sehr gut, egal ob es sich um grosse oder kleine Dateien handelt die übertragen werden müssen. Erreicht wird dies dadurch, dass OBEX die zu übertragenden Daten in mehreren Paketen schickt. Sobald der Client eine PUT oder GET Anforderung sendet, wird eine OBEX-Operation gestartet. Diese Operation besteht solange, bis die Datei vollständig übertragen ist oder ein Fehler eintritt. Um eine PUT Operation durchzuführen, bricht der Client (Applikation oder Protokoll-Stack) das zu sendende Objekt in kleine Stücke und sendet jedes einzeln an den Server. Der Client sendet das nachfolgende Stück nicht bevor der korrekte Versand des Vorgängerstücks bestätigt wurde (Vgl. : [M+05, S. 93]).

OBEX bietet, wie das *Hypertext Transfer Protocol (HTTP)*[33], die Möglichkeit, zusätzliche Informationen zwischen Client und Server auszutauschen. Diese zusätzlichen Meta-Informationen befinden sich im OBEX-Header, der im vergleich zum HTTP-Header lediglich Byte-Werte akzeptiert[34]. OBEX bietet Standard-Header an, wie z. B. Name-, Länge- und Beschreibungsfelder. Zusätzlich existieren noch 64 Header die vom Anwender definiert – und belegt – werden dürfen (Vgl. : [M+05, S. 93]).

Das vom JSR-82 definierte API erlaubt es einer Applikation OBEX-Operationen zwischen Client und Server auszuführen. Allerdings adressiert das API das verbindungslose OBEX nicht, wie dies in der OBEX Spezifikation der IrDA der Fall ist. Das bereitgestellte OBEX API stellt die in [Tab. : 2.8, S. 29] dargestellten Operationen zur Verfügung.

[33] Siehe `http://www.ietf.org/rfc/rfc2616.txt`
[34] OBEX wird aus diesem Grund auch gerne als binäres HTTP bezeichnet.

OPERATION	BESCHREIBUNG
CONNECT	Stellt die Verbindung zum Server her.
PUT	Überträgt Daten vom Client an den Server.
GET	Überträgt Daten vom Server an den Client.
SETPATH	Ändert das Arbeitsverzeichnis auf dem Server.
ABORT	Bricht eine PUT oder GET Operation ab.
CREATE-EMPTY	Erzeugt ein leeres Objekt mit dem im Header angegebenen Namen auf dem Server.
PUT-DELETE	Löscht das im Header angegebene Objekt auf dem Server.
DISCONNECT	Beendet die Verbindung zum Server.

Tabelle 2.8: OBEX Operationen

Da das OBEX Protokoll über verschiedene Transportprotokolle verwendet werden kann, muss der String, der die Verbindung spezifiziert, das Transportprotokoll beinhalten. Die URL, um eine Verbindung zu einem entfernten Gerät herzustellen, hat folgenden Aufbau:

`{scheme}:[{target}{params}]`

Um die zusätzliche Verwendung des Protokolls bei OBEX deutlich zu machen, wird, mit Ausnahme von OBEX über RFCOMM, folgendes Schema verwendet:

`{transport}obex://{target}{params}`.

Wird RFCOMM als Transportprotokoll verwendet, ist der als `{scheme}` gekennzeichnete Teil mit `btgoep` zu ersetzen und `{target}` spezifiziert die Bluetooth-Adresse, sowie die RFCOMM Kanalnummer. Alle für RFCOMM gültigen Parameter (`{params}`) sind ebenfalls für OBEX über RFCOMM gültig[35]. Ein gültiger OBEX über RFCOMM Verbindungskennzeichner hat also die Form:

Client: `btgoep://0123456789ab:1;authenticate=yes`
Server: `btgoep://localhost:DEADBEEFCAFEBABE5DADD115A1D1DEAD`

Die von den anderen Verbindungskennzeichnern differente Schreibweise rührt daher, dass das OBEX Protokoll mit Hilfe des in der Bluetooth-Spezifikation definierten *Generic Object Exchange Profiles (GOEP)*[36] realisiert wurde [M+05, S. 98].

2.5.9 JSR-82 Sicherheit

Im JSR-82 werden Methoden beschrieben, die es ermöglichen, eine gesicherte Bluetooth-Kommunikation herzustellen. Es ist möglich, bereits beim Aufbau der Verbindung, durch

[35] Siehe [Tab. : 2.7, S. 28]
[36] Spezifikation: [Blu05]

einen entsprechend angepassten *Connection String*, die Sicherheitsparameter festzulegen. Methoden der `javax.bluetooth.RemoteDevice` Klasse können benutzt werden, um Sicherheitsänderungen der bestehenden Verbindung zu veranlassen. Die Definition und Belegung der Parameter ist sehr umfangreich und wird in Kapitel 8 des JSR-82 detailliert beschrieben. Die Beschreibung umfasst dabei, wie die Sicherheitsanforderungen bereits beim Verbindungsaufbau festgelegt oder Client und Server Authentifikations- oder Verschlüsselungsforderungen stellen können [M+05, S. 52ff].

KAPITEL 3

OBEX Object Passing

Mobile Endgeräte verfügen durch das GCF der CLDC über einen Mechanismus, der es ihnen erlaubt, eine Verbindung zu einem anderen Gerät über ein Netzwerk herzustellen. Das Kommunikationsverhalten ist jedoch, auf Grund der fehlenden Möglichkeit Java-Objekte zu übertragen, als eingeschränkt einzustufen.

Als Vorstufe der Übertragung ist eine Serialisierung des Objekts notwendig, sodass dieses übertragen und beim Empfänger wieder deserialisiert werden kann. RMI beinhaltet einen solchen Mechanismus um Objekte zu übertragen. Mit dem JSR-66[1] – *RMI Optional Package (RMI OP)* – wurde der RMI Mechanismus, der im J2EE Umfeld häufig im Zusammenhang mit bspw. *Enterprise JavaBeans (EJB)* genannt wird, innerhalb der J2ME verfügbar gemacht. Als Nachteil dieser Lösung erweist sich jedoch der Umstand, dass die RMI OP Erweiterung abhängig von der CDC ist, die von dem Vorhandensein einer VM, welche die JVM Spezifikation vollständig unterstützt, abhängig ist. Ein Einsatz in Verbindung mit der CLDC ist, wegen der fehlenden, vollständigen JVM Unterstützung nicht möglich[2].

Es existieren allerdings andere Frameworks, die sich des Problems der Serialisierung von Objekten mit der J2ME annehmen. Ein Vergleich der Frameworks Javolution, kSOAP, Burlap/Hessian und RMI OP wurde in [BSM06] durchgeführt. Als Ergebnis der Untersuchung wurde festgestellt, dass keines der untersuchten Frameworks in der Lage ist, ein komplexes Objekt innerhalb der J2ME (auf Basis der CLDC) zu deserialisieren. Wäre eine Serialisierung der Objekte mit Hilfe einer der o. g. Frameworks möglich gewesen, würden sie allerdings das Problem der Datenübertragung – abgesehen von RMI – nicht lösen können.

[1] Siehe http://www.jcp.org/en/jsr/detail?id=66
[2] Architekturelle Einordnung siehe [Abb. 2.3, S. 14]

Der Fokus der OOP Bibliothek liegt nicht auf der Serialisierung von Objekten, wie dies auf Grund der Schnittstellendefinition evtl. angenommen werden könnte, sondern vielmehr auf der effizienten, einfachen Übertragung der Daten einfacher Objekte. Tiefe Kopien von komplexen Objektstrukturen können, sofern sie vom Programmierer in eine Byte-Array-Repräsentation überführt und von dieser auch wieder hergestellt werden können, ebenfalls mit dem Mechanismus übertragen werden. Die einfachste Methode jedoch ist es, die komplexe Objektstruktur in eine einfache Darstellung zu überführen. Falls es das Projekt zulässt, kann auf die CDC gewechselt werden, die einen Mechanismus zur Serialisierung von Objekten bereitstellt, sodass die zu übertragenden Objekte sehr leicht in einer Byte-Array-Repräsentation erhalten werden können.

Das *OBEX Object Passing (OOP)* stellt eine signifikante Vereinfachung der Übertragung von Objekten über eine Bluetooth-Funkstrecke dar.

Das Peer2Me Projekt[3] scheint einen, dem OOP ähnlichen, Mechanismus zu implementieren. Die Schnittstelle `Persistent` des Projekts weist die selbe Signatur wie die Schnittstelle `IObexObjectPassing` der OOP Bibliothek auf. Die Peer2Me Bibliothek baut auf der RFCOMM Schnittstelle auf. Um Transportprotokollunabhängigkeit zu erreichen, wurde ein eigener Mechanismus entwickelt (Vgl. : [LN05, S. 70] und [LN05, S. 58]). Eine genauere Untersuchung des Projekts liess sich, auf Grund der späten Entdeckung, nicht mehr realisieren.

OOP setzt im Vergleich zu Peer2Me auf der OBEX Schicht auf. Die Transportprotokollunabhängigkeit wird beim OOP durch eben diesen Umstand erreicht. Das OOP passt sich wie in [Abb. : 3.1, S. 33] dargestellt in den Bluetooth-Protokollstapel ein.

3.1 Technische Voraussetzungen

Die Implementierung der Bibliothek sowie die Umsetzung der Beispielapplikation, ist mit der Netbeans IDE[4] realisiert worden. Die Netbeans IDE wurde auf Grund persönlicher Präferenzen des Autors sowie der beispiellosen Integration des *Wireless Toolkits (WTK)*[5] gewählt. Die Integration des WTK erfolgt durch die zusätzliche Installation des sogenannten *Mobility Pack*s, das unter der selben URL gefunden werden kann wie die IDE selbst. Die Applikationsentwicklung wird durch die gute Integration der vom WTK bereitgestellten Emulatoren wesentlich vereinfacht.

Die Netbeans IDE ist auf nahezu allen Betriebssystemen, auf denen eine JVM unterstützt wird, verfügbar. Allerdings gilt dieser Umstand nur für die IDE selbst. Das für die Entwicklung von J2ME Applikationen notwendige *Wireless Toolkit* ist nicht für Macintosh Plattfor-

[3] Siehe `http://www.peer2me.org/`
[4] Erhältlich unter `http://www.netbeans.org/`
[5] Separat erhältlich unter `http://java.sun.com/products/sjwtoolkit/`

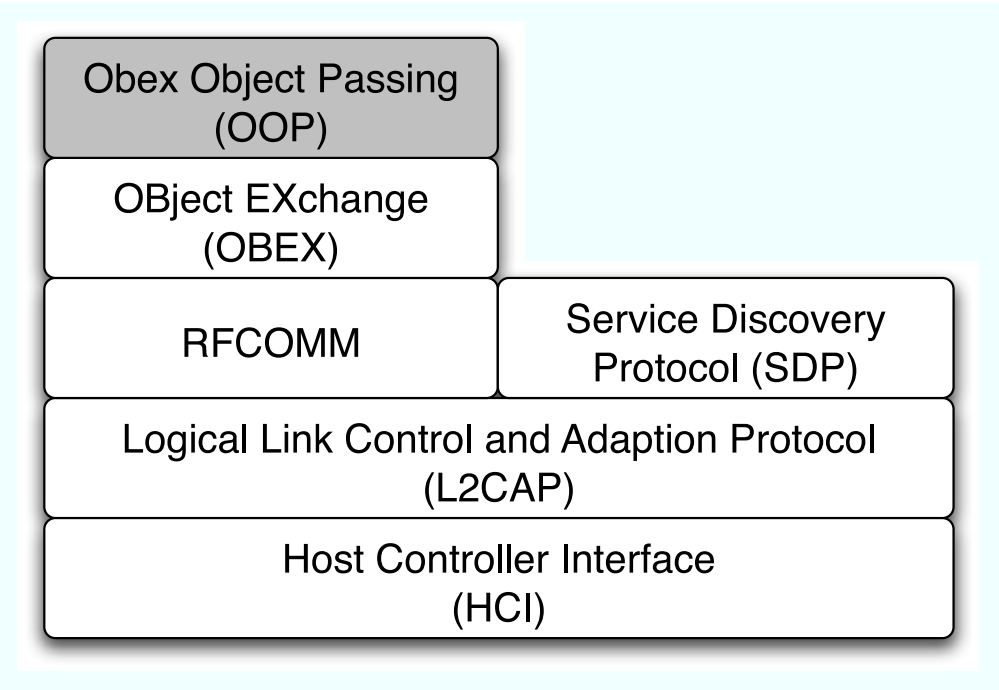


Abbildung 3.1: Einordnung von OOP im Bluetooth-Protokollstapel

men verfügbar. Es ist jedoch möglich, durch den Einsatz eines alternativen Emulators, J2ME Applikationen auf dem Macintosh zu entwickeln[6].

Möchte man, abgesehen von Applikationen für mobile Endgeräte, Applikationen entwickeln, die auf einem Desktop-Computer verwendet werden sollen, ist die zusätzliche Installation eines Bluetooth-Stacks sowie einer JSR-82 Implementierung notwendig.

3.1.1 Bluetooth-Stack

Beim Kauf eines Bluetooth-Adapters wird der Stack i. d. R. vom Hersteller des Produkts mitgeliefert. Der Funktionsumfang des mitgelieferten Stacks ist jedoch vom Hersteller abhängig. Ein nicht mit dem Produkt gelieferter Bluetooth-Stack kann (wahrscheinlich) installiert werden, jedoch ist die Beschaffung desselben auf legalem Wege nicht zu erreichen. Linux verwendet einen eigenen Bluetooth-Stack; den sogenannten BlueZ Stack[7]. Die Macintosh Plattform verwendet ebenfalls ihre Eigenentwicklung.

3.1.2 JSR-82 Implementierung

Bei der Wahl der JSR-82 Implementierung ist ebenfalls Vorsicht geboten. Die Implementierungen sind abhängig vom zu Grunde liegenden Bluetooth-Stack. So kann es passieren,

[6] Anleitung zur Konfiguration der IDE unter [Has06]
[7] Erhältlich unter http://www.bluez.org/

dass die gewünschte JSR-82 Implementierung nicht mit dem auf dem System installierten Bluetooth-Stack zusammenarbeitet.

Um den JSR-82 auf realen Geräten einsetzen zu können, muss das Endgerät Unterstützung für diesen bereitstellen. Dies ist bei den meisten mobilen Endgeräten, wie z. B. Handys, oder PDAs gegeben. Möchte man allerdings auf einem Desktop-Computer eine Bluetooth-Anwendung in Betrieb nehmen, muss zusätzlich zum Bluetooth-Stack eine JSR-82 Implementierung installiert werden.

Die meisten Implementierungen werden in Form eines Java-Archivs und einer Systembibliothek ausgeliefert. Es gibt allerdings auch Ausnahmen, wie z. B. die Harald-[8] oder JavaBluetooth-Implementierung[9], die eine vollständig in Java realisierte JSR-82 Implementierung bereitstellen. Da die beiden Alternativen allerdings das `javax.comm` Paket verwenden, muss dieses zusätzlich installiert werden. Hier hat man die Wahl zwischen dem Paket von Sun[10], und einer RXTX[11] genannten Implementierung. Die Installation und Konfiguration der Bibliotheken erweist sich jedoch, im Vergleich zu den Paketen mit Systembibliothek und Java-Archiv, schwieriger. Es existieren allerdings auch Implementierungen, die lediglich in einem einzigen Java-Archiv ausgeliefert werden. Sie laden die notwendigen nativen Bibliotheken intern nach.

Bei den verschiedenen verfügbaren JSR-82 Implementierungen muss zusätzlich darauf geachtet werden, ob sie den JSR-82 vollständig (inklusive `javax.obex`) implementieren. Die Spezifikation lässt bei der Implementierung des OBEX Protokolls die Wahl, ob OBEX im zu Grunde liegenden Bluetooth-System oder mit dem JSR-82 implementiert wird[12]. Dieses Faktum scheint von einigen JSR-82 Implementierungen falsch verstanden oder bewusst weggelassen zu werden. In jedem Fall sollte die JSR-82 Implementierung die Schnittstellen exportieren bzw. sollten sie vom JSR-82 API angesteuert werden können.

Das Problem des fehlenden `javax.obex` Pakets kann durch den Einsatz der avetanaOBEX[13] Bibliothek kompensiert werden. Da viele mobile Endgeräte ohnehin nur den `javax.bluetooth` Teil der Spezifikation unterstützen, können diese Geräte durch den Einsatz der Bibliothek ebenfalls adressiert werden. Ein allgemeines Nachrüsten der fehlenden OBEX Unterstützung ist hingegen nicht möglich. Die von der CLDC gesetzten Sicherheitsstandards verbieten eine Nachrüstung von Funktionalität durch externe Bibliotheken[14], was zur Folge hat, dass die Bibliothek in jeder Applikation mitgeliefert werden muss, was auf den ohnehin schon mit knappem Speicher bemessenen Geräten zusätzlichen Speicherplatz in Anspruch nimmt.

[8] Siehe http://www.control.lth.se/~johane/harald/
[9] Siehe http://www.javabluetooth.org/
[10] Siehe http://www.sun.com/download/products.xml?id=43208d3d
[11] Siehe http://www.rxtx.org/
[12] Siehe [Kapitel 2.5.1, S. 22]
[13] Erhältlich unter http://sourceforge.net/projects/avetanaobex/
[14] Siehe [Kapitel 2.3.2, S. 17]

Für die vorliegende Arbeit ist auf dem Desktop-System der mit dem *Service Pack 2* für Windows XP ausgelieferte Bluetooth-Stack verwendet worden. Als JSR-82 Implementierung ist BlueCove[15] auf Grund seiner freien Verfügbarkeit und einfachen Installation zum Einsatz gekommen.

3.2 Design

Das OBEX Protokoll ist prädestiniert zur Übertragung grösserer Objekte über ein Netzwerk. Die korrekte Fragmentierung der Daten, die ansonsten manuell durchgeführt werden müsste, ist innerhalb des Protokolls vorgesehen und kann somit bei jeder OBEX Implementierung vorausgesetzt werden. Bei Bluetooth bspw. muss die Fragmentierung, sofern ein HCI vorhanden ist, spätestens von der L2CAP Implementierung durchgeführt werden[16]. Ein weiterer Vorteil des OBEX Protokolls besteht in der Unabhängigkeit der zu Grunde liegenden Transportprotokolle. So ist es bspw. möglich OBEX über TCP/IP zu verwenden. Dieses Verhalten sollte die Adaption des OOP Verfahrens auf andere Transportprotokolle wesentlich vereinfachen. Für die Proof-Of-Concept Implementierung wurde OBEX über RFCOMM gewählt[17]. Eine paketorientierte Übersicht über die OOP Bibliothek ist in [Abb. : 3.2, S. 35] zu sehen.

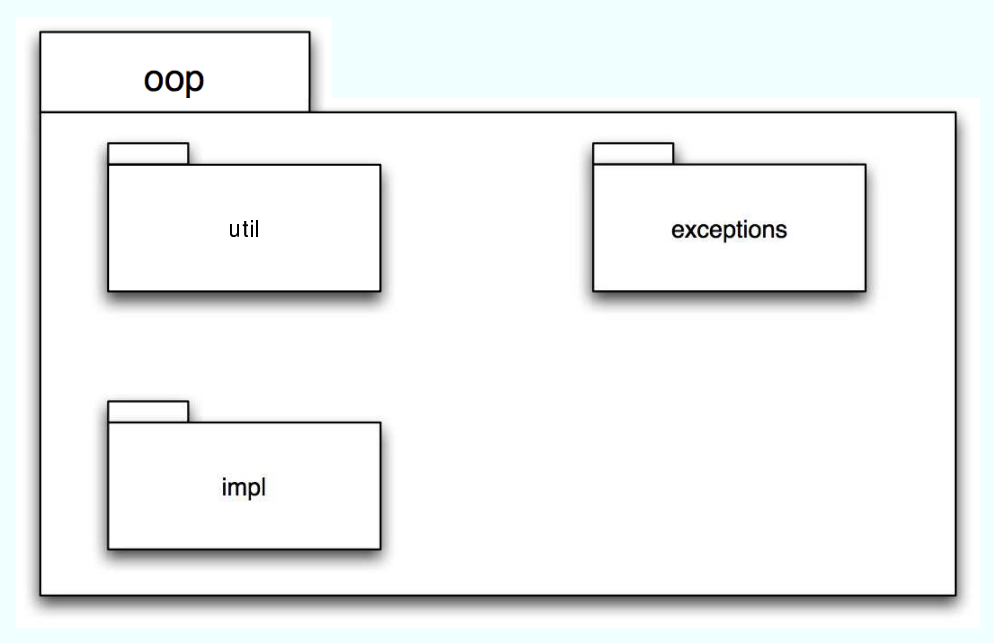


Abbildung 3.2: Paketübersicht der OBEX Object Passing Bibliothek

Der Aufbau der Bibliothek ist relativ simpel gehalten. Das »oop« Paket beinhaltet lediglich das Interface der Bibliothek sowie die Konstanten. Das »util« Paket beinhaltet Klassen, die

[15] Erhältlich unter http://sourceforge.net/projects/bluecove/; Installationsanleitung unter [Lab06]
[16] Siehe [Kapitel 2.1.3, S. 11]
[17] Siehe [Kapitel 2.5.8, S. 26]

zur Realisierung der Bibliothek notwendig waren. Das »impl« Paket beinhaltet diejenigen Klassen, welche die Übertragung der Daten letzten Endes realisieren. Das »exceptions« Paket enthält die von der OOP Bibliothek definierten Ausnahmen. Die Funktionalität der Klassen der jeweiligen Pakete wird im folgenden erläutert.

3.2.1 Paket »oop«

Wie in [Abb.: 3.3, S. 36] zu sehen ist, befinden sich lediglich zwei Klassen im »oop« Paket. Die Klassen haben folgende Funktionalität:

IObexObjectPassing Dieses Interface muss von Entitätsobjekten, welche die Daten kapseln, implementiert werden, um übertragen werden zu können. Es zwingt den Programmierer dazu, die Methoden `setObjectData(byte[] objectData)` und `getAsByteArray()` zu implementieren, die zur korrekten Funktionsweise der Bibliothek notwendig sind.

Constants Diese Klasse enthält die in der Bibliothek verwendeten Konstanten, sodass diese an einer zentralen Stelle geändert werden können.

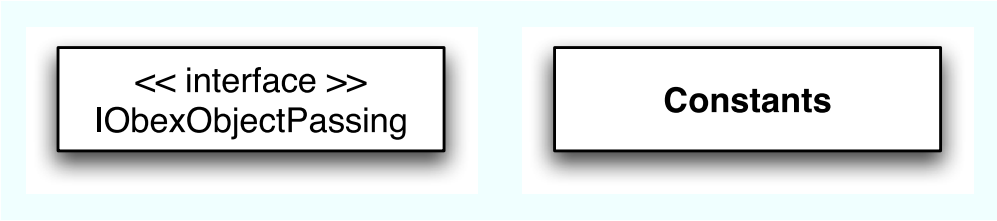


Abbildung 3.3: Klassen des »oop« Pakets

Durch Implementierung des `IObexObjectPassing` Interfaces ist zusätzlich eine einfache Weiterverarbeitung der Daten möglich, sollen diese persistent auf dem Gerät gespeichert werden. Begründet werden kann dies durch die Limiterung, welchen den mobilen Endgeräten auferlegt ist. Da sie laut der CLDC keine Kenntnis eines Dateisystems vorweisen müssen[18], ist ein alternativer Mechanismus im MIDP zur Verfügung gestellt worden; das RMS[19]. Lese- und Schreiboperationen auf das RMS können jedoch ausschliesslich in Byte-Strömen durchgeführt werden. Durch die Implementierung der `IObexObjectPassing` Schnittstelle sind die Entitätsobjekte bereits auf diese Aufgabe vorbereitet. Auf den Umgang mit dem RMS soll an dieser Stelle jedoch nicht näher eingegangen werden[20].

Die Bibliothek wurde bewusst mit einer Schnittstelle und nicht als abstrakte Basisklasse modelliert. Gemäss Balzert wird eine Schnittstelle wie folgt beschrieben:

[18] Siehe [Kapitel 2.3, S. 16]
[19] Siehe [Kapitel 2.4.1, S. 19]
[20] Eine Einführung ist unter [Gho06] zu finden.

> *„Es werden **funktionale Abstraktionen** in Form von Operationssignaturen bereitgestellt, die das »Was«, aber* nicht *das »Wie« festlegen. Eine Schnittstelle besteht also im Allgemeinen nur aus Operationssignaturen, d. h. sie besitzt keine Operationsrümpfe und keine Attribute. Schnittstellen können jedoch in Vererbungsstrukturen verwendet werden. Eine Schnittstelle ist äquivalent zu einer Klasse, die keine Attribute und ausschließlich **abstrakte Operationen** besitzt.“* [Bal00, S. 817]

Da in der Java Programmiersprache allerdings der Mechanismus der Mehrfachvererbung nicht vorgesehen ist, wäre die Modellierung als abstrakte Basisklasse für den Programmierer unvorteilhaft gewesen, da eine Schnittstelle, im Gegensatz zur abstrakten Basisklasse, auf n-ter Stufe der Erbhierarchie implementiert werden kann, wohingegen die abstrakte Basisklasse an der Wurzel der Erbhierarchie implementiert werden müsste.

3.2.2 Paket »impl«

Wie in [Abb. : 3.5, S. 38] zu erkennen ist leiten diejenigen Klassen, die zur Übertragung verwendet werden, von der Observer Klasse im »util« Paket ab. Das Observer-Pattern, oder zu deutsch Beobachter-Muster, ermöglicht es, Klassen, die von einem Objekt abhängig sind, zu informieren, dass sich der Zustand des zu überwachenden Objekts geändert hat. Dieses Pattern wird häufig in der Programmierung graphischer Oberflächen eingesetzt, wenn eine graphische Repräsentation der Daten automatisch geändert werden soll, sobald sich der Datenbestand ändert. Die Struktur des Patterns ist in [Abb. : 3.4, S. 37] abgebildet.

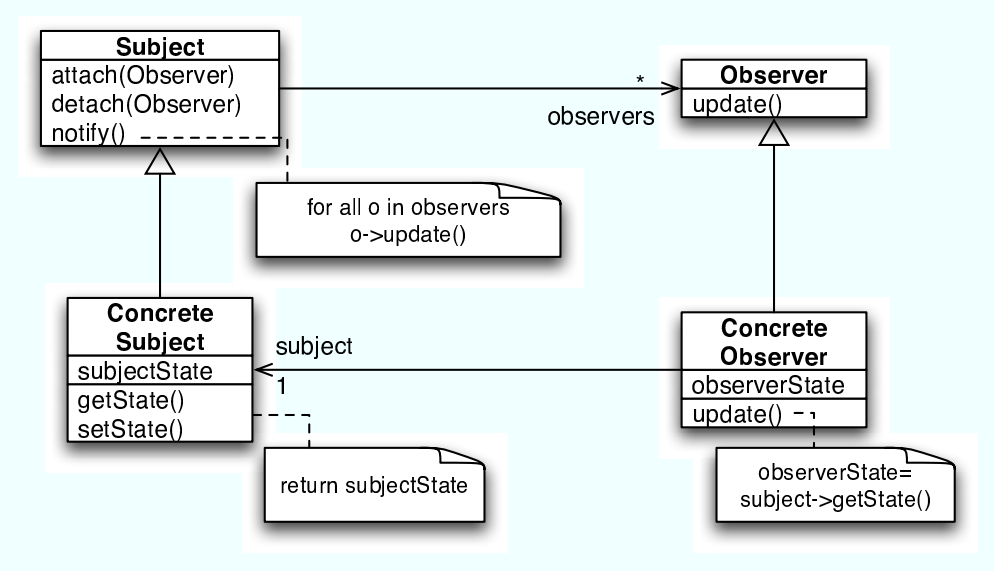


Abbildung 3.4: Struktur des Observer-Patterns (Vgl. : [Bal00, S. 849])

Die Klassen `ObjectPusher`, `ObjectReceiver` und `BulkObjectPusher` aus [Abb. : 3.5, S. 38] stellen dabei die in [Abb. : 3.4, S. 37] als *Concrete Subject* bezeichnete Klasse dar.

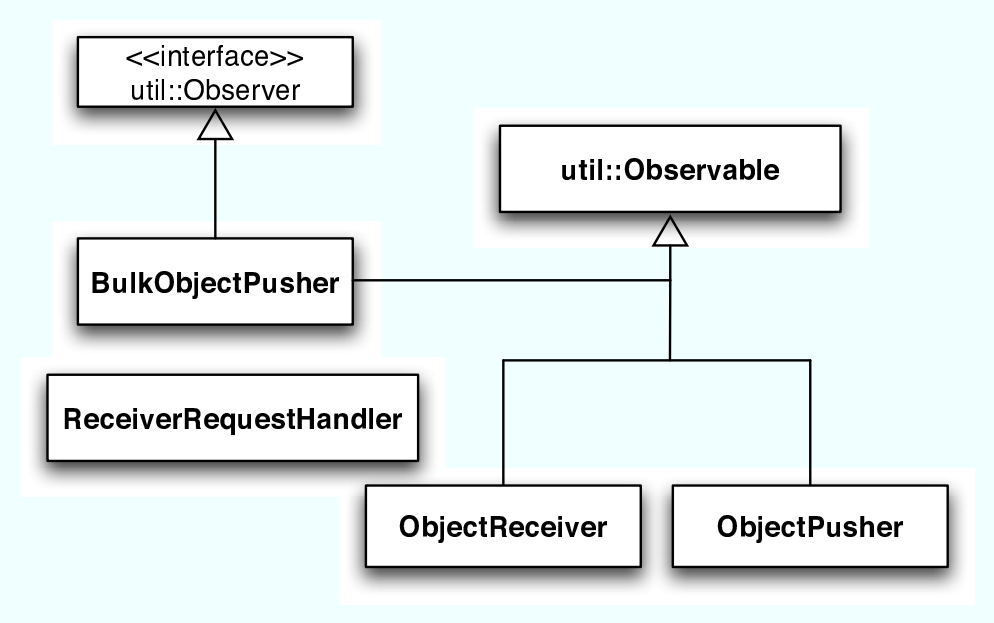


Abbildung 3.5: Klassen des »impl« Pakets

Die Funktionalität der im »impl« Paket enthaltenen Klassen ist folgende:

ObjectReceiver Die `ObjectReceiver` Klasse wird verwendet um Objektdaten, die von der `ObjectPusher` Klasse versandt wurden, zu empfangen, um sie dann in einem Speicher empfangener Objekte abzulegen.

ObjectPusher Die `ObjectPusher` Klasse dient dem Zweck ein Objekt, welches das Interface `IObexObjectPassing` implementiert, zu versenden.

BulkObjectPusher Die Klasse `BulkObjectPusher` ist, wie der Name schon andeutet, dazu gedacht mehrere Objekte sequenziell in einem Aufruf zu übertragen.

ReceiverRequestHandler Diese Klasse ist notwendig, da in Java die Möglichkeit der Mehrfachvererbung nicht gegeben ist. Sie wird innerhalb der Klasse `ObjectReceiver` verwendet und bei der Initiierung einer Verbindung durch einen Client instanziert. Die Behandlung der PUT Anfrage[21] wird durch sie durchgeführt.

Den Klassen `ObjectReceiver`, `ObjectPusher` und `BulkObjectPusher` ist gemein, dass sie in einem separaten Thread ablaufen. Dies ist bei der Anwendung der Klassen von Vorteil, da sie den weiteren Programmablauf nicht stören. Der Transfer der Daten wird dadurch im Hintergrund durchgeführt. Um über die Beendigung des Datentransfers informiert zu werden, muss die aufrufende Klasse das Interface `Observer` implementieren. Die Verwendung

[21] Siehe [Tab. : 2.8, S. 29]

der angesprochenen Klassen ist auch ohne Implementierung des `Observer` Interfaces möglich, jedoch verliert man die Möglichkeit über die Beendigung der Übertragung informiert zu werden.

3.2.3 Paket »util«

Im Paket »util« sind diejenigen Klassen enthalten, die nicht in direktem Zusammenhang zur Implementierung stehen, jedoch zur Umsetzung derer notwendig waren [Abb. : 3.6, S. 39].

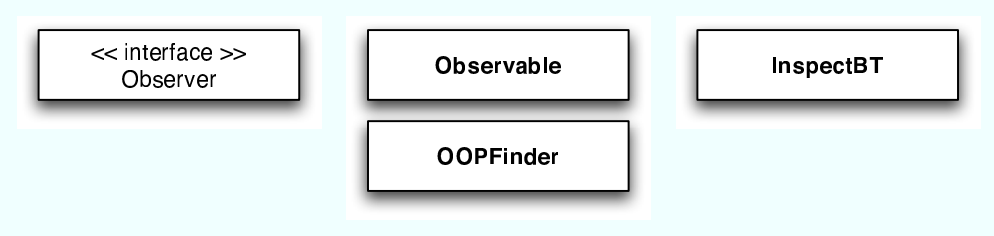


Abbildung 3.6: Klassen des »util« Pakets

Observer Diese Klasse implementiert die in [Abb. : 3.4, S. 37] dargestellte *Observer* Klasse. Die Implementierung ist an die aus der J2SE bekannte `java.util.Observer` Klasse angelehnt. Beim Entwurf der Klasse wurde auf Signaturgleichheit der Methoden geachtet.

Observable Diese Klasse implementiert die in [Abb. : 3.4, S. 37] dargestellte *Subject* Klasse. Die Implementierung ist an die `java.util.Observable` Klasse aus der J2SE angelehnt. Ebenso wie bei der *Observer* Implementierung wurde auf die Signaturgleichheit der Methoden geachtet.

OOPFinder Diese Klasse ist hilfreich, wenn die Methode `DiscoveryAgent.selectService()` von der verwendeten JSR-82 Implementierung nicht unterstützt wird[22]. Diese Klasse stellt die Methode `getConnectionURL()` bereit, welche das Verhalten der Methode `DiscoveryAgent.selectService()` imitiert.

InspectBT Diese Klasse stellt die Methode `getAllProperties()` zur Verfügung, mit deren Hilfe sämtliche Eigenschaften des zu Grunde liegenden Bluetooth-Systems als String erhalten werden können. Sämtliche abgefragten Eigenschaften sind dabei, inklusive Erklärung, in [Tab. : 2.6, S. 24] aufgelistet.

Möchte man auf einem J2SE System aus bestimmten Gründen die *Observable* und *Observer* Klassen der J2SE verwenden, müssen, dank der Signaturgleichheit der Klassen, lediglich die `import` Anweisungen der Bibliothek angepasst werden.

[22] Dies ist z. B. beim BlueCove-Stack der Fall.

3.2.4 Paket »exceptions«

Das »exceptions« Paket definiert die in der Bibliothek verwendeten Ausnahmen. Das Paket enthält momentan lediglich eine Ausnahme [Abb. : 3.7, S. 40].

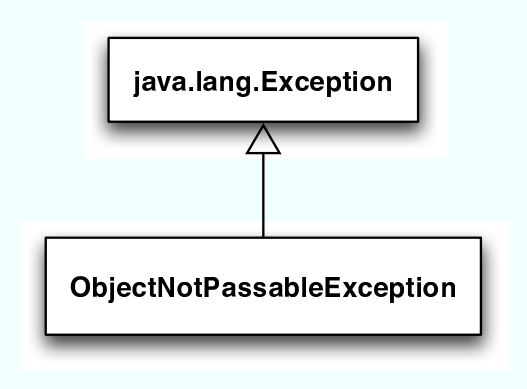


Abbildung 3.7: Klassen des »exceptions« Pakets

ObjectNotPassableException Diese Ausnahme wird geworfen wenn das zu übertragende Objekt die Schnittstelle `IObexObjectPassing` nicht implementiert.

3.3 Deus ex machina

Die Funktionsweise des *OBEX Object Passing* wird anhand des Sequenzdiagramms in [Abb. : 3.8, S. 41] im Folgenden erklärt. Das dargestellte Diagramm zeigt den Versand eines einzelnen Datenobjekts. Auf den Versand mehrerer Datenobjekte, der durch den `BulkObjectPusher` realisiert ist, wird nicht eingegangen. Beim `BulkObjectPusher` wird lediglich für jedes zu übertragende Objekt die Klasse `ObjectPusher` aufgerufen. Die Funktionsweise des `ObjectReceivers` wird im Zuge der Erklärung des `ObjectPushers` behandelt. Weitere Informationen zur Implementierung des `ObjectReceivers` sind in [Kapitel 3.3.1, S. 42] zu finden.

Nach der Erzeugung des `ObjectPusher` Objekts ruft es, durch die interne Thread-Erzeugung, die Methode `run()` auf, in der die eigentliche Arbeit des Objekts verrichtet wird.

Schritt 1.1) In diesem Schritt wird die URL der Verbindung ermittelt, um das mobile Endgerät ansprechen zu können.

Schritt 1.2) Die Verbindung zum entfernten Gerät wird hergestellt und die Sitzung etabliert.

Schritt 1.3) Mit Hilfe der vorhandenen Sitzung wird ein `HeaderSet` erzeugt.

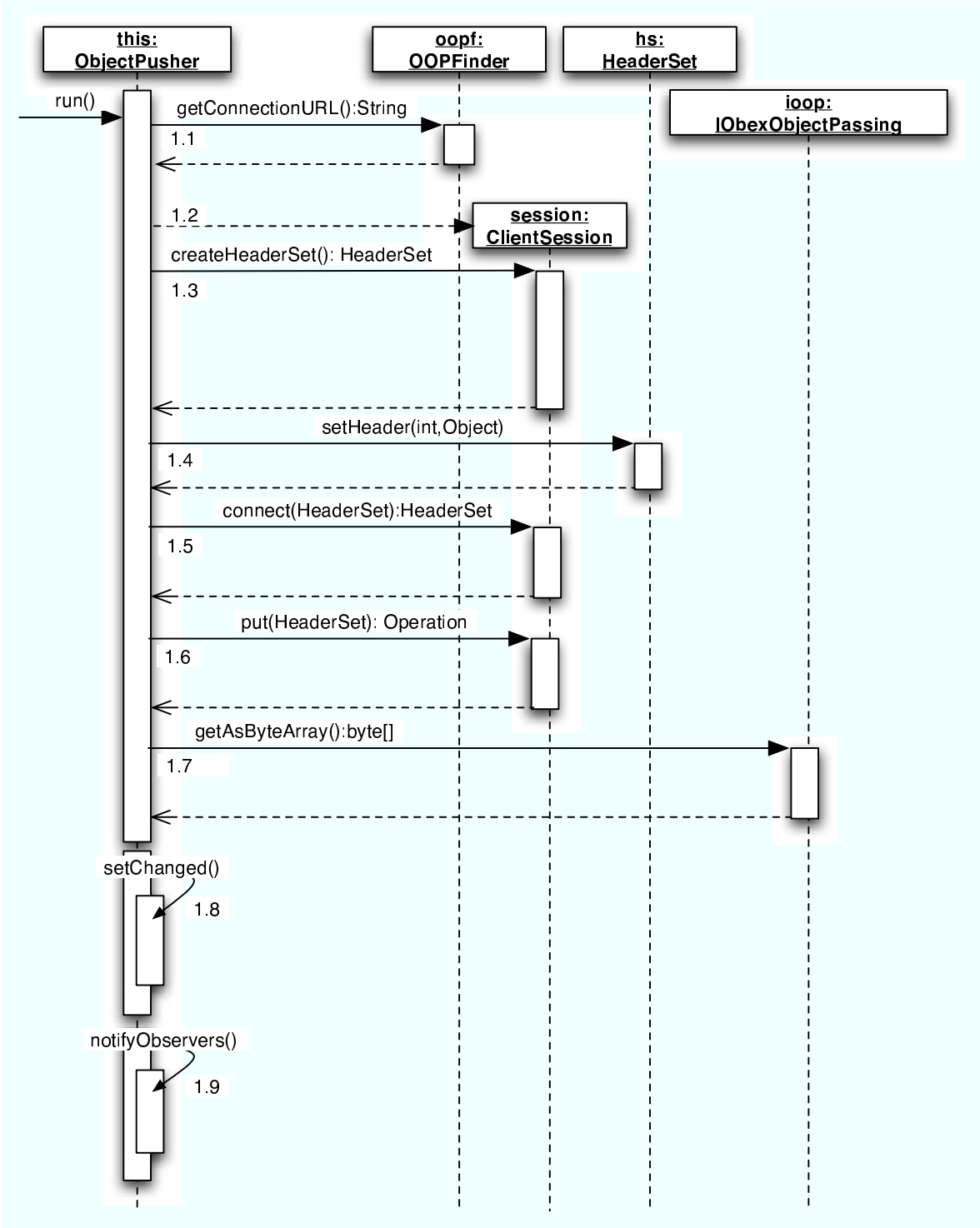


Abbildung 3.8: `ObjectPusher` Sequenzdiagramm

Schritt 1.4) In das erzeugte `HeaderSet` wird der Name der zu übertragenden Klasse in das Feld `HeaderSet.NAME` gesetzt.

Schritt 1.5) Der OBEX Transfer wird eröffnet und das `HeaderSet` zum entfernten Gerät übertragen.

Schritt 1.6) Nachdem ein Ausgabestrom geöffnet wurde, werden die Daten mit der Methode `getAsByteArray()` aus dem Objekt extrahiert und zum entfernten Gerät übertragen. Dieses liest die eintreffenden Daten und speichert sie zunächst temporär in einer Variablen ab. Der `ObjectReceiver` erzeugt aus der im `HeaderSet` übermittelten Information des Klassennamens, mit Hilfe des *Classloaders* ein neues Objekt des übermittelten Typs. Nachdem die Erzeugung abgeschlossen ist, werden die empfangenen Daten mit Hilfe der Methode `setObjectData()` in das erzeugte Objekt geschrieben. Das Objekt wird im nächsten Schritt in einem `Vector` gespeichert, sodass dieses später zugänglich ist.

Schritt 1.7) Der Zustand des Objekts wird auf geändert gesetzt, da die Übertragung der Daten abgeschlossen ist.

Schritt 1.8) Die angeschlossenen `Observer` werden über das Ende der Übertragung informiert. (Die Klasse `BulkObjectPusher` realisiert durch diese Benachrichtigung ihr sequenzielles senden der Objekte).

3.3.1 ObjectReceiver

Die Implementierung des `ObjectReceivers` verändert, bedingt durch die Tatsache, dass sie einen Dienst zur Verfügung stellt, den Auffindbarkeitszustand des Geräts. Das Gerät wird in den GIAC Modus[23] versetzt, um permanent auffindbar zu sein. Ist dieses Verhalten nicht gewünscht, so muss es, in der aktuellen Version der Bibliothek, im Quellcode geändert werden. Die Registrierung des Dienstes in der SDDB[24] wird von der avetanaOBEX Bibliothek, beim Aufruf der Methode `OBEXConnector.open()`, vorgenommen. Diese Registrierung ist notwendig, damit der OOP Dienst gefunden werden kann. Die Implementierung der avetanaOBEX Bibliothek weist an dieser Stelle jedoch einen kleinen Schönheitsfehler auf; so wird der Dienst in der SDDB als SPP Dienst, anstelle eines OBEX Dienstes eingetragen, was jedoch die Funktion nicht weiter stört.

Ist die Änderung des Auffindbarkeitszustands, sowie die Registrierung in der SDDB abgeschlossen, wartet das Objekt auf eingehende Verbindungen. Da bei einem eintreffenden Client ein nebenläufiger Prozess gestartet wird, erfolgt in der `ObjectReceiver` Klasse eine synchronisation der Prozesse des `ReceiverRequestHandlers` und der `ObjectReceiver` Klasse selbst.

[23] Siehe [Kapitel 2.4, S. 13]
[24] Siehe [Kapitel 2.5.5, S. 25]

Der `ObjectReceiver` wartet auf die Termination der `ReceiverRequestHandler` Klasse bis er mit seiner Ausführung fortfährt.

3.4 Tests

Es sind im Rahmen der Entwicklung der Bibliothek rudimentäre Tests durchgeführt worden. Diese Tests sind auf Grund des relativ hohen Einarbeitungsaufwands nicht mit einem automatisierten Test-Rahmenwerk wie z. B. JUnit[25] durchgeführt worden. Tests sollten wegen der Affinität des Entwicklers zu seinem Projekt, normalerweise niemals vom Entwickler selbst durchgeführt werden. Da die Tests jedoch lediglich die während der Entwicklungsphase durchgeführten Testszenarien reflektieren, möge dies entschuldigt werden. Die durchgeführten Testfälle sind in [Tab. : 3.1, S. 44] zusammengefasst.

Die Testreihe konnte auf Grund mangelnder Hardware nicht auf zwei reale Mobiltelefone ausgedehnt werden. Der Versuch mit Hilfe der Bibliothek Daten von einem Laptop auf ein Mobiltelefon zu übertragen, schlug ebenfalls fehl. Das entfernte Gerät konnte weder vom Laptop noch vom Mobiltelefon aus gefunden werden.

Eine Untersuchung der Eigenschaften des Geräts, mit Hilfe der Klasse `InspectBT`, förderte zu Tage, dass das verwendete Mobiltelefon (Nokia 6230) während es mit einem Gerät gekoppelt ist, weder auf eine Suchanfrage antworten, noch eine initiieren kann. Dieses Verhalten jedoch ist für die korrekte Funktionsweise der OOP Bibliothek relevant.

Die verwendete JSR-82 Implementierung des Laptops lieferte bei allen abgefragten Eigenschaften `null` als Ergebnis. Das bedeutet, dass die Eigenschaft nicht implementiert ist oder die JSR-82 Implementierung falsch reagiert. Weiterhin bedeutet dies, dass selbst bei korrektem Verhalten des Mobiltelefons, immer noch die verwendete JSR-82 Implementierung fehlerhaft sein könnte.

[25] Erhältlich unter http://www.junit.org/

TESTBESCHREIBUNG	ERWARTETES ERGEBNIS	TEST ERFÜLLT
Es wird versucht ein Objekt zu übertragen, dass die Schnittstelle `IObexObjectPassing` nicht implementiert.	Die Übertragung wird abgebrochen.	Ja
Sind die Daten am Empfangsgerät korrekt rekonstruiert worden?	Korrekte Rekonstruktion der Daten.	Ja
Es existieren zwei Geräte auf denen der Server `ObjectReceiver` gestartet wurde.	Die Implementierung informiert den Anwender, dass mehrere Geräte verfügbar sind.	Nein[a]
Es wird versucht zu senden, jedoch ist am Empfangsgerät keine `ObjectReceiver` Instanz erzeugt worden.	Die Methode `OOPFinder.getConnectionURL()` liefert `null`.	Ja
Wird nach einer abgelehnten Verbindung durch das BCC eine erneute Rückfrage an den Benutzer gestellt?	Rückfrage wird erneut gestellt.	Nein[b]
Besteht die Möglichkeit `ObjectReceiver` und `BulkObjectPusher` parallel auf einem Gerät zu instanziieren?	Beide können parallel gestartet werden.	Ja

Tabelle 3.1: Während der Entwicklung durchgeführte Testfälle

[a] Die Implementierung wählt das erste verfügbare Gerät.
[b] Ergebnis ist abhängig von der Implementierung des BCC [M+05, S. 53]

KAPITEL 4

Entwickeln mit OOP

In diesem Kapitel wird demonstriert, wie die OOP Bibliothek eingesetzt werden kann. Um die zur Demonstration entwickelte Beispielapplikation verstehen zu können, wird an dieser Stelle kurz auf die Grundlagen der J2ME Programmierung eingegangen. Die Kenntnis der Grundlegenden Sprachkonstrukte von Java sowie das Verständnis einfacher Datenstrukturen und -container wird vorausgesetzt.

4.1 Grundlagen der J2ME Programmierung

Die J2ME ist, wie bereits in [Abb. : 2.3, S. 14] gezeigt, im Java Umfeld einzuordnen. Java Programme, die auf einem mobilen Endgerät ausgeführt werden können, werden als MIDlets bezeichnet. Die Namensgebung folgt aus dem Zusammenschluss von MID *(Mobile Information Device)* und Applet. Der hauptsächliche Unterschied zur J2SE besteht im limiterten Sprachumfang der J2ME. Der signifikante Unterschied der beiden Umgebungen ist auf die Beschaffenheit der Hardware zurückzuführen, die wie bereits in [Kapitel 2.3, S. 16] und [Kapitel 2.5.1, S. 21] gezeigt, im J2ME Umfeld mit starken Einschränkungen belegt ist.

4.1.1 Lebenszyklus eines MIDlets

Jedes MIDlet ist gezwungen von der `MIDlet` Klasse zu erben. Sie ermöglicht das korrekte starten, stoppen und aufräumen des MIDlets. Eine MIDlet darf, im Gegenteil zu J2SE Programmen, nicht über eine `public static void main()` Methode verfügen. Ist dies trotzdem der Fall wird sie von der *Application Management Software (AMS)* ignoriert [VW02, S. 438].

Die AMS ist Bestandteil des Betriebssystems des mobilen Endgeräts. Sie ermöglicht die Ausführung des MIDlets (und ggf. auch die Installation/Deinstallation dessen). Weiterhin ist die AMS in der Lage das MIDlet zu pausieren oder es zu stoppen. Der Zustandsautomat eines MIDlets ist in [Abb. : 4.1, S. 46] dargestellt.

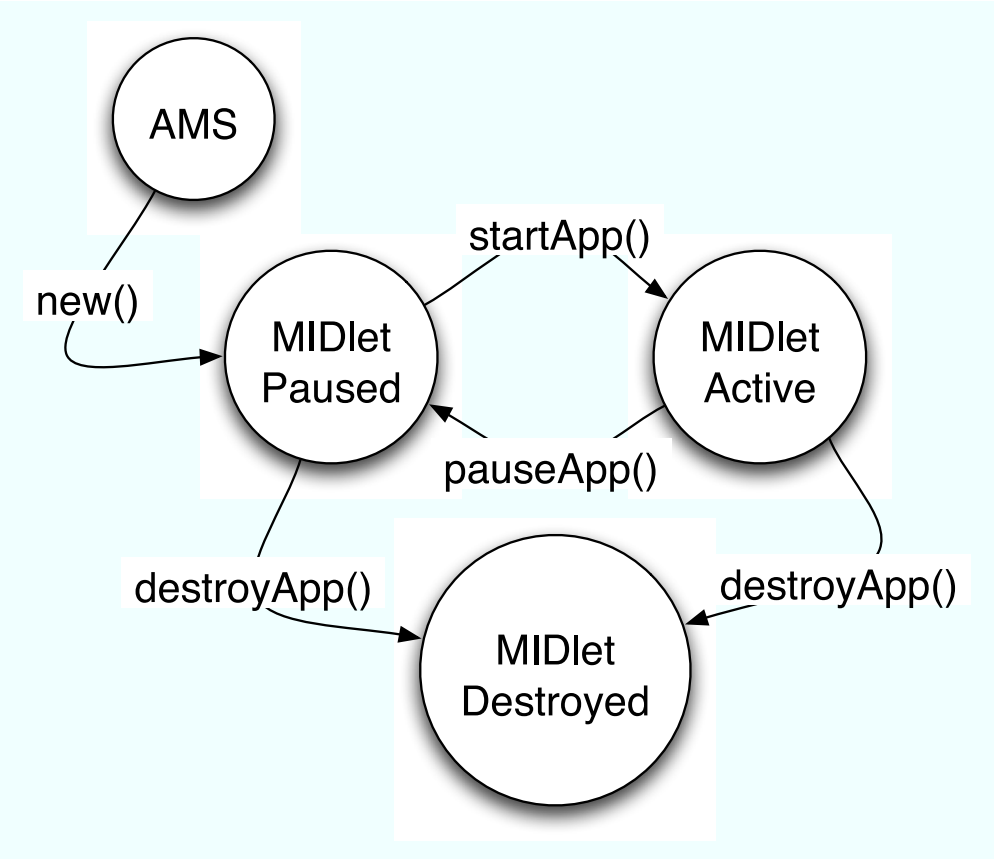


Abbildung 4.1: Zustandsautomat eines MIDlets (Vgl.: [VW02, S. 440])

Die Struktur des Zustandsautomaten wird im Code durch die an den Zustandübergängen in [Abb. : 4.1, S. 46] annotierten Methoden realisiert. Das Betriebssystem ist dadurch in der Lage beim Eintreffen eines externen Ereignisses, wie z. B. einem eingehenden Anruf, die Methode `pauseApp()` aufzurufen, durch die das MIDlet bis zu Fortführung pausiert wird. [Listing 4.1, S. 46] zeigt ein einfaches „Hello World“ MIDlet um den Sachverhalt zu verdeutlichen.

```
package hello;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class Test extends MIDlet {
    private TextBox tf;

    public Test() {
```

```
        tf = new TextBox("Hello", "World", 20, TextField.UNEDITABLE);
    }

    public void startApp() {
        Display.getDisplay(this).setCurrent(tf);
    }

    public void pauseApp() {}
    public void destroyApp(boolean unconditional) {}
}
```

Listing 4.1: Hello World MIDlet

4.1.2 User-Interface Entwicklung

Um die User-Interface Entwicklung eines MIDlets zu vereinfachen, kann ein sogenannter *Screen Designer* eingesetzt werden. Die Netbeans Entwicklungsumgebung bietet einen solchen mit dem *Mobility Pack* an[1]. Des weiteren bietet es einen sogenannten *Flow Designer*, mit dessen Hilfe einzelne „Seiten“, sogenannte *Forms*, leicht untereinander verküpft werden können. Möchte man keine Spiele oder Programme mit anspruchsvollen graphischen Elementen entwickeln, lässt sich mit den dargebotenen Werkzeugen relativ schnell das Frontend einer Applikation entwickeln. Um das Basisset der graphischen Elemente dennoch mit einer ansprechenderen Optik zu versehen, wurde das *J2ME Polish*[2] Projekt ins Leben gerufen. J2ME Polish lässt sich, zumindest laut Angabe der Entwickler, ebenfalls in die Netbeans IDE integrieren[3].

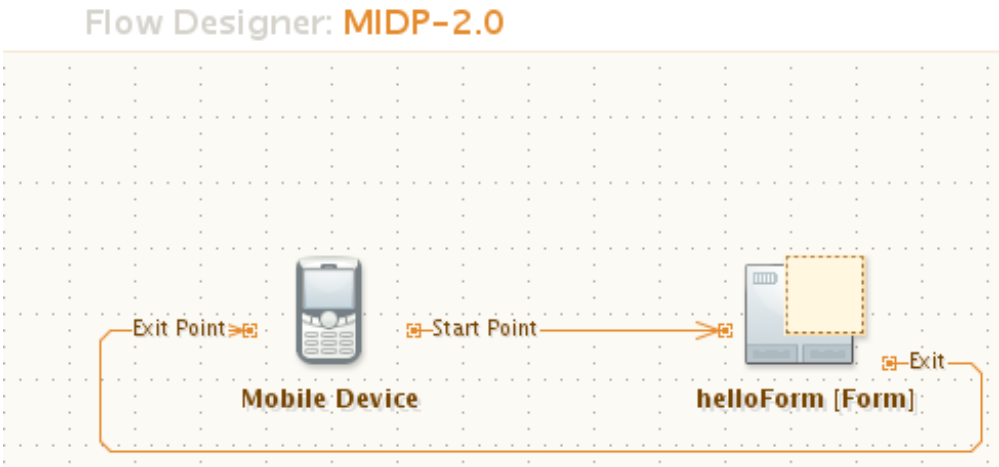


Abbildung 4.2: Netbeans Flow Designer

Die [Abb.: 4.2, S. 47] zeigt den *Flow Designer* für ein „Hello World“ Projekt. Wie zu sehen ist, verbindet man den Startpunkt der Applikation direkt mit der nach dem Start des MIDlets

[1] Entwicklungsumgebung und Mobility Pack können unter http://www.netbeans.org bezogen werden.
[2] Siehe http://www.j2mepolish.org/
[3] Siehe http://www.j2mepolish.org/docs/install.html#netbeans5

anzuzeigenden Form. Wird der mit Exit beschriftete Button des MIDlets gedrückt, terminiert die Applikation. Die Handhabung der Netbeans IDE, bezüglich der Applikationsentwicklung im J2ME Umfeld, soll an dieser Stelle nicht weiter vertieft werden[4].

4.2 Projektorganisation

Wie bei jedem grösseren Projekt, ist es auch bei der Entwicklung von Applikationen für mobile Endgeräte sinnvoll, für die verwendeten Entitätsobjekte eine Bibliothek zu erzeugen. Dadurch kann eine mehrfache Implementierung der Entitätsobjekte für differente Plattformen vermieden werden. Möchte man also eine Applikation entwickeln, die es ermöglicht, Daten zwischen einem Desktop-Computer und einem mobilen Endgerät auszutauschen, ist dieser Ansatz jedem anderen vorzuziehen. Da die OOP Bibliothek in der Lage ist, sowohl im J2SE, als auch im J2ME Umfeld zu operieren, bietet es sich an diese Form der Projektorganisation prinzipiell zu verwenden, da man sich so die Möglichkeit offen hält, einen Client für andere Plattformen zu implementieren. Eine beispielhafte Projektkonfiguration ist dann wie folgt gestaltet.

MeineApplikation-J2ME Dieses Projekt beinhaltet den Code der notwendig ist, um die Logik und die Anzeige des mobilen Endgeräts entsprechend der Anforderung zu manipulieren.

MeineApplikation-J2SE Dieses Projekt beinhaltet den Code der Applikationslogik sowie die notwendigen Manipulationen der Anzeige, um eine Änderung der Entitätsobjekte zu ermöglichen.

MeineApplikation-Entitätsobjekte In diesem Projekt sind einzig die zu beiden Projekten gehörenden Entitätsobjekte zu finden. Sollte eine Änderung der Entitätsobjekte erfolgen, wirkt sich die Änderung unweigerlich auf beide Projekte aus, womit sich eine Duplikation des Codes vermeiden lässt.

4.3 Beispielapplikation

Die Funktionsweise sowie der konkrete Einsatz der Bibliothek wird anhand einer J2ME Beispielapplikation demonstriert. Die Applikation ist bewusst sehr einfach gehalten. So existieren keine zusätzlichen *Forms* mit deren Hilfe die Entitätsobjekte manipuliert werden können. Eine Manipulation der Daten ist für die Demonstration nicht erforderlich. Die Applikation

[4] Eine Einführung in die MIDP Entwicklung mit Hilfe des Mobility Packs für Netbeans ist unter [Net06] zu finden.

bietet allerdings die Möglichkeit zur visuellen Repräsentation der Daten, damit die korrekte Übertragung derer verifiziert werden kann.

Die Beispielapplikation ist mit Hilfe des in Netbeans integrierten Emulators entwickelt worden. Zur Demonstration der Bibliothek muss der Emulator zweimal gestartet werden. Durch die Fähigkeit des Emulators eine Bluetooth-Schnittstelle zu emulieren, kann die Applikation ohne einen vorhandenen Bluetooth-Stack getestet werden. Netbeans ist allerdings nicht in der Lage die Bluetooth-Schnittstelle von einer J2SE Applikation auf ein J2ME Projekt zu emulieren, weswegen sich die Beispielapplikation auf ein J2ME Programm beschränkt.

Die Funktionsweise des Programms ist relativ einfach. Nachdem das Projekt in Netbeans importiert wurde, muss das Programm zweimal gestartet werden. Auf einem der beiden simulierten Geräte wird nun der Empfangsdienst gestartet. Man kann sich nun auf Wunsch die vorhandenen Daten anzeigen lassen. Das Programm reagiert mit einer Fehlermeldung, dass keine Daten vorhanden sind. Auf dem anderen Gerät wird im nächsten Schritt die Übertragung der (im Code konfigurierten) Daten veranlasst. Ist die Übertragung der Daten abgeschlossen, wird ein Dialogelement angezeigt, welches das Ende der Übertragung signalisiert. Auf dem empfangenden Gerät können die Daten jetzt angezeigt werden. Die Zustandsübergänge der Beispielapplikation sind in [Abb. : 4.3, S. 49], ein bebilderter Ablauf in [Abb. : 4.4, S. 53] dargestellt.

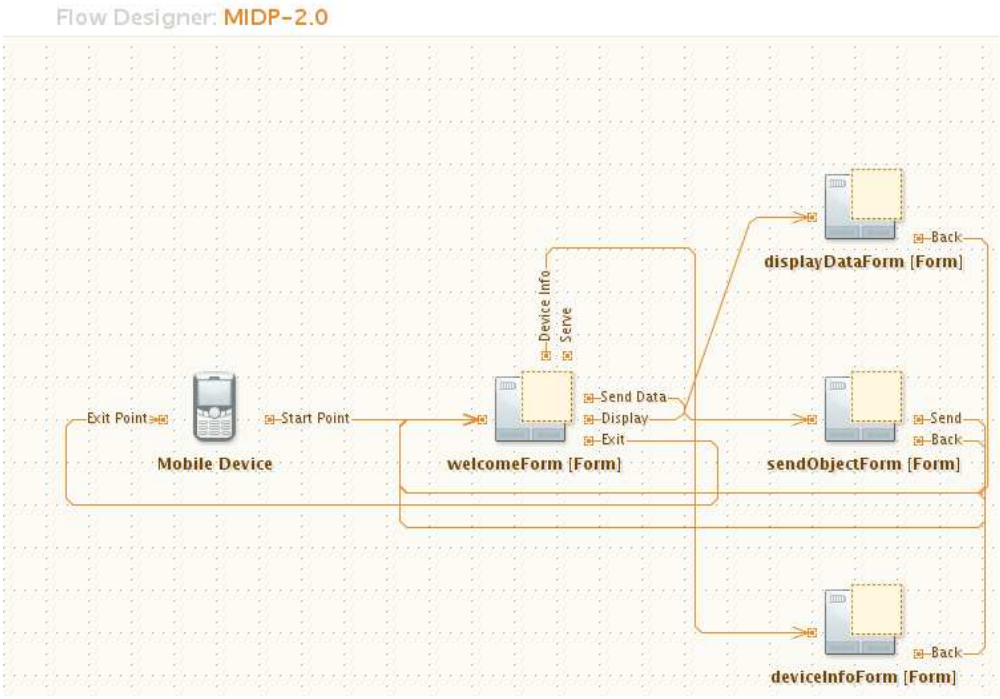


Abbildung 4.3: Zustandsübergänge der J2ME Beispielapplikation

Da innerhalb der Applikation keine Möglichkeit besteht die zu übertragenden Daten zu manipulieren bzw. zu setzen, werden die Daten innerhalb des Programmcodes direkt zugewiesen.

Die Konfiguration eines `Person` Objekts, das die Schnittstelle `IObexObjectPassing` implementiert, erfolgt wie in Listing [Listing 4.2, S. 50] gezeigt. Das Beispiel zeigt wie zwei `Person` Objekte konfiguriert, und anschliessend mit der Klasse `BulkObjectPusher` versendet werden können.

```
Person p = new Person();
  p.setName("Schneider");
  p.setVorname("Rosemarie");
  p.setAge(37);

Person hans = new Person();
  hans.setName("Peter");
  hans.setVorname("Hans");
  hans.setAge(37);

Vector transferringObjects = new Vector();
  transferringObjects.addElement(p);
  transferringObjects.addElement(hans);

new BulkObjectPusher(transferringObjects).addObserver(this);
```

Listing 4.2: Verwendung von `BulkObjectPusher`

Die Klassen der OOP Bibliothek sind in ihrer Anwendung sehr einfach. Der in [Listing 4.2, S. 50] gedruckte Code führt den Transfer der Daten in Zeile 15 durch. Wie zu sehen ist, wird die Klasse ohne Zuweisung initialisiert. Eine Zuweisung auf eine lokal verfügbare Variable ist nicht notwendig. Durch den an die Klasse angeschlossenen *Observer*[5] ist es möglich, über das Ende der Datenübertragung informiert zu werden. Es ist weiterhin zu erkennen, dass dem Objekt keine Zieladresse übergeben wurde. Das ist ebenfalls nicht notwendig, da die Bibliothek in der Lage ist, ihren Kommunikationspartner selbstständig zu lokalisieren.

Um auf dem entfernten Gerät Daten empfangen zu können muss der `ObjectReceiver` instanziiert werden. Er ist, ebenso wie die Klassen `ObjectPusher` und `BulkObjectPusher` nicht blockierend implementiert.

```
Vector receivedObjects = new Vector();

new ObjectReceiver(receivedObjects);
```

Listing 4.3: Verwendung von `ObjectReceiver`

Der vorgestellte Code in [Listing 4.3, S. 50] empfängt die übertragenen Objekte und speichert sie in den Vector `receivedObjects`, aus dem die empfangenen Daten zu einem späteren Zeitpunkt wieder ausgelesen und weiterverabeitet werden können.

Damit dies alles möglich ist, muss das zu übertragende Objekt lediglich das Interface `IObexObjectPassing` implementieren. Ein partieller Ausschnitt der Implementierung des

[5] Siehe [Kapitel 3.2.2, S. 37]

`Person` Objekts ist in [Listing 4.4, S. 51] dargestellt. Die restlichen Methoden sind ausschliesslich Accessor/Mutator Methoden des Objekts.

```
public byte[] getAsByteArray()
{
    bout = new ByteArrayOutputStream();
    dout = new DataOutputStream( bout );
    byte[] ret = null;

    try
    {
        /* Write values */
        dout.writeUTF(name);
        dout.writeUTF(vorname);
        dout.writeInt(age);
        dout.flush();

        /* do temp copy so we can close
         * the writers
         */
        ret = bout.toByteArray();
        dout.close();
        bout.close();
    }
    catch (IOException ex)
    {
        ex.printStackTrace();
    }

    return ret;
}

public void setObjectData(byte[] ba)
{
    bin = new ByteArrayInputStream(ba);
    din = new DataInputStream(bin);
    try
    {
        this.name = din.readUTF();
        this.vorname = din.readUTF();
        this.age = din.readInt();

        din.close();
        bin.close();
    }
    catch (IOException ex)
    {
        ex.printStackTrace();
    }
}
```

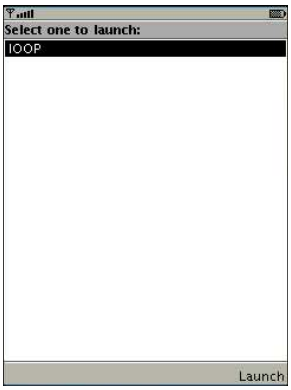
Listing 4.4: Implementierung von `IObexObjectPassing`

Die wichtigsten Elemente, des in [Listing 4.4, S. 51] dargestellten Codes sind die Zeilen 10-12 sowie die Zeilen 36-38. Mit diesen Zeilen werden die Daten des Objekts in ein Byte-Array

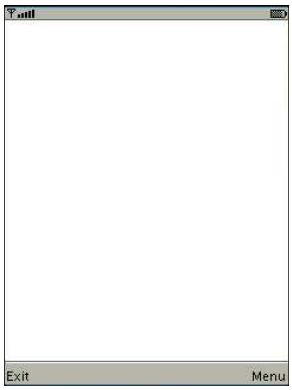
geschrieben (10-12) und können ebenfalls von einem Byte-Array gelesen werden (36-38). Dieser Mechanismus ist zur korrekten Funktionsweise der OOP-Bibliothek notwendig[6].

Der Ablauf der Beispielapplikation ist in [Abb. : 4.4, S. 53] dargestellt. Da die Applikation auf zwei Mobiltelefonen gestartet werden muss, wird im Begleittext das jeweilige Gerät mit »1« oder »2« markiert. Ist keine Markierung angegeben, so bezieht sich die Anzeige auf beide Telefone.

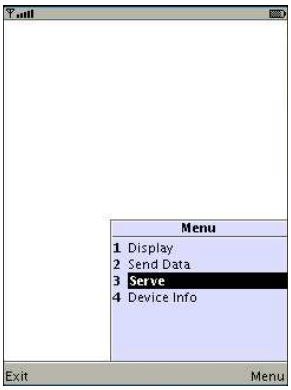
[6] Siehe [Kapitel 3.3, S. 40]



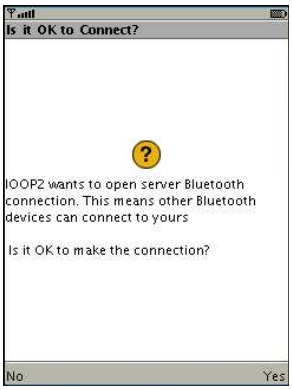
(a) Auf beiden Telefonen muss zunächst das Programm mittels “Launch” gestartet werden.



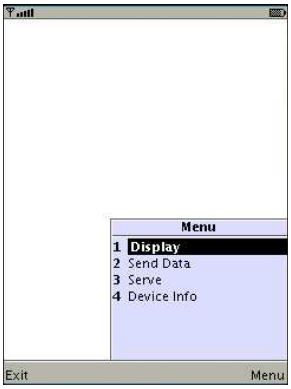
(b) Anzeige nach dem Programmstart.



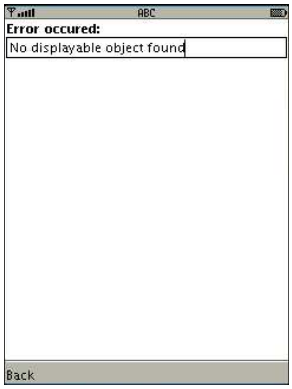
(c) »1« Der Server muss gestartet werden.



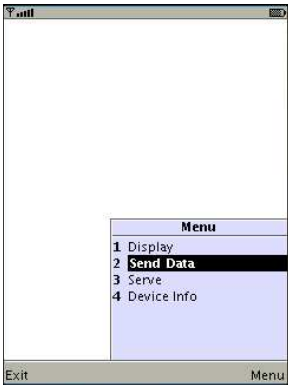
(d) »1« Der Start des Servers muss einmalig bestätigt werden, da ein Zugriff auf das Bluetooth API erfolgt.



(e) »1« Die vorhandenen Daten sollen angezeigt werden.

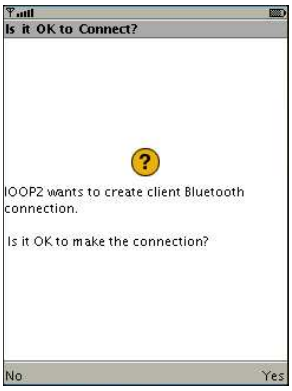


(f) »1« Es sind keine Daten verfügbar, mit “Back” verlassen

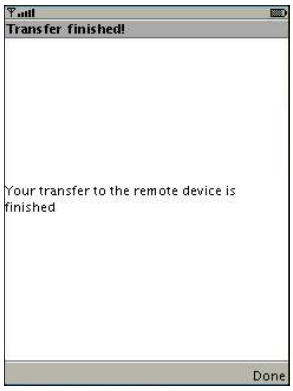


(g) »2« Daten sollen übertragen werden.

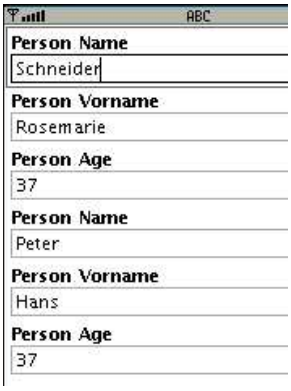
(h) »2« Zum senden “Send” wählen.



(i) »2« Der Zugriff auf das Bluetooth API muss einmalig bestätigt werden.



(j) »2« Die Übertragung wird bestätigt, mit “Done” verlassen.



(k) »1« Empfangene Daten können angezeigt werden.

Abbildung 4.4: Ablauf der Beispielapplikation

KAPITEL 5

Konklusion und Ausblick

Durch den Einsatz der OOP Bibliothek wird die Entwicklung von Applikationen, die Daten über eine Bluetooth Funkstrecke versenden möchten, wesentlich vereinfacht. Durch die Implementierung des von der Bibliothek vorgegebenen Interfaces erhält man zusätzlich die Möglichkeit, die erstellten Objekte direkt mit dem RMS zu verwenden, was den Zugewinn durch den Einsatz der Bibliothek erhöht.

Die notwendigen Zeilen Code, die normalerweise zum Verbindungsauf- und -abbau geschrieben werden müssten, sinken durch den Einsatz der Bibliothek auf ein Minimum. Mit welchem minimalen Aufwand eine Applikation erstellt werden kann, die in der Lage ist Daten zu einem entfernten Endgerät zu senden, ist in Kapitel 4 demonstriert worden. Wieviel Aufwand es bedeuten würde die selbe Applikation mit Hilfe des Peer2Me Frameworks zu entwickeln, konnte auf Grund der späten Entdeckung, nicht evaluiert werden.

Beim Umgang mit der Bluetooth-Technologie war festzustellen, dass sich die Anwendungsentwicklung für mobile Endgeräte, teilweise problematisch darstellt. So ist bspw. bei vielen Mobiltelefonen der JSR-82 lediglich teilweise implementiert, was zur Folge hat, dass die OBEX Bibliotheken nachgerüstet werden müssen, was auf Kosten der geringen Speicherkapazität der Geräte teuer erkauft werden muss. Die strikten Sicherheitsmechanismen der J2ME Architektur legen einem in diesem Fall ebenfalls Steine in den Weg. So ist eine globale Nachrüstung der OBEX-Funktionalität, auf Grund des Verbots der nachträglichen Installation externer Bibliotheken, nicht möglich. Als weiteres Problem stellt sich die Vielzahl verfügbarer Bluetooth-Stacks dar. Möchte man eine Applikation für ein Desktop-System schreiben, so muss für fast jeden verfügbaren Bluetooth-Stack eine entsprechende Binärdatei zur Verfügung gestellt werden.

Die Sicherheit der Bluetooth-Architektur sollte jedoch nicht überschätzt werden. Die Behauptung, dass sich Bluetooth wegen seiner kurzen Distanz nur schwer abhören lasse, konnte erfolgreich widerlegt werden. Mit Hilfe spezieller Richtantennen ist es möglich, Geräte aus Entfernungen bis zu 1.6 Km anzusteuern und erfolgreich auszuspähen [Zet04, S. 2]. Allein durch die Tatsache, dass es sich bei Bluetooth um eine Kurzstreckenfunktechnologie handelt, sollte man sich demnach nicht in falscher Sicherheit wiegen.

Bezüglich der Sicherheit der Bibliothek gilt es zu prüfen, ob sich im konkreten Einsatz, der OBEX Header manipulieren lässt, sodass das Gerät dazu veranlasst werden kann, eine nicht den Daten zugehörige Klasse zu laden.

Durch die Transportprotokollunabhängigkeit des OBEX Protokolls lässt sich die Bibliothek leicht auf andere Transportprotokolle portieren. Eine Weiterentwicklung der Bibliothek, die zusätzliche Transportprotokolle unterstützt, wäre in diesem Zusammenhang vorstellbar. So wäre es in Zukunft denkbar, dass die eingangs erwähnte „Einkaufszettel“-Applikation das mobile Endgerät über Bluetooth und den vorhandenen Desktop-Computer über TCP/IP ansteuert.

ANHANG A

Entwicklungsumgebung

Im Folgenden wird beschrieben, wie sich die entwickelte Bibliothek innerhalb der Netbeans IDE einsetzen lässt. Die Beschreibung umfasst dabei die Erzeugung eines neuen Projektes welches die Bibliothek verwendet, sowie die notwendigen Schritte um die Beispielapplikation in Betrieb zu nehmen.

A.1 Anlegen eines J2ME Projekts mit OOP

Nachdem die IDE gestartet ist, kann mit der Tastenkombination »STRG+SHIFT+N« ein neues Projekt eröffnet werden. Man wählt nun, wie in [Abb.: A.1, S. 58] gezeigt, „Mobile“→„Mobile Application“ und fährt mit dem „Next“ Button fort.

Als nächstes zeigt sich einem der Dialog aus [Abb.: A.2, S. 58], in dem man einen Projektordner, sowie einen Namen für das Projekt spezifiziert. Die Erzeugung des „Hello MIDlets“ kann deaktiviert werden, bleibt jedoch in diesem Beispiel aktiv.

Der Dialog kann jetzt mit einem Klick auf „Finish“ beendet werden. Anschliessend zeigt sich einem [Abb.: A.3, S. 59].

Nachdem die Applikation erzeugt wurde, wird gemäss der Projektorganisation aus [Kapitel 4.2, S. 48] eine Entitäsbibliothek erzeugt. Durch die Tastenkombination »STRG+SHIFT+N« wird erneut der „New Project“ Dialog geöffnet. Diesmal fällt die Auswahl jedoch auf „Mobil“→„Mobile Class Library“, wie in [Abb.: A.4, S. 59] zu sehen ist.

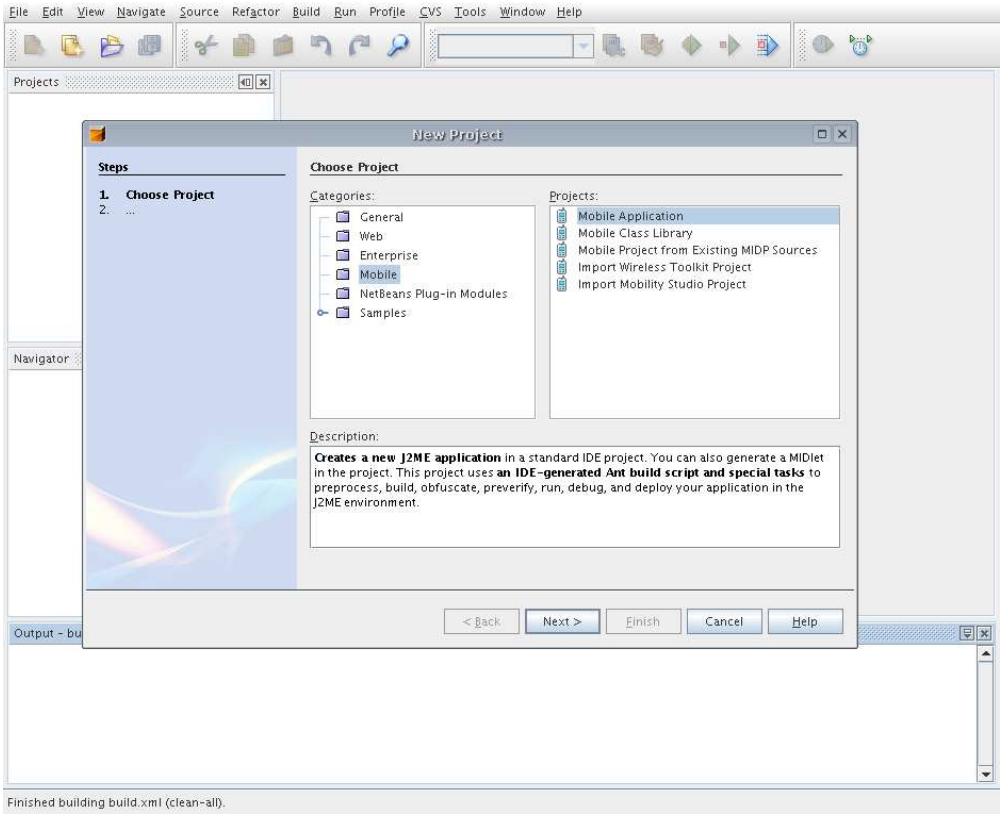


Abbildung A.1: Neue Applikation anlegen

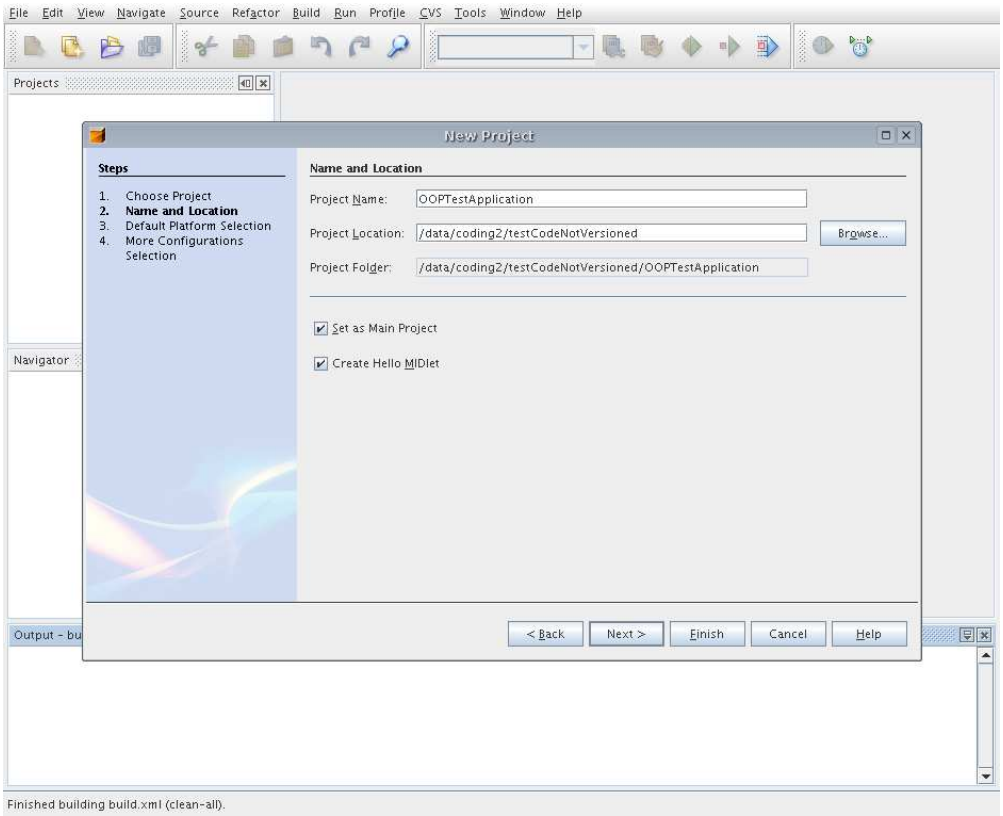


Abbildung A.2: Applikation benennen

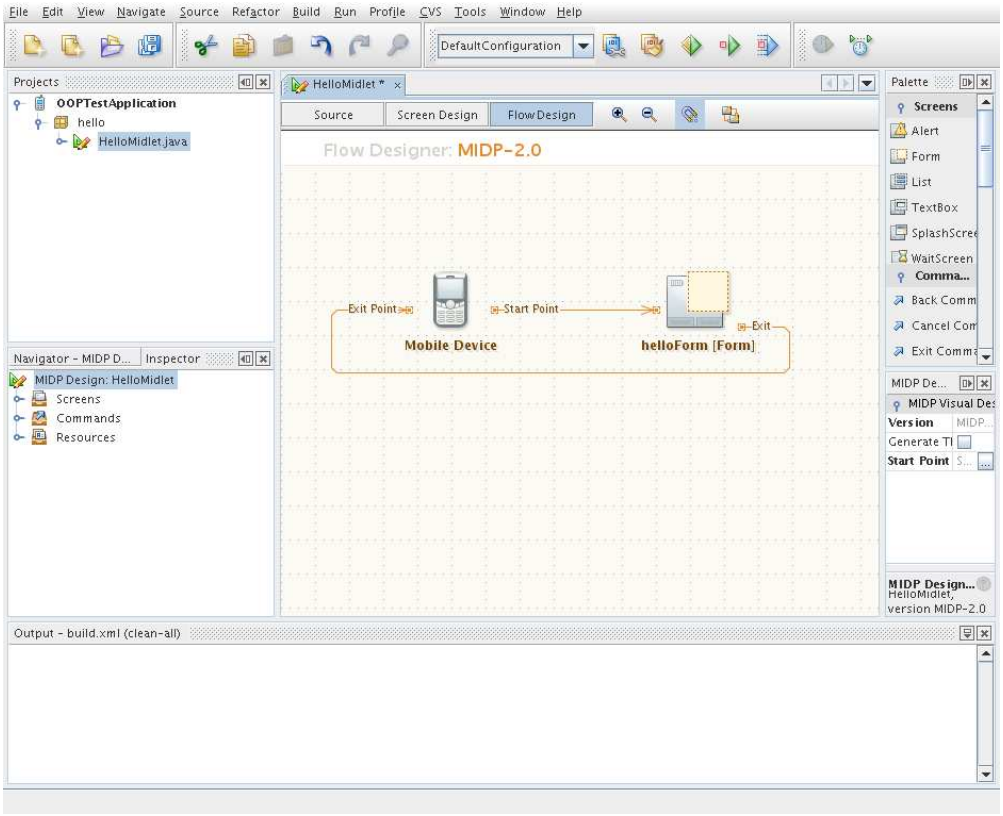


Abbildung A.3: Arbeitsplatz nach anlegen der Applikation

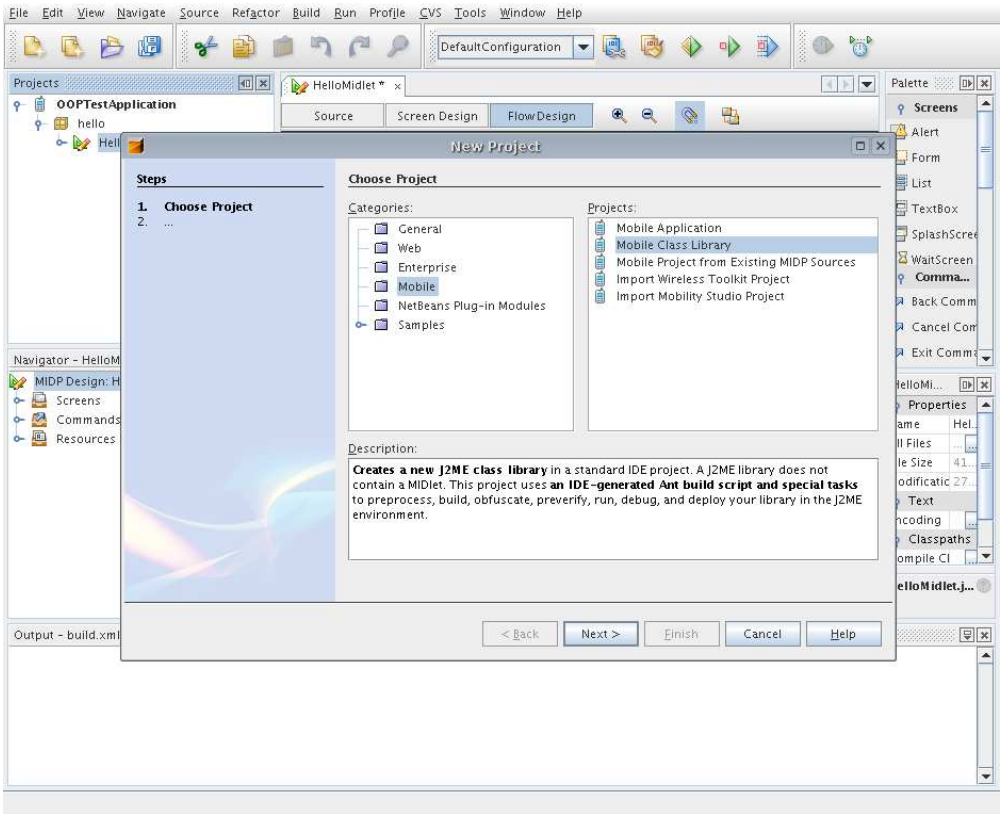


Abbildung A.4: Entitätsbibliothek anlegen

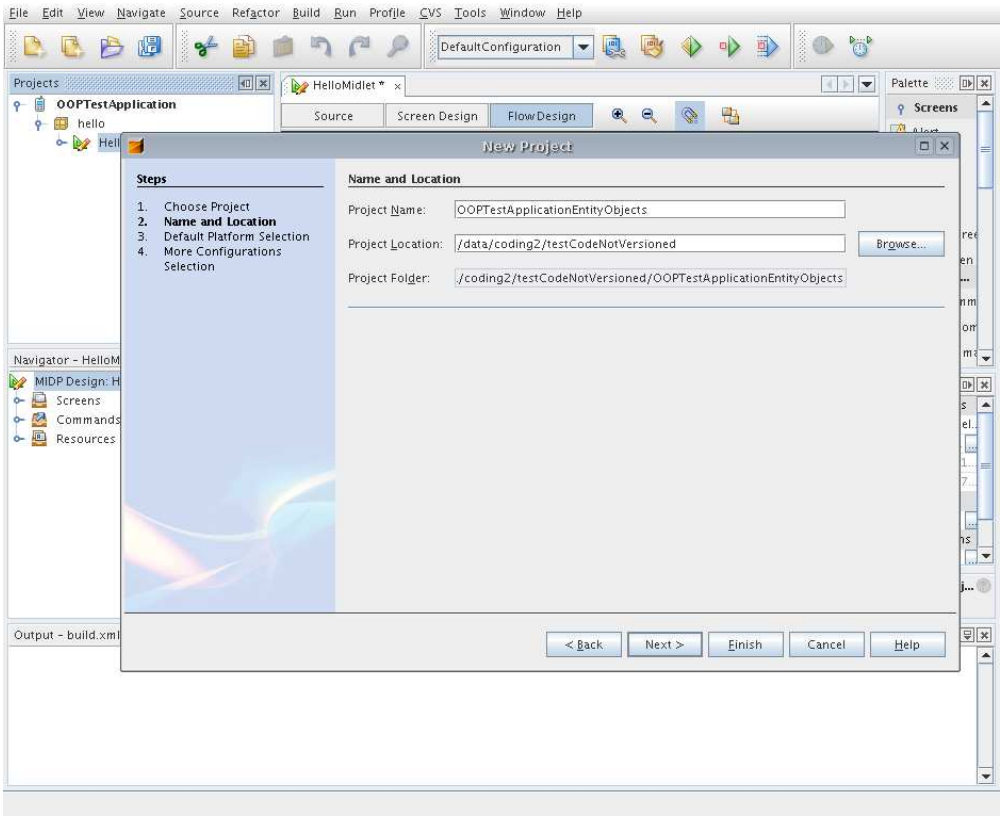


Abbildung A.5: Entitätsbibliothek benennen

Man spezifiziert für die Bibliothek, wie in [Abb. : A.5, S. 60] gezeigt, ebenfalls einen Namen und einen Projektordner und beendet den Dialog mit einem Klick auf „Finish“.

Der vorzufindene Arbeitsplatz sieht nun aus wie in [Abb. : A.6, S. 61] dargestellt.

Als nächstes muss die OOP Bibliothek Netbeans bekannt gemacht werden. Dies geschieht über den „Library Manager“ der durch die Tastenkombination »ALT+T+L« geöffnet werden kann. In dem folgenden Dialog selektiert man „Mobile Libraries“, und klickt auf „New Library“, woraufhin ein neuer Dialog erscheint, in dem der Name der Bibliothek eingetragen werden muss (siehe [Abb. : A.7, S. 61]).

Ist dieser Schritt abgeschlossen muss der Bibliotheksbezeichnung noch ein Java-Archiv (JAR) hinzugefügt werden. Dies geschieht durch einen Klick auf „Add JAR“. In dem aufkommenden Dialog muss nun die JAR-Datei der OOP Bibliothek von dem lokalen Dateisystem gewählt werden (ObexObjectPassing.jar). Ist dieser Schritt abgeschlossen, zeigt sich einem [Abb. : A.8, S. 62]

Die Bibliothek ist Netbeans nun bekannt und kann in den der IDE bekannten Projekten verwendet werden. Die Bibliothek muss zunächst dem Entitätsprojekt zugeordnet werden. Dazu klickt man mit der rechten Maustaste auf das Entitätsprojekt, und selektiert, wie in

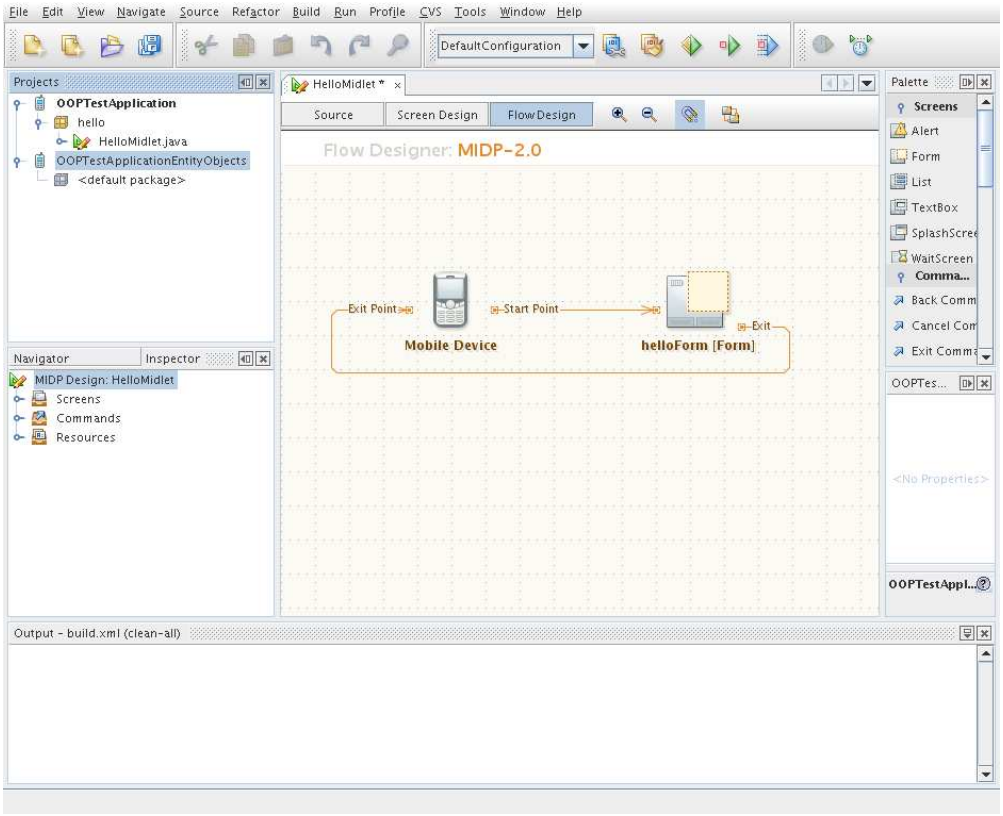


Abbildung A.6: Arbeitsplatz nach anlegen der Entitätsbibliothek

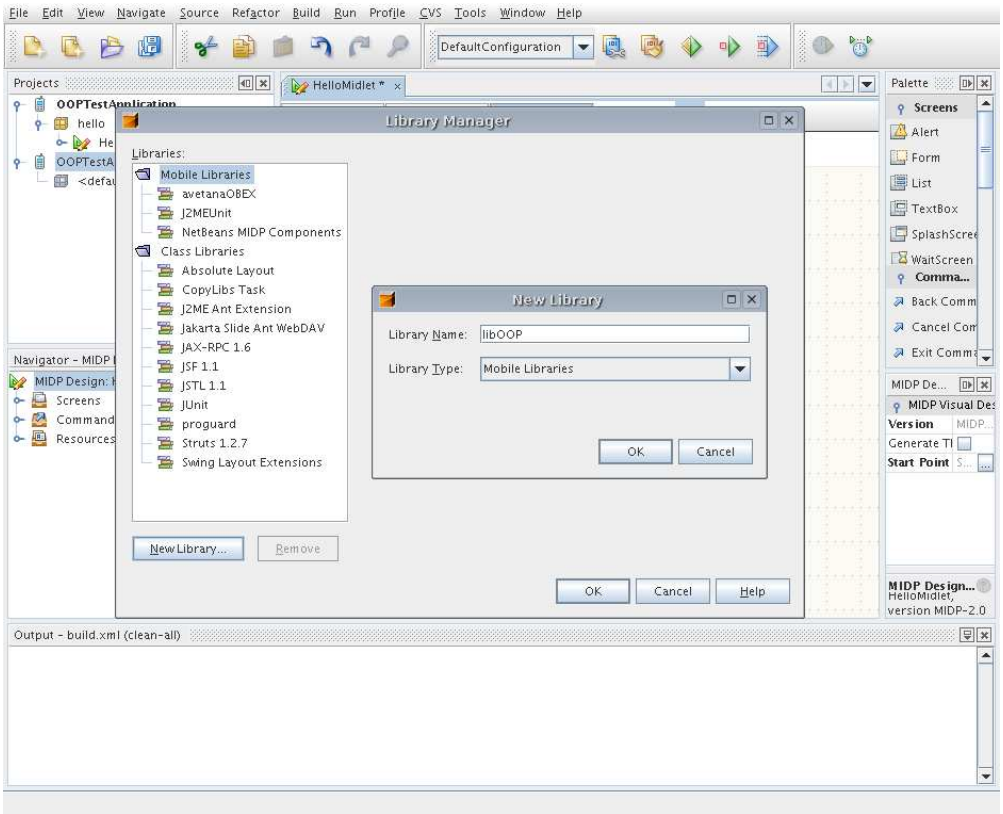


Abbildung A.7: Erzeugen einer neuen Bibliothek („Library Manager“)

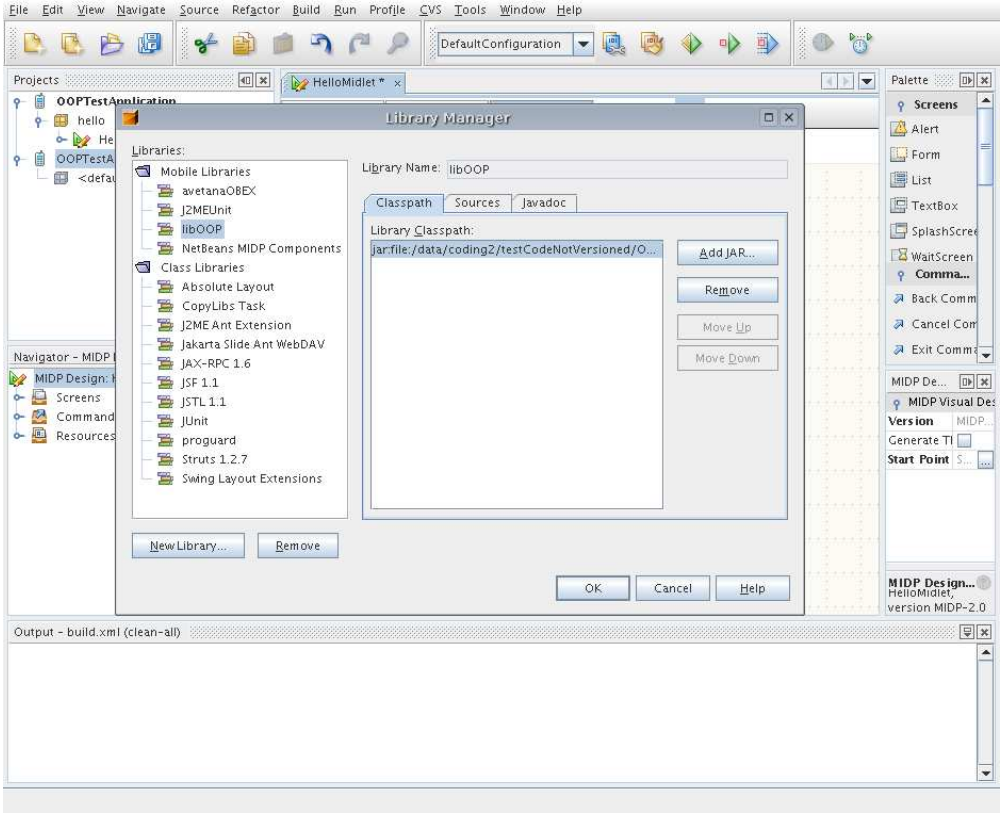


Abbildung A.8: JAR Datei selektiert und hinzugefügt

[Abb. : A.9, S. 63] gezeigt „Libraries & Resources“. Dort ordnet man die Bibliothek mit einem Klick auf „Add Library“ dem Projekt zu.

Nachdem die Bibliothek mit einem Klick auf „Add Library“ und „Ok“ hinzugefügt wurde, muss die Entitätsbibliothek dem J2ME Projekt zugeordnet werden. Dazu wählt man , nach einem Rechtsklick auf das J2ME Projekt „Properties“→„Libraries & Resources“. Das Entitätsprojekt kann nun, mit einem Klick auf „Add Project“ vom lokalen Dateisystem gewählt werden (siehe [Abb. : A.10, S. 63]).

Damit die OOP Bibliothek auch in der J2ME-Anwendung zur Verfügung steht muss sie, nach dem obligatorischen Rechtsklick, und der Wahl von „Properties“→„Libraries & Resources“, mit der Selektion von „Add Library“ hinzugefügt werden.

Nach einem Klick auf „Add Library“→„Ok“ ist die Bibliothek korrekt in das Projekt eingebunden worden. Ab diesem Zeitpunkt kann entwickelt werden.

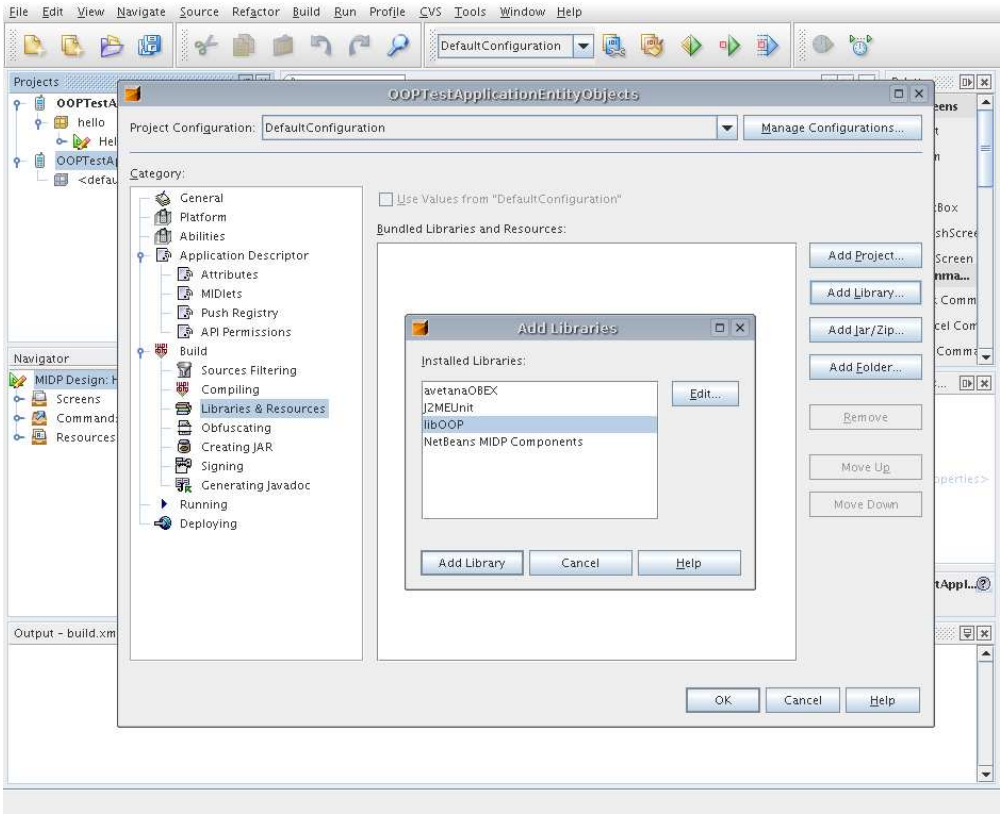


Abbildung A.9: Bibliothek zum Entitätsprojekt hinzufügen

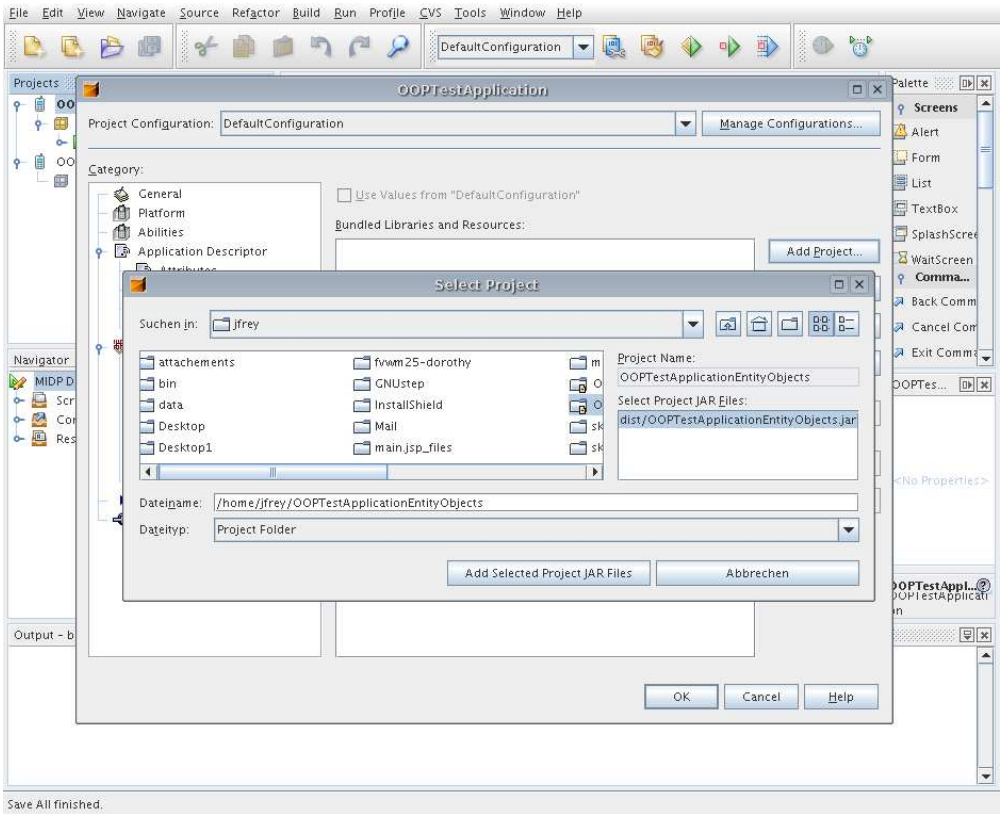


Abbildung A.10: Entitätsprojekt der Applikation hinzufügen

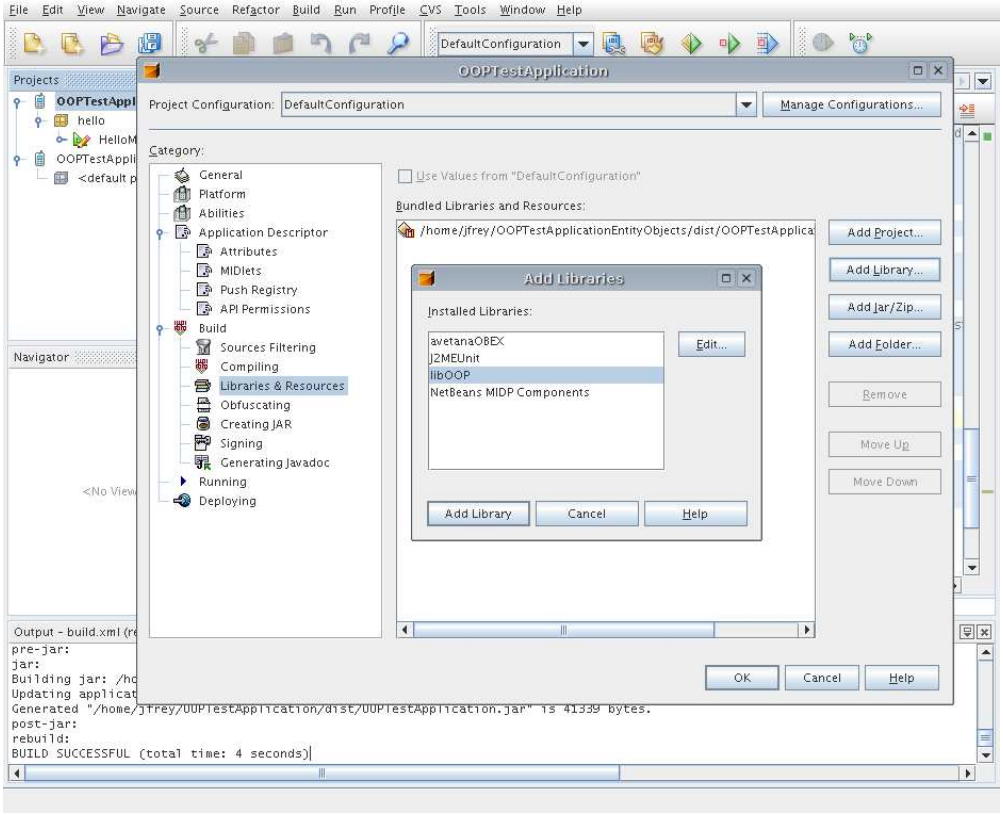


Abbildung A.11: OOP Bibliothek der Applikation hinzufügen

A.2 Beispielapplikation in Betrieb nehmen

Um die Beispielapplikation in Betrieb zu nehmen, müssen die Dateien *IOOP2.zip*, *IOOPEntityLib.zip*, *ObexObjectPassingSrc.zip* und *avetanaObex.jar* auf dem lokalen Dateisystem – mit Ausnahme der JAR Datei – entpackt werden.

Ist dies geschehen, muss die avetanaOBEX Bibliothek unter dem Namen „avetanaOBEX“ im „Library Manager“ der IDE unter „Mobile Libraries“ registriert werden. Der Vorgang ist in [Abb. : A.7, S. 61] und [Abb. : A.8, S. 62] bebildert und wird in [Kapitel A.1, S. 57] für ein neues Projekt beschrieben. Ist der Import der Bibliothek abgeschlossen, können die drei Projektordner mit der Tastenkombination »STRG+SHIFT+O« geöffnet, und der IDE hinzugefuegt werden. Hat alles geklappt muss ggf. das „IOOP“ Projekt mit der rechten Maustaste als „Set Main Project“ markiert werden. Die Applikation kann anschliessend durch einen Tastendruck auf »F6« augeführt werden. Es ist zu beachten, dass zur korrekten Demonstration, wie sie in [Abb. : 4.4, S. 53] dargestellt wird, das Projekt zweimal gestartet werden muss.

ANHANG B

Glossar und Abkürzungsverzeichnis

A

ACL, S. 9 *Asynchronous Connection-Oriented (logical transport)*

ACL-C, S. 10 *ACL Control*

ACL-U, S. 10 *User Asynchronous/Isochronous*

AFH, S. 7 *Adaptive Frequency-Hopping spread spectrum*, Verfahren zur Reduktion der Störanfälligkeit gegenüber anderen Funktechnologien (z. B. WLAN)

AMS, S. 45 *Application Management Software*

API, S. 16 *Application Programming Interface*

ARQ, S. 10 *Acknowledgement/Repeat Request*

ASB, S. 9 *Active Slave Broadcast (logical transport)*

B

BCC, S. 23 *Bluetooth Control Center*, Zentrale Anlaufstelle welche die Bluetooth-Einstellungen des lokalen Geräts verwaltet.

BD_ADDR, S. 11 *Bluetooth Device Address*

C

CDC, S. 16 *Connected Device Configuration*

CID, S. 11 *Channel Identifier*

CLDC, S. 15 *Connected Limited Device Configuration*, Grundlage von J2ME™. Geht aus den JSRs 30 und 139 hervor (Version 1.0/1.1)

CLDC HotSpot Implementation, S. 15 Nachfolgeimplementierung der KVM mit verbesserter Performanz (8-10x schneller)

D

Discovery, S. 25 dt. Entdeckung; bezeichnet den Prozess der es zwei Bluetooth-Geräten ermöglicht sich gegenseitig zu lokalisieren.

E

EDR, S. 7 *Enhanced Data Rate*

EJB, S. 31 Enterprise JavaBeans

eSCO, S. 9 *Extended Synchronous Connection-Oriented (logical transport)*

eSCO-S, S. 10 *User Extended Synchronous*

F

FTP, S. 9 *File Transfer Protocol*

G

GAP, S. 12 *Generic Access Profile*

GCF, S. 16 *Generic Connection Framework*, offeriert die Möglichkeit, auf Ein- und Ausgabeoperationen sowie Netzwerkressourcen zuzugreifen.

GFSK, S. 6 *Gaussian Frequency Shift Keying*, Bei Bluetooth oder DECT Technologie eingesetztes Modulationsverfahren.

GIAC, S. 13 *General/Unlimited Inquiry Access Code*

GOEP, S. 29 *Generic Object Exchange Profile*, Bluetooth-Profil welches das OBEX Protokoll implementiert.

H

HCI, S. 11 *Host Controller Interface*, befindet sich zwischen logischem und L2CAP-Layer und stellt eine einheitliche Schnittstelle zum Bluetooth-Controller dar.

HTTP, S. 28 *Hypertext Transfer Protocol*

I

ID, S. 19 Identifikationsnummer

IDE, S. 48 *Integrated Development Environment*

ISM, S. 6 Industrial, Scientific, and Medical Band

J

J2ME , S. 14 Java 2 Platform, Micro Edition

JABWT, S. 21 *Java API for Bluetooth Wireless Technology*

JSR, S. 15 Java Specification Request

K

KVM, S. 15 *K Virtual Machine*, manchmal auch Kilobyte Virtual Machine Virtuelle Maschine, die mit sehr geringen Hardwareanforderungen auskommt.

L

L2CAP, S. 11 *Logical Link Control and Adaption Protocol*

LC, S. 10 *Link Control*

LIAC, S. 13 *Limited Dedicated Inquiry Access Code*

LLID, S. 10 *Logical Link Identifier*, Feld im Basisband-Paketkopf um die logische Verbindung zu identifizieren

LM, S. 10 *Link Manager*, verantwortlich für die Erzeugung, Modifikation und Freigabe logischer Verbindungen.

LMP, S. 10 *Link Manager Protocol*, Protokoll, mit dessen Hilfe Kontrollinformationen zwischen den *Link Manager* Instanzen zweier Endgeräte ausgetauscht werden können.

M

MID, S. 18 *Mobile Information Device*

MIDlet, S. 18 Programm, dass auf einem MID ausgeführt werden kann.

MIDP, S. 18 *Mobile Information Device Profile*

MTU, S. 24 *Maximum Transfer Unit*, Anzahl an Daten die unfragmentiert in einem Paket übertragen werden können.

O

OBEX, S. 22 OBject EXchange (Protocol)

OEM, S. 22 *Original Equipment Manufacturer*

OOP, S. 32 *OBEX Object Passing*

P

PDA, S. 15 Personal Digital Assistant, kleiner Computer, der typischerweise in einer Handfläche Platz findet.

PSB, S. 9 *Parked Slave Broadcast (logical transport)*

R

RAM, S. 22 *Random Access Memory*

RAND, S. 11 *Pseudo-Random-Number*, Pseudozufallszahl

RFCOMM, S. 27 Emulation mehrerer Serieller RS-232 Ports zwischen zwei Bluetooth-Geräten

RMI OP, S. 31 *RMI Optional Package*

RMS, S. 19 *Record Management System*, vom MIDP spezifizierter Mechanismus zur persistenten Speicherung von Daten

ROM, S. 22 *Read Only Memory*

RSSI, S. 7 *Received Signal Strength Indicator*

S

SCO, S. 9 *Synchronous Connection-Oriented (logical transport)*

SCO-S, S. 10 *User Synchronous*

SDDB, S. 25 *Service Discovery Database*

SDP, S. 22 *Service Discovery Protocol*

SDU, S. 11 *Service Data Units*

SPP, S. 27 *Serial Port Profile*

T

TCK, S. 22 *Technology Compatibility Kits*, Test-Suite die eine korrekte Implementierung der Spezifikation überprüfen kann.

TCP/IP, S. 35 *Transmission Control Protocol/Internet Protocol*

U

URL, S. 27 *Uniform Resource Locator*, einheitliche Ortsangabe für Ressourcen.

V

VM, S. 14 Virtual Machine, Programm, das für die entsprechende Plattform übersetzten Bytecode interpretiert.

W

WLAN, S. 7 *Wireless Local Area Network*

WTK, S. 32 *Wireless Toolkit*, Ein Set an Emulatoren und zugehörigen Bibliotheken, welche die Entwicklung von Applikationen für mobile Endgeräte ermöglicht.

Abbildungsverzeichnis

Tabellenverzeichnis

Listings

Literaturverzeichnis

[Ano06] ANONYMOUS. *HCI Layer Tutorial*. http://www.palowireless.com/. Jan. 2006

[Bal00] BALZERT, Helmut: *Lehrbuch der Software-Technik*. Bd. 1. 2. Auflage. Spektrum, Akad. Verl., 2000 37

[Bal05] BALDWIN, Richard G. *Digital Signatures 101 using Java*. http://www.developer.com/. 2005

[Bec05] BECKER, Markus. *Drahtlose lokale Netze (Ad-hoc / Bluetooth Kommunikation)*. http://jerry.c-lab.de/. Jun. 2005 6

[BJJ04] BALA KUMAR, C ; J. KLINE, Paul ; J. THOMPSON, Timothy: *Bluetooth Application Programming with the Java APIs*. Morgan-Kaufmann Verlag, 2004 28

[Blu03] BLUETOOTH SIG. *Specification of the Bluetooth System - Wireless connections made easy - Version 1.2*. http://bluetooth.org/. Nov. 2003 5, 6, 7, 8, 9, 10, 11, 12, 13

[Blu04] BLUETOOTH SIG. *Specification of the Bluetooth System - Wireless connections made easy - Version 2.0 + EDR*. http://www.bluetooth.org/. Nov. 2004

[Blu05] BLUETOOTH SIG. *Generic Object Exchange Profile*. http://www.bluetooth.org/. Feb. 2005 29

[Blu06] BLUETOOTH SIG. *Assigned Numbers - Bluetooth Baseband*. https://www.bluetooth.org/. Mar. 2006 13

[BSM06] BÜRKI, Reto ; SCHÄRER, Raffael ; MÜLLER, Martin: J2ME-Objektserialisierung: Frameworks im Vergleich. In: *Java Spektrum* (2006), Feb., Nr. 1, S. 35–38 31

[Bv05] BACHFELD, Daniel ; ŽIVADINOVIĆ, Dušan: Wurzelbehandlung – Sicherheitslücken bei Bluetooth-Handys. In: *c't - magazin für computer technik* (2005), Dez., Nr. 26, S. 212–216

[C+05] COURTNEY, Jon et al. *Connected Device Configuration (CDC) 1.1*. http://jcp.org/. Aug. 2005 16

[Fit05] FITTON, Dan. *Java Bluetooth HOWTO*. http://www.caside.lancs.ac.uk/. Dez. 2005

[For04] FORUM NOKIA GLOBAL WEB SITE. *Introduction To The FileConnection API (With Example) v1.1*. http://www.forum.nokia.com/. Nov. 2004

[FS06] FREY, Jens ; SCHWARZ, Matthias. *Kryptographie - Skriptum zur Vorlesung*. http://www.coffeecrew.org/. Mar. 2006 12

[Gho06] GHOSH, Soma. *J2ME record management store*. http://www-128.ibm.com/. Mar. 2006 36

[Has06] HASIK, Lukas. *Mobility Pack on Mac*. http://blogs.sun.com/. Mar. 2006 33

[HHT03] HANDY, M. ; HAASE, M. ; TIMMERMANN, D. *Der Verlust der informationellen Selbstbestimmung? Anonymitätsaspekte bei Bluetooth und WLAN*. http://www.vs.inf.ethz.ch/. 2003

[Hol03] HOLTMANN, Marcel. *Bluetooth und ISDN unter Linux*. http://www.holtmann.org/. Sep. 2003

[Hop05a] HOPKINS, Bruce. *Bluetooth boogies, Part 1: File transfer with JSR-82 and OBEX*. http://www-128.ibm.com/. Sep. 2005

[Hop05b] HOPKINS, Bruce. *Bluetooth boogies, Part 2: Creating the Bluetooth Music Store*. http://www-128.ibm.com/. Nov. 2005

[Jar05] JARVI, Keane. *RXTX: The Prescription for Transmission*. http://users.frii.com/. Jan. 2005

[Kje05] KJELL, J.H. *Bluetooth - Part 9: Service Discovery*. http://www.vs.inf.ethz.ch/. Apr. 2005

[Krü00] KRÜGER, Guido: *GoTo Java™2*. 2. Auflage. Addison-Wesley Verlag, 2000. – Studentenausgabe

[Lab06] LABAYE, Denis. *Bluecove documentation*. http://bluecove.sourceforge.net/. Jan. 2006 35

[LN05] LUND, Carl-Hendrik W. ; NORUM, Michael S.: *The Peer2Me Framework - A Framework for Mobile Collaboration on Mobile Phones*, Norwegian University of Science and Technology, Diplomarbeit, Jun. 2005 32

[M+05] MILIKICH, Mike et al. *Java™APIs for Bluetooth™Wireless Technology (JSR 82) - Specification Version 1.1*.
http://www.jcp.org/. Sep. 2005 5, 13, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 44

[MO03] MEHTA, Pratik ; O'CONNOR, Clint H. *Personal Area Connectivity with Bluetooth™Wireless Technology*.
http://www1.jp.dell.com/. 2003

[Net06] NETBEANS COMMUNITY. *NetBeans Mobility Pack 5.0 Quick Start Guide*.
http://www.netbeans.org/. Mar. 2006 48

[Sch96] SCHNEIER, Bruce: *Angewandte Kryptographie: Protokolle, Algorithmen und Sourcecode in C*. Addison-Wesley, 1996 12

[Sin01] SINTES, Tony. *Speaking on the Observer pattern*.
http://www.javaworld.com/. May 2001

[Sin06] SINTES, Tony. *Vector or ArrayList – which is better and why?*
http://www.javaworld.com/. Jan. 2006

[Son04] SONY ERICSSON. *Developing Applications with the Java APIs for Bluetooth™(JSR-82)*.
http://www.microjava.com/. Jan. 2004

[Sun00] SUN MICROSYSTEMS, INC. *J2ME Building Blocks for Devices*.
http://java.sun.com/. May 2000

[Sun05] SUN MICROSYSTEMS, INC. *CLDC HotSpot™ Implementation Virtual Machine*.
http://java.sun.com/. Feb. 2005

[Sun06] SUN MICROSYSTEMS, INC. *Java™ 2 Platform, Micro Edition*.
http://java.sun.com/. Mar. 2006 14

[T+00] TAIVALSAARI, Antero et al.
Connected, Limited Device Configuration. http://www.jcp.org/. May 2000 5, 15

[T+03] TAIVALSAARI, Antero et al. *Connected Limited Device Configuration 1.1*.
http://www.jcp.org/. Mar. 2003 5, 15, 16, 17

[Van00] VAN PEURSEM, Jim. *Mobile Information Device Profile (JSR-37)*.
http://www.jcp.org/. Sep. 2000 5, 18

[VW02] VAN PEURSEM, Jim ; WARDEN, James. *Mobile Information Device Profile for Java™ - Version 2.0*.
http://www.jcp.org/. Nov. 2002 5, 18, 19, 20, 45, 46

[Wik06a] WIKIPEDIA COMMUNITY. *Basisband*.
http://de.wikipedia.org/. Mar. 2006 8

[Wik06b] WIKIPEDIA COMMUNITY. *Bluetooth*.
http://de.wikipedia.org/. Mar. 2006 6, 7

[Wik06c] WIKIPEDIA COMMUNITY. *ISM-Band*.
http://de.wikipedia.org/. Feb. 2006 6

[Wik06d] WIKIPEDIA COMMUNITY. *Multiplexverfahren*.
http://de.wikipedia.org/. Feb. 2006 8

[Wik06e] WIKIPEDIA COMMUNITY. *Multiplexverfahren*.
http://de.wikipedia.org/. Mar. 2006

[Xil05] XILINX. *Bluetooth HCI Bridging*.
http://www.xilinx.com/. 2005

[Zet04] ZETTER, Kim. *Security Cavities Ail Bluetooth*.
http://www.wired.com/. Aug. 2004 56

Index