Die Android Plattform
Aktueller Stand und Ausblick

Markus Schlichting
Hochschule der Medien
Nobelstr. 10, 70569 Stuttgart
e-mail: markus.schlichting@hdm-stuttgart.de

Abstract

Android ist das neue Betriebssystem für Mobiltelefone und wird von einem durch Google angeführten Konsortium aus 33 namhaften Firmen aus den Bereichen der Mobilfunk-, Software- und Internetbranche entwickelt. Erste Geräte soll es im Herbst 2008 geben. Was sich hinter dem System und der Plattform verbirgt und welches Potenzial darin liegt, zeigt diese Arbeit.

1. Einleitung

1.1. Android

Android ist ein neues, durch die Open Handset Alliance (OHA) vorangetriebenes, Betriebssystem für mobile Endgeräte. Android basiert im Kern auf Linux, verwendet darüber hinaus jedoch kaum von anderen Linux-Varianten bekannte Komponenten. Als Programmiersprache kommt Java zum Einsatz - die ausführende virtuelle Maschine ist jedoch zum Java-Standard inkompatibel. Android geht so neue Wege und bietet das Potenzial, den nicht zuletzt durch das iPhone in Unruhe versetzten Mobilfunkmarkt weiter aufzuwirbeln - zumal die OHA angekündigt hat, Android zu weiten Teilen als Open Source unter der Apache 2 Lizenz freizugeben.

1.2. Zielsetzung

Diese Arbeit soll eine kurze Einführung in das Android System und die verfügbare Plattform bieten, den aktuellen Entwicklungsstand und die Konzepte beschreiben. Abschließend wird ein Ausblick auf die Möglichkeiten der Plattform und das mögliche Potenzial auf dem Mobilfunkmarkt gegeben.

1.3. Abgrenzung

Das Ziel dieser Arbeit ist dementsprechend weder Referenz noch Leitfaden für die Softwareentwicklung zu sein. Es wird im Fortlaufenden nicht darauf eingegangen, welche Schritte zur Installation des SDK[1] und dem Erstellen einer Applikation im Einzelnen notwendig sind. Hierfür sind z. B. die Android Dokumentation[1], der IBM developerWorks Artikel[7] und das im Oktober 2008 als Printausgabe[2] erscheinende Buch von Frank Ableson[6] zu empfehlen.

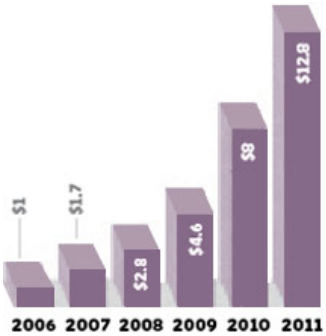

Abbildung 1: Bisheriges und erwartetes Wachstum des Marktes für Werbeanzeigen auf Mobiltelefonen, in Mrd. $ [5]

1.4. Motivation

Die Nutzung des Internets wird sich nach gängiger Meinung fundamental ändern: Nicht mehr der PC, sondern mobile Endgeräte, mit denen der Benutzer jederzeit und von jedem Ort aus auf das Web zugreifen kann, werden der zentrale Zugangspunkt zum Internet sein.

Vor diesem Hintergrund werden die auf den Geräten verfügbaren und installierten Softwarepakete zukünftig eine bedeutende Schlüsselrolle in der Informationstechnologie spielen.

Die Motivation für Google, das Startup Android zu kaufen und die Open Handset Alliance ins Leben zu rufen wird hier sehr deutlich. Noch plausibler wird es, wenn man das in Abbildung 1 gezeigte erwartete Wachstum für Werbung

[1] *Software Development Kit*

[2] Eine Vorabversion ist im PDF Format bereits auf der Website des Verlages verfügbar.

auf mobilen Geräten mit der Tatsache verknüpft, dass Google's primäre Einnahmequelle *Google Ads*, eine Plattform für Werbeanzeigen im Internet, ist.

Während die Zahl der als Internetzugang genutzten PCs derzeit bei etwa zwei Mrd. liegt, wird die Anzahl der Mobiltelefone auf etwa acht Mrd. geschätzt - Tendenz zugunsten der Mobiletelefone steigend. Trifft die eingangs erwähnte Einschätzung zu - der Erfolg des iPhone, das in der zweiten Version allein am ersten Wochenende in den USA über eine Million Mal verkauft wurde (vgl. [13]), scheint dies zu bestätigen - verliert der PC weiter an Bedeutung(vgl. [5]). Für Google Ads stellt dies eine große Herausforderung dar, da die bisherigen Anzeigen auf die klassischen PCs ausgerichtet sind. Eine bedeutende Rolle von Android und somit auch Google im Markt für Mobile Plattformen würde den weiteren Erfolg doch zumindest begünstgen.

Die Aufmerksamkeit für Android in den Medien und das dargestellte mögliche Szenario sowie die daraus resultierende große Bedeutung für den zukünftigen IT-Markt begründet die Motivation, mit dieser Arbeit Android genauer zu betrachten.

2. Marktüberblick

Die OHA wurde durch Google ins Leben gerufen, um Android, welches Google selbst mit dem Startupunternehmen Android Inc. 2005 gekauft hat, voranzutreiben. Sie umfasst zur Zeit 34 Mitglieder, die sowohl aus der Mobilfunk- (T-Mobile, Telecom Italia, HTC, Motorola, Samsung, LG Electronics, ...) als auch aus der Internet- und Softwarebranche (eBay, Intel, Nvidia, ...) stammen. Diese differenzierte Aufstellung der OHA ist notwendig, um Android direkt zu Beginn mit dem nötigen Gewicht an den Markt zu bringen und unterstreicht den Nachdruck, mit dem Google das neue System etablieren will.

Am Markt der verfügbaren Systeme hat Android mit Konkurrenz aus zwei Bereichen zu kämpfen: Zum einen mit den klassischen Betriebsystemen, die derzeit auf Handies und Smartphones[3] zum Einsatz kommen. Zum anderen tritt es gegen die auf den bisherigen Systemen vorhandenen (und zum Teil systemübergreifenden) Entwicklungsplattformen wie Java Micro Edition (JME), Flash Mobile oder die neu auf den Markt drängenden Silverlight von Microsoft und JavaFX von Sun an.

Wie in Abbildung 2 am Beispiel des amerikanischen Marktes zu sehen, wird der Markt für Smartphone-

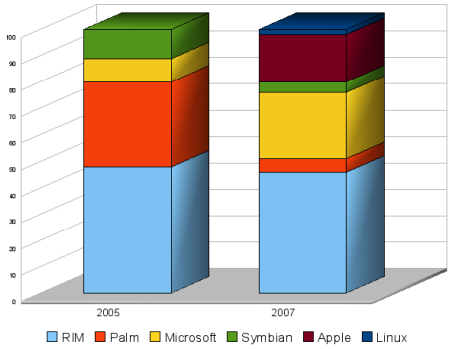

Abbildung 2: Smartphone OS, US Markt [5]

Betriebssysteme vor allem durch RIM[4], Microsoft und Apple beherrscht. Symbian spielt eine wesentlich bedeutendere Rolle, als die Darstellung vermuten lässt, da es überwiegend außerhalb der USA eingesetzt wird - am weltweiten Markt erreicht es einen Anteil von rund 70 Prozent (vgl. [9]).

Microsoft strengt sich mit Windows Mobile an, seine Position als Markführer von PC Systemen auch auf mobile Geräte zu übertragen. Durch eine hinreichend gute Integration in die Systemlandschaft der Unternehmen und Verbraucher scheint dies auch bisher zu gelingen. Vor dem derzeitigen Erfolg stand sich das System lange Zeit mit Kinderkrankheiten und einer hakeligen Bedienung selbst im Weg, auf dem Gesamtmarkt für Smartphones wird ihm ein Markanteil von etwa 9 Prozent zugerechnet (vgl. [9]) .

Mit dem iPhone hat es Apple auf Anhieb geschafft, einen bedeutenden Marktanteil zu erlangen und die bisherigen Spielregeln auf dem Mobilfunkmarkt zu verändern. Die mobile Variante von MacOS X feiert durch ein neuartiges Bedienkonzept viele Erfolge. Aber gerade auch beim Einfluss auf den Vertrieb, der Beteiligung an den Einnahmen aus den Mobilfunkverträgen und mit Exklusivverträgen für den Vertrieb hat Apple die Branche deutlich beeinflusst.

RIM hat sich mit dem BlackBerry System eine bemerkenswerte Position erarbeitet, wobei gerade die „bisher beste mobile E-Mail-Applikation" ([14]) für eine weite Verbreitung unter Geschäftskunden gesorgt hat. Ein Vorstoß in den Privatkundenmarkt soll weiteres Wachstum bringen. Eine alternative Strategie wurde bisher nicht vorgestellt, so dass aufholende Wettbewerber die Markposition von RIM in Frage stellen.

Symbian ist wie oben erwähnt mit seinem Symbian-OS derzeitiger Marktführer für Smartphonebetriebssysteme. Eine Vielzahl an verschiedenen Versionen auf Geräten unterschiedlicher Hersteller macht es für Entwickler jedoch verhältnismäßig schwierig, Software für die gesamte Masse

[3] Die Unterscheidung zwischen Handies, auch als *feature phones* bezeichnet, und Smartphones liegt im Funktionsumfang. Erste sind die *normalen* Mobiltelefone, die neben Telefon- über rudimentäre Adress- und Kalenderfunktionen verfügen und u.U. mit Erweiterungen wie einer Kamera oder MP3-Spielern versehen sind. Smartphones hingegen sind schon fast mit PCs in Taschengröße vergleichbar. Sie verfügen über ein ausgereiftes und umfassenderes Betriebsystem, ermöglichen prinzipiell beliebige Applikationen zu installieren und können in ihrem Funktionsumfang auch nachträglich noch deutlich erweitert werden.

[4] *Research in Motion* - Hersteller und Anbieter des BlackBerry Systems

an Symbian-Installationen zu entwickeln. Als „vorweggenommene Reaktion auf Android“ [9] hat Nokia inzwischen den vollständigen Kauf[5] der Symbian Inc. bekanntgegeben. Symbian soll aufgelöst und in eine nicht kommerzielle Stiftung überführt werden. Weitere an dieser Stiftung beteiligte Firmen wie NTT Docomo, Sony Ericsson und Motorola steuern bisher in Eigenregie entwickelte Komponenten für Symbian bei. Das System wird dann den an der Stiftung beteiligen Firmen kostenlos zur Verfügung gestellt. Mit diesem Schritt soll erreicht werden, was bisher nicht gelang: Das Erstellen einer einheitlichen Software-Plattform für Mobilgeräte. Darüber hinaus soll das System binnen zwei Jahren komplett unter der Eclipse-Lizenz als Open Source zur Verfügung stehen. So will man seinen Vorsprung gegenüber Android und anderen Plattformen, den man mit rund 200 Millionen ausgelieferten Geräten und rund vier Millionen Entwicklern hat, sichern (vgl. [12]).

Betrachtet man die erwähnten Entwicklungsplattformen, zeigt sich ebenfalls ein sehr uneinheitliches Bild. Die beiden Platzhirsche sind Flash Lite und JME. Flash Lite wird durch Adobe als Mobilversion des im Internet als Quasi-Standard für multimediale Inhalte geltenden Flash vertrieben. Laut Adobe unterstützen mehr als 450 Millionen Endgeräte Flash Lite[6] - es ist also vor allem auch auf herkömmlichen Mobiltelefonen verfügbar, nicht nur auf Smartphones. Verglichen mit den geschätzen 8 Mrd. Mobiltelefonen, die weltweit im Einsatz sind, erscheint diese Zahl niedrig. Betrachtet man jedoch nur einzelne Märkte wie Eurpa oder die USA kommt Flash Lite immerhin auf einen Marktanteil von rund 15 Prozent.(vgl. [17]) Vor diesem Hintergrund erscheint es doch verwunderlich, dass Apple für das iPhone bisher keine Flashunterstützung zugesagt hat und auch für Android bisher keine Unterstützung erwähnt wurde. Der kritische Punkt für Flash ist die Versionsvielfalt (fünf verschiedene Versionen), die auf den Geräten installiert sein können. [15]

Die Java Micro Edition erreicht vermutlich einen höheren Verbreitungsgrad[7] - kämpft dennoch mit ähnlichen Problemen. Was bei Flash die Versionskonflikte sind, sind für JME die möglichen Kombinationen aus Gerätekonfiguration und -profilen. Das Versprechen[8] der einheitlichen Plattform zur problemlosen Entwicklung von mobilen Anwendungen konnte die JME bisher nicht einlösen.

[5] Nokia war bisher bereits zu rund 48 Prozent an Symbian beteiligt.

[6] Eine Aufschlüsselung nach Version und Ländern findet sich bei Adobe: http://www.adobe.com/devnet/devices/pdfs/fl_version_by_country_ib_jan_2008.pdf

[7] Quellen hierzu sind leider kaum auszumachen,jedoch lässt die weite Unterstützung für JME seitens der Gerätehersteller, sowie der im Vergleich zu Flash Lite frühere Start darauf schliessen.

[8] Welches Sun mit der Vermarktung der Plattform gibt.

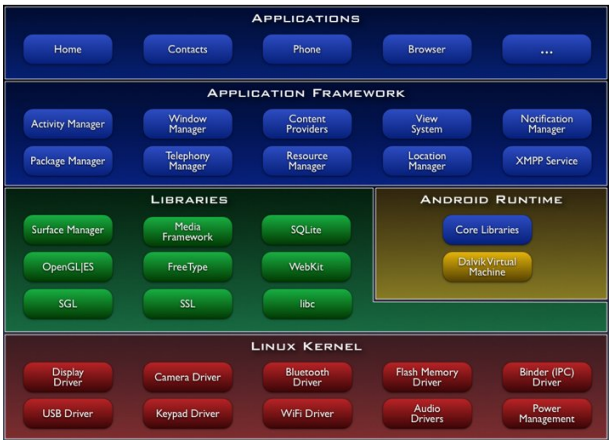

Abbildung 3: Android Architektur [1]

3. Plattform

3.1. Architektur

Abbildung 3 zeigt die grundsätzliche Architektur von Android.

Als Kern des Systems dient ein Linux-Kernel 2.6, der für alle Systemdienste zuständig ist. Zu diesen zählen Security-Modell, Speicher- und Prozessmanagement, Netzwerkfunktionen und - nicht zuletzt - das durch die weite Verbreitung von Linux erprobte Treibermodell.

Der Kernel wurde an einigen Stellen angepasst und erweitert, insbesonderne in der Speicherverwaltung, dem Powermanagement, bei Debug- und Protokollfunktionalitäten. Alle Änderungen am Kernel werden entsprechend der GPL[9] des Linux-Kernels verfügbar gemacht[10] und fließen teilweise auch wiederum in den Hauptentwicklungszweig von Linux ein.

Aufbauend auf Linux als Hardware-Abstraktionsschicht stehen die in Grün dargestellten Bibliotheken zur Verfügung, die - im Wesentlichen aus Performancegründen - in C und C++ implementiert sind. Interessant sind dabei die Bausteine für die Grafikverarbeitung, wo für 2D das im Rahmen von Android entwickelte SGL[11] und für 3D eine zum JSR[12] 239[13] kompatible Bibliothek für OpenGL|ES zur Verfügung steht.

Außerdem ist das, durch die an der OHA beteiligte Packetmedia Inc. beigesteuerte, Mediaframework[14], mit

[9] GNU Public Licence

[10] siehe http://git.android.com/

[11] *Scalable Graphics Library*, bisher nicht im Quelltext verfügbar und ohne Bezug zu anderen Bibliotheken, die dieselbe oder ähnliche Abkürzungen verwenden wie „SDL“ oder „3D SGL“

[12] *Java Specification Request*

[13] Das JSR definiert Java-Bindings für OpenGL|ES, vgl. http://java.sun.com/javame/reference/apis/jsr239/

[14] basierend auf der PaketVideo OpenCORE Plattform

dem alle gängigen Formate wie AAC, MP3, MPEG4, H264, etc. verarbeitet werden können, interessant. Selbiges gilt für SQLite, dass als internes Datenbanksystem eingesetzt wird, sowie für die WebKit-Bibliothek. Diese ist das Herz des Android Browsers und lässt - da identisch mit der auf dem iPhone eingesetzten Bibliothek - auf eine gute Kompatibilität zu iPhone-optimierten Seiten hoffen. Darüber hinaus ist die eingesetzte C-Bibliothek (*„libc“*) einen genaueren Blick wert.

Android greift hier nicht auf die weit verbreitete und in praktisch allen Linux-Systemen eingesetzte glibc[15] zurück, sondern implementiert eine eigene libc - *Bionic*, was im Wesentlichden durch die folgenden drei Punkte begründet ist(vgl. [8]):

Lizenz Android soll oberhalb der Kernelschicht frei von der GPL bleiben, um es Partnern und Nutzern zu ermöglichen, Eigenentwicklungen und Anpassungen nicht zwingend ebenfalls unter der GPL veröffentlichen zu müssen und als Closed-Source Software vertreiben zu können. Die Wahl für Bionic fiel auf die BSD-Lizenz, die deutlich weniger restriktiv ist. Andere Bestandteile von Android, die als Open Source freigegeben werden, stehen unter der Apache 2 Lizenz, die ebenfalls weniger restriktiv ist.

Größe Da nahezu jeder Prozess auf diese Bibliothek zurückgreift und sie somit immer mitgeladen wird, sollte sie sehr ressourcenschonend sein. Bionic benötigt derzeit etwa 200kB, was der Hälfte des durch die glibc benötigten Speichers entspricht. Um dies zu erreichen sind z. B. einige POSIX[16]-Funktionen weggefallen und somit die Kompatibilität zur glibc und anderen C-Bibliotheken nicht mehr gegeben.

Geschwindigkeit Durch den geringeren Speicherverbrauch und die Optimierungen auf mobile Geräte mit geringer CPU-Leistung (u. a. eine eigene, sehr schnelle Implementierung von *pthread*[17]) wird den deutlich höheren Performanceanforderungen Rechnung getragen.

Auf den Linux Kern und den Kernbibliotheken setzen die Schichten auf, die für die Entwicklung von Androidanwendungen unmittelbar von Bedeutung sind, da hier die Kontaktstellen des Entwicklers mit dem System liegen: Die Android Runtime, bestehend aus der Dalvik Virtual Machine, der Kernbibliothek (Core Libraries) und das Application Framework. Diese Komponenten werden in den folgenden Abschnitten genauer beschrieben.

[15] Die GNU Implementierung der C-Bibliothek

[16] „Portable Operating System Interface“ - Beschreibt eine Standard-API, die von dem überwiegendem Teil unixoider Betriebsysteme implementiert wird.

[17] *pthread* ist die geläufige Kurzform für *Native POSIX Thread Library*

3.2. Dalvik VM

Obwohl Android Java als zentrale Programmiersprache einsetzt und Anwendungen ausschliesslich mit dieser entwickelt werden, setzt es nicht die virtuelle Maschine (VM) für Java von Sun ein. Es wurde speziell für Android eine eigene VM mit dem Namen *DalvikVM* entwickelt.

Im Wesentlichen entstehen durch diese Eigenentwicklung zwei Unterschiede: Zum einen löst sich die OHA, da der durch die DalvikVM verarbeitete Bytecode zu Java-Bytecode inkompatibel ist, von Lizenzverpflichtungen gegenüber Sun und umgeht den Java Community Process. Zum anderen kann der durch die VM verarbeitete Bytecode schon beim Kompilieren auf geringen Speicherverbrauch und damit für den Einsatz auf mobilen Endgeräten optimiert werden. Darüber hinaus arbeitet die DalvikVM als Registermaschine und nicht wie bisherige VMs als Stapelverarbeitungsmaschine[2]. So ist sie näher auf die ARM Architektur, die in den meisten aktuellen mobilen Geräten zum Einsatz kommt, angepasst und benötigt weniger Verarbeitungsschritte um den Bytecode zur Laufzeit zu verarbeiten.

Die Architektur von Android erlaubt es mehrere Instanzen der VM gleichzeitig auszuführen. Jede Anwendung läuft in einer eigenen VM, was vor allem, wie sich später zeigen wird, für die Anwendungsentwicklung und Sicherheitsfunktionen wichtig ist.

Obwohl der Bytecode inkompatibel ist, lassen sich vorhandene Java .class/.jar-Dateien in das Dalvik-Executeable (.dex)-Format umwandeln und so bestehende Komponenten für Anwendungen auf Android weiterverwenden. Zu beachten ist hierbei, dass ausschliesslich Kompilate des Sun Java Compiler aus dem JDK 5 oder 6 weiterverarbeitet werden könnnen. Mit anderen Produkten, wie zum Beispiel dem GNU-Compiler, verweigert das Android SDK seinen Dienst (vgl. [16]).

Zusammen mit der DalvikVM steht in den Core Libraries die bekannte Java API zur Verfügung. Wichtig ist hierbei, dass es sich um die gesamte API handelt, nicht nur, wie bei JME, um eine Teilmenge. Folglich können für die Entwicklung von Android-Anwendungen unmittelbar alle Javaentwickler angesprochen werden, da keine neue Programmiersprache erlernt werden muss und auf die bekannten mächtigen und etablierten Entwicklungsumgebungen zurückgegriffen werden kann.

3.3. Android Application Framework

Das Android Application Framework baut auf die System- und Core-Libraries auf und läuft innerhalb der DalvikVM. Innerhalb des Frameworks finden sich die grundlegenden Plattformdienste die z.B. die Verwaltung des Anwendungs-Lebenszyklus, die Paket-, Ressourcen- oder die Fenster- und Oberflächenverwaltung übernehmen.

Alle durch das Framework zur Verfügung gestellten Funktionen sind durch jede auf dem System installierte Anwendung gleichberechtigt nutzbar. Dies bedeutet, das unter Android prinzipiell jede Anwendung mit den gleichen Berechtigungen ausgeführt wird und auf alle verfügbaren Schnittstellen zugreifen kann.[18] Außerdem wird auch ermöglicht, dass jede Anwendung austauschbar ist. Möchte ein Anbieter sich z. B. duch eine erweiterte Kontaktverwaltung differenzieren, ist es kein Problem, die standardmäßige Anwendung durch eine eigene auszutauschen.

Der Location Manager z. B. bietet eine einheitliche Schnittstelle für Positionsinformationen. Je nach Ausstattung des Gerätes und verfügbaren Diensten wird die aktuelle Position über GPS- oder WLAN Signal oder mit Hilfe des Mobilfunknetzes (z.B. anhand der CellID) ermittelt. Für die nutzende Anwendung ist dies jedoch völlig transparent, der Entwickler braucht sich nicht darum kümmern, woher die Information kommt, sondern kann sich vollkommen auf die Weiterverarbeitung konzentrieren.

Die angedeuteten Actvitiy und der Notification Manager sind ebenso wie die Content Provider und das View System Bestandteile des Konzepts von Android Anwendungen, das im nächsten Kapitel genauer beschrieben wird.

4. Entwicklung

Die Softwareentwicklung für Android erfolgt ausschliesslich mit Java. Dabei muss der Entwickler sich nicht, wie bei der JME, auf einen eingeschränkten Befehlssatz einrichten, sondern kann die bekannte Java API sowie die durch Android zur Verfügung gestellten Dienste nutzen.

Darüber hinaus bietet Android ein neues Konzept für die Entwicklung mobiler Anwendungen, das als einer der Schlüsselpunkte dafür gesehen wird, warum Android großes Potenzial zugetraut wird (vgl. [7], Seite 4, „Why is Android important?“).

Das große Potenzial des Frameworks liegt dabei in der Art mit der es das Verständnis von Webanwendungen auf mobile Anwendungen überträgt. Dabei wird nicht einfach ein guter Browser bereitgestellt und die Nutzung von JavaScript und der Zugriff auf serverseitige Ressourcen ermöglicht, sondern es wird als ein zentrales Konzept der Plattform umgesetzt und dadurch die Art der Benutzerführung wesentlich beeinflusst.

Wenn man das Web und die dort verfügbaren Anwendungen betrachtet, kann man schnell erkennen, dass die gesuchten Informationen im Prinzip immer nur einen Klick entfernt sind. Hinter jedem Klick steht ein URI[19], umgangssprachlich als Link bezeichnet, der den schnellen und unkomplizierten Zugriff auf die durch den Nutzer verlangten Informationen erlaubt. Die fast alltäglich benutzte Aufforderung “schick mir den Link“ unterstreicht dies.

Die Bedeutung des URI-Vergleichs wird noch einmal deutlicher, wenn man nach Gründen für den Erfolg von Benutzeroberflächen sucht. Diejenigen, die einfach vom PC bekannte Oberflächen mit ihren verschachtelten Menüs und der Vielzahl an benötigten Klicks nachbilden, sind auf mobilen Geräten nur für einen geringen Teil technisch versierter Benutzer akzeptabel. Oberflächen mobiler Endgeräte verlangen nach intuitiven Benutzerschnittstellen - Anleitungen werden normalerweise nicht gelesen (vgl. [6]).

Das Konzept der URI für die Kommunikation von Anwendungen untereinander und mit dem Betriebssystem zu nutzen ist Teil des angedeuteten neuen Konzepetes. Wie dies im Detail gestaltet ist und welche Konsequenzen sich daraus ergeben, wird nachfolgend beschrieben.

4.1. Konzept

Anwendungen für Android sind nicht monolithisch wie es bisherige Anwendungen für den PC oft sind. Vielmehr wird der Komponenten-Aspekt wesentlich deutlicher hervorgehoben. Wie die folgenden Abschnitte zeigen werden, erlaubt es die Architektur und das Anwendungskonzept von Android, stark anpassbare und beliebig austauschbare Anwendungen zu erstellen. Ermöglicht wird dies durch die späte Bindung zur Laufzeit von angeforderter Aktion und dem ausführenden Code.

Intent & Intent-Filter Intent und Intent-Filter sind zentraler Bestandteil des innovativen Konzepts für Anwendungen und die Benutzeroberfläche, welches das zuvor beschriebene “klick-es-an“-Paradigma vom Web auf mobile Anwendungen - sowohl bzgl. der Benutzung als auch der Entwicklung - überträgt.

Ein *Intent* bezeichnet eine Anforderung für eine Aktion, also beispielsweise “eine Adresse im Adressbuch suchen“ oder “rufe diese URL auf“. Das passende Gegenstück wird als *Intent-Filter* bezeichnet. Es ist die Erklärung einer Anwendung, dass sie in der Lage ist, bestimmte Anforderungen zu verarbeiten.

Die angeforderte Aktion wird typischerweise durch ein Verb (z. B. PICK, VIEW, EDIT, ...) bezeichnet. Eine Grundmenge ist im Android Framework bereits in der Intent-Klasse vordefiniert, Entwickler können jedoch auch problemlos weitere, eigene Aktionen implementieren.

Ein Intent wird bei einem Aufruf außerdem durch Daten begleitet, die zum einen den Intent selbst näher spezifizieren und zum anderen die zu verarbeitenden Daten adressieren. Diese Daten werden als URI angegeben, die Tabelle 1 zeigt einige Beispiele.

Intent-Filter können sich speziell auf die Aktion, die zu verarbeitenden Daten oder beides beziehen. Wenn ein In-

[18] Dies ist ein wesentlicher Unterschied zu bisherigen Plattformen, auf denen die vorinstallierten und durch den Hersteller bereitgestellten Anwendungn oft auf Funktionen zurückgreifen konnten, die dem allgemeinen Entwickler nicht zugänglich gemacht wurden.

[19] Uniform Resource Indentifier

Beispiele für Intent URI	
Datentyp	**URI**
Adressbucheintrag	content://contacts/23
Adressenliste einer Gruppe	content://contacts/friends
Lokation auf Karte	Geo:0,0?q=10+Nobelstrasse+Stuttgart
Webseite	http://www.google.com
Notiz aus der Notepad-Anwendung	data=content://com.google.provider.NotePad/notes/42

Tabelle 1: Beispiele für Intent URI

tent abgesetzt wird, durchsucht das System die verfügbaren Activities, Services und Intent-Receiver (auf alle drei wird nachfolgend eingegangen) und leitet den Intent dann an das am besten passende Gegenstück weiter.

Intent-Filter werden in der für jede Anwendung obligatorischen `AndroidManifest.xml`[20] beschrieben und mittels dieser während der Installation der Anwendung dem System bekannt gemacht.

Activities Ob eine Anwendung eine grafische Oberfläche hat oder nicht, ist auch unter Android nicht festgelegt. Wenn eine Anwendung jedoch über eine solche verfügt, muss es auch mindestens eine Activity haben. Am besten lässt sich eine Activity mit einer Bildschirmmaske vergleichen. Die eigentliche Maske, also das graphische Layout, ist ein View - zusammen mit der für die Anzeige und die Verarbeitung von Eingaben notwendigen Logik ergibt sich die Activity. Eine Activity ist vergleichbar mit Controller-Klassen aus dem MVC-Pattern[21].

Activities werden durch Android mit Hilfe eines Activity-Stack verwaltet und können zu jedem Zeitpunkt in einen der folgenden Zustände versetzt werden:

active/running Die Activity wird ausgeführt und das von ihr gezeigte VIEW im Vordergrund auf dem Gerät dargestellt.

paused Die Activity ist pausiert und wird von einer anderen, derzeit aktiven Activity teilweise oder durch ein halbtransparentes Element überdeckt. Es werden alle Status- und Benutzerinformationen zwischengespeichert.

stopped Wird die Ansicht einer Activity durch eine andere komplett überdeckt, gilt sie als gestoppt. Wie im *paused*-Modus werden auch hier die Statusinformationen gespeichert.

Pausierte und gestoppte Anwendungen können bei Ressourcenmangel durch das System beendet werden, wobei auch dann die Statusinformationen gespeichert bleiben und bei einer Rückkehr des Benutzer zu dieser Activity der alte Status wiederhergestellt wird.

Diese verschiedenen Zustände, die Steuerung durch das System und die Verankerung des Mechanismus in den Grundkonzepten ist wichtig, um auf externe Ereignisse (eingehender Anruf, SMS) reagieren zu können, ohne auf die gerade aktive Anwendungen Rücksicht nehmen zu müssen.

Activities werden immer von der Framework Basisklasse `android.app.Actitity` abgeleitet. Der Wechsel zwischen verschiedenen Activities erfolgt mit `startActivity()` oder `startSubActivity()`, Daten werden wiederum als Parameter in Form von Intents weitergegeben.

Views Wie bereits erwähnt, werden konkrete Bildschirmmasken als Views bezeichnet. Im Gegensatz zu den von Desktopanwendungen bekannten Toolkits SWT, AWT und Swing können Views auch mit XML-Dateien definiert werden und müssen nicht - oft umständlich - im Java-Quellcode programmiert werden. Dies führt zu einer weiteren Trennung von Layout und Logik (dem zentralen Paradigma des erwähnten MVC-Pattern) und hilft, den Quellcode der Anwendungen übersichtlich zu halten.

Intent Receiver Unter einem *Intent Receiver* werden die Intent-Filter zusammengefasst, die eine Anwendung global beim Betriebssystem registriert. Dies dient dazu, bei der beschriebenen Auswahl durch Android zur Verarbeitung eines Intent berücksichtigt oder bei globalen Benachrichtigungen, wie z. B. dem Eingang einer SMS, informiert zu werden. Um dies zu erreichen, stehen einer Anwendung drei Wege zur Verfügung. Der offensichtlichste Weg ist es, über die bereits erwähnte `AndroidManifest.xml` die Intent-Filter zu nutzen. Als Alternative dazu kann eine Anwendung auch zur Laufzeit über die Methode `registerReceiver()` des Systemcontextes registrieren. Als dritten Weg stehen auch bei den IntentReceivern eine Reihe von durch das Fra-

[20] Die `AndroidManifest.xml` ist vergleichbar mit der in jar-Paketen enthaltenen MANIFEST.MF Datei, enthält jedoch weitere für Android spezifische Eigenschaften.

[21] *Model-View-Controller*, ein Designmuster zur Trennung von Benutzeroberfläche, Anwendungslogik und Daten

mework vordefinieren Intent Receiver zur Verfügung, die durch eine Anwendung instantiiert und genutzt werden können. Der PhoneStateIntentReceiver kann beispielsweise genutzt werden, um Statusinformationen zu den Telefonfunktionen zu erhalten, wie z. B. Klingeln oder Auflegen.

An dieser Stelle ist es wichtig, hervorzuheben, dass die durch einen IntentReceiver verarbeiteten Intents von denen durch Intent-Filter angesteuerten und in Activities verarbeiteten grundverschieden sind. IntentReceiver werden unter keinen Umständen durch einen mittels `Context.startActivity()` abgesetzten Intent angesprochen, sie dienen ausschliesslich der Verarbeitung globaler Ereignisse. Dies bedeutet wiederum, dass ein IntentReceiver jedoch auch nicht unmittelbar eine Activity starten kann. Die Kommunikationsmöglichkeiten mit dem Benutzer beschränken sich auf das Absetzen einer *Notification*, die z. B. eine Informationsmeldung anzeigt (z. B. als Symbol am Bildschirmrand) und dem Benutzer die Möglichkeit bietet, aus dieser Mitteilung heraus eine passende Activity zu starten.

Services Soll eine Anwendung auch dann weiter aktiv sein, wenn ihre Oberfläche nicht im Vordergrund angezeigt wird, z. B. ein Mediaplayer oder ein Download, muss man die benötigte Funktionalität in einen Service auslagern. Services erlauben es, Dienste mit einem längeren Lebenszyklus zu realisieren.

Gesteuert wird ein Service wiederum durch Activities, die jedoch nur die Oberfläche zum Steuern des Service darstellen und diesen wiederum über Intents ansprechen.

Content Provider Werden durch eine Anwendung Daten verwaltet, die durch andere Anwendungen weiterverwendet werden sollen, muss ein ContentProvider implementiert werden. Dies stellt sicher, dass immer nur die Daten einer Anwendung weitergegeben werden, die auch dafür vorgesehen sind und jede Anwendung selbst für die Konsistenz ihrer Daten verantwortlich ist - ohne dass von außen an den Daten etwas verändert werden kann.

Erzwungen wird dies durch die Tatsache, dass jede Anwendung auf Dateisystemebene ausschliesslich Zugriff auf die eigenen Dateien hat und nicht auf Dateien, die durch andere Anwendungen erstellt wurden.[22]

Ein ContentProvider ist von der Basisklasse `android.app.ContentProvider` abgeleitet und implementiert eine Reihe von standardisierten Methoden. Er kann Daten sowohl an Activities und Services derselben Anwendung als auch an fremde Anwendungen liefern und dabei auf jede beliebige auf Android verfügbare Form (Dateien, SQLite Datenbanken, HashMaps, etc.) von gespeicherten Daten zurückgreifen. Im Prinzip ist ein ContentProvider also die Daten-Abstraktionsschicht, um Anwendungsdaten für eine oder mehrere Anwendungen an einem zentralen Ort in einer definierten Form vorzuhalten und den definierten Zugriff zu gewährleisten.

Ressourcen Ressourcen in Form von Bildern, Texten (Strings), Layoutbeschreibungen, u. ä. gehören zu jeder Anwendung. Android unterstützt die Verwaltung und Handhabung dieser auf komfortable Art. Abgelegt werden sie dafür standardisiert innerhalb des Unterverzeichnisses `res/` im Projektverzeichnis ebenso wie im Softwarepaket (s. u.). Dies ermöglicht es dem SDK, die vorhandenen Ressourcen automatisch mit der für jede Anwendung erstellten Klasse `R` synchron zu halten, in der Verweise auf die vorhandenen Ressourcen-Dateien in der Form statischer innerer Klassen vorgehalten werden, wie nachfolgend zu sehen:

```
public final class R {
    public static final class drawable {
        public static final int beach=0x7f020001;
        public static final int icon=0x7f020002;
    }
    public static final class string {
        public static final int
                message_notification=0x7f040004;
    }
}
```

Die Integer-Werte sind jeweils Referenzen auf die im Ressourcenverzeichnis vorhandenen Dateien (bei Strings in XML Dateien vorhandener Text), die beim Kompilieren mit in das Programmpaket eingebunden werden.

Dies ermöglicht z. B. den einfachen Zugriff auf einen in einer XML-Datei definierten Text über den Aufruf `R.string.message_notification` und unterstützt zusätzlich die Trennung von Layout und Anwendungslogik. In XML Dateien ausgelagerte Texte sind z. B. zur Übersetzung einer Anwendung in andere Sprachen notwendig.

Lifecycle Android Anwendungen laufen jeweils in einem eigenen Linux-Prozess. Der Prozess läuft unter dem für die jeweilige Anwendung angelegten Benutzer und bleibt solange erhalten, bis die Anwendung nicht mehr benötigt wird und das System die Ressourcen zur weiteren Verwendung einfordert.

Wesentlich ist, dass der Status einer Anwendung nicht durch die Anwendung selbst sondern durch das Betriebsystem verwaltet wird. Hierbei ist die Unterteilung der Anwendungen in die zuvor beschriebenen Activities mit den dort vorgestellten Zuständen wichtig. Um entscheiden zu können, welche Prozesse bei geringen Ressourcen beendet werden können, erstellt es eine interne Prioritäten-Hierarchie, indem es nach beteiligten Komponenten und aktuellem Status bewertet.

[22] Im Detail wird dies durch Zugriffsrechte in Verbindung mit dem von Linux bekannten Benutzermanagement erreicht. Für jede auf Android installierte Anwendung wird ein eigener Benutzer angelegt, der exklusiv Berechtigungen für die Dateien der jeweiligen Anwendung besitzt.

4.2. Werkzeuge

Zusammen mit dem SDK werden eine Reihe für die Entwicklung nützliche Werkzeuge mitgeliefert, von denen im Folgenden die Wichtigsten vorgestellt werden.

4.2.1 Eclipse Plugin

Zusätzlich zum SDK ist ein Eclipse Plugin verfügbar (beides kostenlos unter [1], für Windows, Mac OS X (Intel) und Linux (i386)). Die Dokumentation beschreibt zwar auch, wie man die einzelnen Schritte vom Java-Sourcecode bis zum Test im Emulator von Hand vollzieht, welche Konfigurationsdateien wie befüllt und verarbeitet werden müssen und stellt Werkzeuge bereit, die hierbei unterstützen. Letztenendes wird aber deutlich, dass die Nutzung des Eclipse-Plugins deutliche Vorteile bietet.

Das Plugin selbst offenbart seine Stärke durch die richtige Unterstützung im Detail und an schwierigen Stellen. Nach der Installation erscheint es erst einmal ohne besondere Merkmale, als Quelltext-Editor dient der standardmäßig in Eclipse enthaltene Java-Editor. Beim Erstellen eines Projekts und der standardmäßigen Verzeichnis- und Dateistruktur unterstützt, wie man es bei Projekten unter Eclipse kennt, ein Wizard.

An zwei Stellen wird die Unterstützung durch das Plugin besonders deutlich: Zum einen steht zum Bearbeiten der bereits erwähnten *AndroidManifest.xml* ein Editor zur Verfügung, der die Einstellung und Verwaltung der vielfältigen Optionen und ihre Verschachtelung sowie die Unübersichtlichkeit, die mit ihnen einhergeht, deutlich vereinfacht. Zum anderen arbeiten im Hintergrund Automatismen, die das Entwicklerleben vereinfachen. So wird z. B. die oben beschriebene zentrale Ressourcen-Klasse „R“ ständig mit dem `res/`-Verzeichnis synchron gehalten oder das Erstellen von Installationpaketen vereinfacht. Darüber hinaus wird der nachfolgend vorgestellte Emulator so eingebunden, dass man ihn bequem aus der Entwicklungsumgebung heraus starten kann, der aktuelle Entwicklungsstand der Anwendung paketiert und installiert wird und der Debugger in vollem Umfang zur Verfügung steht.

Für die Nutzung des Debugger existiert die „DDMS“[23]-Perspektive, über die man, wie von anderen Java Anwendungen bekannt, laufende Prozesse, genutzte Variable, Haltepunkte, etc. im Blick hat und verwalten kann. Darüber hinaus sind Steuerelemente vorhanden, über die man spezielle Funktionen des Emulators (s. folgender Abschnitt) komfortabel ansteuern kann.

Bisher nicht enthalten ist ein graphischer Editor für die XML-Dateien der Views, also der einzelnen Bildschirmmasken.

[23] „Dalvik Debug Monitor Service“

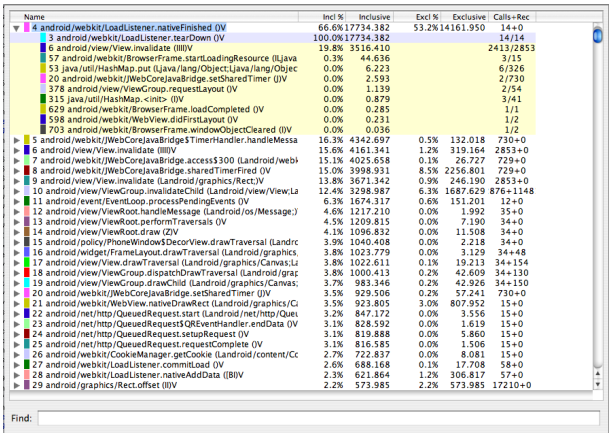

Abbildung 4: Traceview einer Anwendung [1]

4.2.2 Emulator

Der für Android zur Verfügung stehende Emulator basiert auf dem in der Open Source Szene weit verbreiteten Qemu[24]. Dieser wurde in vielen Bereichen für Android angepasst, insbesondere für die Zusammenarbeit mit den Entwicklungswerkzeugen des SDK.

So bietet der Emulator komfortable Funktionen um z.B. Dateien zwischen Host- und Emulatorsystem auszutauschen, eine Remote-Shell, um auf die Kommandozeile zuzugreifen (zum Auswerten von Kernel-Logs) oder spezielle Befehle um Netzverhalten (Status, Geschwindigkeit) oder Telefonie- und SMS-Kommunikation zu simulieren. Des Weiteren verfügt der Emulator über vier verschiedene Profile, um Geräte mit QVGA[25]- und HVGA[26]-Auflösung jeweils im Quer- und im Hochformat zu simulieren.

Mehrere Instanzen des Emulators zeitgleich auszuführen ist problemlos möglich.

4.2.3 Debugging & Profiling

Zum Debuggen erstellter Anwendungen stehen die von Eclipse bekannten Funktionalitäten zur Verfügung, die durch das Android SDK noch erweitert werden. Startet man die Anwendung im Emulator, wird der Eclipse-Debugger mit diesem verbunden und man kann wie gewohnt Breakpoints benutzen und Variablen inspizieren. Darüber hinaus steht im Android Framework ein ausgereifter Protokoll-Mechanismus zur Verfügung, der mit zwei Zeilen Code-Aufwand in eigenen Programmen genutzt werden kann. Die Ausgabe so protokollierter Daten erscheint in einem Ausgabefenster in der Debug-Ansicht der Entwicklungsumgebung.

[24] Homepage http://bellard.org/qemu/

[25] QVGA steht für Quarter-VGA und entspricht 320x240 Pixel

[26] HVGA steht für Half-VGA und entspricht 320x480 Pixeln

Zusätzlich stellt das SDK Werkzeuge für das Profiling[27] von Anwendungen zur Verfügung. Das Erstellen von Protokolldateien für eine zu analysierende Anwendung ist dabei ebenso wenig Aufwand, wie für das zuvor erwähnte Logging. Praktisch reichen die Zeilen *Debug.startMethodTracing("/tmp/protocol");* und `Debug.stopMethodTracing()` um eine Protokolldatei über zwischen diesen beiden Methodenaufrufen Codeteile zu erstellen. Mit dem im SDK enthaltenen Traceviewer[28] kann man dann aus dieser Protokolldatei graphische Auswertungen erstellen lassen - z. B. als Zeitstrahl, als Aufrufliste mit Angaben, wieviel Zeit in welchem Codeteil verbraucht wurde (s. Abbildung 4) oder auch als Baumansicht, die die Aufrufliste darstellt.

4.2.4 Softwarepakete

Für die Paketverwaltung verwendet Android das eigene Paketformat **.apk (Android Package)**. Bei .apk Dateien handelt es sich um Zip-Dateien, die eine spezielle Verzeichnis- und Dateistruktur enthalten. Dies ist neben den .dex Dateien (Bytecode für die DalvikVM, analog zu .class Dateien für die JavaVM) die AndroidManifest.xml sowie zur Anwendung gehörige Ressourcen wie Bilder und andere Medien oder Layoutdefinitionen für die Bildschirmmasken im XML Format.

Der Gesamtaspekt der Paketverwaltung auf Android ist noch nicht offengelegt. Zwar wird immer wieder betont, dass es möglich sein soll, beliebige Pakete zu installieren, jedoch gibt es bisher keine detaillierten Informationen darüber, ob auf Systemseite Zertifikate (wie z. B. bei dem bei Symbian eingesetzten Verfahren) oder ähnliche Konzepte zurückgegriffen werden soll, um vertrauenswürdige Pakete von solchen aus zweifelhafter Quelle zu unterscheiden.

4.3. Dokumentation

Die für Android verfügbare Dokumentation[1] deckt alle mit dem bisherigen SDK vorgestellten Teile der Plattform sehr gut ab. Durch das große Aufsehen und dem später erwähnten Developer Contest wurde darüber hinaus soviel Aufmerksamkeit erregt, dass es unzählige Foren und Blogs gibt, die sich mit Android und der Softwareentwicklung für die Plattform beschäftigen.

Die Android Dokumentation selbst umfasst neben einer reinen API-Dokumentation des Frameworks auch eine Vielzahl an Beispielen sowie Beiträgen, die das Zusammenspiel der Komponenten und die Architektur vorstellen. Anleitungen für die Benutzung des Eclipse-Plugins oder der Kommandozeilen-Tools fehlen ebensowenig wie 'Best Practices' für die Entwicklung von mobilen Anwendungen.

Gerade da der angekündigte Erscheinungstermin der ersten Androidgeräte[29] immer näher rückt, wächst die Spannung auf die Plattform ebenfalls.

5. Software

Mit der verfügbaren Software steht und fällt die Akzeptanz von Android. Nur wenn Benutzer die Möglichkeit haben, die dazugehörigen Geräte entsprechend ihren Vorstellungen und Bedürfnissen zu nutzen, werden sie sie akzeptieren und weiterempfehlen. Dabei dürfen sich ihnen keine Stolperfallen wie Instabilitäten oder benutzerfeindliche Oberflächen in den Weg stellen.

Für konsistente Benutzeroberflächen legt die Architektur von Android einen wichtigen Grundstein. Die Austauschbarkeit und die leichte Interaktion der Anwendungen untereinander ermöglichen es, dass Benutzer für einzelne Aufgaben auch immer genau eine Anwendung benutzen können. Dies kommt insbesondere in Kombination mit anderen Anwendungen als 'Workflow' zum tragen. Somit müssen die Benutzer nicht häufig neue Anwendungen und deren Bedienung (kennen-) lernen, um im Endeffekt zum selben Ergebnis zu kommen.

Der folgende Abschnitt zeigt, welche Anwendungen derzeitig verfügbar sind und stellt aktive Initiativen und Projekte vor, aus denen weitere Anwendungen - oder vielleicht die durch die OHA erhoffte „Killer-Applikation“[30] - hervorgehen werden.

5.1. Momentaufnahme

Im SDK enthaltene Anwendungen beschränken sich auf zentrale Funktionen (Telefonie- und Kontaktverwaltung), einen auf WebKit[31] basierenden Browser und einer Beispielanwendung für Google Maps. Darüber hinaus sind einige Beispielanwendungen und solche, die bei der Entwicklung von eigenen Anwendungen und dem Umgang mit dem Emulator unterstützen verfügbar.

Durch die Nutzung von WebKit als Kern des Browsers ist zu erwarten, dass der Browser zu dem in Apple's iPhone genutzten kompatibel sein wird und somit für das iPhone kompatible Seiten ebenfalls problemlos darstellen kann.

[27] *Profiling* bezeichnet die Laufzeitanalyse von Anwendungen. Es wird z. B. benötigt, um den Grund für Performanceschwächen oder übermäßigem Speicherverbrauch ausfindig zu machen.

[28] siehe auch http://code.google.com/android/reference/traceview.html

[29] Angekündigt für die zweite Jahrehälfte 2008

[30] Als *Killer Applikation* wird eine konkrete Anwendung bezeichnet, die einer Technologie zum schlagartigen Durchbruch verhilft und dabei ähnliche und oft ältere Konkurrenz-Technologien schnell verdrängt, also „tötet“.

[31] Bei WebKit handelt es sich um eine Open Source Browser Engine, die aus dem KHTML-Code des KDE Projekts hervorgegangen ist und als Basis der Browsers Safari von Apple bekannt geworden ist. Homepage: http://webkit.org/

5.2. Projekte

Um die Entwicklung von Anwendungen für Android zu forcieren, das Interesse für die Plattform zu erhöhen und natürlich die Verfügbarkeit einer beachtlichen Anzahl von Anwendungen zum Erscheinen der ersten Geräte sicherzustellen, veranstaltete Google in der ersten Jahreshälfte 2008 den „Android Developer Challenge“ (ADC, vgl. [3]) mit Preisgeldern von insgesamt 10 Millionen US-Dollar.

Die 50 Sieger der ersten Runde, die je ein ein Preisgeld über 25.000 USD erhielten, wurden im Mai 2008 aus über 1700 Einsendungen aus aller Welt (nur rund ein Drittel davon aus den USA) gekürt. Diese haben in der zweiten Runde, die mit der Verfügbarkeit der ersten Geräte beginnt, die Möglichkeit ihre Anwendungen weiterzuentwickeln und im richtigen Einsatz zu beweisen.

Die vorgestellten Anwendungen[32] waren dabei so vielfältig wie es die Möglichkeiten von Android bereits erahnen lassen. Location Based Services (LBS) wurden mit Social Networks verknüpft, die Kamera des (virtuellen) Smartphones zu einem Barcodescanner für den Onlinepreisvergleich umfunktioniert oder die Kontaktverwaltung um LBS Informationen und Instantmessaging-Funktionen angereichert. LBS treten bei der Betrachtung stark hervor und insbesondere die Tatsache, das durch den ADC bereits jetzt, Monate bevor Android als Produkt für Endkunden verfügbar ist, schon halb so viele Anwendungen wie für BlackBerry und ein Zehntel soviel wie für Windows Mobile (vgl. [5]) existieren, darf Google den ADC bis jetzt als Erfolg verbuchen - man darf gespannt sein, welche Anwendungen den Benutzer bei Erscheinen der Geräte erwarten und ob die erhoffte „Killer Applikation“ dabei sein wird.

6. Zusammenfassung und Ausblick

Android steht - trotz der überzeugenden Konzepte, durchdachten Schnittstellen und Werkzeuge, der sauberen und gut verwendbaren Dokumentation - auf wackeligen Beinen. Denn die Softwareplattform ist nur ein Teil des Erfolgs. Bringen die Hardwarehersteller unausgereifte Geräte an den Markt, die den Kunden verärgern oder blockieren die Netzoperatoren bestimmte Funktionalitäten für Android so, dass es seine Stärken nicht mehr voll ausspielen kann, wird der durch Google und die OHA angestrebte Erfolg sehr fraglich.

Android ist skalierbar genug um nicht nur auf Smartphones (derzeitig einen Marktanteil von 9 Prozent am gesamten Markt für Mobiltelefone), sondern auch auf schwächer ausgestatteten Mobiltelefonen zum Einsatz zu kommen. Damit kann ein wesentlich größerer Markt angesprochen werden und der kritische Verbreitungsgrad wesentlich schneller erreicht werden.

Geräte sind dabei von einigen namhaften Herstellern, insbesondere HTC, Motorola und LG, angekündigt. HTC, bisher hauptsächlich durch Smartphones mit Windows Mobile am Markt vertreten, hat ein eigenes rund 200 Kopf starkes Entwicklerteam mit Fokus auf Android aufgebaut. Für Motorola geht es noch ein Stück weiter. Die dortige Smartphone-Sparte steht am Rand der Pleite und benötigt einen Erfolg mit Android für ein Comeback. Bekannte Spezialisten, unter anderem die Designer des Erfolgs-Handys *Razr* arbeiten an Android. Für Motorola ist es entscheidend, mit Android am Markt erfolgreich zu sein.

Die Betreiber der Mobilfunknetze zu überzeugen, ist der bisher undurchsichtigste Schritt auf dem Weg zum Erfolg von Android. Auf der einen Seite bietet Android einen unschätzbaren Vorteil: das kostenlose System. Bisher machen Softwarekosten rund 20 Prozent der Kosten für ein Mobiltelefon aus. Auf der anderen Seite beschwört Android jedoch - ebenso wie das iPhone - den Tag herauf, an dem die bisherigen Platzhirsche der Branche - Vodafone, T-Mobile usw. - zu einfachen Datenleitungen derangiert werden. Zu alledem gibt es „wenig Anzeichen ernsthafter Schritte dagegen. Es ist zu bezweifeln, dass die Manager überhaupt verstanden haben, wie der Markt sich wandelt.“[33]. Ein Weg aus diesem Dilemma könnte die Bereitschaft von Google sein, die Einnahmen aus seinen Diensten mit den Providern zu teilen - ohne derlei Partnerschaften könnte es schließlich sein, dass es seine marktbeherrschende Stellung verliert (vgl. [5]).

Die Entscheidung von Google hier eine Allianz ins Leben zu rufen, an der Firmen aus allen für den Mobilfunkmarkt relevanten Bereichen zusammen mit einigen Größen aus der Software- und Internetbranche beteiligt sind, scheint dennoch die richtige Strategie zu sein, um den Weg für den Erfolg von Android bestmöglich zu ebnen.

Für Google selbst scheint es dabei unerheblich, ob Android selbst eine marktbeherrschende Position erlangen kann, oder ob allein die Präsenz und der Einfluss auf den Markt mehr Nutzer dazu bringt, mobile Dienste zu nutzen. Kann man über Android an den Formaten und Standards mitwirken, hat Google einen entscheidenen Schritt getan, sich weiter am Markt zu etablieren. Je mehr Nutzer - gleich über welchen Zugang - sich im Web bewegen und Google's Dienste nutzen, desto besser für Google, desto eher kann das Engagement an Android als Erfolg verbucht werden.

Der - nicht zuletzt durch das gerade veröffentlichte iPhone 3G - immer wieder aufkommende Vergleich von Android zum iPhone hinkt allerdings etwas. Das iPhone ist ein abgeschlossenes, fertiges Produkt, als Meisterstück angepriesen, kaum oder schwer durch Dritte änder- und erweiterbar. Android hingegen soll eine offene Plattform werden, ist die Basis für weitere Entwicklungen, jeder kann es nutzen, ändern, weiterentwickeln, Anwendungen darauf aufsetzen, ohne mit Google zusammenarbeiten zu müssen. Es gibt

[32] Vgl. Slideshow der 50 Finalisten: http://code.google.com/android/images/adc1r1_deck.pdf

[33] Dan Bieler, Marktforscher IDC, [10]

nicht *das Android* - es soll hunderte Mobiltelefone mit Android geben. Android ist kein fertiges Produkt, es kann das Saatgut für einen neuen Zweig am Mobilfunkmarkt sein.

Eine Herausforderung für Google und die OHA wird es dabei mit Sicherheit auch sein, das Projekt als ein Produkt weiterzuführen. Ist es erstmal offen für jeden zugänglich, müssen die Provider und Hersteller in einem Boot gehalten werden, um eine Zersplittung, wie es bei Symbian zum Problem wurde und Linux als Desktopsystem seit Jahren auszubremsen scheint, zu verhindern.

Das Potenzial von Android erscheint also gewaltig. Es existieren einige Unwägbarkeiten, nichts desto trotz darf man in den kommenden Monaten noch mit einigen Überraschungen und Veränderungen am Mobilfunkmarkt rechnen. Ob Androiden mit den Initiative der OHA alltäglich werden, werden wir in nicht allzu langer Zeit wissen.

Literatur

[1] Google Inc., “Android Online Documentation”, Google Inc., http://code.google.com/android, zuletzt abgerufen am 10.07.2008

[2] Dan Bornstein, “Dalvik VM Internals”, Google I/O Session Videos and Slides, http://sites.google.com/site/io/dalvik-vm-internals, abgerufen am 02.07.2008

[3] Google Inc., “Android Developer Challenge”, Google Inc., http://code.google.com/android/adc.html, abgerufen am 12.07.2008

[4] Thomas Grothaus, Bodo Hinüber, “Android im Überblick” *Java Sprektrum*, 2/2008, Seite 8-12

[5] Daniel Roth, “Google´s Open Source Android OS will free the wireless web” *wired magazine 16.07*, 23.06.2008, http://www.wired.com/techbiz/media/magazine/16-07/ff_android, abgerufen am 01.07.2008

[6] W. Frank Ableson, “Unlocking Android - A Developer´s Guide” (Manning Early Access Program), Manning Publications Co., March 2008, http://www.manning.com/ableson/

[7] Frank Ableson, “Develop Android applications with Eclipse”, ibm developerWorks, 26.02.2008, https://www6.software.ibm.com/developerworks/education/os-eclipse-android/index.html, abgerufen am 05.04.2008

[8] Ed Burnette, “Patrick Brady dissects Android”, ZDNET.COM, 04.06.2008, http://blogs.zdnet.com/Burnette/?p=584, abgerufen am 20.06.2008

[9] Volker Müller, “Kampf gegen Google: Nokia gibt Betriebssystem Symbian frei”, Financial Times Deutschland, Ausgabe vom 25.06.2008

[10] Volker Müller, “Magere Zeiten für dicke Fische”, Financial Times Deutschland, Ausgabe vom 07.07.2008

[11] Stephen Shankland, “Google carves an Android path through opensource world”, c|net News.com, 22.05.2005, http://news.cnet.com/8301-13580_3-9949793-39.html, abgerufen am 15.06.2008

[12] Christian Kirsch, “Nokia kauft Symbian”, heise online, 24.06.2008, http://www.heise.de/newsticker/Nokia-kauft-Symbian-Update--/meldung/109873, abgerufen am 05.07.2008

[13] Volker Briegleb, “iPhone 3G am ersten Wochenende eine Million Mal verkauft”, heise online, 14.07.2008, http://www.heise.de/newsticker/iPhone-3G-am-ersten-Wochenende-eine-Million-Mal-verkauft--/meldung/110895, abgerufen am 14.07.2008

[14] Konrad Lischka, Frank Patalong, “Google revolutioniert die Handy-Welt”, Spiegel online, 05.11.2007, http://www.spiegel.de/netzwelt/mobil/0,1518,515504,00.html, abgerufen am 03.04.2008

[15] Jens Franke, “Mobile Plattformen im Überblick”, PAGE Magazin, Ausgabe 06.08, Seite 26ff, PAGE Verlag 2008

[16] Markus Stäuble, “Komplettpaket”, iX Ausgabe 2/2008, Seite 34ff, Heise Verlag 2008

[17] Flashdevices.com, “Flash Lite Market Percentage (as of Jan 2008)”, http://www.flashdevices.net/downloads/FL-Market-Percentage-Jan-2008.pdf, abgerufen am 23.07.2008