



**TECHNISCHE UNIVERSITÄT ILMENAU**

FAKULTÄT FÜR INFORMATIK UND AUTOMATISIERUNG

INSTITUT FÜR THEORETISCHE UND TECHNISCHE INFORMATIK

FACHGEBIET RECHNERARCHITEKTUREN

**Studienjahresarbeit**

Realisierung einer mobilen P2P-Anwendung  
mittels JXTA für J2ME

Datum	20.10.2004 (WS 2004/2005)
Autor	Mario Kubek (27976)
Studiengang	Informatik
Betreuer	Dr.-Ing. Jürgen Nützel

# Inhaltsverzeichnis

1. Einleitung und Ziel .....	3
2. Eingesetzte Hard- und Software .....	4
2.1. <i>Restriktionen heutiger mobiler Endgeräte</i> .....	4
2.2. <i>Die "Java 2 Micro Edition" (J2ME)</i> .....	4
2.3. <i>Das Projekt "JXTA for J2ME" (JXME)</i> .....	7
2.3.1. <i>Die Komponenten von JXME</i> .....	7
2.3.2. <i>Das Application Programming Interface (API) von JXME</i> .....	10
3. Das Programm "JXMEShare" .....	11
3.1. <i>Die Entwicklungs- und Testumgebung</i> .....	11
3.2. <i>Programmbeschreibung</i> .....	12
3.3. <i>Codebeschreibung</i> .....	16
4. Die Programmtests .....	21
4.1. <i>Die Nachrichtenlaufzeiten</i> .....	21
4.2. <i>Die Zuverlässigkeit der Übertragung</i> .....	21
5. Abschlussbetrachtungen .....	22
Abbildungsverzeichnis .....	23
Literaturverzeichnis .....	24

# 1. Einleitung und Ziel

Drahtlose Kommunikation hat in unserer heutigen mobilen Welt einen hohen Stellenwert eingenommen. Dies zeigen unter anderem die in den letzten Jahren enorm gestiegenen Verkaufszahlen drahtloser Endgeräte wie Mobiltelefone und PDAs sowie die mannigfaltigen Anwendungen auf diesen Geräten. Peer-to-Peer-Software (P2P) jedoch ist in diesem Segment noch rar, bedingt natürlich durch die begrenzten Ressourcen (geringer Arbeitsspeicher, niedrige CPU-Verarbeitungsgeschwindigkeit) solcher Kleingeräte und die hohen Datenübertragungspreise in Funknetzen wie GSM und UMTS. Da die Hardware jedoch stetig weiter entwickelt wird, sind sie heute in der Lage, größere Datenvolumen zu verarbeiten und gegebenenfalls persistent zu speichern. P2P-Anwendungen lassen sich im Allgemeinen durch mehrere Fähigkeiten charakterisieren. Sie können dynamisch einzelne Peers, sowie Peer-Gruppen erzeugen, Peers ausgewählten Gruppen beitreten lassen und Kommunikation zwischen ihnen etablieren, wozu auch das Anbieten und Versenden von Ressourcen gehört. Das Projekt JXTA, 2001 durch Sun Microsystems ins Leben gerufen und nun durch eine große industrielle und akademische Gemeinschaft weiterentwickelt, stellt eine Plattform zur Anwendungsprogrammierung in verteilten Umgebungen dar [1], jedoch mit speziellem Augenmerk auf das P2P-Computing. Die JXTA-Technologie ist im Hinblick auf Interoperabilität, Plattformunabhängigkeit und universeller Einsetzbarkeit entworfen worden. So können sich Peers leicht zum Datenaustausch finden, verbinden und gegenseitig Dienstleistungen anbieten. Dabei macht es keinen Unterschied, in welcher Programmiersprache eine JXTA-Anwendung geschrieben wird, gleichfalls welches Betriebssystem, noch welches Protokoll der Datenübertragung im Netzwerk (z.B. TCP/IP oder Bluetooth) eingesetzt wird.

Durch eingangs erwähnte Verbesserungen der Hardware heutiger mobiler Geräte, werden P2P-Anwendungen auf diesen für die Endnutzer immer reizvoller, erweitert solche Software doch die Kommunikationsmöglichkeiten erheblich, etwa durch "Instant Messaging", "Push-to-talk"-Dienste und natürlich das drahtlose Filesharing. Deshalb wurde das Projekt "JXTA for J2ME", kurz JXME, mit dem Ziel erstellt, die JXTA-Plattform und damit P2P-Funktionalität auf mobilen Endgeräten verfügbar zu machen.

Das Ziel dieser Studienarbeit bestand darin, eine Filesharing-Anwendung mittels JXME zu erstellen und diese auf mobilen Endgeräten zu testen. Zunächst soll es darum gehen, zu besprechen, was J2ME und JXME bedeuten und wie JXME auf J2ME aufbaut. Danach werden die entwickelte Anwendung "JXMEShare" beschrieben und grob die Aufgaben der Java-Klassen erläutert. Zum Abschluss folgen eine Auswertung der Tests der Software und Möglichkeiten ihrer Verbesserung, sowie ein Ausblick auf die Zukunft von JXME.

## **2. Eingesetzte Hard- und Software**

### **2.1. Restriktionen heutiger mobiler Endgeräte**

Um mobile Anwendungen zu entwickeln, ist es notwendig, die Grenzen ihrer Realisierbarkeit hinsichtlich der Leistungsfähigkeit der Zielhardware zu kennen, liegt ihre Rechenleistung doch im Bereich der PCs von vor 20 Jahren. Folgende Beschränkungen drahtloser Kleingeräte sind hervorzuheben:

- moderate CPU-Leistung (um 20 MHz)
- geringer Arbeitsspeicher (etwa 160 kbyte)
- persistenter Anwendungsspeicher gering
- mobile Stromversorgung nur durch Akku
- schmalbandige Netzwerkverbindungen und hohe Datenlaufzeiten

Trotz dieser Einschränkungen, die natürlich teilweise mit der Zeit durch stetige Hardwareverbesserungen an Relevanz verlieren, sind sie dennoch vorhanden. Der größte Pluspunkt mobiler Anwendungen ist daher ihre Fähigkeit, mit anderen Systemen zu kommunizieren, denen leistungsfähigere Ressourcen zur Verfügung stehen. Auf diese Weise können sie andere Rechner benutzen, um eventuell rechenintensive Probleme durch geschickte Aufgabenzuteilung an diese bewältigen zu können.

### **2.2. Die "Java 2 Micro Edition" (J2ME)**

Neben den Java-Versionen "Standard" (J2SE) und "Enterprise" (J2EE) hat Sun die "Java 2 Micro Edition" (J2ME) entwickelt. Diese ist speziell auf die geringer zur Verfügung stehenden Ressourcen kleinerer Plattformen ausgerichtet. Wie in den anderen Editionen sind bei J2ME die Portabilität von Code und dessen Hardwareunabhängigkeit die hohen Ziele. Jede der Editionen umfasst folgende Werkzeuge zur Programmentwicklung und Ausführung:

- virtuelle Maschinen, die auf die jeweilige Plattform zugeschnitten sind
- spezielle APIs für jede Umgebung
- Werkzeuge zur Geräteanpassung
- Spezifikationen für Geräteprofile

Die J2ME beinhaltet speziell ausgerichtete Konfigurationen und Profile [2][3] für Kleingeräte, nämlich die „Connected Limited Device Configuration“ (CLDC) und die „Connected Device Configuration“ (CDC). Am Funktionsumfang der Konfigurationen lässt sich die Leistungsfähigkeit einer Klasse von Endgeräten erkennen. Die CLDC ist für stark ressourcenbeschränkte Umgebungen ausgelegt, die CDC benötigt eine leistungsfähigere Grundlage. Jede Konfiguration basiert auf einer virtuellen Maschine und einer Klassenbibliothek, wobei die CDC auf eine vollständige „Java Virtual Machine“ (JVM) angewiesen ist, die CLDC hingegen eine „Kilobyte Virtual Machine“ (KVM) einsetzt. Die Profile sind logisch über den Konfigurationen, jedoch direkt an sie

gebunden. So basieren die Profile MIDP V1.0, MIDP V2.0 und IMP (Information Module Profile) auf der CLDC, die Profile „Foundation Profile“, „Personal Basis Profile“ und „Personal Profile“ jedoch auf der CDC. Für diese Studienarbeit wurde MIDP V1.0 eingesetzt, so dass an dieser Stelle mit einem Hinweis auf entsprechende Literatur nicht auf die anderen Profile eingegangen wird. Interessante optionale Pakete (APIs) für die CLDC sind unter anderen die „Wireless Messaging API“ (WMA), die „Webservices API“, die „Mobile Media API“, die „Java Bluetooth APIs“ und die „Location API“ in unterschiedlichen Versionen. Diese Profile und APIs stellen die Kernklassen für drahtlose Geräte zur Verfügung, mit deren Hilfe es möglich ist, Programme zu erstellen, die die Fähigkeit haben, mit Java-Lösungen auf Workstations, Servern und Mainframes zu interagieren. CLDC zielt auf Kleingeräte wie Mobiltelefone und PDAs ab. Diese Konfiguration ist für 16/32-bit RISC/CISC Mikroprozessoren und Speichergrößen um 160 kbyte (128 ROM, 32 RAM) gedacht. Im ROM befinden sich bei CLDC die Klassenbibliotheken und die virtuelle Maschine. Der RAM steht den Anwendungen zur Verfügung. Darüber hinaus kann ein persistenter Speicher vorhanden sein. Die drahtlose Netzwerkanbindung erreicht eine Datenübertragungsrate im GSM-System bei verbindungsorientierten Verbindungen von 9600 bits/s und gegebenenfalls im gleichen Modus bei Nutzung eines optimierten Funkprotokolls 14400 bits/s. Paketorientierte GPRS-Verbindungen übertragen heutzutage Daten mit 25 bis 56 kbits/s, abhängig von der Konfiguration des Netzbetreibers und der momentanen Last der Funkzelle. Die CLDC definiert eine virtuelle Maschine mit eingeschränktem Funktionsumfang, eine damit einhergehende Teilmenge von Java-Sprachkonstrukten und eine minimale Klassenbibliothek. Die Spezifikation der KVM basiert mit Einschränkungen auf der der JVM. Diese werden nun kurz angerissen:

- keine Fließkomma-Arithmetik, da in Hardware nicht durchgehend unterstützt
- fest in KVM integrierter und nicht austauschbarer Klassenlader
- keine Reflections: keine Bereitstellung von Metadaten über Struktur von Klassen Methoden und Variablen vorhanden
- Threadprogrammierung möglich, jedoch keine Thread-Gruppen und Daemon-Threads
- kein Finalizing: Garbage Collector kann anwendungsspezifische Aufräumarbeiten nach Objektverdrängung nicht in Gang setzen
- keine schwachen Referenzen

Obwohl der J2SE-Compiler javac zur Übersetzung von J2ME-Programmen eingesetzt wird, kommt es trotzdem nicht zur Ausführung von etwaig vorhandenem mit der KVM nicht kompatiblen Bytecode wegen des Zwischenschritts der Bytecodeverifikation in der KVM nach Erzeugung der .class Dateien durch den Compiler.

Das MIDP ist das wichtigste auf der CLDC aufbauende Profil. Seine Klassenbibliothek ist darum eine Ergänzung zu der der CLDC. Die darin enthaltenen Klassen ermöglichen die Entwicklung von mobilen Java-Anwendungen durch Vorgabe von Schnittstellen zur Steuerung der Lebensphasen der Applikation, zur 2D-Spieleentwicklung, der Gestaltung grafischer Benutzeroberflächen, der Netzwerkkommunikation und der persistenten Datenspeicherung. Diese APIs sind in den jeweiligen Klassenpaketen enthalten. Ihre genaue Beschreibung an dieser Stelle würde jedoch den Rahmen dieser Arbeit sprengen, zumal bereits andere wissenschaftliche Arbeiten zu diesem Thema an der TU-Ilmenau entstanden sind. Wichtig jedoch ist, dass man zur Programmentwicklung

auf API-Pakete verschiedener Hersteller zurückgreifen kann, die mitunter nicht kompatibel sind, weil sie z.B. Funktionen mitliefern, die auf einem herstellerfremden Geräten nicht lauffähig sind. Ein Beispiel hierfür ist die Socket-API von Siemens im Paket `com.siemens.mp.io.Connection`, die Unterstützung für Raw-Sockets bietet, jedoch mit Mobiltelefonen von Nokia, etwa dem 7650, nicht eingesetzt werden kann, da Raw-Sockets auf diesem Gerät nicht unterstützt werden, was aber der Spezifikation MIDP V1.0 entspricht, die nur HTTP-Kommunikation vorsieht. Aufgrund solcher und anderer herstellerbedingter Unterschiede, der Programmierschnittstellen, wie die Bereitstellung von Dateioperationen, ist eine hardwareunabhängige Programmentwicklung schwer zu realisieren, was dem eigentlichen Konzept der hardwareunabhängigen Ausführung von Java-Programmen entgegent läuft.

Folgende Abbildung demonstriert die Abgrenzungen der Java-Editionen untereinander:

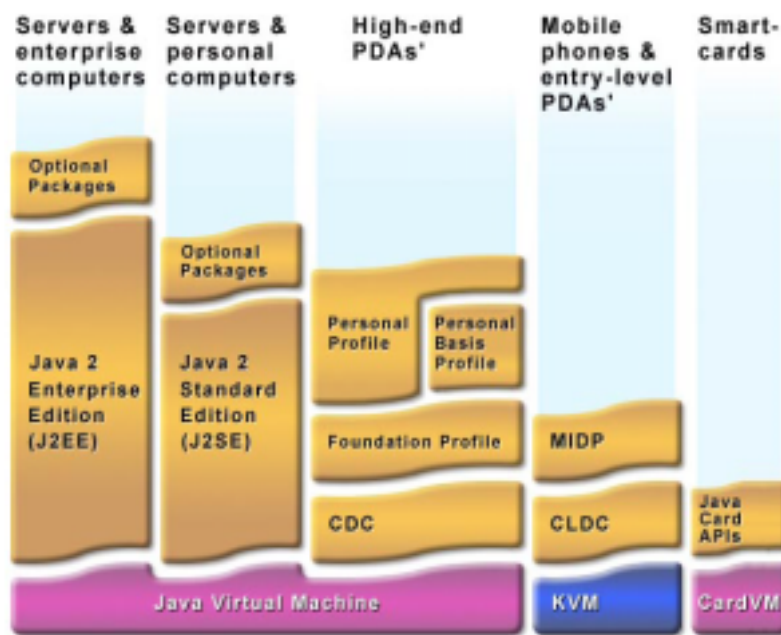


Abbildung 1 Java-Editionen [4]

Eine J2ME–Anwendung wird als MIDlet bezeichnet. Dabei handelt es sich um eine Java-Klasse, welche von der abstrakten Klasse MIDlet der MIDP-Laufzeitbibliothek abgeleitet ist und zwingend die Methoden „startApp“, „pauseApp“ und „destroyApp“ zur Lebenszyklussteuerung der Anwendung beinhaltet. Zusätzlich kann die Applikation eigene Klassen und Methoden umfassen. Ein oder mehrere MIDlets können als sogenannte MIDlet-Suite in Form eines .jar-Archivs verpackt werden. Es enthält die präverifizierten .class-Dateien der MIDlets, etwaig nötige Hilfsdateien und ein Manifest. Dieses enthält Metainformationen, die Auskunft über die MIDlets geben, wie die MIDlet-Namen, den Hersteller und die verwendete Konfigurationen und Profile. Neben dem .jar-Archiv kann optional den Applikationsdeskriptor .jad vorhanden sein, der weitgehend die gleichen Attribute beinhaltet wie das Manifest und dazu dient, vor dem Herunterladen einer Applikation, die .jar-Größe und die Kompatibilität mit der Laufzeitbibliothek zu testen. Dies erspart den damit vermeidbaren Datenverkehr.

## **2.3. Das Projekt "JXTA for J2ME" (JXME)**

JXTA wurde nicht nur dazu entworfen, eine P2P-Infrastruktur bereitzustellen, sondern auch als virtuelles Netzwerk einige der Plagen verteilter Systeme wie unterschiedliche Hardware und unterschiedliche Datenrepräsentation in hoch dynamischen Netzwerken zu überdecken. Wie in der Einleitung erwähnt, stellt JXME eine Technologie dar, die P2P-Anwendungen auf drahtlosen Geräten ermöglicht. Damit ist es basierend auf J2ME möglich, mobile JXTA-Peers zu erstellen, die untereinander und mit Peers z.B. auf Desktops für P2P-Aktivitäten in Verbindung treten. JXME fußt dabei auf der CLDC und dem MIDP V1.0. Die beiden Module Peer und Relay sind die grundlegenden Komponenten, auf denen dieses Projekt basiert.

Die Hauptziele von JXME [1] sind:

- Interoperabilität mit JXTA auf Desktops und Workstations
- Bereitstellung einer P2P-Infrastruktur für Kleingeräte
- intuitive Erlernbarkeit für Programmierer durch simple API
- Einsetzbarkeit auf Mobiltelefonen und PDAs
- geringer Speicherbedarf
- MIDP V1.0 Kompatibilität

Obwohl MIDP V1.0 schon eine Menge der nötigen APIs mitliefert, die man zur Erstellung mobiler Applikationen benötigt, besitzt es doch einige Einschränkungen wie das Fehlen eines XML-Parsers, keine Sicherheitsverwaltung bei Datenverbindungen und die Begrenzung der Kommunikation via HTTP, was sich auf die Fähigkeiten von JXME niederschlägt. Anstelle von XML-Nachrichten wird das Binärformat von JXTA eingesetzt. Damit umgeht man die ansonsten notwendige Integration eines XML-Parsers, was die Ablaufgeschwindigkeit eines mobilen Java-Programmes schmälern kann, spart Speicherplatz im Anwendungsspeicher der Zielhardware und verringert das entstehende Datenvolumen und damit die Kosten mobiler Datenübertragung erheblich. Die J2ME besitzt einen HTTP-Client, jedoch keinen HTTP-Server. Daraus folgt, dass eine J2ME-Applikation nur HTTP-Requests versenden, jedoch keine Port öffnen kann, um an einem solchen nach eingehenden Requests zu lauschen.

### **2.3.1. Die Komponenten von JXME**

In JXTA sind mehrere Typen von Ressourcen, wie Peers, Peer-Gruppen, Kommunikationskanäle (Pipes) und zu übertragende Daten definiert. Die für JXME wichtigsten Komponenten werden nun vorgestellt [1][6].

#### **JXME-Peers**

Ein Peer im JXTA-Netzwerk wird durch einen Knoten repräsentiert. Dazu hat jeder Peer eine einzigartige Identifikationsnummer, genannt Peer ID. Diese wird dynamisch an seine IP gebunden.

Durch oben genannte Einschränkungen, sowohl bezogen auf die eingesetzte Plattform als auch die damit in Zusammenhang stehenden Beschränkungen in MIDP V1.0, können JXME-Peers nur als sogenannte Edge-Peers im JXTA-Netzwerk auftreten. Dies bedeutet, dass sie nicht aufwändigere Services, wie das Suchen nach Ressourcen im Netzwerk und rechenintensive Aufgaben übernehmen können. Dies müssen andere Peers in einer Gruppe übernehmen.

### ***Peer-Gruppen***

Eine Peer-Gruppe stellt eine logische Verbindung einzelner Peers dar. Sie ist eine Gemeinschaft, bestehend aus mehreren Peers, die die gleichen Interessen besitzen z.B. eine bestimmte Musikrichtung.

### ***Pipes***

Kommunikationskanäle in JXTA, genannt Pipes, existieren wie die Peer-Gruppen nur logisch. Eine Pipe ist ein virtueller Pfad zwischen zwei Kommunikationsendpunkten, von denen ein Peer mindestens einen besitzen muss und welcher dynamisch an die IP des Peer gebunden wird.

### ***JXTA-Relays***

Um mobile Peers mit dem JXTA-Netzwerk zu verbinden, kommunizieren diese unabhängig von den unter den JXTA-Protokollen liegenden Netzwerkprotokollen mit den sogenannten JXTA-Relays. Diese dienen als Stellvertreter für die Kleingeräte und lassen darüber hinaus solche Peers als uneingeschränkt in erscheinen. Auf diesen Relay-Peers läuft der sogenannte „JXTA for J2ME“-Proxy Service [1], dem folgende Aufgaben zugedacht sind:

- Wahrung der Interoperabilität mit den JXTA-Protokollen
- Reduzierung der XML-Advertisements durch Herausfilterung unnötiger Informationen
- Übersetzung der XML-Nachrichten in das von JXTA spezifizierte Binärformat, welches von den JXME-Peers genutzt wird und umgekehrt
- Durchführung der Suche nach Peers, Pipes und Gruppen Peers, der Erstellung von Pipes und Gruppen, der Verwaltung der Gruppenbeitritte, und der Kommunikation mit anderen Peers zu ermöglichen im Interesse des JXME-Peers
- Publikation von Advertisements im Auftrag eines mobilen Peers
- Speicherung der an einen mobilen Peers adressierten Daten, bis dieser ihn durch HTTP-Polling abrufen
- Behandlung der von einem mobilen Peers ins JXTA-Netzwerk zu sendenden Daten

Dabei sind Relays keine Neuentwicklung für JXME, dienen sie doch dazu, Peers hinter einer Firewall/NAT mit dem JXTA-Netzwerk zu verbinden. Abbildung 2 soll dies verdeutlichen:

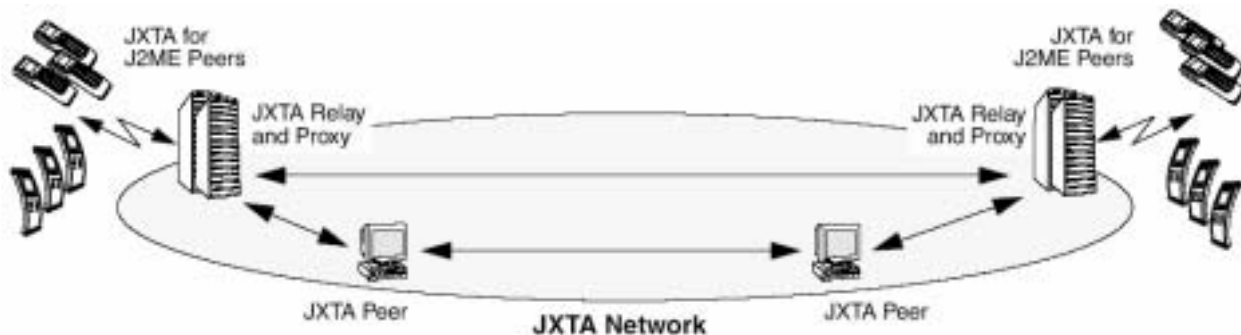


Abbildung 2 JXME-Peers und Relays im JXTA-Netzwerk [1]

Wenn also ein JXME-Peer z.B. eine Suchoperation startet, sendet er diese an einen JXTA-Relay. Dieser propagiert die Suche ins JXTA-Netzwerk. Eintreffende Antworten werden vom Relay gesammelt und in der Größe reduziert, um sie für Geräte mit niedriger Bandbreite handhabbarer zu machen. Das Ergebnis einer Peer-Suche etwa ist ein Peer-Advertisement. Aus diesem extrahiert der Relay die retournierte PeerID und hält sie für den JXME-Peer als HTTP-Antwort bereit. Auf diese Weise wird das zu übertragende Datenvolumen drastisch reduziert. Währenddessen sendet der JXME-Peer periodisch durch HTTP-Polling Anfragen, ob denn neue Nachrichten für ihn bereitstehen. Wenn dem so ist, erhält er in der Reply-Nachricht eine Antwort. JXME-Peers sind nicht dazu verpflichtet, sich an einem einzigen JXTA-Relay anzumelden oder diesen bei späteren Verbindungen wieder zu kontaktieren. Mehrere unabhängige JXME-Peers können durch unterschiedliche Relays mit dem JXTA-Netzwerk verbunden sein und trotzdem in der Lage sein, sich gegenseitig zu finden und zu kommunizieren. Darüber hinaus kann ein Peer dynamisch seinen Relay wechseln und sogar mit mehreren Relays in Verbindung stehen [5]. Momentan existiert in JXME für einen mobilen Peer kein Mechanismus, um selbständig nach verfügbaren Relays zu suchen. Deswegen muss einer mobilen JXTA-Anwendung die IP-Adresse eines zu kontaktierenden Relay vorgegeben werden, etwa durch Einstellung mittels einer GUI.

### **Die Nachrichten (Messages)**

Die Kommunikation eines JXME-Peers mit einem Relay basiert auf dem Austausch von Nachrichten. Eine Nachricht (message) besteht dabei aus einem oder mehreren Elementen. Um eine versendbare Nachricht zu konstruieren, erstellt man ein Array aus gegebenenfalls mehreren Elementen, weist ihnen die zu versendenden Daten zu und übergibt das Feld dem Message-Konstruktor. Jedes Element im HTTP-Request an den Relay enthält dann strukturierte Daten, die vom Relay geparkt werden. Dieser wandelt sie zum weiteren Versand in XML-Advertisements um.

### **2.3.2. Das Application Programming Interface (API) von JXME**

JXME wurde mit dem Ziel entworfen, die Details ihrer Implementation in den unteren Ebenen, etwa der HTTP-Kommunikation und der Zusammensetzung der Datenformate, dem Anwendungsprogrammierer zu verbergen, damit dieser die JXME-API im „Blackbox“-Verfahren einsetzen kann, ohne weitere Kenntnisse über sie zu haben. So braucht man sich nur um die Anwendungslogik Gedanken zu machen, es sei denn, man möchte die freien Quellcodes von JXME verändern.

Die schlanke Architektur von JXME spiegelt sich neben obigen Gesichtspunkten auch in der Größe ihrer API [1] wider, welche aus nur drei Klassen namens

- Message, zur Erstellung und Bearbeitung von JXTA-Nachrichten
- Element, zur Bearbeitung einzelner Nachrichtenkomponenten
- PeerNetwork, zum Aufruf von Operationen im JXTA-Netzwerk

besteht.

Um ihre Größe zu reduzieren wurde auf den Einsatz von Interfaces, Factories, Listeners, Threads und inneren Klassen verzichtet. Folgende Dienste werden in den Methoden der Klasse „PeerNetwork“ implementiert.

Die Methode "search" bietet die Suche nach:

- Peers
- Gruppen
- Pipes

Die Methode "create" leistet:

- die Erstellung von Point-to-Point Pipes und Propagate Pipes
- das Anlegen von Gruppen

Das Öffnen und Lauschen an einer Pipe wird durch die Methode "listen" ermöglicht.

Die Methode „close“ hingegen schließt eine geöffnete Pipe. Die Kommunikation mit dem Relay erfolgt durch die Methoden "send" und "poll".

### 3. Das Programm "JXMEShare"

Das in dieser Studienjahresarbeit entwickelte Programm soll zeigen, dass es mit JXME möglich ist, mobile P2P-Anwendungen zu entwickeln, die in der Lage sind, außer dem Austausch von Textnachrichten auch Dateitransfer (Filesharing) zu bewältigen.

#### 3.1. Die Entwicklungs- und Testumgebung

Bevor mit der Programmentwicklung begonnen werden konnte, war es nötig, die „Java 2 Runtime Environment“ (J2RE), das „Java 2 Software Development Kit“ (J2SDK) (hier eingesetzt Version 1.4.2\_04) und das „J2ME Wireless Toolkit“ (JWTK) [3] zu installieren, wobei letzteres nötig ist, um J2ME-Applikationen zu erstellen. Dieses umfasst unter anderem die Erstellung von J2ME-Projekten, die Generierung der jar und jad Dateien nach der Übersetzung und Präverifikation, das Debuggen einer J2ME-Anwendung und das Testen der Applikation in mitgelieferten Handy-Emulatoren.

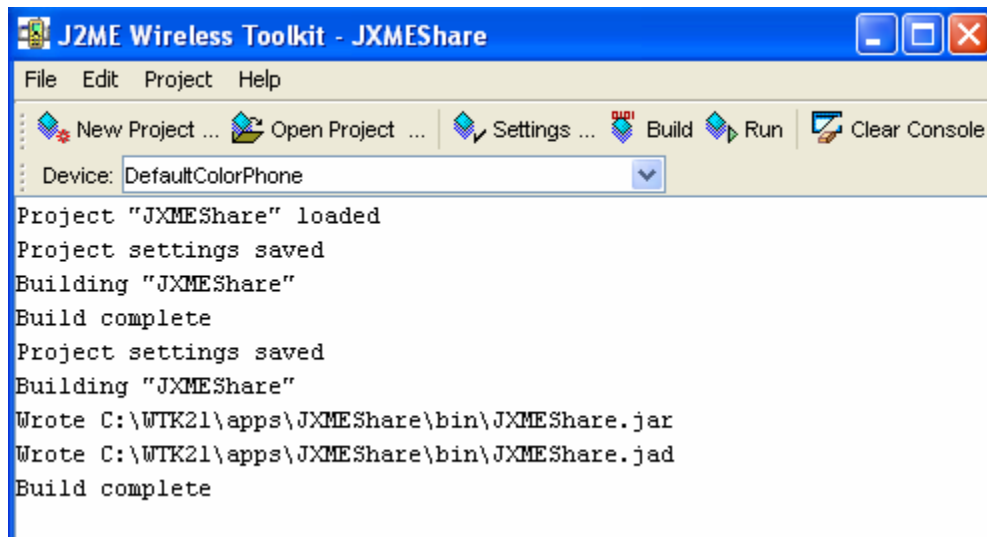


Abbildung 3 Oberfläche des JWTK

Zur Erstellung der Quelldateien wurde der Texteditor UltraEdit benutzt, der die Funktion des Syntaxhighlightings beherrscht. Leider war es nicht möglich, JXME-Anwendungen in den Emulatoren des JWTK zum Laufen zu bringen. Darum wurde sich dafür entschieden, das Programm für Handys von Siemens zu programmieren, im speziellen für das S55 und das C55. Aus diesem Grund wurde das „Siemens Mobility Toolkit“ (SMTK) [7] für MIDP V1.0 installiert und die von Siemens erweiterten Klassen in das Bibliotheksverzeichnis des JWTK kopiert, um spezielle Funktionen der Siemens-API in der Anwendung einsetzen zu können. Die von der Website von Siemens herunter geladenen Emulatoren für das S55 und das C55 wurden in das dafür vorgesehene Verzeichnis des SMTK kopiert. Die Emulatoren ermöglichen den Test und das Debugging mobiler Java-Software mit hohem Realitätsbezug ohne den Zwang, die Software auf einem echten Handy zu testen. Damit werden der Entwicklungsaufwand minimiert und im Fall dieser Studienarbeit die Kosten für mobile Datenübertragung stark reduziert, weil die Kommunikation mehrerer "JXMEShare"-Instanzen, jede in einem

Emulator, über einen Relay-Peer auf einem leistungsstarken Rechner stattfinden kann. Solch ein Relay-Peer wurde durch die Aktivierung der entsprechenden Optionen im Konfigurationsfenster der „instantp2p“-Applikation zum Laufen gebracht. „InstantP2P“ besitzt eine GUI und unterstützt Chatting und Filesharing über das JXTA-Netzwerk. Man kann sie unter <http://www.jxta.org> in dem Paket „JXTADemo\_Windows\_VM.exe“ herunterladen, genauso die Klassen von JXME im entsprechenden Projekt. Diese sind in dem Archiv „jxta-cldc.jar“ enthalten und werden in das Verzeichnis „lib“ der jeweiligen JXME-Applikation gelegt, welches durch das JWTK bei der Projekterstellung neben den Verzeichnissen „src“, „classes“, „tmpclasses“, „bin“, „res“ und „tmplib“ erzeugt wurde. Dadurch werden die JXME-Klassen bei der Kompilierung dem Projekt zugebunden. In das Verzeichnis „src“ werden die Quellcodes der Software gelegt. Das Verzeichnis „res“ beinhaltet die von der Anwendung möglicherweise benötigten Ressourcen wie etwa Bilder oder Textdateien. Die kompilierten Anwendungsklassen findet man in „classes“ und die ausführbare J2ME-Anwendung im Verzeichnis „bin“.

### **3.2. Programmbeschreibung**

Das in dieser Studienjahresarbeit entstandene Programm "JXMEShare" realisiert als J2ME-Anwendung die Möglichkeit, mobil durch Einsatz des virtuellen JXTA-Netzwerkes, mit anderen Nutzern des gleichen Programms überall auf der Welt unabhängig von der Art der Datenübertragung zu kommunizieren. Dazu implementiert es Funktionen zum Nachrichtenaustausch und dem Senden und Empfangen von Dateien. Jede Instanz des Programms generiert genau einen Peer, jeder mit einer eigenen PeerID im JXTA-Netzwerk. Es ist möglich, dass Peers den gleichen Namen im Netz haben, denn die PeerID ist das Unterscheidungsmerkmal. Im Programm werden zwei Arten von Pipes als Kommunikationskanäle eingesetzt. Zum einen die Propagate-Pipe, welche fest im Programm integriert ist und nicht vom Nutzer geändert werden kann, zum anderen die Unicast-Pipe, welche jedem Peer nach der Verbindungsaufnahme mit einem Relay durch eine Nachricht mitgeteilt wird. Die erstgenannte dient der Gruppenkommunikation. Jeder Peer lauscht an dieser. Wenn eine Nachricht oder Datei über sie eintrifft, werden alle Peers diese erhalten. Die Unicast-Pipe dient dazu, jeden Peer einzeln adressieren zu können, ohne dass andere Peers die Daten der Kommunikation erhalten. Das bedeutet, dass jeder Peer seine eigene Unicast-Pipe besitzt. Das Programm ist so konzipiert, dass nach erfolgreichem Verbindungsaufbau und empfangener Unicast-Pipe, eine Nachricht über die Propagate-Pipe gesendet wird, damit andere Peers über die Existenz des neuen Peers erfahren. Darin wird der Name des neuen Peers und seine Unicast-Pipe übertragen. Ein diese Nachricht empfangender Peer, sendet selbst darauf die eigenen Daten an die Unicast-Pipe des neuen Peer, damit dieser ihn seinerseits kennen lernt. Auf diese Weise, können sich alle Peers gegenseitig erreichen, sei es nun in Form einer Multicast-Nachricht oder gezielt als Botschaft an einen bestimmten Peer. Obwohl es auch auf der JXME/J2ME-Plattform technisch machbar ist, auf eine eingehende Suchnachricht, eine Ressource lokal zu suchen und diese ohne Wissen des Nutzers an einen anderen Peer zu transferieren, wurde darauf verzichtet, um mobilen Datenverkehr so gering wie möglich zu halten. Der Benutzer muss selbst eine Datei zum Versenden angeben und den Versand initiieren. Zur Steuerung besitzt das Programm eine grafische Benutzeroberfläche.

Diese bietet in mehreren Teilen (Screens) Möglichkeiten der Interaktion des Nutzers mit dem Programm. Diese Screens sollen nun erläutert werden.

### ***Der Main-Screen***

Dieser dient dazu, eintreffende Nachrichten sowie Programm Meldungen anzuzeigen. Desweiteren bietet er die Möglichkeiten, den Settings-Screen zu erreichen sowie über das Options-Menü die verschiedenen Programmfunktionen auszuwählen. Die auswählbaren Optionen sind die folgenden:

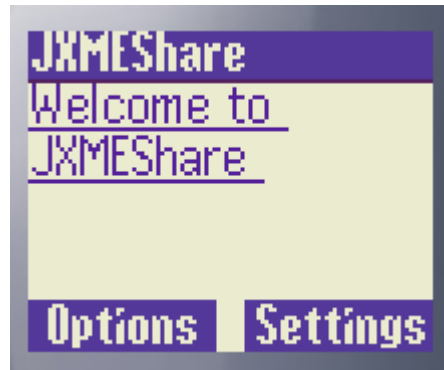


Abbildung 4 Main-Screen

- Settings:            Programmeinstellungen s.u.
- Write Msg         Verfassen von Textnachrichten
- Connect            Verbindungsherstellung mit dem Relay
- View Contacts     Anzeige der verbundenen Kontakte
- Load File         Laden einer Datei aus dem Internet
- Send File         Senden einer Datei an einen oder alle Peers
- Disconnect        aktuelle Verbindung zum Relay aufheben
- Exit                Beenden des Programms

### ***Der Settings-Screen***

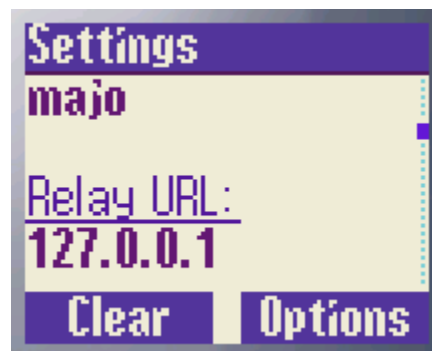


Abbildung 5 Settings-Screen

In dieser Anzeige kann man kommunikationsrelevante Einstellungen vornehmen. Dazu gehören die Wahl eines Namens für den Peer, die IP-Adresse des Relays, mit dem sich der Peer verbinden soll und die Port-Nummer, unter welcher der Relay angesprochen werden soll. Standardmäßig ist dies die 9700, welche auch nicht geändert werden sollte.

### ***Der "Write Msg"-Screen***

Durch Anwahl dieser Option, gelangt der man zu einem Fenster, in welchem man eine Textnachricht verfassen kann. Diese kann man dann sofort versenden oder zum Main-Screen zurückkehren. Entscheidet man sich für ersteres, öffnet sich ein neues Fenster mit der Auswahl der möglichen Zielkontakte. Darauf kann die Nachricht endgültig verschickt werden.



Abbildung 6 "Write Msg"-Screen

### ***Der "View Contacts"-Screen***

Dieses Fenster zeigt alle mit dem JXTA-Netzwerk verbundenen Kontakte (Peers), die im Programm bekannt sind.



Abbildung 7 "Online Contacts"-Screen

Es werden solche Peers angezeigt, die durch eine Instanz dieses Programms entstanden sind, nicht etwa eine möglicherweise riesige Menge anderer Peers.

### ***Der "Load File"-Screen***

In diesem Fenster kann man eine URL zu einer Datei auf einem Server angeben, die man dann gegebenenfalls herunterladen möchte. Wenn dies geschehen ist, wird die Datei im Verzeichnis "storage" dieses Programms abgelegt. Diese kann nun über die Option "Send a File" des Hauptbildschirms an andere „JXMEShare“-Peers gesendet werden. Diese Funktion hat nichts mit JXTA zu tun, sondern wird über eine HTTP-Netzwerkverbindung durch die Siemens-API realisiert.



Abbildung 8 "Load File"-Screen

### ***Der "Send File"-Screen***

Durch Auswahl dieser Option, kann ein User dieses Programms nach Angabe des Dateinamens, welcher keine Pfadangaben beinhalten darf, da nur Dateien im Unterverzeichnis "storage" dieses Programmes angesprochen werden können, eine Datei an einen oder alle Peers aus der Kontaktliste versenden.

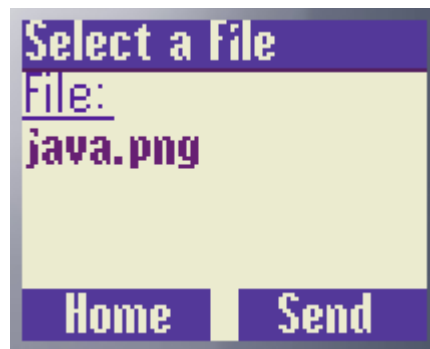


Abbildung 9 "Send File"-Screen

Die bei dem Ziel-Peer eingehende Datei wird auch im Verzeichnis "storage" gespeichert und liegt nun auch dort als Ressource vor.

### **3.3. Codebeschreibung**

Wie oben angesprochen, ist das Programm "JXMEShare" in Java unter Zuhilfenahme der "MIDP V1.0"-API von Siemens und der JXME-Klassen programmiert worden. Es besteht aus zwei Klassen namens "JXMEShare" als Hauptklasse und "IntroCanvas", die für die Anzeige des Begrüßungsbildes zuständig ist.

#### ***Die Hauptklasse "JXMEShare"***

Alle wichtigen Methoden zur Grundfunktionalität des MIDlets befinden sich in dieser Klasse. Die in ihr deklarierten Variablen haben mannigfaltige Aufgaben. Dazu zählen Variablen für Screens, Commands, JXME (z.B. Propagate-PipeID) und programminterne Zwecke zum Nachrichtenaustausch. Die einzelnen Methoden, welche diese Variablen verwenden, sollen nun erläutert werden.

#### ***Der Kontruktor "JXMEShare"***

Diese Methode dient primär dazu, die Bedienelemente des Programms zu initialisieren. Dazu werden die menübezogenen Kommandoelemente generiert und den entsprechenden Screens zugewiesen. Teilweise erhalten Textfelder vordefinierte Werte z.B. die Portnummer 9700 im entsprechenden Element.

#### ***Die Methode "startApp"***

Diese in jedem J2ME-Programm unverzichtbare Methode, stellt für die MIDP-Laufzeitumgebung den Einsprungpunkt zur Programmabarbeitung dar. In ihr wird der Intro-Bildschirm aufgerufen und ein Thread für das Polling nach eingehenden Nachrichten gestartet, da dieses für Netzwerkoperationen speziell für blockierende Operationen sinnvoll ist, die sonst den übrigen Programmablauf unnötig ausbremsen könnten. Die für den "pollThread" genannten Thread unabdingbare Methode "run" wird weiter unten erläutert.

#### ***Die Methode "pauseApp"***

Hierin wird der pollende Thread angehalten, etwa wenn ein anderes Programm im Mobiltelefon gestartet wird oder ein Anruf eingeht. Die Programmausführung wird bis zur Reaktivierung pausiert.

#### ***Die Methode "destroyApp"***

Auch hier wird der pollende Thread angehalten, mit dem Unterschied, dass beim Aufruf dieser Funktion das MIDlet kurz vor der Beendigung steht.

### **Die Methode "commandAction"**

Damit eine Applikation high-level-events wie Benutzereingaben bearbeiten kann, muss eine Klasse die das Interface `CommandListener` implementiert, diese Methode besitzen. Je nach gerade angezeigtem Objekt der Klassen `Displayable`, wozu in diesem Programm die Screens zählen, kann nach Auswertung eines gerade aktivierten Kommandos eine bestimmte Aktion ausgeführt werden. Die Behandlung der Eingaben geht also in zwei Schritten vor sich. Zuerst wird getestet, welcher Screen gerade angezeigt wird und dann eine Auswertung des gewählten Kommandos vorgenommen. Wird als Beispiel der "Settings"-Screen angezeigt, stehen zur Auswertung die Kommandos "homeCommand" und "ConnectCommand" zu Verfügung. Sollte das erste ausgewählt werden, wird auf das Hauptfenster zurückgeschaltet. Im zweiten Fall wird die Routine "connect" für den Verbindungsversuch mit einem Relay aufgerufen. In diesem Stil laufen auch die anderen Auswertungen der Benutzereingaben.

### **Die Methode "connect"**

Diese Methode ist für den Verbindungsaufbau zu einem JXTA-Relay essentiell. Sie beinhaltet mehrere JXTA-bezogene Methodenaufrufe. Wird sie ohne Fehler bis zum Ende ausgeführt, wurde ein Peer samt seiner eigenen PipeID erstellt und eine Verbindung mit einem Relay etabliert. Dazu sollen die einzelnen Schritte genauer beleuchtet werden, weil diese Funktion eine der wichtigsten im Programm ist.

Im ersten Schritt wird durch den Aufruf

```
pNetwork = PeerNetwork.createInstance(tfpeername.getString());
```

ein neuer JXME-Peer erzeugt, dessen Name aus einem Textfeld des "Settings"-Screen stammt. Danach wird versucht, eine Verbindung mit dem spezifizierten Relay durch

```
state = pNetwork.connect(relayUrl, state);
```

aufzunehmen. War dieser Schritt erfolgreich, wird mit

```
int result = pNetwork.create(PeerNetwork.PIPE, PIPE_NAME, pipeID, PeerNetwork.PROPAGATE_PIPE);
```

eine Propagate-Pipe erstellt, deren Nummer (pipeID) im Programm fest vorgegeben ist, damit alle Peers für Multicast-Nachrichten an genau dieser lauschen. Darauf folgt ein ähnlicher Aufruf mit

```
uniqueRequestId = pNetwork.create(PeerNetwork.PIPE, "Uni_Pipe", null, PeerNetwork.UNICAST_PIPE);
```

um eine neue Unicast-PipeID durch den Parameter null eigens für den Peer zu erzeugen. Dieser Aufruf bewirkt, dass der Relay solch eine neue Pipe kreiert und diese als Antwort zur Abholung durch den Peer bereithält.

Um sie zu erhalten, wird der Peer durch

```
listenQueryId = pNetwork.listen(pipeID);
```

dazu gebracht, an der Propagate-Pipe zu lauschen. In nachfolgender while-Schleife wird versucht, durch Polling des Relays nach neuen Nachrichten, die relevanten Daten der Elemente, hier die Daten der Request-ID, des NAME-TAG und des ID-TAG, zu sichern. Stimmt die Nummer der retournierten Request-ID mit derjenigen der Variablen `uniqueRequestId` überein und sind die anderen beiden Tags vorhanden, so werden diese Daten gesichert. Besonders wichtig hierbei ist die Speicherung des Unicast-PipeID in der Variablen `uniquePipeId`. Das Konzept des Nachrichtenaufbaus aus Elementen wird daran gut deutlich. Eine Nachricht wird nach passenden Elementen durchsucht und die Daten dieser gegebenenfalls gesichert.

Nach dem Empfang der Unicast-PipeID lauscht der Peer durch

```
pNetwork.listen(uniquePipeId);
```

neben der Propagate-Pipe auch an dieser. Darauf wird eine leere Nachricht mittels

```
sendMessage(fixedPipeID);
```

über die Propagate-Pipe zu den an dieser lauschenden Peers gesendet, um sich selbst den anderen vorzustellen. Darauf wird diese PipeID in eine Kontaktliste als Feld „Community“ eingefügt und die Variable "connected" auf true gesetzt. Diese dient dazu, in anderen Methoden sicherzustellen, dass eine Verbindung mit dem Relay besteht.

### ***Die Methode "disconnect"***

Wie unschwer am Namen der Methode zu erkennen, bewirkt sie eine Kappung der Verbindung durch das Schließen der beiden Pipes. Die Variable "connected" wird auf false gesetzt.

### ***Die Methode "sendMessage"***

Diese Methode dient dem Versenden von Textnachrichten. Dabei besteht jede Textnachricht aus den drei Elementen "mySend:Name", "mySend:Message" und "mySend:UniqueID". Das erste Element trägt den Namen des sendenden Peers, das zweite die eigentliche Textnachricht und das dritte die Unicast-PipeID des sendenden Peers, damit ein empfangender Peer dem Sender direkt antworten kann. Diese Elemente werden als Nachricht an die durch den Übergabeparameter "pID" bestimmte Pipe mittels

```
sendRequestId = pNetwork.send(pID, msg);
```

gesendet.

### **Die Methode "sendFile"**

Ähnlich vorhergehender beschriebener Methode, dient diese Methode dem Nachrichtenaustausch, nur mit dem Unterschied dass hierin Dateitransfer implementiert ist. Dazu wird die API von Siemens zur Dateibehandlung eingesetzt, um eine Datei zu öffnen, sie in einen Puffer einzulesen und wieder zu schließen. Die zu übertragende Datei muss im Unterverzeichnis "storage" des Programms liegen. Pfadangaben im entsprechenden Screen führen zu einem Fehler. Diesmal besteht die zu übertragende Nachricht aus den zwei Elementen "mySend:Filename" und "mySend:File". Das erste enthält den Dateinamen, das zweite den Puffer der eingelesenen Daten. Wie oben werden diese Elemente seriell in eine Nachricht verpackt und mit obiger JXME-Anweisung asynchron verschickt.

### **Die Methode "recvMessage"**

Die Aufgabe dieser Methode besteht darin, Nachrichten zu empfangen und die darin enthaltenen Daten elementweise zu extrahieren. Der Empfang einer Nachricht, als erster Schritt, geschieht durch HTTP-Polling des Relays mittels

```
msg = pNetwork.poll(1);
```

nach neuen Nachrichten. Das Polling-Intervall wurde auf eine Sekunde eingestellt. Hält der Relay eine Nachricht für den Peer bereit, wird der Peer sie in der Reply-Nachricht des Relays auf seinen Request erhalten. Leider ist dies für MIDP V1.0 momentan die einzige Form der Kommunikation, die JXME implementiert. Das in diesem Fall entstehende Datenvolumen durch Metadaten des HTTP-Pollings ist negativ zu bewerten. Zwar besteht die Möglichkeit, das Polling-Intervall zu vergrößern, das grundlegende Problem bleibt jedoch weiter bestehen. Bei Tests verursachten der Versand und Empfang dreier Textnachrichten bereits 23 kbyte an Volumen. Kommunikation über Raw-Sockets, die leider nicht Teil der MIDP V1.0 Spezifikation sind, wäre eine Problemlösung. Eine andere Lösung wäre, in ein J2ME-Midlet Serverroutinen zu integrieren, um auf bestimmten Ports Aktionen durchführen zu können. Der Aufwand hierzu wäre jedoch recht hoch. In einer zukünftigen Version von JXME, die ohne Relay funktionieren soll, wird dieser Mangel durch Socket-Support behoben sein. Wenn nun eine Nachricht eingetroffen ist, wird sie nach den Elementen durchsucht, die in den Methoden "sendMessage" und "sendFile" angegeben wurden. Falls eine Textnachricht eingetroffen ist, wird ihr Inhalt mitsamt dem Absender dem Hauptfenster hinzugefügt und damit angezeigt. Ist die mitgeteilte Unicast-Pipe noch nicht in der dafür vorgesehenen Liste gespeichert, wird sie hinzugefügt, sowie der dazugehörige Peername in der Kontaktliste gespeichert. Der Empfang einer Datei funktioniert nach dem gleichen Prinzip. Die Daten der Elemente "mySend:Filename" und "mySend:File" werden, falls vorhanden, extrahiert und in einer neu angelegten Datei gesichert. Auch hier ist die API von Siemens zur Dateibehandlung benutzt worden. Die empfangene Datei wird standardmäßig im Verzeichnis "storage" abgelegt. Eine Mitteilung über den Empfang und die Sicherung der Datei wird ausgegeben.

### ***Die Methode "loadFile"***

Eine andere Möglichkeit, eine Datei in das Dateisystem des Handys zu übertragen, bietet die Methode "loadFile". Sie kann nach Angabe der URL einer Datei auf einem Server als Übergabeparameter, diese per HTTP-Verbindung herunterladen und im Verzeichnis "storage" sichern. Dabei werden keinerlei Routinen von JXME eingesetzt. Der Benutzer wird bei erfolgreicher Sicherung der Datei durch eine Mitteilung darüber in Kenntnis gesetzt.

### ***Die Methode "showAlert"***

Diese Methode hat die Aufgabe, Alarmmeldungen des Programms so lange anzuzeigen, bis eine Bestätigung des Lesens vom Benutzer eintrifft.

### ***Die Methode "deleteScreen"***

Damit das Hauptfenster nicht durch ältere Programmmitteilungen zu unübersichtlich wird, werden diese bei einer Anzahl von mehr als vier Elementen gelöscht.

### ***Die Methode "run"***

Da das Polling nach neuen Nachrichten in einem eigenen Thread stattfinden sollte, der kontinuierlich die Methode "recvMessage" aufruft und die Klasse "JXMEShare" das Interface Runnable implementieren soll, ist diese Funktion zwingend erforderlich. Diese sorgt dafür, dass das Begrüßungslogo eine Sekunde lang angezeigt wird und versucht dann alle 1000 Millisekunden, falls eine Verbindung zum Relay besteht, die Methode "recvMessage" aufzurufen.

### ***Die Klasse "IntroCanvas"***

Als zweite Klasse dieses Programms hat "IntroCanvas", welche von der J2ME-Klasse "Canvas" abgeleitet ist, durch die somit zwingend erforderliche Methode "paint" die Aufgabe, den Begrüßungsbildschirm zu zeichnen. Das anzuzeigende Bild wird in der Mitte des Bildschirms platziert.

## **4. Die Programmtests**

Das entwickelte Programm „JXMEShare“ wurde erfolgreich sowohl auf den Mobiltelefonen S55 und C55 von Siemens als auch mit den Emulatoren dieser Geräte im „Siemens Mobility Toolkit“ getestet. Die Tests in den Emulatoren wurden einmal mit Relay-Verbindungen zu einem lokal eingerichteten Relay aus oben erwähnter „InstantP2P“-Applikation durchgeführt, wobei die beiden Emulatoren und der Relay auf einem Rechner liefen, und zum anderen zu einem entfernten Relay durchgeführt, um die Zuverlässigkeit und die Nachrichtenlaufzeiten der Übertragung zu ermitteln. Diese Tests wurden mit dem auf dem Campus der TU-Ilmenau vorhandenen WLAN-System durchgeführt. Die Programmtests in den Mobiltelefonen S55/C55 wurden über GPRS-Datenverbindungen im GSM-Netz des Netzbetreibers o2 getestet. Diese Verbindungen liefen ebenfalls über ein Relay auf einem mit dem WLAN-System verbundenen Rechner. Das Polling-Intervall nach neuen Nachrichten wurde auf 1 Sekunde eingestellt. Weil sich herausstellte, dass das Polling nach dem Versenden einer Datei mit einer Größe von mehr als 8 kbyte nicht mehr funktionierte, wurden Dateien geringerer Größe im Test verschickt.

### **4.1. Die Nachrichtenlaufzeiten**

Die Laufzeit der Übertragung reiner Textnachrichten etwa „Guten Tag!“ dauerte von einem Emulator zu einem anderen bei lokaler Relay-Verbindung im Schnitt 3 Sekunden. Lief die Übermittlung über den entfernten Relay, waren die Laufzeiten signifikant höher. So dauerte es mitunter bis zu 13 Sekunden bis an eine Unicast-Pipe gesendete Nachricht übertragen wurde. Textsendungen an die Propagate-Pipe kamen interessanterweise schneller bei den Peers an. Hier dauerte die Übertragung im Schnitt 5 Sekunden. Dateiübermittlungen dauerten bei Dateigrößen von 3 kbyte und 7 kbyte durchschnittlich 7 Sekunden mit entferntem Relay und 3 Sekunden bei lokalen Relay-Verbindungen.

Das Versenden einer Textnachricht von einem realen Handy zu einem anderen nahm mit bis zu 25 Sekunden zusätzlich Zeit in Anspruch, wohl aufgrund von Latenzen innerhalb des Mobilfunknetzes. Laufzeitunterschiede durch Verwendung der beiden Pipe-Arten waren nicht erkennbar. Dateiübertragungen dauerten sogar bis zu 45 Sekunden.

### **4.2. Die Zuverlässigkeit der Übertragung**

Unter Verwendung lokaler Relay-Verbindungen trafen im Test alle Nachrichten bzw. Dateien bei der Zielinstanz des Programms ein. Anders sah dies bei entfernten Relay-Verbindungen im Emulatortest aus. Hier kam es teilweise zu Nachrichtenverlusten. Glücklicherweise waren diese Ausfälle sehr selten. Bei den Programmtests mit den Mobiltelefonen traten Verluste häufiger auf. Bei Laufzeiten von 25 Sekunden, bis eine Textnachricht eintrifft, braucht es also schon eine Portion Geduld, um eine sicher davon ausgehen zu können, dass die Übertragung fehlschlug.

## 5. Abschlussbetrachtungen

Das in dieser Studienjahresarbeit entstandene Programm „JXMEShare“ sollte zeigen, dass es mittels JXME möglich ist, mobile P2P-Anwendungen zu erstellen. Eine Reihe von Verbesserungsmöglichkeiten, sowohl in diesem Programm, als auch in den zu Verfügung stehenden JXME-Klassen sollen nun erläutert werden.

Das Programm „JXMEShare“ demonstriert grundlegende Fähigkeiten der mobilen JXTA-Implementierung, wie das Verbinden mit einem Relay, die Erstellung von Peers und Pipes, den Versand von Textnachrichten und die Übertragung von Dateien. Um es nicht zu überladen, wurde darauf verzichtet, JXTA-Peermanagement zu integrieren, was eine aufwändige Verwaltung und persistente Speicherung des Status der JXTA-Elemente PeerID und Unicast-PipeIDs der an der Kommunikation beteiligten Peers mit Hilfe des „Record Management System“ (RMS) im MIDP nach sich ziehen würde, um etwa einen Peer nach Beendigung und Neustart des Programms erneut kontaktieren zu können, weil man entsprechende User-Daten in einer früheren Sitzung gesichert hat.

Das gleiche gilt für die anderen programmrelevanten Daten, wie Relay-IP, Portnummer, Peername und URL für die zu ladende Datei. Dies könnte in einer Erweiterung implementiert werden. Ebenso wäre denkbar, die Beschränkung der maximal per JXME versendbaren Dateigröße von 8 kbyte durch sukzessives Einlesen einer Datei in einen Puffer, welcher iterativ an den Ziel-Peer geschickt wird, aufzuheben. Dieser müsste dann die eingehenden Pakete in einer Datei zusammenfügen und persistent speichern. Momentan kann der Benutzer dieses Programms alle anderen mit dem JXTA-Netzwerk verbundenen Instanzen sehen. Die Verwaltung einer speicherbaren Buddy-Liste, bekannt aus Chat-Systemen wie ICQ, wäre ebenfalls eine nützliche Funktion. Realisierbar wäre sie über JXTA-Gruppenmanagementfunktionen. Denkbar ist natürlich auch die Haltung einer „Blacklist“ zur Filterung der Nachrichten eines bestimmten Peers. Sehr hilfreich wäre ebenso eine Anzeige des in einer Sitzung verbrauchten Datenvolumens, was momentan bei den hohen Preisen für mobile Datenkommunikation eine nützliche Information darstellt. In diesem Kontext ist natürlich auch der Nachteil des HTTP-Pollings an den Relay zu sehen. Diese kontinuierlichen Requests des Peers nach neuen Nachrichten an den Relay verursachen vermeidbaren Traffic durch Metainformationen, obwohl der Relay in den meisten Fällen keine neuen Daten für den JXME-Peer bereithält. Dies ist jedoch einzig auf die JXME-Implementierung zurückzuführen, die basierend auf der MIDP V1.0 Spezifikation nur HTTP-Kommunikation unterstützt. Ändern soll sich dies in einer neuen Version [7] von JXME für MIDP V2.0. Ihr Ziel ist es, mobile P2P-Kommunikation ohne den Gebrauch eines zusätzlichen Proxy zu realisieren, indem die Funktionalität des Proxy in die J2ME-Anwendung selbst integriert wird. Dies soll durch die Implementierung der JXTA-Protokolle „Peer Discovery Protocol“, „Peer Resolver Protocol“, „Endpoint (Routing) Protocol“ und „Pipe Binding Protocol“ ermöglicht werden. Ein Peer, der als Proxy, Relay und Rendezvous agieren müsste, entfällt in diesem Szenario ganz. Kompatibilität mit JXME für MIDP V1.0 soll ebenfalls gegeben sein. Somit wird ein mobiler Peer in Zukunft in der Lage sein, als vollwertiger JXTA-Peer zu gelten. Falls nötig, wird solch ein Peer selbst die Suche nach anderen Peers mit leistungsfähigerer Hardware starten und dessen Dienste in Anspruch nehmen können. Auf diese Weise wären mobile P2P-Anwendungen wie Filesharing ohne Netzabhängigkeiten sofort nutzbar und würden sicher eine breite Schar von Anwendern auf der ganzen Welt finden.

## Abbildungsverzeichnis

Abbildung 1	Java-Editionen.....	6
Abbildung 2	JXME-Peers und Relays im JXTA-Netzwerk.....	9
Abbildung 3	Oberfläche des JWTK.....	11
Abbildung 4	Main-Screen .....	13
Abbildung 5	Settings-Screen.....	13
Abbildung 6	"Write Msg"-Screen .....	14
Abbildung 7	"Online Contacts"-Screen .....	14
Abbildung 8	"Load File"-Screen.....	15
Abbildung 9	"Send File"-Screen .....	15

## Literaturverzeichnis

- 1 Akhil Arora, Carl Haywood, Kuldip Singh Pabla; "JXTA for J2ME – Extending the Reach of Wireless With JXTA Technology"; 2002; <http://jxme.jxta.org>
- 2 Kim Topley, „J2ME in a Nutshell“ (Kap. 1 und 2), März 2002
- 3 Sun J2ME, <http://java.sun.com/j2me/>,
- 4 Bill Day, "Developing Wireless Applications using MIDP 2.0, WMA, and MMA", 2003, <http://billday.com>
- 5 Bilal Siddiqui, "JXTA for Wireless Programmers", 2002, <http://www.developer.com/java/j2me/article.php/1501461>
- 6 Michael J. Yuan, „ Mobile P2P messaging, Part 2: Develop mobile extensions to generic P2P networks“, 2003, <http://www-106.ibm.com/developerworks/java/library/wi-p2pmsg2/>
- 7 Siemens Wireless Toolkit, 2004, <http://www.siemens-mobile.com/developer/>
- 8 Proxyless JXME, 2003, <http://jxme.jxta.org/plan2.html>, <http://jxme.jxta.org/planproxyless.html>