Entwicklung eines mobilen Clients für den Handel mit digitalen Objekten

Bachelorarbeit

von

Viet Anh Nguyen

aus

Düsseldorf

vorgelegt am

Lehrstuhl für Rechnernetze und Kommunikationssysteme
Prof. Dr. Martin Mauve
Heinrich-Heine-Universität Düsseldorf

November 2005

Betreuer:
Dipl. -Inform. Michael Stini

Danksagung

Zuerst möchte ich mich ganz besonders bei Michael Stini bedanken für die äußerst kompetente und hilfsbereite Betreuung meiner Bachelorarbeit. Die konstruktiven Kritiken von Michael, waren für mich stets sehr lehrreich und haben mich an viele Erfahrungen bereichert. Dies weiß ich sehr zu schätzen und bedanke herzlich dafür.

Ein großer Dank von mir gilt Professor Martin Mauve, der für mich um die administrative Arbeit gesorgt hat. Bei Professor Mauve habe ich viel über Rechnernetze und Mobilkommunikation gelernt. Für dieses Wissen bedanke ich mich aufrichtig.

Bei allen Kommilitonen möchte ich mich für eine schöne Studienzeit bedanken. Und ganz besonders: Oliver, Sadet, Matthäus, Duy, Alex, Nizar, Adel, und Nabil, sie haben alle großes Interesse an meiner Bachelorarbeit gezeigt. Dies wird bei mir immer in guter Erinnerung bleiben.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Kapitel 1

Einleitung

1.1 Motivation

Der Mobiltelekommunikationsmarkt ist in den letzten Jahren stetig gewachsen. Mehrere Netzbetreiber bieten ein zunehmend flächendeckendes GSM-Netz. Nach einem Bericht von BITKOM [1] basierend auf EITO (European Information Technology Observatory) Daten, besaßen 2004 in Deutschland im Durchschnitt 87 von 100 Einwohnern ein Handy, die Tendenz ist weiter steigend. Ein modernes Mobiltelefon bietet seinem Benutzer viel mehr als nur die gewohnte Sprachkommunikation. Die Anzahl der integrierten Multimediafunktionen wie Mp3-Player, hochauflösende Digitalkamera etc. nimmt kontinuierlich zu. Gleichzeitig werden die neuen Handys mit hochauflösenderen Displays, besserem Sound und mehr Rechenleistung ausgestattet. Die Leistungsfähigkeit eines solchen modernen Geräts entspricht der eines Desktop PCs anfang der 90er Jahre. Die Fähigkeit zur Kurzstrecken-Datenkommunikation wurde in den letzten Jahren ständig verbessert und erweitert, zum Beispiel mittels Infrarot und Bluetooth .

Unter den jungendlichen Handybenutzern hat es sich zum Trend entwickelt, die neusten Java-Games, Klingeltöne und Wallpaper-Logos auf das Handy zu laden, um diese sich gegenseitig zu präsentieren und auszutauschen. Das Geschäft mit diesem, sogenannten *mobilen Content* ist in den letzten Jahren zu einem eigenständigen umsatzreichen und gewinnbringenden Geschäftszweig in dem mobilen Unterhaltungsmarkt gewachsen.

Ein System zur Verwaltung von virtuellen Objekten auf Mobiltelefonen ist eine denkbare Anwendung. Hierzu würde auch ein Austauschsystem für virtuelle Sammelkarten gehören. Viele haben in der Jugend die Panini-Sammelkarten kennengelernt. Diese kann

man am Kiosk päckchenweise kaufen und mit Freunden gegen andere Karten, die man lieber mag, austauschen. Es liegt nahe, in dem digitalen Zeitalter diese Idee auf digitale Sammelkarten zu übertragen. Man kauft sich die Karte an einem virtuellen Kioskschalter und bietet diese per Bluetooth zum Tausch gezielt gegen eine andere Karte, die man noch nicht in der Sammlung hat, an. Die Sammelobjekte können mit der zunehmenden Vielfalt und Leistungsfähigkeit von mobilen Endgeräten im Hinblick auf die Multimedia Darstellung auf mobilen Endgeräten wesentlich aufwendiger und interessanter gestaltet werden als eine herkömmliche Sammelkarte aus Papier.

1.2 Problemstellung und Lösungen

Bisher kann man den mobile Content meist nur über kostenpflichtige Datenverbindungen wie zum Beispiel GPRS downloaden und anschließend über SMS oder über herstellerspezifische Kommunikationskanäle austauschen. Die in Frage kommenden Tauschpartner beschränken sich hierbei auf den eigenen Bekanntenkreis, da für einen Tausch eine vorherige mündliche Absprache benötigt wird. Es existiert bisher noch kein System mit dem man Tauschpartner finden und den mobilen Content Gerätehersteller unabhängig sicher verwalten, präsentieren und automatisch austauschen kann.

In Rahmen der Bachelorarbeit soll ein Programm für mobile Endgeräte entwickelt werden, mit dem man digitale Objekte verwalten und austauschen kann. Die Entwicklung umfasste auch ein Software Tool, mit dem man exemplarisch schnell und zuverlässig einfache virtuelle Objekte erstellen kann.

Die Lösung für diese Anforderungen wurde in Form einer J2ME Applikation implementiert. Diese ermöglicht die Verwaltung und Austausch von digitalen Objekten. Für die Datenverwaltung wurde das so genannte »Record Management System« von J2ME benutzt. Das Auffinden von potentiellen Tauschpartnern und die Datenkommunikation zum Tausch der Objekte basiert auf einem Client-/Server-Dienst über eine Bluetooth Funkverbindung.

Grundlage der Applikation ist eine funktionelle Benutzeroberfläche, die zur Anzeige und Verwaltung der Objekte sowie für Abfrage und Rückmeldung an den Benutzer verwendet wird.

1.3 Related Work

An der Technischen Universität Darmstadt existiert das Projekt namens *iClouds* [3], das einen ähnlichen Ansatz wie die Thematik der Bachelorarbeit verfolgt. Mit Hilfe von Kurzstrecken-Funkkommunikation von kleinen mobilen Endgeräten, wie PDAs und Pocket PC, tauscht man Werbeinformationen aus. Zur Filterung von Informationen werden sogenannte *iWish* und *iHave* Listen ausgetauscht. Hier liegt die gemeinsame Grundidee mit den in der Bachelorarbeit verwendeten Listen (Abschnitt 4.2.3). Anhand der Listen vergleicht man, ob die in der Umgebung verfügbaren Informationen erwünscht sind und nimmt diese je nach Wunsch in dem Gerät auf. Das Ziel, das iClouds verfolgt, ist gänzlich ein anderes als das der Bachelorarbeit. Man versucht mit der Informationsaustausch eine elektronische *Mund-zu-Mund-Proganda mit Bonusssystem* [4] um Werbung für kommerzielle Artikel zu verbreiten.

Ein weiteres interessantes Forschungsprojekt, das sich mit mobilen Datenobjektaustausch befasst, wird an der Swedish Institute of Computer Science unter den Name *MobiTip* betrieben [14]. Dort befasst man sich mit der Möglichkeit ein virtuelles Pinboard zu erschaffen. An bestimmten Bluetooth-Hotspot bekommt man die Aushanginformationen, die von anderen Benutzer eingetragen wurden, übertragen. MobiTrip arbeitet wie in der Bachelorarbeit mit Java-Bluetooth-API fähigen Endgeräten, insbesondere mit dem Sony Ericsson P900.

1.4 Struktur der Arbeit

Kapitel 2 bietet eine Einführung in die Java Technologie für mobile Endgeräte mit Erläuterung von Begriffen, die in der Dokumentation teilweise wieder aufgegriffen werden. Weiter in Kapitel 3 wird die Installation des Programms zur Bachelorarbeit mit dem Namen *ATVO (Applikation zum Tausch und Verwaltung von Objekten)* und, das für dessen Erstellung wichtigen, Wireless Toolkit vorgestellt. Die Datenverwaltung von ATVO wird in Kapitel 4 besprochen. Anschließend wird in Kapitel 5 auf die Bedeutung und Funktion der verschiedenen Menüs in der GUI von ATVO eingegangen. In Kapitel 6 wird die Datenkommunikation zum Austausch von Objekten behandelt. Und abschließend in Kapitel 7 folgt ein Resümee über die Bachelorarbeit.

Kapitel 2

Einführung in die J2ME-Technologie

2.1 Grund für die Verwendung von J2ME

Mobile Endgeräte sind Produkte mit teilweise sehr unterschiedlicher Hardwareausstattung. Dies betrifft die Displayauflösung, die Eingabeschnittstelle, die Rechenleistung, die Speicherkapazität etc. Es ist notwendig eine geräteunabhängige Lösung zu konzipieren, damit sie einfach und schnell funktioniert und sich rasch verbreiten lässt. Die Java 2 Micro Edition ist die zurzeit am weitesten verbreitete Programmiersprache für Mobiltelefone. Aufgrund dieser Tatsachen wurde Java als Entwicklungsplattform verwendet.

2.2 J2ME Architektur

Die Java 2 Microedition, kurz J2ME [11] genannt, ist eine Java Technologie, die auf die Bedürfnisse von Kleinstcomputern, hauptsächlich kleine mobile Endgeräte wie Mobiltelefone und PDAs optimiert ist. Diese Systeme sind meist mit relativ wenig Arbeitspeicher und Rechenleistung sowie beschränkten Ein- & Ausgabefunktionalitäten ausgestattet.

Speziell für den Umgang mit den eingeschränkten Ressourcen wurde die Kilobyte Virtual Machine (KVM) entwickelt. Die KVM bildet den Kern der J2ME-Technologie bei

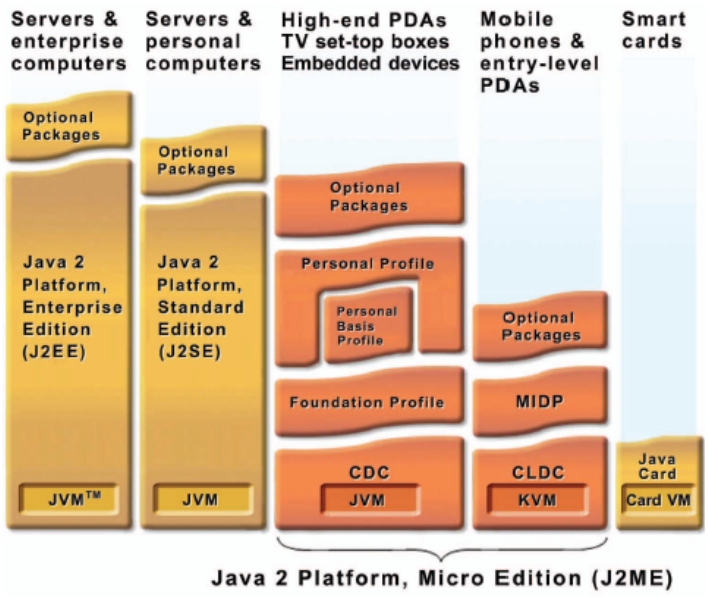

Abbildung 2.1: J2ME Architektur [10]

der Anwendung auf kleinen mobilen Endgeräten (Abbildung 2.1). Diese spezielle Variante der Vitual Machine erfordert, wie der Namen vermuten lässt, nur wenige Kilobyte Speicher.

Neben der KVM-Technologie waren noch weitere Anpassungen der J2SE erforderlich, um die Java Technologie effizient nutzen zu können. Neben der Anzahl der Pakete, die J2SE-Klassenbibliothek bekannt sind, sind weitere spezielle Pakete entwickelt worden (Abbildung 2.2), die auf die Anforderung der Zielgeräte abgestimmt sind. Da die kleinen, meist mobilen Geräte hinsichtlich der Benutzerein-/ausgaben, Dateisystem etc. erhebliche Unterschiede zum Desktop PC vorweisen, sind diese Klassen notwendig. Weil mobile Endgeräte meist kein Dateisystem wie herkömmliche Desktop PCs besitzen, bietet J2ME im Paket `javax.microedition.rms` den Record Store für die persistente Datenverwaltung an. Das Package stellt Klassen zur Verfügung, mit dem die persistente Speicherung in den Flash-Speicher ermöglicht wird. Ein wesentlicher Teil der Bachelorarbeit befasst sich mit der persistenten Datenhaltung, die in Kapitel 4 genau behandelt wird.

Der Hauptteil der J2ME-Architektur für Mobiltelefone besteht aus dem *Connected Limited Device Configuration (CLDC)* und das darauf bauende *Mobile Information Device Profile (MIDP)*. Im Folgenden wird der weitere Aufbau der J2ME-Architektur beschrieben und insbesondere die CLDC und das MIDP vorgestellt.

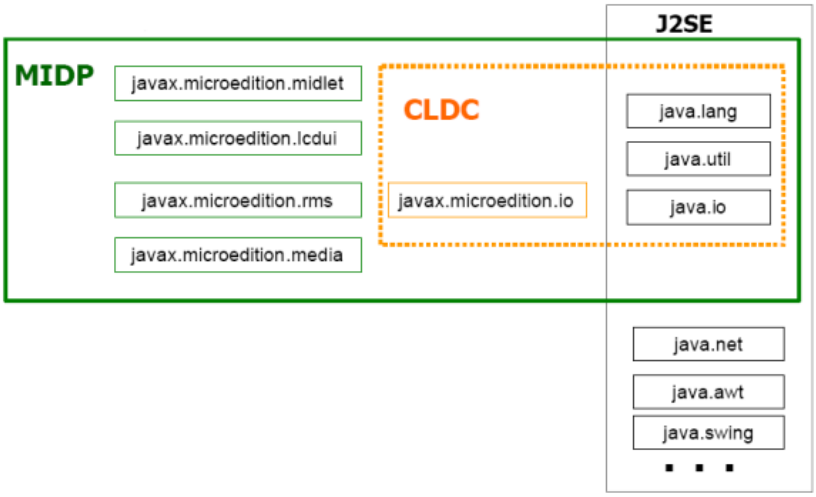

Abbildung 2.2: J2ME Packages [15]

2.3 Das Prinzip von Konfigurationen und Profilen

Um die Entwicklung der Java-Software für eine Gerätesparte zu vereinfachen, werden diese zu sogenannten Konfigurationen zusammengefasst. Die Geräte, die zu einer bestimmten Konfiguration gehören, ähneln sich zum Beispiel hinsichtlich ihrer Rechenleistung, Speicherkapazität und Netzwerkanbindung.

Des weiteren unterscheiden sich die spezifischen Anforderungen von Gerätentypen in der Praxis so sehr, dass man unterschiedliche Profile für Klassen von Gerätentypen definiert. MIDP ist ein weit verbreitetes Profil für die CLDC auf Mobiltelefone. In der J2ME-Architektur basieren die Profile auf den Konfigurationen (Abbildung 2.1)

2.4 Konfigurationen

Zurzeit existieren zwei Konfigurationen für J2ME. Das sind die *Connected Limited Device Configuration (CLDC)* und die *Connected Device Configuration (CDC)*. Für die CLDC sind Geräte wie Mobiltelefone, Pager und PDAs definiert. Diese Konfiguration setzt folgende Hardware voraus:

- 16 oder 32-Bit CPU
- eine Speicherkapazität zwischen 128 Kbyte und 512 Kbyte
- Batterien für die Stromversorgung
- Netzwerkverbindung mit einer Übertragungsrate von mindestens 9600 bps

Die CDC-Konfiguration dagegen ist für Geräte mit mehr Performance, wie Settop-Boxen, high-end PDAs oder Fahrzeug Telematik Systeme, definiert. Diese haben mindestens zwei Megabyte an Arbeitspeicher, eine 32-Bit CPU und sind leistungsfähig genug, um mit einer vollständigen Java-VM zu arbeiten.

2.5 Profile

Die Profile definieren die Programmierschnittstelle, die sehr stark auf einen bestimmten Gerätentyp zugeschnitten ist, beispielsweise das zurzeit weit verbreitete *Mobile Information Device Profile (MIDP)*. MIDP ist ein CLDC-Profil, das den Zugriff auf die grafische Benutzeroberfläche und den persistente Speicher ermöglicht. Das MIDP ist speziell auf die Einschränkungen und besonderen Möglichkeiten von Mobiltelefonen zugeschnitten. Die Schnittstelle zur Programmierung von Grafiken gliedert sich in zwei Bereiche, das High-Level und das Low-Level.

2.6 MIDlet und MIDlet-Suite

Eine Java-Applikation, die für MIDP geschrieben ist, nennt man ein MIDlet. Einen Verbund von zusammengefassten MIDlets nennt man MIDlet-Suite [5]. Indem man eine Applikation in mehrere kleine MIDlets unterteilt, benötigt man unter Umstände bei der Ausführung wenig Arbeitspeicher pro MIDlet und kann somit effizienter arbeiten. Außerdem bietet die modulare Zusammensetzung einer Anwendung mehr Übersicht und bessere Wiederverwertung des Programms.

Kapitel 3

Installation eines MIDlets

Für gewöhnlich installiert man eine neue Software auf einem Desktop-PC, indem man ein Speichermedium mit der gewünschten Software in das Laufwerk einlegt oder sich die Softwares über lokale Netzwerk oder Internet auf dem Rechner lädt. Die Installation wird durch eine Setup-Routine durchgeführt, die meist in einer Datei namens setup.exe enthalten ist.

Die Installation eines *MIDlets* auf kleinen mobilen Endgeräten verläuft anders. Diese wird über den Aufruf der *Java Application Descriptor (JAD)-Datei* initiiert. Im nächsten Abschnitt wird dies in Detail besprochen.

3.1 J2ME Wireless Toolkit

Für die Bachelorarbeit wurde das *J2ME Wireless Toolkit (WTK)* [9] von Sun Microsystems in der aktuellen Version 2.2 benutzt. Dies ist ein Werkzeug, mit dem man MIDP-Projekte erzeugen und verwalten kann. Bei der Erzeugung eines neuen Projekts wird automatisch die für MIDP-Programme erforderliche JAD-Datei angelegt. Die Attribute der JAD-Datei können mittels des WTKs komfortabel über Dialogboxen gesetzt werden. Das WTK kompiliert auf Knopfdruck ein vorhandenes Projekt und kann es im mitgelieferten Emulator ausführen. Bei dem WTK handelt es sich nicht um eine komplette Entwicklungsumgebung, da es unter anderen keinen Editor enthält. Um Quelltexte von Projekten zu editieren, die im Wireless Toolkit verwaltet werden, muss ein externer Edi-

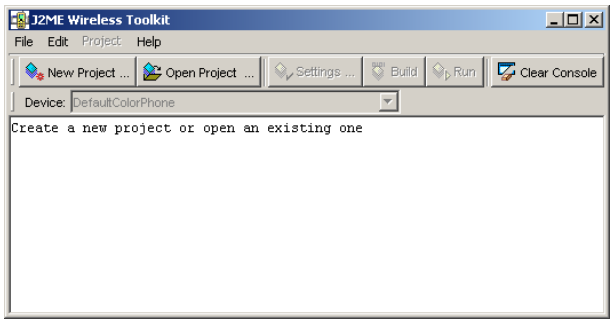

Abbildung 3.1: J2ME Wireless Tookit Hauptfenster

tor hinzugenommen werden. Das WTK enthält zwar die spezifischen Bibliotheken für MIDP, doch zur eigenen Ausführung benötigt es zusätzlich ein JavaTM 2 SDK, Standard Edition (J2SE SDK), version 1.4.2 [8] oder JavaTM 2, Standard Edition Runtime Environment (JRE), Version 1.4.2 [12].

3.2 Erstellung eines Projekts in WTK

Die Erstellung eines Projekts in WTK wird am Beispiel der in Rahmen der Bachelorarbeit realisierten *Applikation zum Tausch und Verwaltung von Objekten (ATVO)* beschrieben. In der Abbildung 3.1 ist das Hauptfenster von dem WTK zu sehen.

Um ATVO bei der erstmaligen Inbetriebnahme zu starten, braucht man ein neues Projekt mit dem MIDlet-Class-Name `GUI`. Da ATVO für das Auffinden von Tauschpartnern und die Datenkommunikation die Bluetooth-API JSR 82 (Abschnitt 6.1), benötigt, muss man mindestens diese als zusätzliche API auswählen (Abbildung 3.2). ATVO besteht aus zwei MIDlets. Um das zweite MIDlet in der selben MIDlet-Suite hinzuzufügen, fügt man in dem Register `MIDlets` (Abbildung 3.3) das zweite MIDlet mit dem MIDlet-Class-Name `ObjectsCreator` ein.

Das WTK kann Hardware mit unterschiedlicher Ausstattung emulieren. Zum Beispiel kann man unterschiedliche Skins verwenden. Abbildung 3.4 zeigt ATVO-Emulationen in unterschiedlichen Skins mit verschiedenen Farbtiefen und Displaygrößen. Außerdem können im WTK Einstellungen für eine bestimmte Speichergröße und Netzwerk-Verbindung der Emulationshardware vorgenommen werden.

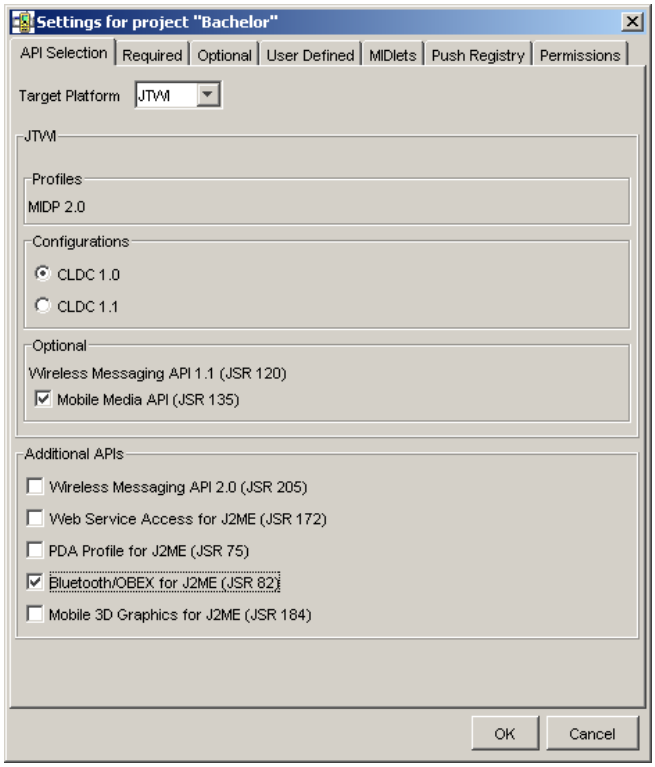

Abbildung 3.2: J2ME Wireless Tookit Bluetooth Settings

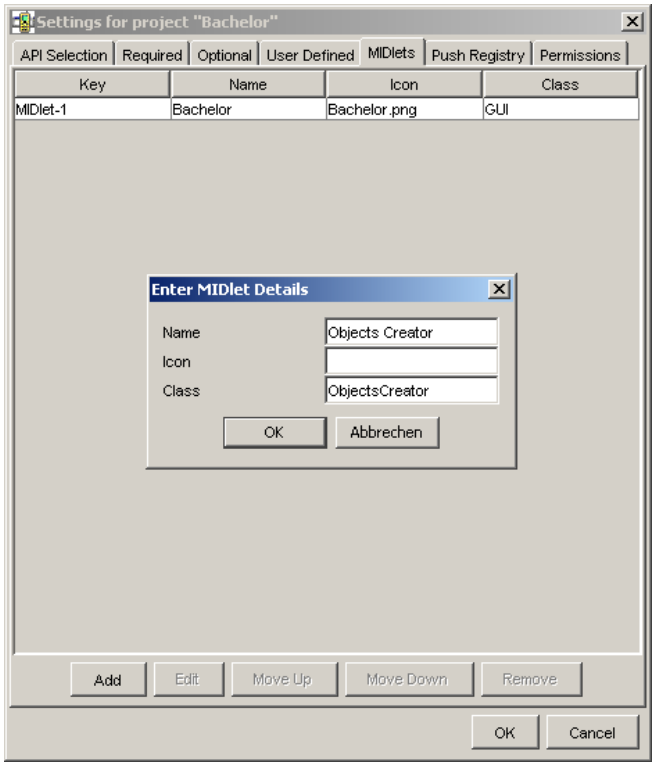

Abbildung 3.3: J2ME Wireless Tookit MIDlets Settings

Abbildung 3.4: WTK Skins: DefaultColorPhone (rechts) und DefaultGrayPhone (links)

3.3 Installation auf Sony Ericsson K750i

ATVO wurde auf einem realen Gerät, das Modell K750i von Sony Ericsson getestet. Es gehört zu den wenigen Modellen auf dem Markt, die das Bluetooth-API JSR 82 (Abschnitt 6.1) unterstützen.

Die für den Test benötigten JAD- und JAR-Dateien erstellt man mit Hilfe des WTK und installiert diese auf dem K750i. Beim Test ist festgestellt worden, dass alle Funktionalitäten von ATVO auf dem K750i voll ausführbar sind.

Kapitel 4

Datenverwaltung

Die persistente Datenspeicherung auf Mobiltelephonen erfolgt im Gegensatz zu Desktop-PC üblicherweise nicht auf einer Festplatte. Stattdessen erfolgt die nicht flüchtige Datenhaltung generell über Flash-Speicher. Die Struktur der Daten auf dem Flash-Speicher unterscheiden sich deutlich von der des herkömmlichen Dateisystems auf einer Festplatte. J2ME bietet das sogenannte *Record Management System (RMS)* als Mechanismus für die persistente Datenhaltung an.

4.1 Grundlagen der persistenten Datenspeicherung

Das Paket `javax.microedition.rms` von MIDP beinhaltet das RMS-API. MIDlets können beliebige Daten in dem RMS persistent speichern. Die Größe des RMS ist gerätespezifisch festgesetzt.

Die zu speichernden Daten werden in Form von Byte-Arrays als *Records* in einem *Record Store* gespeichert. Die Länge der Records kann unterschiedlich lang sein. Die Größe von Records und Record Stores wird durch das RMS begrenzt.

Einen Record Store kann man sich als eine sehr einfache Datenbank vorstellen, in der nur die Zuordnung *Record ID* zu Byte Array existiert (Tabelle 4.1). Der Name des Record Stores kann bis zu 32 Unicode-Zeichen lang sein, wobei Groß-/Kleinschreibung unterscheiden wird. Alle Zugriffe auf einen Record Store werden atomar ausgeführt. Das

bedeutet, dass falls zwei Threads zeitgleich auf einem Eintrag zugreifen, werden die Zugriffe hintereinander ausgeführt, um somit mögliche Datenkonflikte zu verhindern.

Die *Record-ID* ist der Primärschlüssel zu den jeweiligen Records eines Record Store.

Record-ID	Data
1	Byte Array
2	Byte Array
4	Byte Array
...	...

Tabelle 4.1: Aufbau eines Record-Stores [13]

Dieser ist immer ein Integerwert, der automatisch vom RMS erzeugt wird und eindeutig ist. Die Nummerierung der Records beginnt mit dem Wert 1. Jeder weitere nachfolgende Record erhält jeweils eine um eins erhöhte ID.

Ein MIDlet kann über mehrere Record Stores verfügen, welche zur Unterscheidung innerhalb einer MIDlet-Suite über eindeutige Namen verfügen müssen, weil alle MIDlets innerhalb einer MIDlet-Suite auf die Record Stores von anderen MIDlets zugreifen können (Abbildung 4.1). Mittels dieser Funktionalität ist es möglich ATVO in zwei MIDlets zu unterteilen und trotzdem mit einem gemeinsamen Record Store zu arbeiten. Ab der Version 2.0 von MIDP können auch MIDlets von unterschiedlichen MIDlet-Suites Record Stores gemeinsam benutzen. Dazu führt MIDP zusätzliche Methoden ein, die die Zugriffsrechte auf Record Stores spezifizieren.

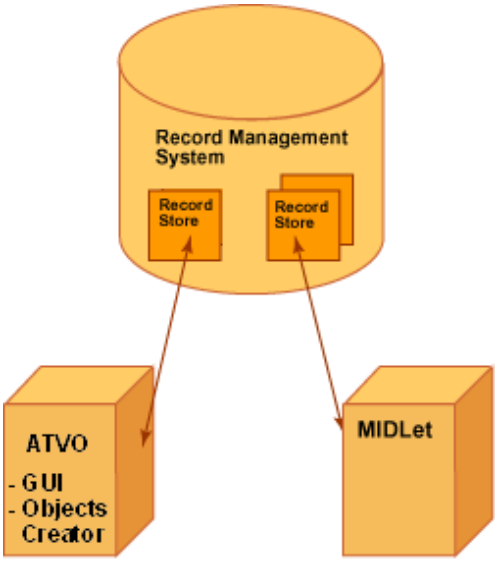

Abbildung 4.1: Nutzung des Record Stores von ATVO [6]

4.2 Objektverwaltungskonzept

ATVO arbeitet mit tauschbaren virtuellen *Objekten*. Ein ATVO-Objekt besitzt bestimmte Eigenschaften und gehört zu einem *Objektyp* (Abschnitt 4.3). Die Eigenschaften von Objekten eines Typs werden in Form von Strings oder Integers dargestellt. Ein Objekt bekommt bei der Erstellung einen eindeutigen *Unique-Name* und jedes wird in einem Record in dem Record-Store *myDB* gespeichert.

Die wesentlichen Elemente, die ATVO zur Verwaltung von diesen Objekten im RMS benutzt, sind:

- *Inhaltverzeichnis namens TOC (Table of Content)*, indem alle Objekte registriert sind. (Tabelle 4.2)

Unique-Name	Record-ID
x	n
...	...

Tabelle 4.2: Inhaltverzeichnis

- *Listenverzeichnis*, das zur Listeverwaltung dient. (Tabelle4.3)

Listenname	Record-ID
offerList	i
wishList	j
...	...

Tabelle 4.3: Listenverzeichnis

- *Liste*, die Objekte in einer Kategorie zusammenfasst. (Tabelle 4.4)

Listenname	Anzahl	Unique-Name	Unique-Name
wishList	2	K750iSchwarz	S-KlasseSilber

Tabelle 4.4: Eine Liste

Im Folgenden werden diese Elemente und das darunter liegende Verwaltungskonzept im Detail erklärt.

4.2.1 Inhaltverzeichnis

In der *TOC* sind alle in *myDB* enthaltenen Objekte anhand ihres Unique-Name und der zugehörige Record-ID adressiert. Die TOC ist ein sogenanntes *well known record*, da sie so erstellt wurde, dass man sie immer in dem ersten Record von myDB findet. Sie wird bei jeder Änderung innerhalb vom myDB aktualisiert. Die TOC ist notwendig, um die Objekte in myDB wieder zu finden, ohne alle Records einzeln zu durchsuchen, da sie jedem Unique-Name eine eindeutige Record-ID zuordnet. Aus Performance Gründen wurde auf die Möglichkeit einen Record Comparator zu implementieren verzichtet und eine eigene Verwaltungsstruktur geschaffen.

4.2.2 Listenverzeichnis

Wie das vorher erwähnte Inhaltverzeichnis, ist das Listenverzeichnis auch ein *well known Record*. Es ist immer in dem zweiten Record der *myDB* zu finden. Tabelle4.3 zeigt einer Ausschnitt aus einem möglichen Listenverzeichnis. Zur Adressierung enthält das Listenverzeichnis alle Listennamen und deren zugehörige Record-IDs innerhalb von myDB. Das Listenverzeichnis wird benötigt, um Listen wieder finden zu können.

4.2.3 Listen

ATVO arbeitet mit Listen die Unique-Names beinhalten. Minimal sind im System die folgenden vier Listen implementiert:

- *offerList*, beinhaltet die Unique-Names der Objekte, die der Benutzer zum Tausch anbieten will.
- *wishList*, beinhaltet die Unique-Names der Objekte, die der Benutzer beim einem Tausch bekommen will
- *allList*, beinhaltet die Unique-Names aller dem System bekannten. Objekte.
- *myList*, beinhaltet die Unique-Names der Objekte im eigenen Besitz.

Die offerList und die wishList werden persistent gespeichert. Die Liste allList wird aus vordefinierten Klassen herausgelesen. Die Unique-Names von allen in myDB befindlichen Objekten sind in myList enthalten. Tabelle 4.4 zeigt beispielsweise die Informationen, die in einer Liste gespeichert werden. Weitere Listen werden automatisch erstellt, sobald ein Objekttyp, der in myDB noch nicht enthalten ist, in dem Record-Store gespeichert wird. Diese enthalten alle Namen der Objekte, die von dem selben Typ wie der Name der Liste sind. Die Listen sind für die Gruppierung und den Tausch der Objekte notwendig.

4.3 Objekttypen

Alle virtuellen ATVO-Objekte werden nach Typen sortiert und gehandelt. ATVO kennt derzeitig drei exemplarische Objekttypen: Car, Mobilephone und Game. Diese Objektypen implementieren das Objektinterface (Tabelle 4.5). In der Implementierung der Applikation ist vorgesehen, dass sie um beliebige weitere Objekttypen erweiterbar sind. Der neue Objekttyp muss ebenfalls die Methoden aus dem Objektinterface siehe Abschnitt 4.4 implementieren.

Methode	**Beschreibung**
`byte[] toBA()`	Wandelt ein Objekt in ein Byte Array um.
`void fromBA(byte[] rs)`	Erstellt ein Objekt, aus einem Byte Array,
`String getUname()`	Gibt den Unique-Name des Objekts in Form eines Strings wieder.
`String toHumanReadable()`	Gibt den Inhalt des Objekts in Form eines Strings wieder.

Tabelle 4.5: Objektinterface definiert die notwendigen Methoden für jeden Objektyp

4.4 Objektinterface

Um ein Objekt in den Record-Store zu schreiben oder um es für den Datentransport über eine Bluetoothverbindung (Kapitel 6) vorzubereiten, benutzt ATVO das Datenformat Byte Array. Für jeden in ATVO benutzten Objektyp ist die Möglichkeit zur Serialisierung von Objekten in ein Byte Array und die Wiederherstellung aus einem Byte Array

in ein Objekt zwingend notwendig.

Durch die Benutzung eines selbst definierten Objektinterfaces wurde die Serialisierung von Objekten vereinheitlicht. Indem die Objekttypen das Objektinterface implementieren, beinhalten sie das gleiche Interface für die Datenspeicherung und den Datentransfer. Tabelle 4.5 zeigt die Methoden, die das Objektinterface definiert.

4.5 Objekterzeugung

Die Funktionen der Objekterzeugung sind als ein MIDlet in der Midlet Suite implementiert worden. Es trägt den Namen *Objects Creator*. Dieses funktioniert nach einem Baukasten-Prinzip, wobei der Benutzer die Eigenschaften des zu erzeugenden Objekts auswählt und der Objects Creator erstellt anhand dieser Daten das erwünschte Objekt. Abhängig von den ausgewählten Eigenschaften kann das erstellte Objekt zusätzliche Werte, die von dem Objects Creator automatisch hinzugefügt werden, enthalten.

Zum Beispiel, wenn der Benutzer ein Mobilephone der Marke Sony Ericsson in der Farbe Schwarz erstellt, bekommt das erstellte Objekt von den Objects Creator den errechneten Preis von 400 Euro automatisch eingetragen.

Jedes Objekt ist einzigartig und kann maximal nur einmal in jedem Record Store erzeugt werden. Das oben genannte Baukasten-Prinzip wurde mit Hilfe eines sogenannten *Initkey* implementiert. Jede einzelne Ziffer in dem Initkey repräsentiert eine bestimmte Eigenschaft eines Objekts. Durch die Auswahl des Benutzers werden diese Werte festgelegt. Zum Erzeugen eines Objekts wird im Objects Creator ein Mechanismus aufgerufen, der die Information aus dem Initkey liest und daraus das gewünschte Objekt erstellt.

Kapitel 5

Benutzeroberfläche

In Rahmen der Bachelorarbeit wurde ein MIDlet-Suite erstellt, das aus den zwei MIDlets *ATVO* und *Objects Creator* besteht. Das ATVO-MIDlet enthält die in der Bachelorarbeit realisierten Hauptfunktionalitäten. Mit diesem MIDlet kann man Objekte verwalten, Tauschpartner suchen, Objekte zum Tausch anbieten, den Tausch aushandeln und die Objekte transferieren. Das Objects Creator-MIDlet ist ein Tool, das zur exemplarischen Erstellung von Objekten für das ATVO-MIDlet benötigt wird. Im Folgenden wird die Funktion der Menüs in den beiden MIDlets erläutert. Der Startbildschirm des MIDlet-Suites bietet dem Benutzer die Wahl zwischen den beiden MIDlets.

5.1 GUI vom ATVO-MIDlet

Das ATVO-Midlet beginnt mit einem Login-Bildschirm, der den Benutzer nach dem Zugangspasswort fragt. Dieses ist in dem Grundzustand auf **0000** gesetzt. Bei einer erfolgreichen Eingabe des Passworts gelangt der Benutzer in das Hauptmenü von ATVO. Hier hat er die folgenden Optionen zur Auswahl: *Objektverwaltung*, *Objekttausch* und Passworteinstellung. Die Passworteinstellungsoption dient zur Veränderung des Zugangpassworts durch den Benutzer. Auf die beiden, die Objekte betreffenden Menüpunkte wird in den nachfolgenden Absätzen eingegangen. Die folgende Auflistung stellt die Menüstruktur des ATVO-MIDlet dar:

- Hauptmenü
  - Objektverwaltung
    - myObjects
      - Cars
      - Games
      - Mobilephones
      - All
    - wishList
    - offerList
    - wishObjects
  - Objekttausch
    - Server Modus
    - Client Modus
  - Passworteinstellung

5.1.1 Objektverwaltung

Das Menü der Objektverwaltung (Abbildung 5.1) beinhaltet die folgenden Funktionalitäten:

- die in eigenem Besitz befindlichen Objekte darzustellen,
- die in eigenem Besitz befindlichen Objekte zum Tausch anzubieten,
- gewünschten Objekte auszuwählen.

Alle Objekte im Besitz des Benutzers findet er im *myObjects*-Menü (Abbildung 5.1). Dort ist eine Menüauswahl von allen bekannten Objekttypen vorhanden. Im *Cars*-, *Games*-, *Mobilephones*-Menü befinden sich die Objekte des jeweiligen Objekttyps. ATVO sortiert automatisch Objekte nach ihrem Objekttyp in das passen die Menü ein. In Menü *All* sind alle Objekte, welche sich im Besitz von dem angemeldeten Benutzer befinden, zu sehen. In den Menüs *Cars*, *Games*, *Mobilephones* und *All* hat der Benutzer die Möglichkeit die Eigenschaften von den dort befindlichen Objekten anzeigen zu lassen. Er hat dort außerdem die Möglichkeit diese Objekte zum Tausch anzubieten, indem er diese

zu seiner *offerList* hinzufügt. Zusätzlich können Objekte aus dem Besitz des Benutzers entfernt werden.

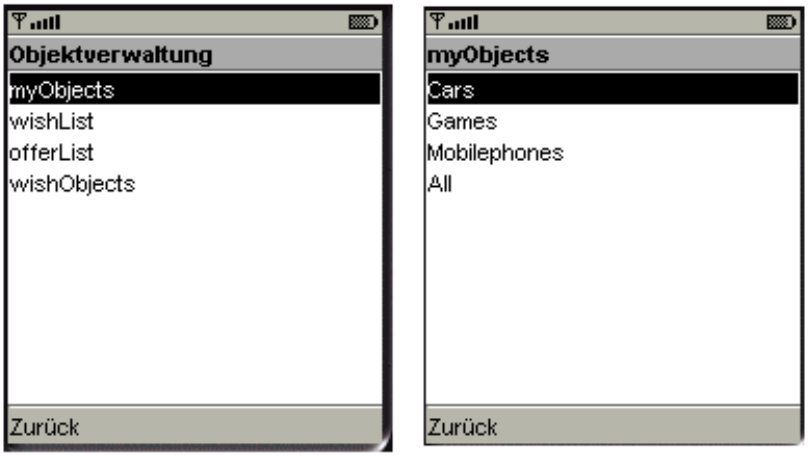

Abbildung 5.1: Objektverwaltung-Menü (links) und myObjects-Menü(rechts)

Unter *wishList*-, *offerList*-Menü im Objektverwaltung-Menü befinden sich die Namen der Objekte, die der Benutzer sich wünscht, beziehungsweise die er bei einem Tausch abgeben möchte. In diesen Menüs kann er die entsprechenden Objektnamen entfernen, falls er sie nicht mehr wünscht oder nicht mehr zum Tausch anbieten will. Die Funktionalität die Eigenschaften des Objekts anzuschauen, ist im *offerList*-Menü vorhanden. Die möglichen Wunschobjekte stehen in *wishObjects*-Menü zur Auswahl.

Jedes Objekt existiert nur einmal und dessen Unique-Name kann maximal in zwei Liste-Menüs vorhanden sein. Entweder in myObjects und offerList oder in wishObjects und wishList.

5.1.2 Objekttausch-Menü

Die Tausch-Funktionalität befindet sich im *Objekttausch*-Menü (Abbildung 5.2). Dort hat der Benutzer die Auswahl, ob er die Bereitschaft zu Tauschen aktivieren möchte, indem er den Server Modus wählt, oder ob er den Tausch selbst initiiert, indem er im Client Modus nach einem Tauschpartner sucht.

Wird der Server Modus gewählt, wird der Benutzer im nächsten Bildschirm einmalig gefragt, ob er eine Bluetooth-Verbindung zulässt. Nur wenn er dies erlaubt, kann er von

einem Tauschpartner gefunden werden, denn die Bluetooth-Kommunikation ist für den Datenaustausch zwingend notwendig. Bei der Zulassung einer Bluetooth-Verbindung wird in dem darauf folgenden Bildschirm angezeigt, dass das Gerät sich zurzeit in Server Modus befindet und auf einen Tauschpartner wartet. Das Warten auf Tauschpartner wird beendet, wenn ein Tauschpartner gefunden wurde, oder wenn der Benutzer es manuell abbricht. Falls der Dienst von einem Tauschpartner gefunden wird, der Interesse an einem Tausch hat, wird der Benutzer darüber informiert. Er bekommt einen Tauschvorschlag von dem Tauschpartner und kann diesen annehmen oder ablehnen. Nach einer Bestätigung werden die Objekte zum Tausch versandt.

Im Clientmodus wird auf dem Bildschirm angezeigt, dass der Suchvorgang nach einem Server gerade stattfindet. Dieser Vorgang wird beendet, wenn ein Tauschpartner gefunden wurde, oder wenn der Benutzer ihn manuell abbricht. Falls das Gerät einen Tauschpartner findet, wird der Benutzer gefragt, ob er eine Bluetooth-Verbindung zu diesem aufbauen möchte. Um einen Datenkommunikationskanal aufzubauen, muss er an dieser Stelle zustimmen.

Als nächstes bekommt der Benutzer eine Liste von möglichen Tauschvorschlägen (Abbildung 5.2) angezeigt. Der Benutzer kann in dieser Liste die Anzahl der abzugebenden und gewünschten Objekten reduzieren und als Tauschvorschlag an den Partner senden. Um einen gültigen Tausch vorzuschlagen, muss die Anzahl der Angebote und der Wünsche jeweils mindestens eins sein. Je nachdem, ob der Partner den Vorschlag annimmt oder ablehnt, wird der Benutzer von ATVO informiert.

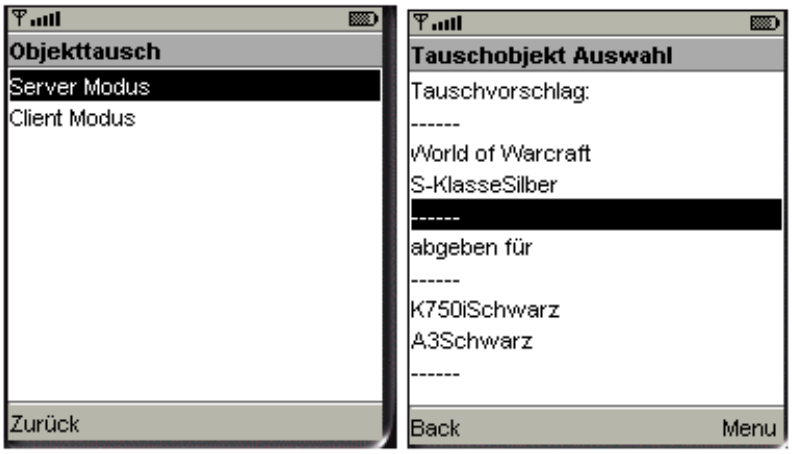

Abbildung 5.2: Tauschbetriebmodus-Menü (links) und Tauschvorschlag-Menü(rechts)

Nach einem erfolgreichen Objekttausch bekommt der Benutzer, sowohl in Server- als auch in Client-Modus, die Information über die getauschten Objekte auf dem Display angezeigt.

5.2 GUI vom Objects Creator-MIDlet

Im Hauptmenü des Objects Creator (Abbildung 5.3) stehen alle bekannten Objekttypen zur Verfügung. Mit der *Delete all objects*-Funktion löscht man alle im Besitz befindlichen Objekte aus dem persistenten Speicher. Um ein Objekt zu erstellen, wählt man als erstes den gewünschten Objekttyp aus. Danach folgt man den Bildschirmanweisungen bis der Objects Creator alle Eigenschaften, die er braucht um das Objekt zu erstellen, gesammelt hat. Nach erfolgreicher Erstellung von einem Objekt wird der Benutzer abschließend über alle Eigenschaften, die das Objekt besitzt, informiert.

Abbildung 5.3: Objects Creator-Menü

Der Objects Creator bietet mit *Alle Autos*, *Alle Games*, *Alle Mobilephones* die Möglichkeit zur schnellen Erstellung von allen Objekten eines bestimmten Objekttyps an.

5.3 Realisierung der GUI mittels High-Level-API

Die Ein-/Ausgabe-Bildschirme von ATVO wurden mit der High-Level API erstellt. Sie bestehen aus Listen und Formularen mit verschiedenen Dialogelementen. Man hat in der High-Level API keinen detaillierten Einfluss auf die Anordnung der Grafikelemente auf dem Bildschirm. Sie werden automatisch entsprechend dem spezifischen graphischen Design des jeweiligen Handsets angepasst. Aus diesem Grund unterscheiden sich Programme, die im Emulator und auf realen Endgeräten ausgeführt werden, leicht in Darstellung und Benutzerführung.

Die hauptsächlichen Klassen der High-Level-API, die zur Darstellung der GUI genutzt wurden, sind *Alert*, *List* und *Form* sowie die Formularelemente. Abbildung 5.4 stellt die Vererbungshierachie der Klassen innerhalb der High-Level-API dar.

Alert wurde benutzt um Textmeldungen an dem Benutzer auszugeben. Es ist zu be-

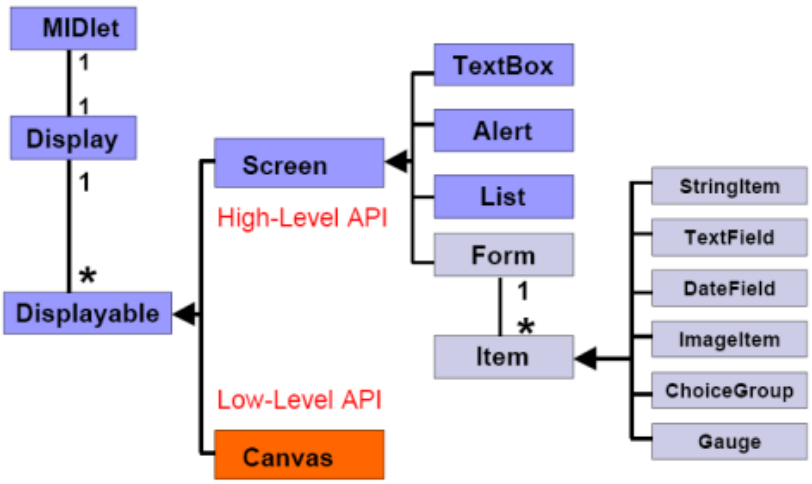

Abbildung 5.4: Vererbungshierachie der Klassen zur Darstellung der GUI [15]

merken, dass die Alert-Bildschirme auf dem Sony Ericsson K750i anders als im Emulator mit zusätzlichen Ton- und Vibrationseffekten ausgegeben werden. Dies ist wie vorher erwähnt darauf zurück zu führen, dass die Darstellung der High-Level-API gerätespezifisch erfolgt. Die Login-, Passwortoptions- und Tauschinformations-Bildschirme von ATVO wurden mit der Klasse Form dargestellt. Die restlichen Menü-Bildschirme wurden mit Hilfe der Klasse *List* realisiert.

Kapitel 6

Datenkommunikation

Die Bluetooth-Technologie wurde für die kabellose Kommunikation von mobilen Endgeräten über eine kurze Reichweite (zwischen 10 bis 100 Metern) entwickelt. Die meisten der zur Zeit auf dem Markt befindlichen Mobiltelefone, unterstützen Java und können über Bluetooth kommunizieren, meist für die Benutzung eines Headsets. Doch nur wenige dieser Modelle verfügen über die *Java Bluetooth-API JSR 82*, welche für die Bluetoothkommunikation von Java-Anwendungen notwendig ist. Um gerätenunabhängige Programme wie ATVO zu entwicken, die über Bluetooth kommunizieren, gibt es als einzige Möglichkeit nur die Verwendung von JSR 82. Im Folgenden wird eine kurze Einführung in JSR 82 und das Grundprinzip des hier genutzten Bluetooth Service Record gegeben.

6.1 Java Bluetooth-API JSR 82

Für die Datenkommunikation zum Austausch von Objekten in ATVO wurden die *Java APIs for Bluetooth Wireless Technology (JABWT)* benutzt. JABWT benutzt die Bibliotheken des *Java Specification Requests 82*, abgekürzt *JSR 82*. Diese beinhaltet die zwei Pakete `javax.bluetooth` und `javax.obex`, die in der Tabelle 6.1 aufgelistet sind.

Diese Pakete ermöglichen eine Programmierung mit der JABWT-API, wobei sie auch separat verwendet werden können. Speziell für den Zweck zum Austausch von Objek-

Package	Provides
`javax.bluetooth`	The core Bluetooth API.
`javax.obex`	The Object Exchange (OBEX) API.

Tabelle 6.1: JABWT Pakete [7]

ten wurde OBEX entwickelt. Doch auf dem Markt gib es so gut wie kein Handsets, das diese API unterstützt. Aus diesem Grund wurde in ATVO ein eigenes Verfahren entwickelt, um die Datenkommunikation zum Versand von Objekten lediglich mit dem Paket `javax.bluetooth` von JABWT zu realisieren. JABWT-Pakete sind standardgemäß nicht in MIDP oder CLDC enthalten. Abbildung 6.1 zeigt die Anordnung von JABWT in der J2ME-Architektur.

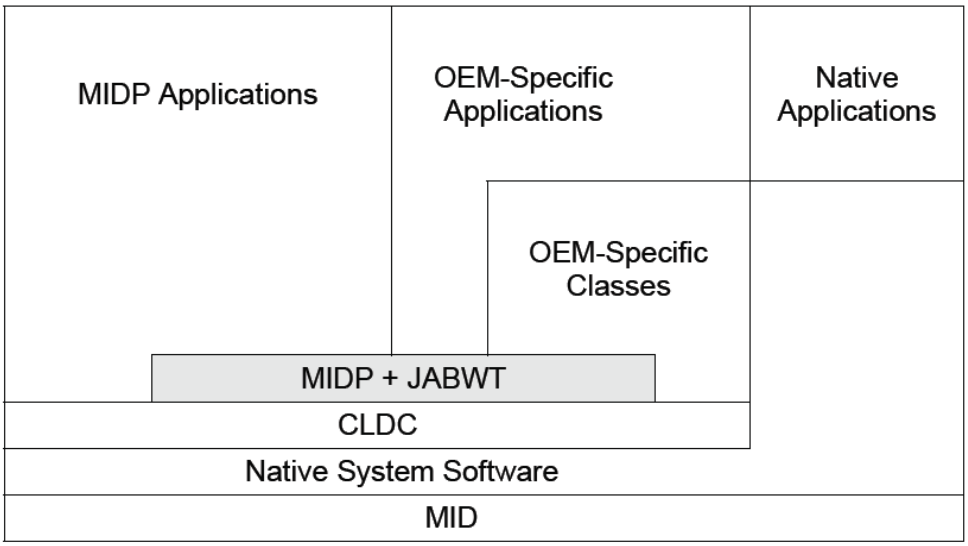

Abbildung 6.1: Position von JABWT in der J2ME-Architektur [7]

6.2 Tauschpartner Suche

Im *Bluetooth protocol stack* gibt es das *Service Discovery Protocol (SDP)*. Dieses Protokoll arbeitet mit sogenannten *Service Records*, die jeweils eine oder mehrere Attribute-IDs besitzen. Mittels Attribute-IDs wird ein über Bluetooth angebotener Dienst identifiziert.

Der ATVO Tauschserver initialisiert im Betrieb einen Dienst, der unter der *Universally*

Unique Identifier (UUID)[1] FFF0D0C0B0A000908070605040302010 zu finden ist, und bietet den Tauschdienst unter der *Attribute-ID* 0x2005 an. Der Tauschdienst ist zeitlich nicht begrenzt. Er läuft solange bis ein Tauschpartner gefunden wurde, oder bis der Benutzer diesen manuell abschaltet.

Der ATVO-Tauschclient sucht nach einem aktivierten Tauschdienst auf Bluetoothgeräten in der Umgebung, der die oben genannte *UUID* besitzt, und einen Dienst mit dem oben genannten *Attribute-ID*. Ist dies der Fall nimmt der Tauschclient den Dienst an und verbindet sich, nach einer Erlaubnis von dem Benutzer, zu dieser.

6.3 Tauschprotokoll

Das *Tauschprotokoll* von ATVO arbeitet wie folgt. Der Server startet einen Dienst, der auf die Verbindung eines Tauschclients wartet. In dem Fall, dass sich ein Client mit dem Server verbindet, wird dieser die *wishList* und *offerList* vom Server erhalten. Der Client vergleicht die *offerList* vom Server mit seiner eigenen *wishList* und die *wishList* vom Server mit seiner eigenen *offerList* und erstellt daraus einen möglichen Tauschvorschlag, der dem anderen Benutzer (auf der Seite des Clients) angezeigt wird. Der Benutzer kann diese Liste bearbeiten, um diese auf seine individuellen Wünsche anzupassen. Der Client schickt im Folgenden den Tauschvorschlag an den Server und wartet auf dessen Bestätigung.

Der Server zeigt dem Benutzer den Tauschvorschlag des Clients an, und wartet auf eine Bestätigung oder Ablehnung des Vorschlags von dem Benutzer. Bei einer Ablehnung beendet der Server von seiner Seite aus den Tausch und übermittelt eine Ablehnungsnachricht an den Client. Bei einer Zustimmung des Tausches versendet der Server die angeforderten Objekte an den Client und bekommt anschließend die geforderten Daten übermittelt. Die persistente Speicherung der gesendeten Objekte wird erst nach dem vollständigen Erhalt der geforderten Daten vom Client gelöscht. Nach dem Erhalt der Objekte vom Client ist der Tauschvorgang sowohl für den Server als auch für den Client beendet.

[1] The Universally Unique IDentifier (UUID), is the data type used for identifying protocols and profiles etc. [7]

Bei einer Ablehnung des Vorschlags vom Server beendet der Client ebenfalls den Tausch. Wird der Vorschlag vom Server angenommen, werden die Objekte dem Client direkt im Datenstrom übermittelt. Nach dem vollständigen Erhalt aller geforderten Objekte vom Server sendet der Client die im Tauschvorschlag versprochenen Objekte an dem Server und entfernt diese aus dem persistenten Speicher. Abschließend beendet der Client den Tausch auf seiner Seite.

Wie in Abschnitt 4.4 bereits erwähnt, läßt sich jeder Objekttyp in ATVO in Byte Arrays umwandeln. Diese wird als Datenformat für den Datentransfer benutzt.

Tabelle 6.2 und Tabelle 6.3 skizzieren grafisch den Ablauf des ATVO-Protokolls. Tabelle 6.2 stellt den Verlauf eines erfolgreichen Objektentauschs dar, während Tabelle 6.3 den Ablauf einer erfolglosen Tauschverhandlung zeigt.

Server	wishList, offerList	Client
	——>	
	Tauschvorschlag	
	<——	
	Objekte	
	——>	
	Objekte	
	<——	Client Dienst beendet
Server Dienst beendet		

Tabelle 6.2: Positiver Verlauf eines Tauschs im ATVO-Protokoll

Server	wishList, offerList	Client
	——>	
	Tauschvorschlag	
	<——	
	Ablehnung	
	——>	
Server Dienst beendet		Client Dienst beendet

Tabelle 6.3: Gescheiterte Tauschverhandlung ohne Objektübermittlung

Kapitel 7

Resümee

Durch ATVO ist eine Grundlage geschaffen worden, um beliebige digitale Objekte auf JSR82 fähigen MIPD Endgeräten zu verwalten und auszutauschen.

ATVO wurde modular konzipiert. Es besteht im Wesentlichen aus drei Teilen: der *persistenten Datenverwaltung*, der *Datenkommunikation* und der *GUI*. Diese sind *universell einsetzbar*, da sie auch mit neuen Objekttypen funktionieren. Alle drei Teile wurden so strukturiert, dass bei einem weiteren Ausbau des Programms für Objekte mit komplexerem Inhalt als den derzeitigen exemplarischen Objekten, sie *einfach und schnell erweitern* kann.

Die *persistente Datenverwaltung* von ATVO, basiert auf die dem *Record Management System*, arbeitet mit einem funktionellen Konzept mit *Inhaltsverzeichnis*, *Listenverzeichnis* und *Listen*.

Die *Datenkommunikation* verfügt über die Funktionalität *Tauschpartner bequem und voll automatisch über Bluetooth zu finden*. ATVO bietet dem Benutzer *die Möglichkeit Objekte für einen Tausch auszuhandeln*. Der *Datentransfer* basiert auf einem eigens entwickelte Datenformat. Um Objekte über eine Bluetooth Verbindung zu transferieren wurde ein *effizientes Tauschprotokoll* entworfen.

Die *GUI* ist für die derzeitige Darstellung einfacher Objekte ausgelegt. Es wurde auf eine *einfache und zweckorientierte Menüführung* geachtet. Sie arbeitet ausschließlich mit dem *High-Level-API*, darum kann sie sich *individuell auf das Design des jeweiligen Handsets* auf dem ATVO läuft anpassen.

In ATVO MIDlet-Suite ist außerdem das Tool *Objects Creator* enthalten, das ein *schnelles und unkompliziertes Erzeugen von Objekten* ermöglicht.

Literaturverzeichnis

[1] BITKOM. http://www.bitkom.org/.

[2] ComputerBase Lexikon. http://www.computerbase.de/lexikon/.

[3] Technische Universität Darmstadt. iClouds. http://iclouds.tk.informatik.tu-darmstadt.de/.

[4] Technische Universität Darmstadt. Mund-zu-Mund Propaganda mit Bonussystem in mobilen Ad-Hoc Netzen . http://iclouds.tk.informatik.tu-darmstadt.de/iClouds/pdf/mobwissen2003.pdf.

[5] Sun Certified Mobile Application Developer. J2ME Glossary. http://scmad.gayanb.com/j2me-glossary1.php.

[6] Soma Ghosh. J2ME record management store. http://www-128.ibm.com/developerworks/wireless/library/wi-rms/.

[7] André N. Klingsheim. J2ME Bluetooth Programming. Master's thesis, Department of Informatics University of Bergen, June 2004.

[8] Sun Microsystems. Download Java 2 Platform. http://java.sun.com/j2se/1.4.2/download.html.

[9] Sun Microsystems. J2ME Sun Java Wireless Toolkit. http://java.sun.com/products/sjwtoolkit/.

[10] Sun Microsystems. Java 2 Platform Micro Edition Datasheet. http://www.sun.com/software.

[11] Sun Microsystems. Java 2 Platform, Micro Edition (J2ME). http://java.sun.com/j2me/.

[12] Sun Microsystems. J2ME Wireless Toolkit 2.2 Documentation, 2004.

[13] John Muchow. Inside the Record Management System. http://www-128.ibm.com/developerworks/library/j-j2me3/.

[14] Swedish Institute of Computer Science. MobiTip. http://www.sics.se/humle/projects/mobitip/about.php.

[15] Michael Kroll und Stefan Haustein. *J2ME Developer´s Guide*. Markt+Technik, 2003.

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Bachelorarbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Düsseldorf, 10.November 2005 Viet Anh Nguyen