Seminar
Android: Plattform für mobile Geräte

Android SDK

von

Simon Kostede

Sommersemester 2010

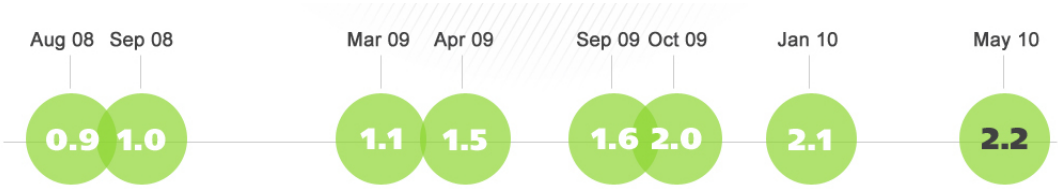
Betreuer:
Julian M. Kunkel

Universität Hamburg
Fachbereich Informatik am DKRZ
MIN Fakultät

Inhaltsverzeichnis

0.1 Vorwort

Diese Ausarbeitung beschäftigt sich mit dem Android SDK für die Android Plattform. Android ist eine der drei großen Plattformen für mobile Geräte, insbesondere für Smartphones. Die anderen großen Systeme sind Blackbarry OS von RIM und IPhone OS von Apple. Android wird im Zuge der OpenHeadsetAlliance hauptsächlich von Google bereitgestellt. Andere Betriebssysteme für mobile Geräte sind Windows Mobile von Microsoft und WebOS von HP (ehemals Palm).
Android ist insofern besonders, als dass es auf dem Linux Kernel aufbaut und komplett unter freien Lizenzen steht. Weiterhin läuft vieles der System-Software und ein Großteil der User-Programme in einer virtuellen Maschine, so dass Android leicht von einer CPU-Architektur auf eine andere übertragen werden kann.
Die aktuelle Android Version ist 2.2 aka Froyo. Diese Version ist die 7. stabile Version, welche in den 2 Jahren seit Veröffentlichung vorgestellt wurde.



Zum tieferen Einstieg in die Grundlagen des Android Systems empfiehlt sich die Lektüre der Ausarbeitung von Markku Lammerz.
Ziel dieser Ausarbeitung ist es eine Übersicht über das Android SDK (Software Developement Kit) zu bekommen, also darzustellen, wie man das SDK benutzt und welche Werkzeuge es einem für die Softwareentwicklung an die Hand gibt.

Kapitel 1

Installation

1.1 SDK

Das SDK kann von der Android Developer Webseite für Windows, Mac OS X, und Linux heruntergeladen werden. Das SDK benötigt das JDK[1] in der Version 1.5 oder neuer und kann einfach an einen beliebigen Ort entpackt werden.

Windows

Damit das SDK unter Windows optimal funktioniert, sollte man den Pfad zum „tools“ Verzeichnis des Android SDK der Windows PATH Variable hinzufügen. Diese findet man in Windows unter „Systemsteuerung\System und Sicherheit\System -> Erweiterte Systemeinstellungen -> Umgebungsvariablen“.

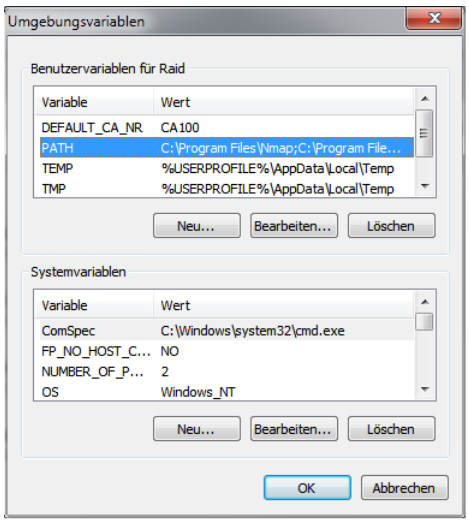


Abbildung 1.1: Path Dialog in Windows 7

[1] Java Developement Kit www.java.com

Linux

Unter Linux/Unix sollte man nach dem Entpacken auch die PATH Variable editieren. Diese findet man bei den meisten Distributionen unter /home/*benutzer*/.bash_profile oder /home/*benutzer*/.bashrc. Auch muss man beachten, dass die Android Tools für 32bit kompiliert wurden und deshalb entsprechende Kompatiblitätsbibliotheken auf 64bit Systemen nötig sind.

1.2 IDE

Prinzipiell ist es möglich jede Art von Texteditor zu benutzen um die Java Dateien zu schreiben, welche dann später zu Dalvik Dateien kompiliert werden.
In den meisten Fällen wird man aber mit einer normalen IDE arbeiten. Das ist meist auch ohne Probleme möglich, da es für diverse IDEs Plugins gibt, die einen Großteil der Arbeit für einen übernehmen.
Offiziell wird nur Eclipse mit einem Plugin bedacht, welches man von der offiziellen Android Developer Seite herunterlanden[2] kann oder per Eclipse über das „New Software“ Feature direkt installieren kann. Das Plugin deckt die meisten Aufgaben, die bei der Softwareentwicklung für Android anfallen, ab, so ermöglicht es z.B. die Kontrolle von Emulatoren oder angeschlossenen Entwickungesgerät und die Bereitstellung von Syntaxvervollständigung und Dokumentation für die Android-Klassen.

[2] http://developer.android.com/sdk/eclipse-adt.html

Kapitel 2

Werkzeuge

Das Android SDK besteht zum großen Teil aus verschiedensten Werkzeugen, welche in diesem Kapitel vorgestellt werden. Diese Werkzeuge übernehmen verschiedenste Aufgaben, von Crosscompiling in ausführbare Dalvik Dateien bis zum Debuggen von Anwendungen.

2.1 Android SDK and AVD Manager

Der Android SDK and AVD Manager ist ein Programm, mit dem man das SDK aktualisieren und seine virtuellen Android Telefone verwalten kann. Auch erlaubt der Manager das Herunterladen von Dokumentation und Quelltext zu den verschiedenen Android Versionen.

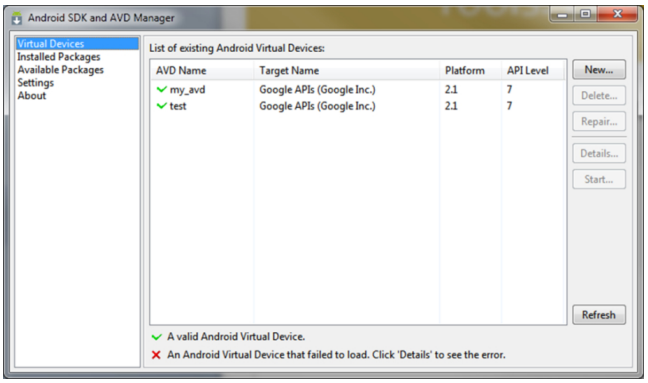


Abbildung 2.1: Android SDK and AVD Manager

2.2 Android Virtual Devices

In diesem Abschnitt geht es um das Erstellen von AVDs. AVDs oder Android Virtual Devices sind, wie der Name andeutet, virtuelle Android Geräte, die Android auf dem eigenen Rechner emulieren, d.h. dass im Gegensatz zu einer Simulation sich die Geräte, bis auf die Geschwindigkeit, genauso verhalten wie echte Android Geräte. Diese Emulation hat den Vorteil, dass man sich sicher sein kann, dass die eigene Software auf Android Systemen funktionsfähig sein wird. Da die Emulatoren normalerweise langsamer sind als echte Geräte, kann man sich sogar einigermaßen sicher sein, dass die eigene Software auch flüssig funktioniert.

Ein AVD kann man einfach über den oben erwähnten Android SDK and AVD Manager erzeugen oder man erstellt sie mit folgendem Konsolenbefehl, wobei <Name> ein frei wählbarer Name ist und <target> eine Id für eine Android Version.

```
# android create avd -n <Name> -t <targetID>
```

Abbildung 2.2: Befehl zum Erzeugen eines AvDs

Da man nicht immer im Kopf hat, welche Android Versionen man installiert hat und welche ID diese haben, kann man sich mit dem folgenden Befehl alle auflisten lassen.

```
# android list targets
Available Android targets:
id: 10 or "Google Inc.:Google APIs:8"
     Name: Google APIs
     Type: Add-On
     Vendor: Google Inc.
     Revision: 1
     Description: Android + Google APIs
     Based on Android 2.2 (API level 8)
     Libraries:
      * com.google.android.maps (maps.jar)
          API for Google Maps
     Skins: WVGA854, WQVGA400, HVGA (default), WQVGA432, WVGA800, QVGA
```

Abbildung 2.3: Beispielausgabe für Android 2.2 aka Froyo.

Sollte man GUIs favorisieren, kann man sich die AVDs auch über den AVD Manager erstellen, indem man diesen selbsterklärenden Dialog nutzt.

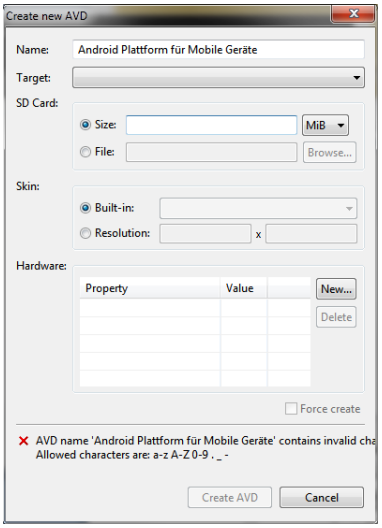


Abbildung 2.4: Dialog zum erstellen neuer AVDs

Skins für AVDs

Normalerweise werden die AVDs in einer ziemlich hässlichen grauen Umgebung ausgeführt:

Abbildung 2.5: Standard Emulator

Aber man kann den Emulator glücklicherweise „skinnen“, also ihm eine individuelle Optik verpassen, indem man entweder selber etwas erstellt oder sich einen der zahllosen Skins aus dem Internet herunterlädt.

Abbildung 2.6: Emulator mit Nexus One Skin

2.3 Neues Projekt erstellen

Es gibt zwei Möglichkeiten ein neues Androidprojekt mit allen dazugehörenden Dateien erstellen zu lassen, entweder über die eigene Entwicklungsumgebung mit passendem Plugin, hier Eclipse, oder über die Konsole mit dem „android“ Befehl.

Konsole

Um ein Projekt über die Konsole zu erzeugen nutzt man den „android create project“ Befehl, welcher, wenn mit den richtigen Paramentern versorgt, einen passenden Projektordner mit allen wichtigen Dateien anlegt. Im folgenden Beispiel ist „target“ wieder die ID für eine Android Version, welche man sich mit „android list targets“ anzeigen lassen kann. Beachten

sollte man, dass der „package“ Parameter, einen richtigen Java Package Namen benötigt, welcher nach dem Schema „<text>.<text>$^{+}$“ aufgebaut ist.

```
# android create project \
        --target <target_ID> \
        --name <dein_Projekt_name> \
        --path /pfad/zu/deinem/Projekt \
        --activity <dein_activity_Name> \
        --package <dein_package_Namensraum>
```

Abbildung 2.7: Auflistung der üblichen Erstellungsparameter

Eclipse

In Eclipse gibt es wie üblich ein Popupmenü, über welches man sein Projekt erstellen kann. Dieses findet man über den normalen „Neue Datei“ Dialog. Im Popupmenü kann man auch die wichtigen Einstellungen treffen, wie Name des Projekts, Name der Anwendung, Auswahl einer installierten Android Version etc.

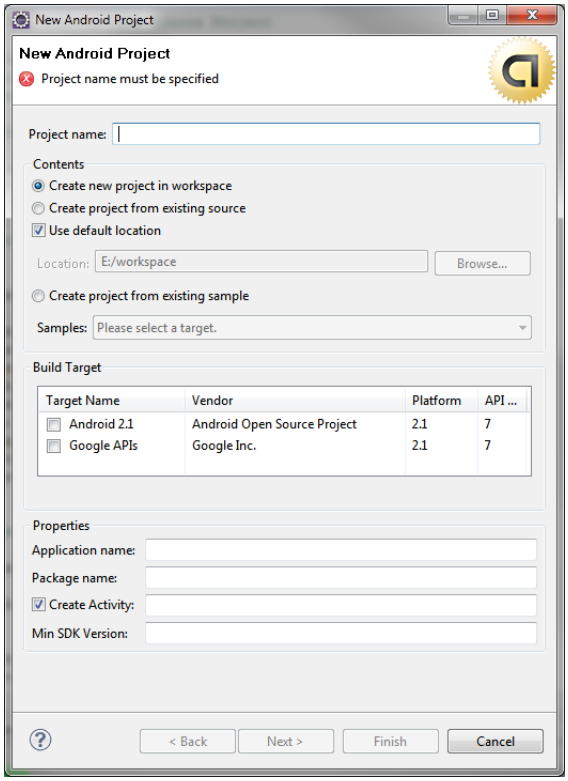


Abbildung 2.8: Dialog zum erstellen eines neuen Projekts

2.4 Anwendungen installieren & starten

Um eine Anwendung zu installieren gibt es zwei Möglichkeiten. Entweder in Eclipse, per Rechtsklick auf das Projekt im Workspace den „Ausführen als“ Menüpunkt, oder man nutzt den Konsolenbefehl „adb install /Pfad/zu/deinem/Programm.apk“. In beiden Fällen sollte die Anwendung, zumindest im Emulator, gleich gestartet werden. Passiert dies nicht, kann man sie einfach über das Programmmenü finden, je nach Android Version dargestellt durch einen Pfeil auf dem Android Hauptbildschirm oder durch ein Icon mit einer Matrix aus 3x3 Punkten. Dort einfach das Icon anklicken und die Anwendung startet.

Abbildung 2.9: Android Programmmenü

2.5 Persistenter Speicher für AVDs

Sollte man beim Erstellen seines AVDs keinen Speicher miterstellt haben oder ist der Speicher für die gewünschte Anwendung zu klein, so kann man mit dem Werkzeug „mksdcard“ eine virtuelle SD-Karte erstellen, welche man dann in seinem AVD einbinden kann. Android unterstützt zur Zeit nur den SDHC Standard, welcher nur Karten bis zu einer Maximalgröße von 32GB zulässt.

```
# mksdcard -l <Label> <Größe>[K|M] <Datei>
```

Abbildung 2.10: Erstellen einer virtuellen SD-Karte

```
# emulator -sdcard <Datei> -avd <VirtuellesDevice>
```

Abbildung 2.11: Einbinden der erstellen Karte

2.6 Graphische Benutzerschnittstelle

Android nutzt bevorzugt XML für das Design der Benutzerschnittstelle, dies führt zu einer besseren Trennung von Code & Design und macht alle damit erstellten Anwendungen leicht skalierbar. Die Mächtigkeit des Benutzerschnittstellendesigns lässt sich erahnen, wenn man sich verdeutlicht, dass Android Anwendungen ohne Änderungen auf kleinen Displays wie denen von kleinen Smartphones laufen, aber auch auf 50"Fernsehern.

Um den notwendigen XML-Code zu bekommen, kann man entweder den, etwas rudimentären, graphischen Editor nutzen, oder man schreibt den Code selber. Die auf XML basierende Syntax ähnelt dabei stark dem vergleichbaren Ansatz XAML von Microsoft, welcher zur „Windows Presentation Foundation“ für das .Net Framework gehört.

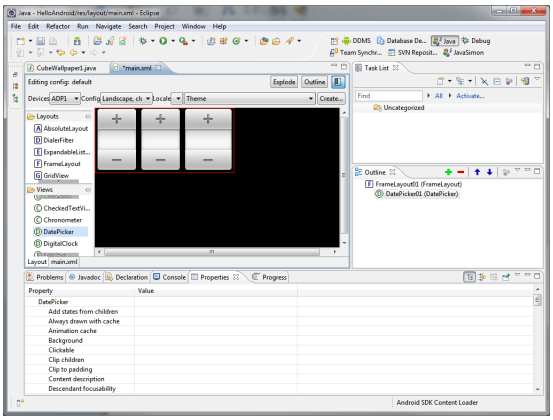


Abbildung 2.12: Rudimentärer graphischer Editor

Das Android SDK stellt auch noch weitere Werkzeuge bereit, welche die Skalierbarkeit von Anwendungen verbessern sollen. So kann man in seine Anwendungen sogenannte „Draw9“ Grafiken einbinden, welche man vorher zum Beispiel im, zum SDK gehörenden, „NinePatch“ Werkzeug erstellt hat. Diese Grafiken haben den Vorteil, dass sie aus einer Art Raster bestehen, in dem bestimmte Gebiete dupliziert werden können um die Grafik größer zu machen. Diese Methode funktioniert allerdings nur bei relativ einfachen Grafiken, wie sie beispielsweise oft für Knöpfe verwendet werden.

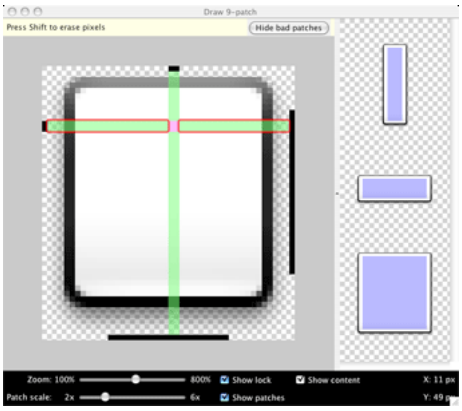


Abbildung 2.13: NinePatch Editor Draw9

UI Analyse

Das Android SDK bietet auch zwei Werkzeuge um erstellte Benutzeroberflächen zu analysieren. Einmal ist da „layoutopt“, welches die XML-Dateien eines UIs analysiert, indem es sie gegen eine Menge von Regeln prüft, das heißt, es findet manche Fehler und überflüssige Elemente.

```
# layoutopt samples /
samples/compound.xml
    7:23 The root-level <FrameLayout/> can be replaced with <merge>
    11:21 This LinearLayout layout or its FrameLayout parent is useless
samples/simple.xml
    7:7 The root-level <FrameLayout/> can be replaced with <merge>
samples/too_deep.xml
    28:81 This LinearLayout layout or its LinearLayout parent is useless
```

Abbildung 2.14: Beispielausgabe von layoutopt

Das zweite Werkzeug ist der „Hierarchy Viewer“, welcher die Struktur der Oberfläche darstellen kann und weitere Informationen zu ausgewählten Elementen anzeigt. Auch bietet er einen Modus, wo ein Bild der derzeitigen Oberfläche angezeigt wird. In dieser Darstellung kann man sich sowohl die Grenzen der einzelnen Elemente als auch die Benutzeroberfäche pixelgenau anzeigen lassen.

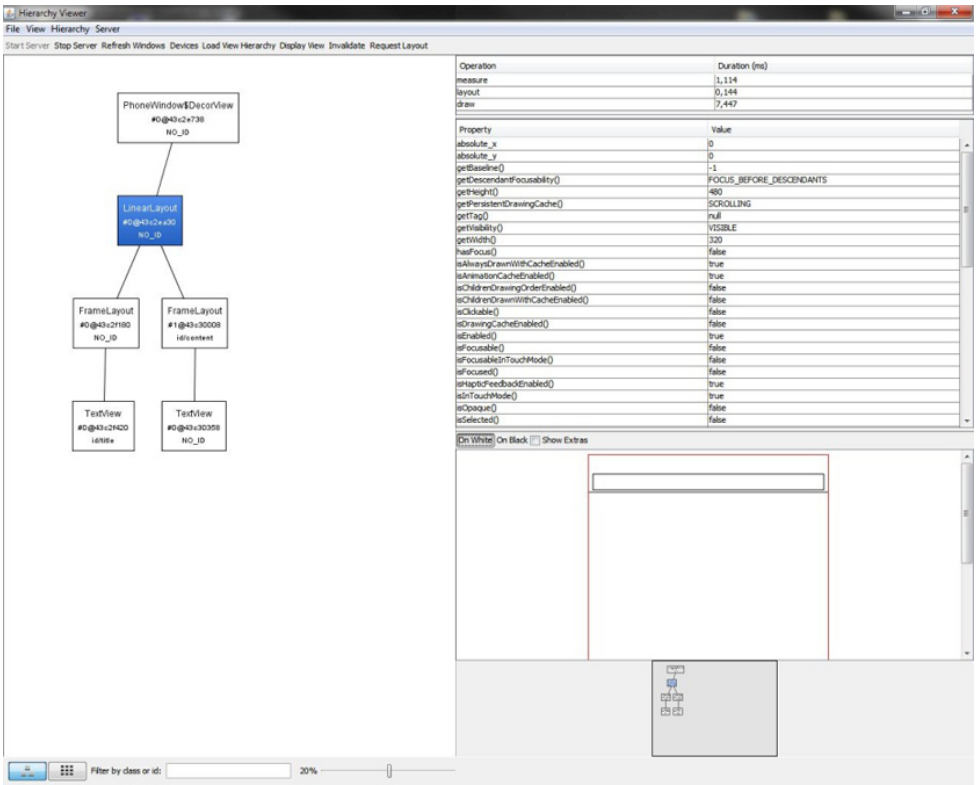


Abbildung 2.15: Rudimentärer graphischer Editor

Kapitel 3

Debugging

Das Android SDK bietet verschiedenste Möglichkeiten um die eigene Anwendung zu debuggen. Die Werkzeuge zeigen Informationen über den Systemzustand sowie auch Informationen über das gerade ausgeführte Programm. Weiterhin bietet das SDK Werkzeuge zum „Step-Based“-Debugging mit Integration in Eclipse durch das Eclipse Plugin. Auch stehen aktive Werkzeuge zur Verfügung, welche beispielsweise die Benutzeroberfläche einem Stresstest unterziehen.

3.1 Anzeige des Systemlogs

Traceview

Um sich das Systemlog im Bezug auf die eigene Anwendung anzeigen zu lassen, muss man zwei Schritte vollziehen. Erst sucht man sich eine Methode aus, die man überwachen möchte, und umschließt sie dann mit den folgenden Codeblöcken:

```
//startet tracing nach „/sdcard/cal.trace“
Debug.startMethodTracing(„calc“);


/** Dein Code **/


//stop tracing
Debug.stopMethodTracing();
```

Abbildung 3.1: Methodenüberwachung in einem Android Programm

Dieser Code speichert eine „*.trace“ Datei auf der notwendigen SD-Karte oder dem SD-Karten Image. Um jetzt das Log dargestellt zu bekommen lädt man die Datei per „adb pull \sdcard\calc.trace“ vom Gerät herunter und nutzt dann das Programm „traceview“ welches im „tools\“ Ordner des SDK zu finden ist.

„Traceview“ bereitet das Log graphisch auf und zeigt zwei Repräsentationen davon.

dmtracedump

„dmtracedump“ ist ein kleines Programm, welches auf Graphviz aufsetzt und Aufrufstacks von Prgrammen erzeugen kann.

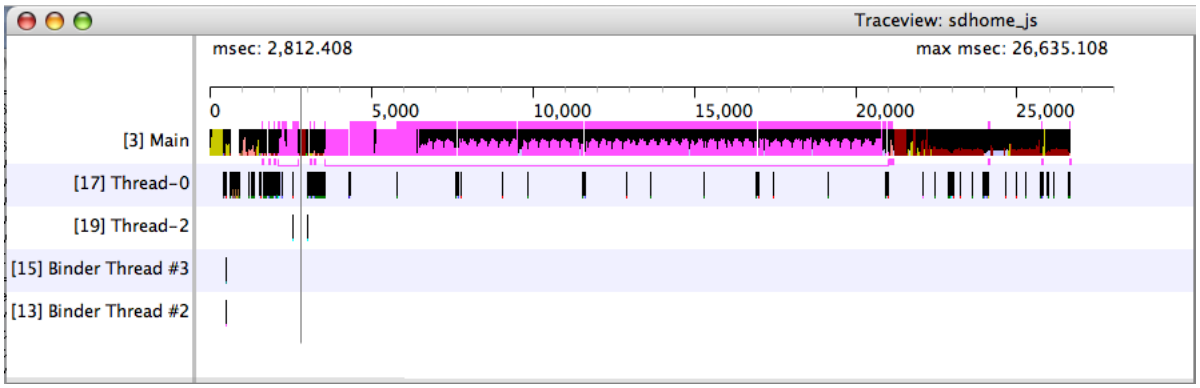


Abbildung 3.2: Traceview

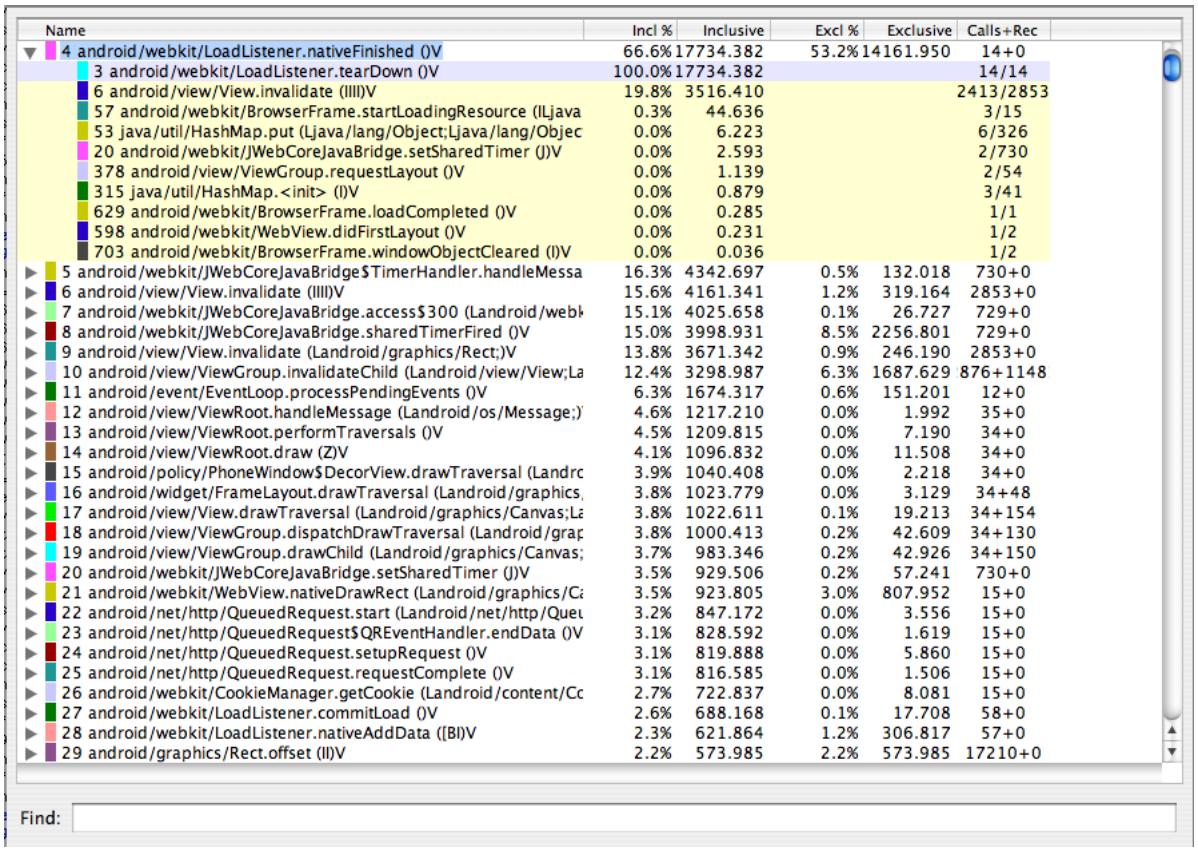


Abbildung 3.3: Traceview

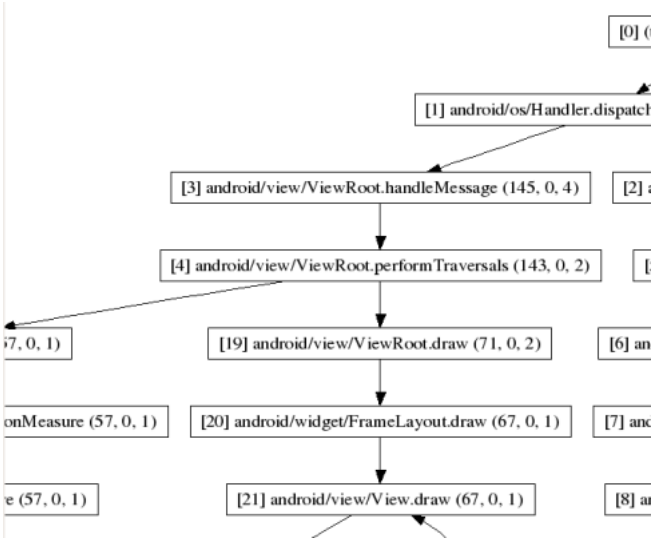


Abbildung 3.4: Partielle Ausgabe von dmtracedump

3.2 The Monkey

„The Monkey“ ist ein Stresstest für Android Programme. Es erzeugt zufällige Ereignisse wie Klicks, Berührungen, Gesten und Systemereignisse und testet dadurch die Stabilität des Programms. Um bessere Ergebnisse zu erhalten kann man den Ereignistyp einschränken und „The Monkey“ auf das eigene Programm beschränken. „The Monkey“ stoppt wenn es zu unbehandelten Exceptions oder anderen Programmabstürzen kommt oder wenn die vorgegebene Anzahl von Ereignissen ausgeführt wurde.

Um „The Monkey“ auszuführen nutzt man den folgenden Befehl:

```
# adb shell monkey -p dein.package.Name -v 500
```

Abbildung 3.5: „The Monkey“ mit Einschränkung auf ein Programm und 500 Ereignissen

3.3 Android Debug Bridge

Die Android Debug Bridge (ADB) ist eines der grundlegenden Werkzeuge wenn man Android Programme schreibt, es bietet Funktionalitäten wie Debugging, Datenübertragung und Shell Zugriff aufs Android Gerät.

Funktionsweise

Um die oben beschriebene Funktionalität bereitzustellen ist ADB in drei Teile unterteilt: Client und Server auf der PC-Seite sowie ein Deamon auf dem Android Gerät. Der ADB Server läuft dabei auf Port 5037 und die ADB Deamons lauschen auf den Ports 5554-5585, genauer gehören zu einem Gerät immer zwei Ports nämlich der kleinere gerade Port für die Konsole und der ungerade Port für die restliche Funktionalität.
Zwar ist der Normalzustand, dass ADB auf einem Rechner und maximal 15 Geräten läuft, allerdings kann man, wenn notwendig, die Struktur auch anders gestalten um Beispielsweise mehreren Entwicklern Zugriff auf ein Gerät zu gewähren.
Sind mehrere ADB Devices angeschlossen so kann man sich mit „adb devices“ die Liste dieser ausgeben lassen.

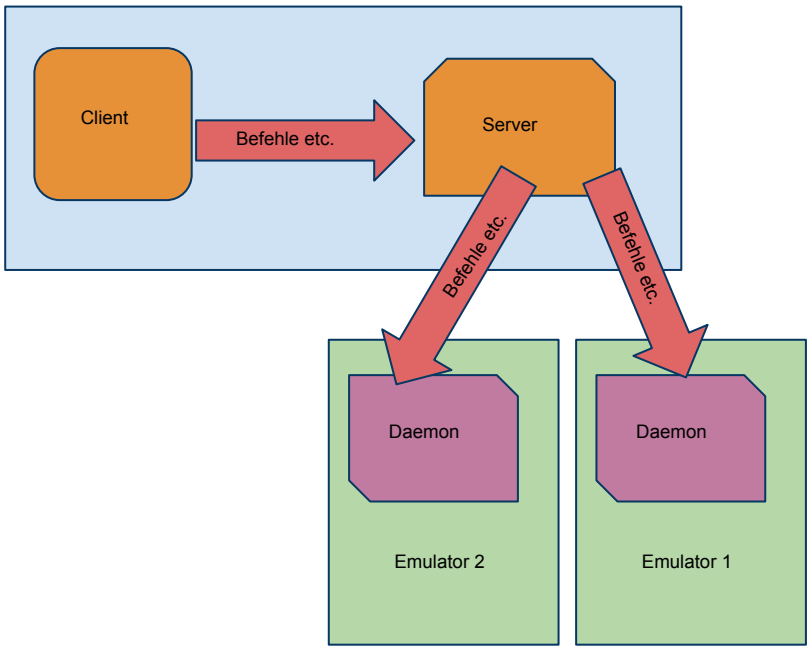


Abbildung 3.6: Struktur von ADB

ADB Befehle

ADB Befehle bestehen immer aus dem Präfix „adb“ und einem Suffix der die Funktionalität steuert, gefolgt von den Parametern.

```
# adb install <Datei> //Installiert ein Programm
# adb -s emulator-5556 install <Datei> //Installiert ein Programm auf einem
    bestimmten Gerät
# adb forward tcp:6100 tcp:7100 //Leitet einen TCP Port weiter
# adb push <Datei> //Schreibt eine Datei auf das Gerät
# adb pull <Datei> //Lädt eine Datei vom Gerät
# adb shell //Öffnet die Geräte Shell
# adb logcat <Filter> //Zeigt das Systemlog in Echtzeit
# adb bugreport //Erstellt einen Bugreport aus dem Gerätestatus
```

Abbildung 3.7: Auflistung der grundlegenden ADB Befehle

logcat

Logcat zeigt die Debuginformationen der letzten Minuten des Systems an, wobei diese in Echtzeit aktualisiert werde. Logcat kann mit „adb logcat“ aufgerufen werden, zusätzlich können noch ein Filterargument und eine Aktivity übergeben werden um die Ausgabe einzuschränken.

- V – Verbose (niedrigste Priorität: alles wird ausgegeben)
- D - Debug
- I - Info
- W - Warning
- E - Error
- F - Fatal
- S – Silent (höchste Priorität: nichts wird ausgegeben)

Abbildung 3.8: Filter für Logcat

Die Filter und Aktivity-Einschränkungen kann man zum Beispiel als „adb logcat ActivityManager:I *:S“ zusammensetzen. In diesem Fall hieße das, dass logcat Ereignisse ab dem Level Info ausgibt, welche vom ActivityManager erzeugt werden, und alle anderen Ereignisse auf Silent setzt, so dass sie nicht angezeigt werden.

Shell

In der Android Shell kann man *nix Programme direkt auf dem Gerät ausführen, allerdings sind meistens nicht sonderlich viele installiert. Welche *nix Programme vorhanden sind, kann man in /system/bin sehen.

Sqlite3

Als besonderes Programm sticht „sqlite3“ hervor, da es auf fast jeder Android Installation zu finden ist und man mit ihm grundlegende SQL Datenbanken verwalten und modifizieren kann. Befehl für sqlite3 ist „adb shell sqlite3“.

3.4 Dalvik Debug Monitor

Der „Dalvik Debug Monitor“ (DDM) ist das zweite große Debuggingprogramm, welches das Android SDK bietet. Es ergänzt die „Android Debug Bridge“ mit Funktionalität, die auf die virtuelle Maschine von Android ausgerichtet ist, die Dalvik VM. In diesem Zusammenhang bietet DDM Funktionen, die denen des JDK Debuggers ähneln.

Grundlegende Funktionen

Der „Dalvik Debug Monitor“ bietet verschiedenste Funktionen die sich auf die VM beziehen, allerdings bietet er auch einige Funktionen, die Geräteeigenschaften steuern. Die grundlegenden Funktionen sind:

- Anzeige des Heaps & Stacks
- Schrittbasiertes Debugging von Anwendungen mit Breakpoints
- Kontrolle der Garbage Collection
- Möglichkeit, das Gerät „virtuell“ anzurufen oder eine SMS zu schreiben
- GPS Daten bereitstellen (aus GPX oder KML (Google Earth) Dateien)

Den „Dalvik Debug Monitor“ gibt es in drei Inkarnationen, als eigenständiges Programm, das den Zustand der VM darstellt und weiterführende Informationen über CPU, RAM o.ä. ausgibt, dieselbe Funktionalität integriert in Eclipse und den „in-Code“-Debugger welcher auch in Eclipse integriert ist.
Die ersten beiden Varianten bieten dabei noch kleinere Funktionen wie einen Dateiexplorer für das Gerät, die Möglichkeit Bildschirmfotos zu machen sowie den einfachen Zugriff auf eine Prozessanzeige, logcat, dumpsys und dumpstate aus der Benutzeroberfläche.

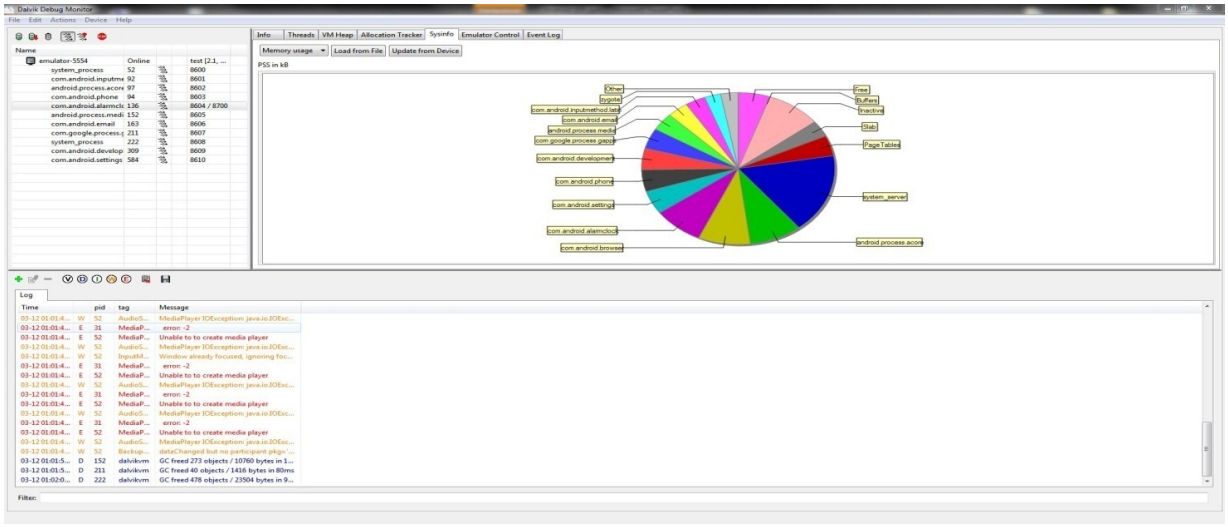


Abbildung 3.9: Dalvik Debug Monitor

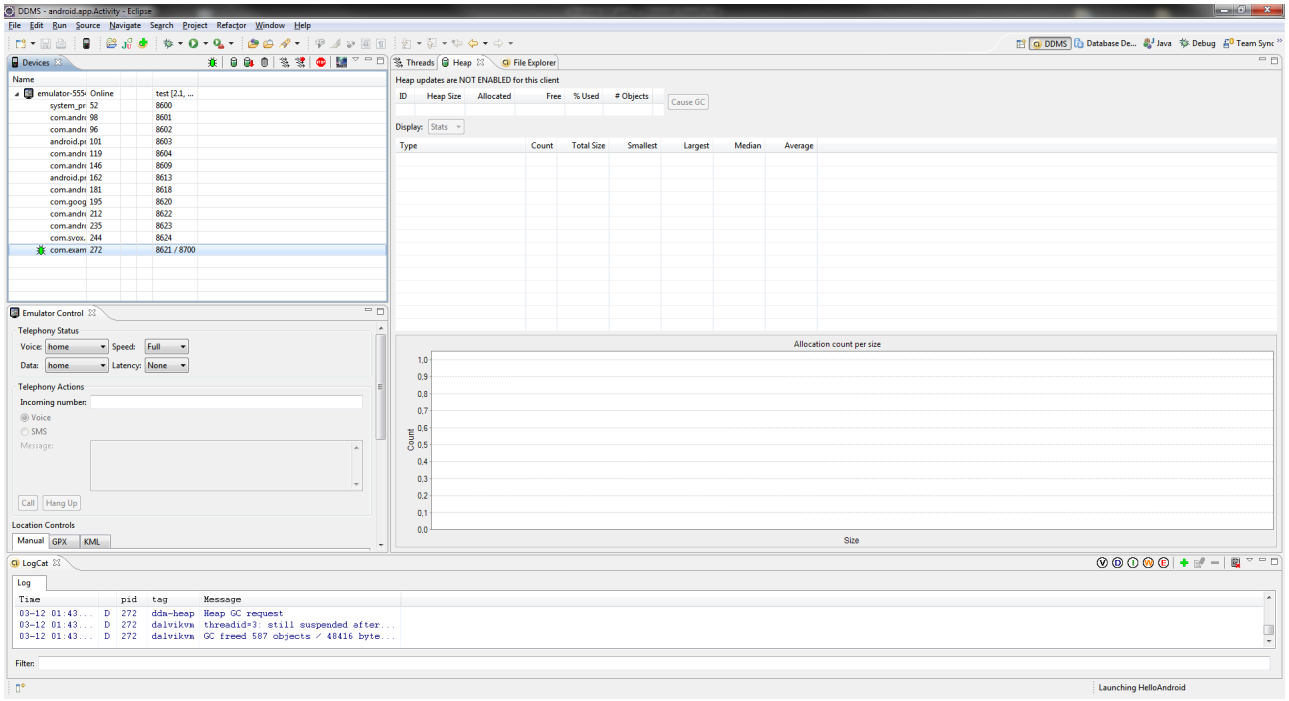


Abbildung 3.10: Dalvik Debug Monitor in Eclipse

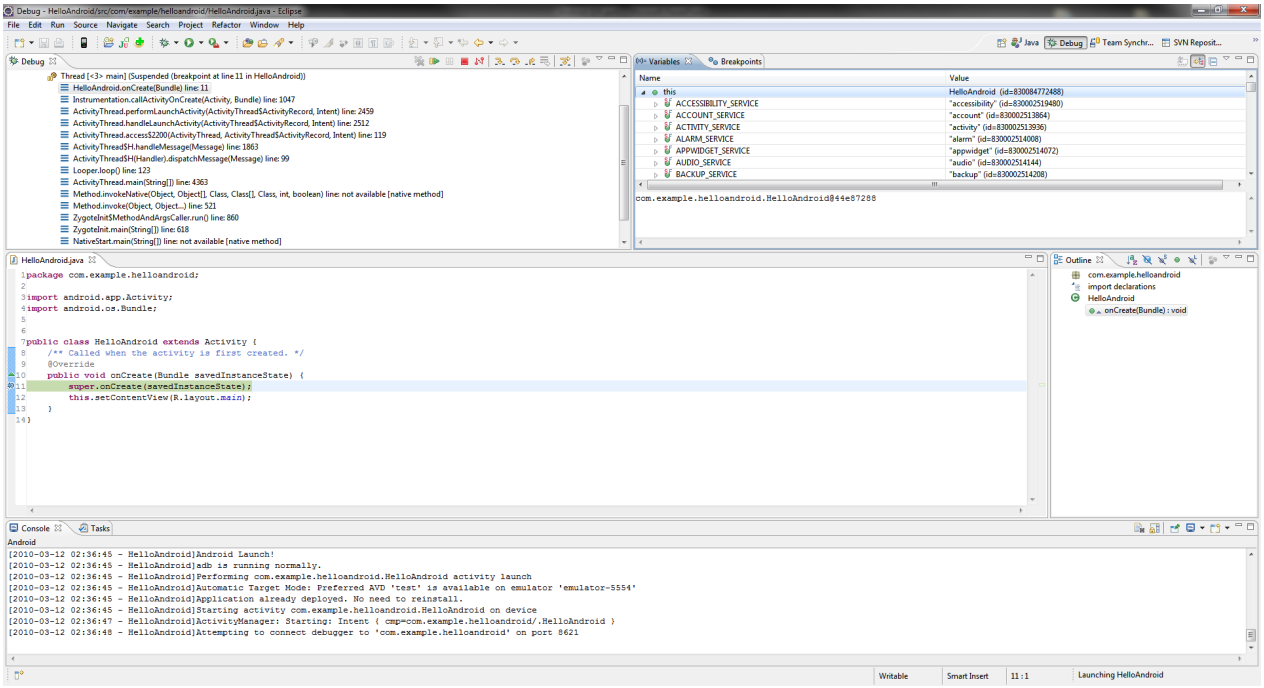


Abbildung 3.11: Dalvik Debug Monitor in Eclipse mit schrittbasiertem Debugging

Kapitel 4

Performance

Da Programm nicht nur möglichst fehlerfrei sein, sondern auch flüssig laufen , bietet das Android SDK einige Werkzeuge, mit denen man das Verhalten seines Programms überwachen kann. Dies ist vor allem wichtig, wenn man für die große Menge an günstigeren Android Geräten programmiert, da diese oft nicht so starke Hardware haben.

4.1 Leistung durch Alignment

Das wohl einfachste Werkzeug um die Leistung des eigenen Programms zu steigern, ist das Programm „zipalign“, welches die Daten in einem Android „*.apk“ so anordnet, dass sie direkt per mmap() in den Speicher geladen werden können. Das Alignment sollte bei derzeitigen Android Versionen immer 4 (Byte) sein, da die Dalvik VM 32bit Wörter nutzt.

```
# zipalign [-f] [-v] [-c] <Alignment> Input.apk Output.apk
```

Abbildung 4.1: Parameter für „zipalign“

Selber muss man dieses Tool nur ausführen, wenn man nicht das Plugin für Eclipse nutzt bzw. das Plugin für die eigene IDE „zipalign“ nicht ausführt.

4.2 Leistungsüberwachung

Will man die Leistung des eigenen Programms überwachen, gibt es mehrere Möglichkeiten, einmal das Tracing[1] und zum anderen den TimingLogger[2]. Tracing umschließt Methoden und misst deren Hardwarenutzung, während der TimingLogger Code-Laufzeiten ermittelt.

```
Debug.startMethodTracing("<Name>");

/** Der Code **/

Debug.stopMethodTracing();
```

Abbildung 4.2: MethodTracing für Android

Wenn man mit „MethodTracing“ eine „.trace“ Datei erstellt hat, kann man diese mit „traceview“ betrachten.

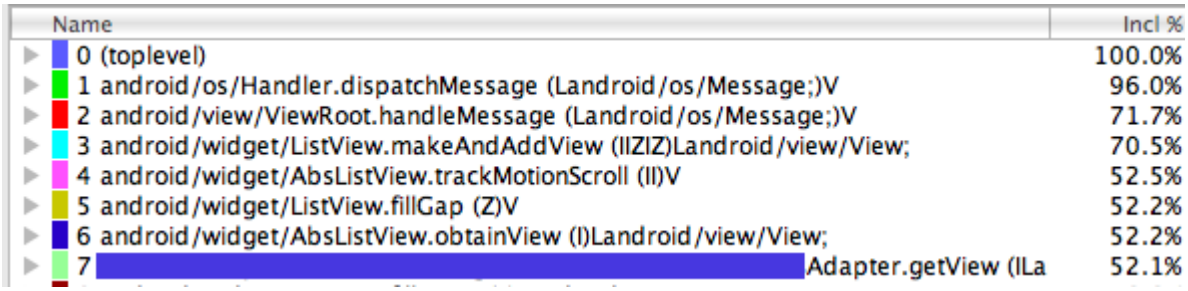


Abbildung 4.3: MethodTracing Ausgabe in Traceview

```
TimingLogger tim = new TimingLogger("TopicLogTag","preparePicturesFromList");

/** Der Code **/

adsplits(„<Label>“)

/** Mehr Code **/

tim.dumpToLog();
```

Abbildung 4.4: TimingLogger für Android

Die Ausgabe vom „TimingLogger“ kann man sich in Echtzeit mit „adb logcat -v time TopicLogTag:V *:E“ anschauen.

[1] `http://newfoo.net/2009/04/18/performance-tuning-android-applications.html`
Entwickler über MethodTracing
[2] `http://tech.artcom.de/blog/?p=256`
Entwickler über TimingLogger

4.3 Geräteleistung

Bei der Android Programmierung sollte man in Betracht ziehen, dass Android auf einer großen Anzahl von verschiedenen Geräten installiert ist, von langsamen wie dem T-Mobile G, dem ersten Android Handy mit 500 MHz[3], bis hin zu Dualcore[4] Tablets oder Fernsehern mit bis zu 1,5 GHz und mehr. Um die Leistung des anvisierten Geräts grob zu erfassen eignen sich Programme wie Linpack für Android und dazugehörige Übersichtsseiten, welche einem eine ungefähre Idee davon geben, wie schnell ein Gerät ist.

- Android Emulator auf Core2Duo@2,4GHz ⇒ 1,1 Mflops
- Android 2.1 auf 1GHz Snapdragon ⇒ 5-8 Mflops
- Android 2.2 auf 1GHz Snapdragon ⇒ 15-40 Mflops
- T-Mobile G1 ⇒ 1,2 Mflops

[3] `http://www.htc.com/www/product/g1/specification.html`
G1 Spezifikationen

[4] `http://www.engadget.com/2010/06/01/qualcomm-ships-first-dual-core-snapdragon-chipsets-clocking-1-2g/`
Engadget über DualCore Snapdragon Prozessoren

Kapitel 5

Native Developement Kit

Das „Native Developement Kit“ ist eine Ansammlung von Programmen die Arm-Bytecode erzeugen, was es erlaubt Teile der eigenen Android Anwendung in „nativem“ Code zu schreiben. Der Hauptvorteil soll dabei sein, dass man bestehenden Code der in C/C++ geschrieben wurde, wiederbenutzen kann. Dies ist vor allem auf Spiele ausgerichtet, die für andere Plattformen unter Zuhilfenahme von OpenGL ES geschrieben wurden.

Abbildung 5.1: Mit OpenGL beschleunigtes Spiel

Kapitel 6

Fazit

Android liefert in Form seines SDKs eine große Anzahl von nützlichen und sinnvollen Programmen und Werkzeugen mit die das Entwickeln für die Plattform deutlich erleichtern. Dies und die sehr verbreitete Sprache Java als Grundlage für die Entwicklung von Android Programmen dürfte zu den starken Steigerungen bei der Anzahl an verfügbaren Programmen für die Plattform führen. Diese Faktoren machen das Entwickeln von Android Anwendungen für Programmierer mit etwas Java Erfahrung sehr einfach. Allerdings muss man auch feststellen, dass es, bis jetzt, abseits dieser von Google bereitgestellten Programme nur wenige Werkzeuge von Drittanbietern für die Programmentwicklung für Android gibt.

Kapitel 7

Literaturverzeichnis

- `http://developer.android.com`
  Die offizielle Android Homepage
- `http://newfoo.net/2009/04/18/performance-tuning-android-applications.html`
  Entwickler über MethodTracing
- `http://tech.artcom.de/blog/?p=256`
  Entwickler über TimingLogger
- `http://www.htc.com/www/product/g1/specification.html`
  G1 Spezifikationen
- `http://bit.ly/aqaI3j`
  Engadget über DualCore Snapdragon Prozessoren