Ausarbeitung zum Masterseminar WS 08/09

# J2ME, Android und iPhone SDK im Vergleich


**Moritz Prinz**
Fachbereich Elektrotechnik und Informatik
Studiengang Informationstechnik

11. Januar 2009

# Inhaltsverzeichnis

# 1 Einführung

## 1.1 Motivation und Ziele der Arbeit

Die Vielfalt an Betriebssystemen und Applikationsumgebungen auf verschiedenen mobilen Endgeräten nimmt ständig zu. Für Softwareentwickler wird es immer schwieriger festzustellen, welche Plattform die am besten geeignete ist und ob es überhaupt möglich ist, geplante Funktionen auf der gewünschten Plattform umzusetzen.
Diese Arbeit gibt als Schwerpunkt einen Einblick in die drei Umgebungen J2ME, Android und das iPhone SDK und vergleicht diese grundlegend miteinander. Daneben werden auch noch weitere Umgebungen, wie zum Beispiel das LiMo Projekt oder OpenMoko, kurz angesprochen.

## 1.2 Aufbau der Arbeit

Im ersten Teil der Arbeit sollen die drei Umgebungen vorgestellt werden.
Dabei wird erläutert woher sie kommen, welche Hersteller an ihnen beteiligt sind und wie weit sie verbreitet sind. Außerdem werden in einem weiteren Abschnitt noch weitere Umgebungen kurz vorgestellt, die in Zukunft auch eine große Rollen spielen könnten.
Als nächster Schritt werden die Voraussetzungen für eine erfolgreiche Entwicklung beschrieben. Dazu gehören die verschiedenen Entwicklungs- und Testumgebungen ebenso wie die Möglichkeit, die eigene Software anschließend verteilen zu können.
Der letzte Teil der Arbeit beschäftigt sich mit der Frage, inwieweit es möglich ist Applikationen zu entwickeln, die auf allen drei Plattformen laufen. Als Alternative dazu soll die Frage beantwortet werden, ob es aktuell möglich ist mit Hilfe von Migrationswerkzeugen eine Applikation von der einen auf die andere Plattform zu portieren.

# 2 Die Plattformen

## 2.1 J2ME

Die *Java Platform Micro Edition* (Java ME) ist eine Umsetzung der Programmiersprache Java für so genannte *embedded consumer products* wie etwa Mobiltelefone oder PDAs. Definiert wird sie in den JSR 30 [2] und JSR 37 [3]. Ursprünglich designed wurde Java ME von Sun Microsystems [1], seit Dezember 2006 steht der J2ME Quelltext allerdings unter der GNU GPL [4]. Die aktuelle Version von Java ME basiert auf der Java Runtime Enviroment 1.3, was eine deutliche Einschränkung der Programmiermöglichkeiten darstellt.

Die Konfigurationen stellen verschiedene Bibliotheken und eine virtuelle Maschine zur

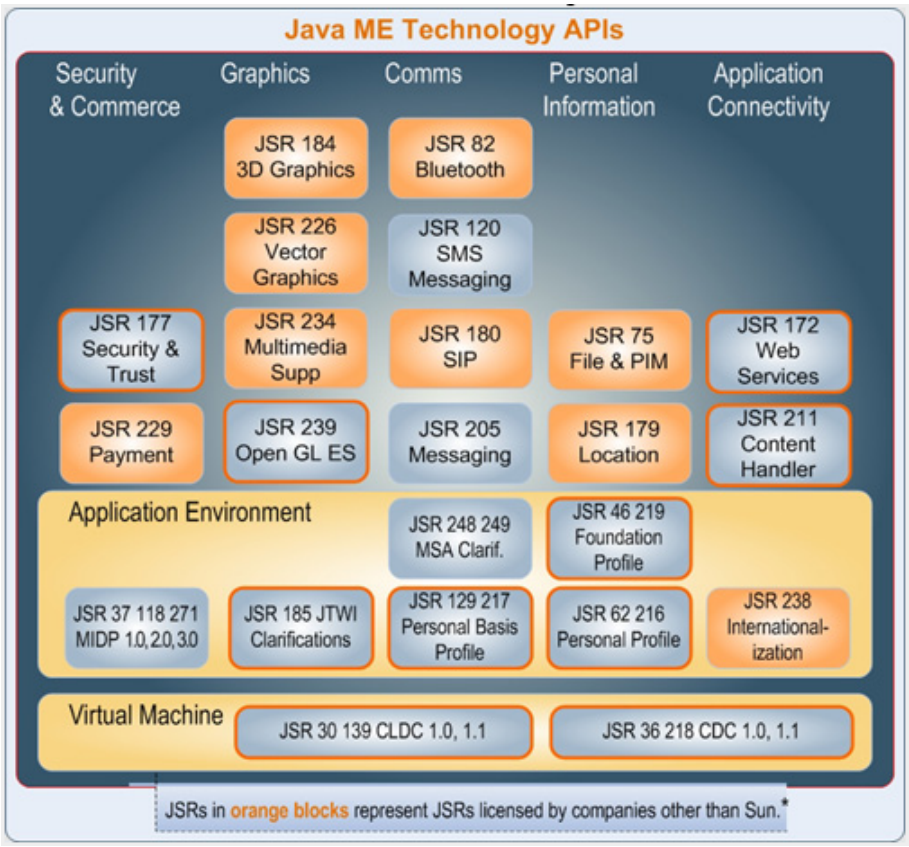


Abbildung 2.1: Architektur Java ME inklusive einiger APIs

Verfügung. Die wichtigste Konfiguration ist dabei die *Connected Limited Device Configuration* (CLDC) [5], da fast jedes aktuelle Mobiltelefon diese Konfiguration zusammen mit dem *Mobile Information Device Profile* (MIDP) [6] nutzt. Das CLDC liegt derzeit in der Version 1.1 (JSR 139 [7]) und das MIDP in der Version 2.0 (JSR 118 [8]) vor, wobei für letzteres die Version 3.0 bereits in JSR 271 [9] zur Verfügung steht.
Wie bereits erwähnt nutzt fast jedes aktuelle Mobiltelefon die CLDC in Verbindung mit dem MIDP und unterstützt somit Java ME. Auch die meisten älteren Telefone tun dies, wenn auch nicht zwingend CLDC 1.1 und MIDP 2.0. Zwei Unterstützungstabellen der gängigsten Mobiltelefone sind im Anhang zu finden [10], [11].
Ende 2007 hat Sun Microsystems angekündigt, Java ME zu Gunsten der Java Standard Edition nicht mehr weiterzuentwicklen. Der Hauptgrund sei, dass die Geräte für die Java ME entwickelt wurde, immer leistungsfähiger werden und somit Java SE genügend Ressourcen zur Verfügung stehen würden. Die Umstellung soll sich über die nächsten Jahre hinziehen. Damit dürfte sichergestellt sein, dass auch die nächsten Generationen mobiler Endgeräte Java ME unterstützen werden.

## 2.2 Android

*Android* [12] ist ein von der *Open Handset Alliance* [13] 2007 ins Leben gerufenes Projekt, das von Google angeführt wird. Der Allianz gehören bereits jetzt viele namenhafte Hersteller an, als Beispiele seien hier folgende genannt: Google, T-Mobile, HTC, Vodafone, ARM, Broadcom, Intel, NVidia, Qualcomm, Texas Instruments, LG, Motorola, Samsung, Sony Ericsson, eBay, Garmin. Anhand dieser Liste lässt sich das Potential von Android bereits erahnen.
Im Gegensatz zu Java Me ist Android allerdings nicht nur eine Programmiersprache, sondern eine komplette mobile Plattform:

- Das zugrunde liegende Betriebssystem basiert auf einem 2.6 Linuxkernel, der für Speicherverwaltung, Prozessverwaltung und Netzwerkkommunikation zuständig ist. Außerdem bildet er die Hardwareabstraktionsschicht für den Rest der Software und stellt die Gerätetreiber für das System bereit.

- Weitere wichtige Bausteine sind die auf der von Sun Microsystems entwickelten Java-Technik basierende virtuelle Maschine Dalvik und die dazugehörigen Android-Java-Klassenbibliotheken.
  Das Besondere an Dalvik ist, dass sie, im Gegensatz zu der normalen virtuellen Maschine von Java, eine Registermaschine darstellt. Dies ermöglicht es, Programme auf aktuellen Plattformen schneller auszuführen als es mit Stapelmaschinen möglich ist.

Diese Software steht in Teilen bereits unter der Apache-Lizenz 2.0 [14], der GPL und der LGPL. Außerdem wurde angekündigt, dass auch weitere Teile unter freie Lizenzen gestellt werden.
Neben der Software wird von der Open Handset Alliance auch die entsprechende Hardware entwickelt und vertrieben. Bisher gibt es nur ein Gerät auf dem Markt, auf dem

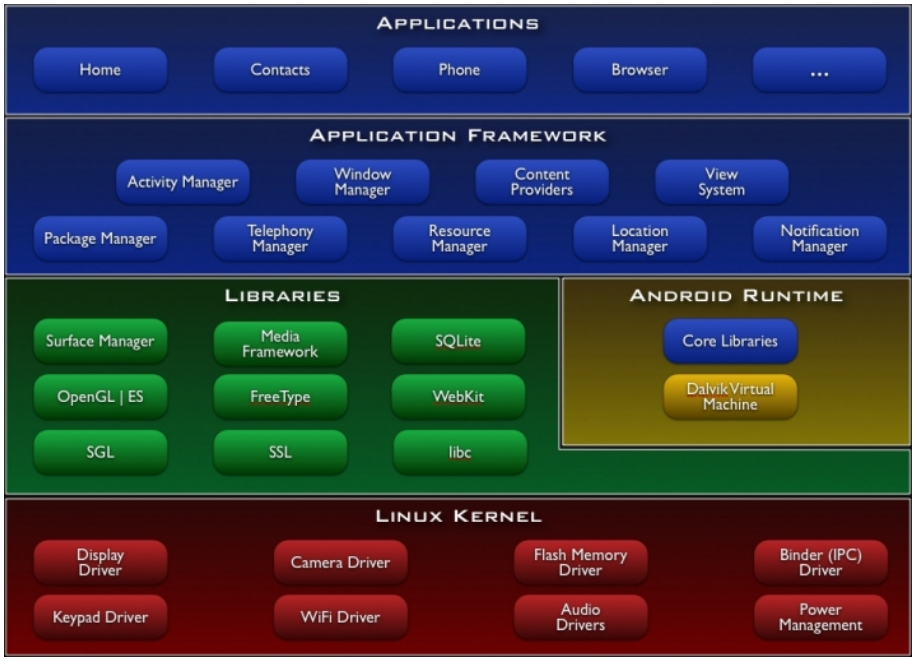


Abbildung 2.2: Architektur von Android

Android ab Werk läuft: das T-Mobile G1 (Entwicklungsname: *HTC Dream*). Allerdings sind bereits weitere Geräte, wie zum Beispiel das *Kogan Agora* angekündigt worden. Auch das Openmoko Projekt [20] arbeitet daran, Android auf seinen Endgeräten einzusetzen. Der derzeitige Stand ist, dass Android kompiliert, aber nicht alle Funktionen zur Verfügung stehen. Die unterstützten Geräte sind das *Neo 1973* und das *Neo FreeRunner*. Neben diesen ist es einigen Usern auch gelungen, Android auf Geräten anderer Hersteller nutzen zu können, so zum Beispiel auf dem *Nokia N810*, dem *HTC Vogue* und dem *Motorola A1200 Ming*.
Aufgrund der an der Allianz beteiligten Unternehmen ist davon auszugehen, dass es in Zukunft einige weitere Mobiltelefone mit Android als Betriebssystem geben wird. Auch die Tatsache, dass Teile des Quellcodes offengelegt werden müssen und dies auch bereits angekündigt wurde, verspricht, dass Android in Zukunft eine große Rolle für mobile Endgeräte spielen wird.

## 2.3 iPhone SDK

Das *iPhone* [16] ist ein von *Apple* [15] entwickeltes Smartphone, das als Betriebssystem ein angepasstes MacOS einsetzt. Um für das iPhone Software zu programmieren benötigt man das iPhone SDK, das von Apple frei zur Verfügung gestellt wird. Das SDK steht unter keiner freien Lizenz und läuft ausschließlich auf einem Mac mit x86-Prozessor und mindestens MacOS X 10.5.3.
Anfangs war es offiziell nicht möglich Anwendungen für das iPhone zu entwickeln, die

nativ, also direkt auf Ressourcen des Geräts zugreifend, auf dem Gerät laufen. Apple sah als Programmieroberfläche für Drittanbieter lediglich den Browser des iPhones, *Safari*, vor und verwies auf dessen Unterstützung von Standards wie *Ajax*. Safari stellt eine Schnittstelle zu nativ laufenden Anwendungen, wie Google-Maps oder Telefonfunktionen zur Verfügung.
Nachdem im Juli 2007 ein Hack publiziert wurde, wie trotz der eingebauten Hürden, Anwendungen nativ auf dem Gerät ausgeführt werden können, hat Apple kurze Zeit später die eigene Politik geändert und unterstützt seitdem auch offiziell nativ in *Objective-C* [28] entwickelte Anwendungen.
Nach Angaben der Telekom wurden bis Anfang November 2008 etwa 450.000 iPhones in Deutschland verkauft [18]. Insgesamt lassen sich im Internet Zahlen von bis zu 4,5 Millionen verkaufter iPhones weltweit finden.

## 2.4 Weitere Alternativen

Neben diesen drei Plattformen gibt es noch weitere, von denen einige sehr vielversprechend sind. Zwei davon sind für die Hersteller aus verschiedenen Gründen äußerst interessant und werden deshalb hier kurz angesprochen.

### 2.4.1 LiMo

Wie Android ist auch die *Linux Mobile Foundation* (LiMo) [19] eine Allianz mehrere großer Hersteller: Motorola, NEC, Panasonic, Samsung, Vodafone, LG Electronics, Texas Instruments, Verizon Wireless, AMD, ARM, Broadcom, Ericsson, Mozilla Cooperation, SAGEM, Qt Technologies (gehört jetzt zu Nokia). Das LiMo Projekt versucht, ähnlich wie das Android Projekt, eine Linux-basierte Plattform für mobile Endgeräte zu schaffen. Das Projekt ist zwar noch sehr jung, aber einige Geräte, aktuell 18, sind bereits auf dem Markt.
Für Hersteller ist dieses Projekt besonders interessant, da es einen großen Pool mit lizenz- und patentkostenfreiem Code aufbaut, aus dem sich jedes Mitglied der Allianz kostenfrei bedienen kann. Darin enthalten sind sowohl Treiber für verschiedene Hardwarekomponenten als auch Software für das Userinterface. Für ein Mitglied der Allianz ist es damit möglich, „mit wenigen Klicks“ ein auslieferungsfähiges Betriebssystem inklusive Userinterface zusammenzustellen. Dies ersparrt, sofern der Pool an Firm- und Software ausreichend groß ist, so gut wie jede weitere Entwicklungsarbeit.
Ein SDK für Entwickler, die der Allianz nicht angehören, gibt es bisher nicht, soll aber bald erscheinen. Für die Entwicklung kann dann auf die LiMo-API zurückgegriffen werden.

### 2.4.2 Openmoko

Ein weiteres Projekt ist *Openmoko* [20], das sich völlige Offenheit auf die Fahnen geschrieben hat. Dabei handelt es sich nicht nur um die Offenheit der Software, sondern auch der Hardware. Die komplette Firmware steht unter der GPL und es wurden auch

schon Teile der Konstruktionspläne der Hardware veröffentlicht. Die noch fehlenden sollen folgen, sobald ein entsprechend ausgereifter Status erreicht ist.
Die große Stärke dieses Projekts ist, dass es sich nicht darauf beschränkt, den Entwicklern eine virtuelle Maschine oder ein umgebautes Linux zur Verfügung zu stellen, sondern es nutzt ein komplettes Linux als Grundsystem inklusive Paket-Management. Das macht es Entwickler und Benutzer möglich, das System zu administrieren und auf die eigenen Bedürfnisse anzupassen. Die sich bietenden Möglichkeiten sind damit nahezu genauso vielfältig wie auf einem herkömmlichen Computer.
Aktuell ist es möglich, für Openmoko in C, Java und Mono, einer .NET Portierung für Linux, zu programmieren.

# 3 Entwicklung für die Plattformen

## 3.1 J2ME

### 3.1.1 Entwicklung

Für Java ME gibt es eine Vielzahl an Entwicklungsumgebungen. Die bekanntesten beiden dürften *netbeans* [21] und *eclipse* [22] sein. Beides sind Frameworks, mit deren Hilfe sich auch alle sonstigen Java Varianten einfach programmieren lassen. Letztere ist sehr weit verbreitet, wird von mehreren großen Unternehmen gefördert und steht für nahezu jedes Betriebssystem zur Verfügung.
Um mit eclipse Java ME programmieren zu können, benötigt man das Sun Java SDK[1] [23], das Sun Java Wireless Toolkit [24] und das eclipse-Plugin *EclipseME* [25]. Das WTK[2] ist erforderlich, um die zur Programmierung nötigen APIs zur Verfügung zu stellen und EclipseME um eclipse mit den nötigen Funktionalitäten auszustatten. Neben diversen APIs bringt das Wireless Toolkit auch einen Emulator mit, mit dessen Hilfe man programmierte Anwendungen auch ohne ein mobiles Endgerät testen kann.
Der Emulator unterstützt neben dem einfachen Ausführen der Applikation auch die Möglichkeit, externe Events an die Anwendung zu senden, wie zum Beispiel GPS-Koordinaten.

### 3.1.2 APIs

Sun stellt für Java ME eine Vielzahl an Spezifikationen zur Verfügung, die die verschiedensten Funktionen verfügbar machen sollen. Die wichtigsten dabei sind folgende:

- JSR 75: File Connection and PIM[3]
- JSR 82: Bluetooth
- JSR 120: Wireless Messaging API (WMA)
- JSR 135: Mobile Media API (MMAPI)
  Stellt Audio, Video und weitere Multimedia Funktionen zur Verfügung
- JSR 172: Web Services
- JSR 177: Security and Trust Services

[1] Source Development Kit
[2] Wireless Toolkit
[3] Personal Information Management

- JSR 179: Location API
  Programmierschnittstelle zum internen GPS-Empfänger
- JSR 180: Session Initiation Protocol API (SIP)
  Dient zum Aufbau von Multimediakommunikationssessions in IP Netzen
- JSR 184: Mobile 3D Graphics
- JSR 185: Java Technology for the Wireless Industry (JTWI)
- JSR 205: Wireless Messaging 2.0 (WMA)
- JSR 211: Content Handler API
- JSR 226: SVG 1.0
- JSR 229: Payment API
- JSR 234: Advanced Multimedia Supplements (AMMS)
  MMAPI (JSR 135) Erweiterungen
- JSR 238: Mobile Internationalization API
- JSR 239: Java Bindings for the OpenGL ES API
- JSR 248: Mobile Service Architecture
- JSR 256: Mobile Sensor API
- JSR 287: SVG 2.0

### 3.1.3 Deployment

Java ME Applikationen lassen sich von jedem vertreiben. Es gibt mehrere Möglichkeiten die Applikationen auf die mobilen Endgeräte zu laden. Die einfachste und kostengünstigste ist der Download der *jar-Datei* auf den PC und der anschließende Upload auf das mobile Endgerät, zum Beispiel über eine USB- oder Bluetooth-Verbindung.
Eine Alternative dazu ist die Bereitstellung einer *jad-Datei* und einer *jar-Datei* auf einem Webserver. Der Nutzer kann dann die jad-Datei per GPRS auf sein Gerät laden, das dann automatisch die Applikation, also die jar-Datei, nachlädt und installiert. Dies ist in den meisten Fällen allerdings nicht umsonst, da die Datenverbindung kostenpflichtig ist.

## 3.2 Android

### 3.2.1 Entwicklung

Um auf und für Android entwickeln zu können benötigt man das Android SDK, das kostenlos von der Open Handset Alliance zur Verfügung gestellt wird [26]. Es ist ein eclipse Plugin und greift somit, wie auch Java ME, auf ein weitentwickeltes Framework zurück.

Allerdings ist es mit den mitgelieferten Tools auch möglich, den Quelltext in einem beliebigen Editor zu schreiben und anschließend per Kommandozeile zu kompilieren.
Zum Test der Anwendungen ist ein Emulator enthalten, so dass Applikationen auch ohne entsprechendes Endgerät ausprobiert werden können; außerdem sind im SDK verschiedene APIs der Mitglieder der Allianz enthalten.
Der Quelltext für Android-Applikationen wird ausschließlich in Java geschrieben. Dieser wird mit einem normalen Java-Compiler übersetzt und dann von einem Cross-Assembler für die Dalvik VM angepasst. Für geschwindigkeitskritische Bereiche stehen zahlreiche, in C und C++ geschriebene, native Bibliotheken zur Verfügung, auf die die Anwendungen zugreifen können. Um diese nutzen zu können, müssen sie mit der Funktion *System.loadLibrary* geladen werden.
Da Android eine komplette Plattform inklusive Betriebssystem darstellt, bietet sich hier allerdings auch die Möglichkeit nativen Code auszuführen. Mit Hilfe des ADB-Debuggers wird eine root-Shell zur Verfügung gestellt, die es ermöglicht, ARM-Code auf das System zu laden und auszuführen. Allerdings ist das Ausführen von nativem Code kompliziert, da Android nicht die Standard C Bibliotheken verwendet, sondern eine als *Bionic* bekannte Bibliothek, welche von der *libc* von BSD abgeleitet ist.
Mit dem angesprochenen Debugger, der auch als Hintergrundprozess auf den aktuell vorhandenen Geräten läuft, bietet Android eine gute Variante, Applikationen oder auch das komplette Betriebssystem zu überwachen und somit Fehler einfacher zu finden und entsprechend zu reagieren.
Zur Zeit steht die Entwicklungsumgebung für Linux (i386), Windows und MacOS (Intel) zur Verfügung. Damit stellt sie kaum Ansprüche an die Vorlieben des Entwicklers.
Trotz dieser anscheinend vollständigen Entwicklungsmöglichkeiten gibt es immer wieder Beschwerden seitens der Entwickler, dass die Umgebungen nicht richtig funktionieren, falsch oder gar nicht dokumentiert sind und somit für den produktiven Einsatz nicht geeignet seien. Dazu sei angemerkt, dass Android als sehr junges Projekt sein volles Potential bei weitem noch nicht ausgeschöpft hat.

### 3.2.2 APIs

Als Beispiel für einige wichtige APIs von Android seien folgende genannt:

- System C library:
  Die System C Bibliothek ist von der BSD Implementierung der *libc* abgeleitet und für eingebettete Linux-Geräte optimiert.

- Media Libraries:
  Diese APIs stellen verschiedene Bild-, Video- und Audiofunktionen zur Verfügung, wie zum Beispiel MPEG4, H.264, MP3, AAC, AMR, JPG, und PNG.

- Surface Manager:
  Mit dieser API wird ein nahtloser Zugriff zwischen dem Displaysystem und den 2D und 3D Grafikengines realisiert.

- LibWebCore:
  Die LibWebCore API ist eine moderne Webbrowserengine.
- Skia Graphics Library (SGL):
  SGL ist die zugrundeliegende 2D Grafikengine.
- 3D libraries:
  Die 3D libraries stellen eine Implementierung der OpenGL ES 1.0 APIs dar. Mit ihnen ist es möglich 3D-Beschleunigung sowohl in Hardware als auch in Software zu realisieren.
- FreeType:
  Diese Bibliothek ist zum Rendern von Schriften vorhanden.
- SQLite:
  SQLite stellt eine leichtgewichtige, relationale Datenbankengine zur Verfügung.
- Location-Based Services (LBS):
  Diese Bibliothek stellt, wie die LocationAPI von Java ME, Funktionen zur Standortbestimmung zur Verfügung.

### 3.2.3 Deployment

Die Verbreitung der eigenen Software für Android ist derzeit über zwei Wege möglich:

- Zum einen kann man Anwendungen selbst zum Download anbieten, die der User dann über seinen Computer auf das Gerät übertragen muss.
- Die Alternative dazu ist dem *Android-Market* beizutreten. Dies ist allerdings kostenpflichtig. Dafür bietet die Open Handset Alliance allerdings eine Plattform, auf der man seine Applikation anbieten und somit ein weitaus größeres Publikum als mit der eigenen Website erreichen kann; die Plattform ist von jedem Android Gerät aus direkt erreichbar und der Download der Software ist kostenfrei. Es besteht allerdings die Möglichkeit seitens der Anbieter, Anwendungen kostenpflichtig anzubieten.

## 3.3 iPhone

### 3.3.1 Entwicklung

Für die Entwicklung von iPhone Software ist, wie bereits in der Beschreibung der Plattformen angesprochen, ein Mac mit einem x86-Prozessor und Mac OS X 10.5.3 notwendig. Für diesen kann dann, nachdem sich kostenlos als Entwickler auf der Apple Website [27] angemeldet wurde, die benötigte Software herunterladen: das Framework *Cocoa touch*. In dieses SDK-Paket hat Apple neben der integrierten Entwicklungsumgebung *Xcode* und dem *Interface Builder* auch Optimierungs- und Debugging-Werkzeuge eingebaut. Auch ein iPhone Simulator ist enthalten, so dass man zum Testen auf Hardware verzichten

kann. Allerdings unterstützt dieser Simulator nicht alle Funktionen die echte Hardware anbietet: Die Drei-Finger-Geste wird nicht unterstützt und auch das Rendering mit OpenGL ES sieht im Simulator unter Umständen ganz anders aus, als später auf einem echten Gerät. Auch der 3D Sound fehlt im Simulator komplett. Problematisch ist an dieser Stelle außerdem, dass der Simulator den Einsatz von Klassen akzeptiert, die zwar in Mac OS X vorhanden sind, aber nicht auf dem iPhone.
Als Alternative zu dem iPhone oder dem Emulator genügt für Anwendungen, die kein Mobilfunk, Mikrofon, Kamera oder GPS benötigen, auch ein *iPod touch*. Für die Standortbestimmung greift dieser auf WLAN-Informationen zurück. Dies ist zwar ungenauer und funktioniert nicht immer, für einen prinzipiellen Funktionstest sollte es allerdings ausreichen. Sowohl das iPhone als auch der iPod touch müssen mindestens mit der Firmware-Version 2.0 ausgestattet sein um entwicklungsfähig zu sein.
Zum Übertragen und Löschen der Daten kann der in Xcode integrierte Organizer verwendet werden. Mit ihm ist es möglich Daten auf das iPhone zu laden und gegebenenfalls durch die Applikation erstellte Dateien wieder zu löschen.
Für die eigentliche Entwicklung steht die Entwicklungsumgebung *Xcode* zur Verfügung. Alle gängigen Funktionen, die von anderen Entwicklungsumgebungen bekannt sind, wie zum Beispiel das Anbinden von Versionskontrollsystemen wie CVS oder subversion, Syntaxhighlighting oder automatische Vervollständigung, sind auch hier enthalten. Als Compiler und Debugger kommen *gcc* und *gdb* zum Einsatz, deren Ausgaben von Xcode interpretiert und optisch aufbereitet dem Entwickler ausgegeben werden.
Die Programmiersprache für Apple Geräte und somit auch das iPhone ist *Objective-C* [28], welche die Programmiersprache C um Sprachmittel zur objektorientierten Programmierung erweitert. Apple hält dafür eine ausführliche Dokumentation bereit, die es dem Entwickler leichter machen soll den Einstieg in Objective-C zu finden.

### 3.3.2 APIs

Generell stehen dem Entwickler alle Funktion des iPhones zur Verfügung und Apple bietet reichlich Dokumentation auf den Entwicklerseiten an. Allerdings gibt es kleine Einschränkungen:

- Offiziell ist es verboten APIs zu nutzen, die durch Reverse-Engineering gefunden wurden und Apple behält sich vor, nicht dokumentierte APIs ohne weitere Kommentare zu ändern.
- Außerdem ist es nicht möglich Hintergrundprozesse zu starten. Sobald eine Anwendung nicht mehr im Vordergrund ist, wird sie beendet. Dies ist Teil einer Strategie, die der Benutzerfreundlichkeit dienen soll.

### 3.3.3 Deployment

Zum Verbreiten der eigenen Software führt kein geeigneter Weg an dem *App Store* vorbei. In diesem kann man Software für jeden Nutzer zum Download zur Verfügung stellen

und auch selbständig einen Preis für die Anwendung festlegen. Sofern man eine kostenpflichtige Applikation anbietet, muss man allerdings 30% vom Preis als Provision an Apple zahlen. Ein Zugang zum App Store kostet $99 pro Jahr. Als Alternative kann eine Enterprise-Variante genutzt werden die wesentlich teurer ist, dafür aber das Verteilen von Software über eine eigene Website ermöglicht.
Desweiteren kann man die Software auch zum Download auf einer Website anbieten. Der Nutzer müsste sie dann mit Hilfe von Xcode auf das iPhone übertragen. Dafür benötigt er allerdings einen Mac mit Mac OS und den Sourcecode der Anwendung.
Anmerkung: ein großer Nachteil des App Stores ist, dass sich Apple vorbehält den Quelltext der Anwendung zu ändern.

# 4 Kompatibilität und Portabilität

## 4.1 J2ME

Sofern es für die Zielplattform eine Implementierung der Java virtuellen Maschine gibt und diese das CLDC und das MIDP unterstützt, sind Java ME Applikationen lauffähig. Wie auf [10] und [11] nachzulesen ist, unterstützen nahezu alle gängigen mobilen Endgeräte Java ME Anwendungen.
Für Android gibt es eine solche virtuelle Maschine aktuell noch nicht, allerdings gibt es Werkzeuge, die es möglich machen, Java ME Applikationen zu konvertieren. Ein Beispiel dafür ist der *MicroEmu* [29], der sich allerdings noch im Anfangsstadium befindet. Selbiger soll es bald auch möglich machen Java ME, Applikationen auf einem iPhone zu nutzen [30].

Viele Blogs und Webseiten bieten weitere Tricks und Tools zum Konvertieren von Java (ME) Anwendungen an, so dass sie anschließend auf dem iPhone laufen. Bisher gibt es keine „Java Virtuelle Maschine“ (JVM) für das iPhone, allerdings wird vermutet, dass Apple daran arbeitet. Sun Microsystems hat bereits publiziert, dass sie eine JVM für das iPhone implementiert und erfolgreich getestet haben. Allerdings darf diese aufgrund von Lizenzproblematiken nicht released werden. Aus diesem Grund unterstützt Sun Microsystems offiziell das *alcheMo* Projekt [31]
Bei Android bleibt abzuwarten, ob die Open Handset Alliance oder die entstehende Community die vorhandene virtuelle Maschine entsprechend erweitern oder eine eigene implementieren; meiner Meinung nach ist die Chance dafür aufgrund der Fülle von vorhandenen Java ME Anwendungen sehr groß.

## 4.2 Android

Da Android Applikationen in Java geschrieben werden, sind sie prinzipiell auch für andere Plattformen geeignet. Sie müssen nur durch einen anderen Java Compiler übersetzt werden und die Zielplattform muss die entsprechenden APIs zur Verfügung stellen. Portabel auf Geräte die nur Java ME unterstützen, sind die Applikationen aber nicht. Dafür müsste die Android Applikation in Java ME geschrieben sein, das sich allerdings stark von aktuellen Java Implementationen unterscheidet.
Eine Kompatibilität oder Portabilität zum iPhone ist auch hier nicht gegeben, da es für das iPhone bisher keine JVM gibt. Sofern man jedoch ein Tool findet, das Java Quelltext in Objective-C Quelltext übersetzen kann, sind Anwendungen portabel. Es gibt im Internet einige Anleitungen und Versuche, allerdings habe ich keine sicher funktionierende

Lösung gefunden, die diese Aufgabe übernimmt.

## 4.3 iPhone

Wie auch schon bei Java ME angesprochen, ist es derzeit nicht ohne Weiteres möglich, Anwendungen zu portieren. Um von Java (ME) zu Objective-C und somit Applikationen, die auf einem iPhone funktionieren, zu kommen, gibt es mehrere Konvertierungstools und Tricks, für den umgekehrten Weg gibt es aber offenbar bisher keine funktionierende Lösung. Aufgrund der geringen Verbreitung des iPhones ist es auch fraglich, ob es eine solche jemals geben wird. Deutlich wahrscheinlicher ist meiner Meinung nach, dass Apple eine JVM released.

# 5 Fazit

J2ME erreicht zwar mit Abstand am meisten Nutzer, ist allerdings nicht mehr zeitgemäß. Die Möglichkeiten der Benutzeroberfläche ohne weitere Frameworks entspricht bei weitem nicht mehr dem, was die Nutzer gewohnt sind und erfüllt die Ansprüche nicht mehr.

Auch mit verschiedenen Frameworks ist abzusehen, dass J2ME in der aktuellen Form mittelfristig vom Markt verschwinden wird und, dank der steigenden Rechenleistung der mobilen Endgräte, durch J2SE oder andere Weiterentwicklungen ersetzt wird. Eine grundlegende Kompabilität zu neuen Plattformen wie Android und LiMo ist ebenfalls schon vorhanden, was darauf schliessen lässt, dass der Übergangsprozess bereits begonnen hat.
Während meiner Recherchen habe ich immer wieder die Meinung gelesen, dass in Zukunft der Markt vor allem von diesen beiden, Android und LiMo, beherrscht werden wird. Obwohl es noch zahlreiche weitere Projekte gibt, die von verschiedenen Marktriesen gefördert werden, bin ich ebenfalls dieser Meinung. Auch wenn beide Projekte noch in den Kinderschuhen stecken, ist ihr Potential enorm groß, was nicht zuletzt an der Beteiligung der vielen großen und wichtigen Hersteller liegt.
Das iPhone hingegen wird häufig als Randerscheinung gesehen und ist auch nur wenig verbreitet: in Deutschland knapp 450.000 Geräte von mehr als 82 Millionen, weltweit spricht Apple von etwa 5 Millionen verkauften Geräten.
Die anderen Hersteller haben die Popularität der geschickten Kombination von verschiedenen Technologien ebenfalls lange erkannt und bieten mittlerweile reichlich Geräte mit ähnlichem Funktionsumfang an, besitzen allerdings deutlich mehr Marktmacht auf dem Sektor als Apple selbst. Auch die fehlende Kompabilität von Anwendungen für das iPhone zu den anderen Plattformen macht das iPhone, im Hinblick auf die Alternativen, nicht sehr attraktiv.

# Literaturverzeichnis


[1] "The Java ME Platform":
http://java.sun.com/javame/

[2] JSR 30:
http://www.jcp.org/en/jsr/detail?id=30

[3] JSR 37:
http://www.jcp.org/en/jsr/detail?id=37

[4] "GNU General Public LicenseÄrtikel bei Wikipedia:
http://en.wikipedia.org/wiki/GNU_General_Public_License

[5] CLDC Artikel bei Wikipedia:
http://en.wikipedia.org/wiki/CLDC

[6] MIDP Artikel bei Wikipedia:
http://en.wikipedia.org/wiki/MIDP

[7] JSR 139, CLDC 1.1:
http://www.jcp.org/en/jsr/detail?id=139

[8] JSR 118, MIDP 2.0:
http://www.jcp.org/en/jsr/detail?id=118

[9] JSR 271, MIDP 3.0:
http://www.jcp.org/en/jsr/detail?id=271

[10] Unterstützungstabelle von Sun:
http://developers.sun.com/mobility/device/device

[11] Unterstützungstabelle vom Java Club:
http://www.club-java.com/TastePhone/J2ME/MIDP_mobile.jsp

[12] Website des Android Projekts:
http://www.android.com

[13] Website der Open Handset Alliance:
http://www.openhandsetalliance.com

[14] Apache-Lizenz 2.0:
http://www.opensource.org/licenses/apache2.0.php

[15] Apple Website:
http://www.apple.com

[16] iPhone Artikel bei Wikipedia:
http://en.wikipedia.org/wiki/Iphone

[17] Website des Openmoko Projekts:
http://www.openmoko.com/

[18] "Bericht: Telekom verkaufte bislang über 450.000 iPhones":
http://www.heise.de/newsticker/meldung/118624

[19] Website der LiMo Foundation:
http://www.limofoundation.org/

[20] Website von Openmoko:
http://www.openmoko.com/

[21] Netbeans Website:
http://www.netbeans.org/

[22] eclipse Website:
http://www.eclipse.org

[23] Sun Java SDK:
http://java.sun.com/

[24] Sun Java Wireless Toolkit:
http://java.sun.com/javame/downloads/index.jsp

[25] eclipse ME Website:
http://eclipseme.org/

[26] Android Developer Website:
http://code.google.com/android/

[27] Apple Developer Website:
http://developer.apple.com/iphone/

[28] Objective-C Artikel bei Wikipedia:
http://en.wikipedia.org/wiki/Objective-C

[29] Website von MicroEmu:
http://microemu.blogspot.com/

[30] "MicroEmulator coming to the iPhone":
http://microemu.blogspot.com/2008/12/
microemulator-coming-to-iphone.html

[31] alcheMo Website:
http://www.innaworks.com/alcheMo.html