Dalvik VM

Reinhard Penn, Sebastian Ratzenboeck

Embedded Systems Design, FH Obersterreich
Campus Hagenberg
[1] S1410567017@students.fh-hagenberg.at
[3] S1419002063@students.fh-hagenberg.at

Fig. 1. Android Logo[1]

***Abstract*—This paper is about the Dalvik Virtual Machine. The goals are to give a short overview on the Dalvik VM and its function. Furthermore the *.dex* file format will be explained. In the end a comparison between the Dalvik VM and the newer Android Runtime will take place.**

I. Allgemeines[2]

Bei Dalvik handelt es sich um eine Open Source Software die von Dan Bornstein entwickelt wurde. Benannt ist sie nach einem Fischerdorf in Eyjafjoerour, Island. Veroeffentlicht wurde die Software unter Apache License 2.0. Die Ausfuehrung findet auf einem Linux Kernel statt.

Verwendung findet Dalvik in Googles Betriebssystem Android. Einsatz findet Dalvik hierbei als virtuelle Maschine. Der Hauptverwendungsbereich liegt im Mobilbereich, wie zum Beispiel bei Smartphones, Tablets und seit neuestem auch bei Smart Tvs und Wearables, wie Smartwatches.

II. Architektur[3]

In diesem Kapitel wird die Architektur der Dalvik Virtual Machine erklaert. Besonderes Augenmerk wird hierbei auf den Aufbau der *.dex* Datei gelegt.

A. Funktion

In Abbildung 2 ist die allgemeine Architektur von Android zu sehen. In ihr sieht man, dass die Dalvik Virtual Machine Teil der Android Runtime ist. In Android bekommt jeder Prozess seine eigene virtuelle Maschine, bei dieser handelt es sich um eine Dalvik VM. Sie aehnelt teilweise einer Java VM. Ein bedeutender Unterschied ist allerdings, dass eine Java VM stapelbasiert und eine Dalvik VM registerbasiert arbeitet. Diese registerbasierte Arbeitsweise lehnt sich an moderne Prozessorarchitekturen an. Sie verarbeitet Registermaschinencode, dadurch wird die Dalivk VM schneller als die Java VM und ist ressourcenschonender.

Ein weiterer bedeutender Unterschied liegt darin, dass eine Dalvik VM klassische Java Bibliotheken nicht unterstuetzt, zum Beispiel *AWT* und *Swing*. Es werden eigene Bibliotheken verwendet, die Apache Harmony als Grundlage verwenden.

Ein wichtiger Bestandteil der *SDK* ist das Tool *dx*. Es sorgt dafuer, dass Java Binaerdaten in Dalvik Executables umgewandelt werden. Das heist es wandelt *.class* Dateien in *.dex* Dateien um. Bei dieser Umwandlung koennen auch mehrere *.class* Dateien zu einer *.dex* Datei zusammengefasst werden, beziehungsweise kann der Speicherbedarf, mithilfe von *.odex* Dateien optimiert werden.

In Abbildung 3 ist ein solcher Vorgang beispielhaft dargestellt. Zuerst werden die *.java* Dateien in *.class* Dateien kompiliert. Nach dem Kompiliervorgang werden die Binaerdateien und die *.class* Dateien mithilfe des *dx* Tools zu eienr *.dex* Datei konvertiert. Mithilfe dieser Datei kann nun eine *.apk* Datei erstellt werden. Bei dieser handelt es sich um die Applikationsdatei mit der, der Endbenutzer die Anwendung auf seinem Mobiltelefon installiert.

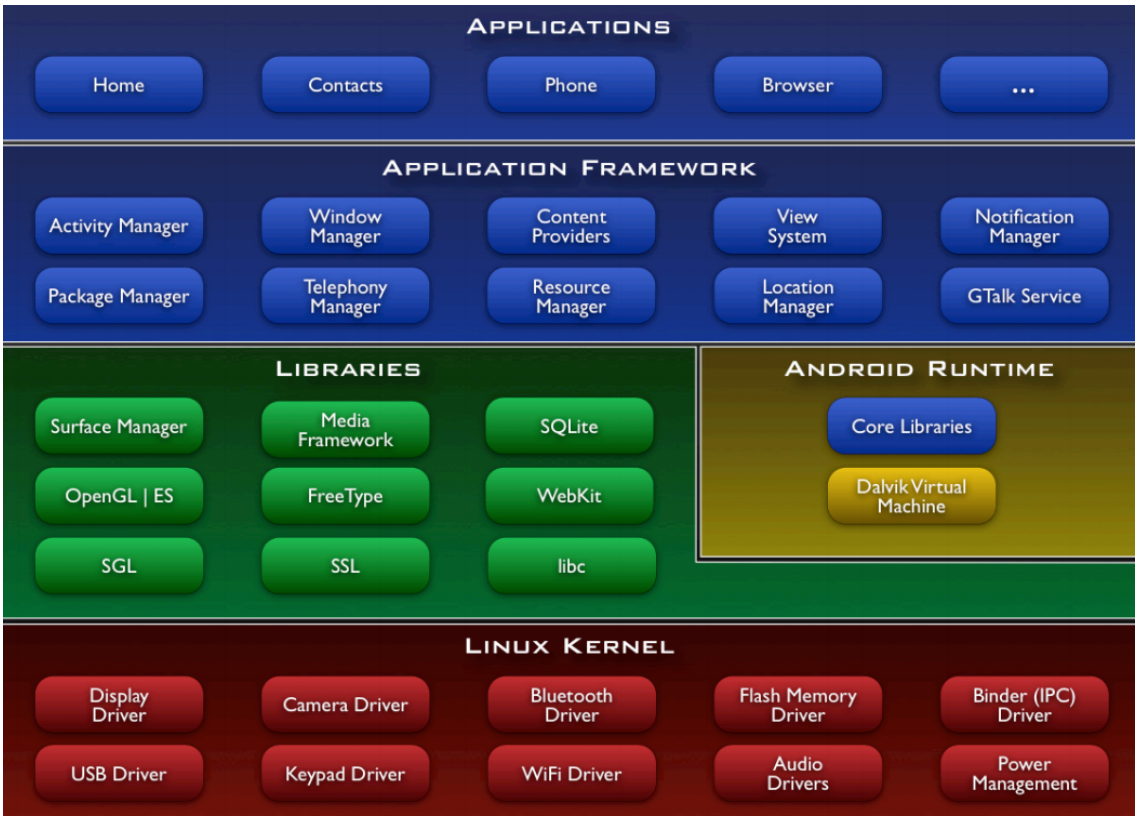


Fig. 2. Android Architektur[4]

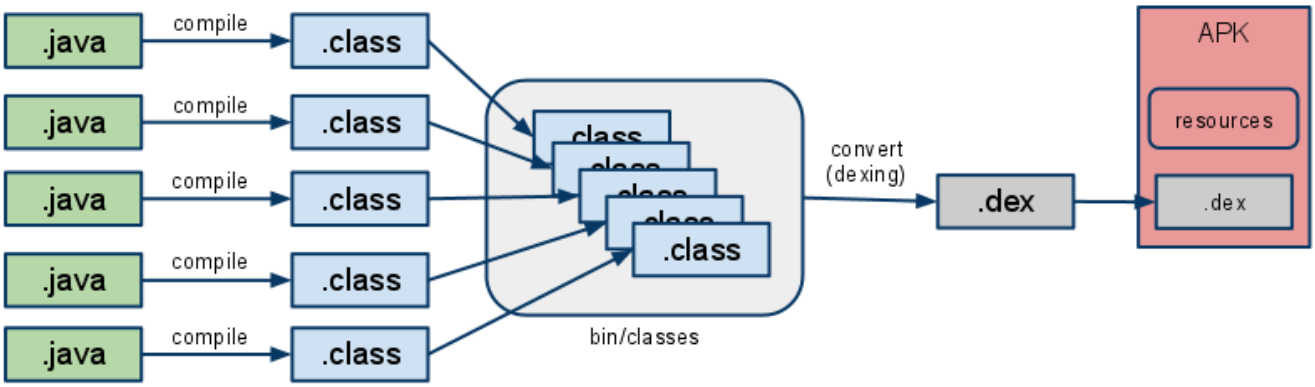


Fig. 3. Android Kompiliervorgang[5]

B. .dex Format

In .dex Dateien werden die Klassen Definitionen und die dazugehoerigen Daten gespeichert. Es handelt sich dabei um die ausfuehrbaren Dateien der Dalvik VM. Android Programme werden zuerst in Java Bytecode kompiliert. Dieser wird im Anschluss in Dalvik Bytecode uebersetzt.

In Abbildung 4 ist der Aufbau einer .dex Datei im Vergleich zu einer .class Datei von Java zu sehen. Fuer alle folgenden Listen der Datei duerfen keine doppelten Eintraege vorhanden sein.

1) Header: In Ausschnitt 1 ist der Header der *.dex* Datei zu sehen. Dabei handelt es sich um eine bestimmte Bytefolge die den Start der Datei bestimmt.

2) StringIds: Hierbei handelt es sich um eine Liste aller Strings die in der Datei verwendet werden. Die Liste muss sortiert sein und darf keine doppelten Ein-

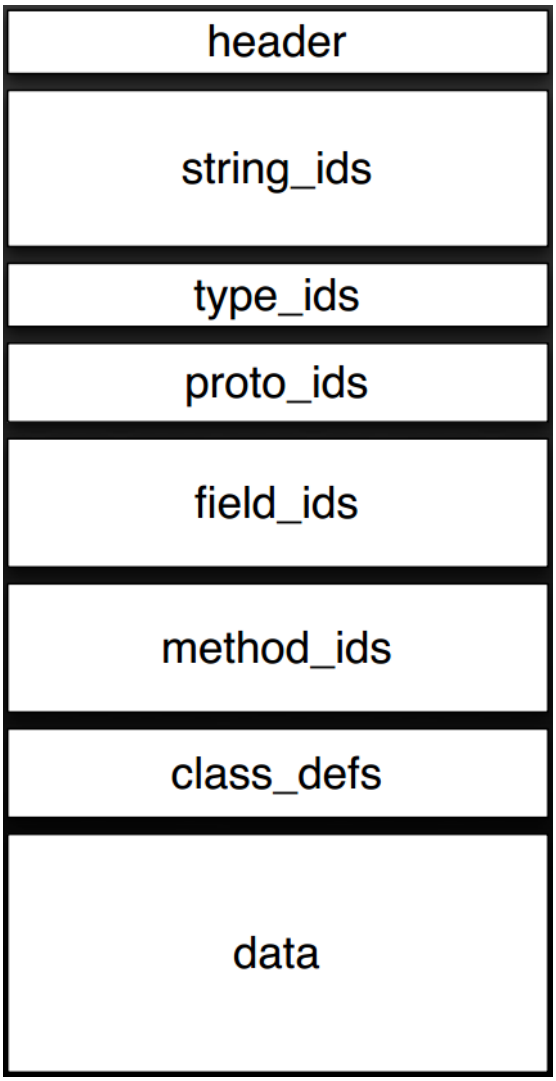


Fig. 4. Dex Dateiformat[4]

```
ubyte[8] DEX_FILE_MAGIC =
{ 0x64 0x65 0x78 0x0a 0x30 0x33 0x35 0x00 }
= "dex\n035\0"
```

Ausschnitt 1. Dex Header[3]

traege enthalten. Moegliche Inhalte sind zum Beispiel konstante Strings und Funktionsnamen.

3) TypeIds: Dies ist eine Liste aller verwendeten Typen dieser Datei. Dazu gehoeren Klassen, Arrays, und Primitive Datentypen. Diese Liste ist nach den *StringIds* sortiert.

4) ProtoIds: In dieser Liste befinden sich alle Methoden Prototypen, die in *.dex* Datei verwendet werden. Die Prototypen sind primaer anhand ihrer Rueckgabetypen sortiert und danach anhand ihrer Argumente.

5) FieldIds: Diese Liste enthaelt alle Felder die von der *.dex* Datei referenziert werden. Diese Liste wird anhand des Feldtypen und Feldnamen sortiert.

6) MethodIds: In der Methoden Liste werden alle von der *.dex* Datei referenzierten Methoden aufgelistet. Diese Liste wird nach dem Methodennamen und dem Methodenprototypen sortiert.

7) ClassDefs: In dieser Liste werden die Klassendefinitionen der *.dex* Datei angegeben. Die Liste muss so geordnet werden, dass uebergeordnete Klassen und Interfaces vor den davon ableitenden Klassen angefuehrt werden.

8) Data: In dem Data werden zusaetzliche Daten fuer die oben angefuehrten Listen gespeichert. Die verschiedenen Elemente haben verschiedene Alignements und padding bytes werden, wenn notwendig, eingefuegt. Des weiteren befinden sich in diesem Bereich die Daten von statisch verlinkten Dateien. Bei nicht verlinkten Dateien ist dieser Teilbereich leer.

C. Bytecode & Instruction Format

Das Modell der Dalvik VM ist an eine echte Architektur angelegt und soll Calling Conventions C hnlich realisieren. Die Dalvik VM ist registerbasiert und ihre Frames bekommen bei der Erstellung eine fixe Gre zugewiesen. Jedes Frame besteht aus einer bestimmten Anzahl an Registern, die von einer Methode aus dem Bytecode Set vorgegeben wird. Zustzlich beinhaltet das Frame noch alle zustzlich bentigten Daten, so wie zum Beispiel Program Counter und die *.dex* Datei die die Methode beinhaltet. Fr Bitwerte, wie zum Beispiel *integer* und Gleitkommazahlen sind Register 32-Bit lang. Fr 64-Bit Werte wird mit einem anliegenden Register ein Paar gebildet. Fr Registerpaare wird kein Alignement bentigt. Wenn die Register fr Objekt Referenzen benutzt werden, dann sind sie gro genug um exakt eine Referenz zu speichern. Bei einem Methodenaufruf werden N Argumente der Methode in den N letzten Registern des Frames der Methode platziert. Grere Argumente bentigen hierbei 2 Register. Methoden einer Instanz bekommen als ersten Parameter zustzlich eine *this* Referenz bergeben.

Instruktionen sind nicht auf Datentypen limitiert, solange sie den Inhalt der Register nicht interpretieren. Ein Beispiel dafr ist der *move* Befehl, ihm ist es egal ob er *integer* oder *floats* verschiebt. Die meisten Instruktionen sind bei ihrer Ausfhrung auf die ersten 16 Register limitiert und knnen auf hhere Register nicht zugreifen. Wenn es mglich ist erlauben einige

Instruktionen, aber auch Zugriff auf die ersten 256 Register. In einigen Ausnahmefllen wie zum Beispiel einzelnen *move* Instruktionen ist es mglich auf Register zwischen 0 und 65535 zuzugreifen. Untersttzt eine Instruktion das gewnschte Register nicht muss das hohe Register vor der Operation in ein niederes Register kopiert und nach der Operation wieder zurck kopiert werden.

Die Dalvik VM besitzt einige Pseudoinstruktionen, die dazu verwendet werden um Daten mit variabler Lnge zu verarbeiten zum Beispiel *fill-array-data*. Diese Instruktionen mssen auf 4 Byte aligned sein. Um dieses Ziel zu erreichen werden bei der Generierung der *.dex* Datei wenn ntig *nop* Instruktionen eingefgt. Wenn die Applikation auf einem System installiert wird knnen manche Instruktionen, aufgrund von Optimierungen, verndert werden. Damit kann eine schnellere Ausfhrung am System erreicht werden.

In Abbildung 5 sind einige Beispielinstruktionen der Dalvik VM zu sehen. Die Syntax ist so aufgebaut das zuerst das Ziel- und danach das Quellregister angegeben wird. Manche Opcodes haben Namenserweiterungen um gewisse Eigenschaften anzuzeigen. Zum Beispiel wird bei 64 Bit Opcodes *-wide* hinzugefgt und bei typenspezifischen Opcodes wird der Typ hinzugefgt, *-char*. Anhand des *move-wide/from16 vAA, vBBBB* Opcodes lassen sich diese Eigenschaften gut erkennen. Der Opcode dieser Instruktion ist *move* daran lsst sich leicht erkennen das es sich um eine Operation zum verschieben von Registerwerten handelt. Das Attribut *wide* gibt an das die Instruktion auf 64-Bit lange Daten angewandt wird. Das Attribut *from16* gibt an das eine 16-Bit Register Referenz als Quelle dient. Die beiden Parameter *vAA* und *vBBBB* geben Quell- und Zielregister an. Wobei das Quellregister im Bereich *v0-v255* liegen muss und das Zielregister im Bereich *v0-v65535*.

III. Dalvik vs Android Runtime

In diesem Kapitel wird die neuere Android Runtime nher betrachtet. Des weiteren wird die Dalvik VM mit der Android Runtime verglichen.

A. Dalvik[6]

Dalvik ist ein Just-in-time-Compiler (JIT Compiler). Dabei werden Programme oder Programmteile zur Laufzeit in Maschinencode bersetzt. Da die Kompilierung whrend der Ausfhrung des Programms durchgefhrt wird, kann sie nicht beliebig aufwendig sein, da dies sonst die Ausfhrungsgeschwindigkeit des eigentlichen Programms merklich beeintrchtigen knnte. Daher beschrnkt man sich meist auf hufig ausgefhrte Programmteile. Diese sind typischerweise fr den Groteil der Ausfhrungszeit des Programms verantwortlich, weshalb sich deren Kompilation und Optimierung besonders lohnt.

Die Aufgabe des JIT-Compilers ist es, diese Programmteile zu identifizieren, zu optimieren und anschlieend in Maschinencode zu bersetzen, welcher vom Prozessor direkt ausgefhrt werden kann. Der erzeugte Code wird meist zwischengespeichert, um ihn zu einem spteren Zeitpunkt der Programmausfhrung wiederverwenden zu knnen.

B. Android Runtime[7]

Android Runtime (kurz ART) ist eine Laufzeitumgebung die von Googles mobilem Betriebssystem Android ab Version 5.0 Lollipop eingesetzt wird. ART lste damit die bisherige virtuelle Maschine Dalvik ab, die bis Version 4.4 (KitKat) im Einsatz war. Laut Google bietet ART eine bessere Performance und niedrigen Energieverbrauch als Dalvik. Dies wird sich in der Praxis in besserer Performance und lngerer Akkulaufzeit bemerkbar machen.

C. Vergleich[8]

Bei Android Runtime handelt es sich um einen Ahead-of-time-Compiler (AOT-Compiler). Ein AOT-Compiler ist ein Compiler, der im Gegensatz zu Just-in-time-Compilern (JIT-Compiler) Programmcode vor der Ausfhrung in native Maschinensprache bersetzt. Dies hat den Vorteil, dass dieser Code zur Laufzeit wesentlich schneller ausgefhrt wird als auf einem JIT-Compiler, da die bersetzung bereits durchgefhrt wurde. Dabei wird der Java-Bytecode bereits bei der Installation einer App in maschinenlesbaren Code vorkompoliert.

Der Nachteil an AOT-Compilern ist aber, dass dieser Code nicht mehr plattformunabhngig ist, wie es bei JIT-Compilern der Fall ist. AOT-Compiler sind die herkmmlichen Compiler wie sie schon von C eingesetzt wurden. Da Bytecode prinzipiell speichermig kleiner als Maschinencode ist, kann die Gre einer installierten App schnell auf bis zu zustzlich 20% ansteigen. Auerdem dauert die Installation einer App lnger als zuvor, da der Bytecode bereits hier in Maschinencode bersetzt wird.

Op & Format	Mnemonic / Syntax	Arguments	Description
00 10x	nop		Waste cycles. **Note:** Data-bearing pseudo-instructions are tagged with this opcode, in which case the high-order byte of the opcode unit indicates the nature of the data. See "`packed-switch-payload` Format", "`sparse-switch-payload` Format", and "`fill-array-data-payload` Format" below.
01 12x	move vA, vB	`A`: destination register (4 bits) `B`: source register (4 bits)	Move the contents of one non-object register to another.
02 22x	move/from16 vAA, vBBBB	`A`: destination register (8 bits) `B`: source register (16 bits)	Move the contents of one non-object register to another.
03 32x	move/16 vAAAA, vBBBB	`A`: destination register (16 bits) `B`: source register (16 bits)	Move the contents of one non-object register to another.
04 12x	move-wide vA, vB	`A`: destination register pair (4 bits) `B`: source register pair (4 bits)	Move the contents of one register-pair to another. **Note:** It is legal to move from `vN` to either `vN-1` or `vN+1`, so implementations must arrange for both halves of a register pair to be read before anything is written.
05 22x	move-wide/from16 vAA, vBBBB	`A`: destination register pair (8 bits) `B`: source register pair (16 bits)	Move the contents of one register-pair to another. **Note:** Implementation considerations are the same as `move-wide`, above.

Fig. 5. Dalvik VM Instruktionen[3]

ART liefert Untersuchungen zufolge, in vielen Apps mehr Leistung als Dalvik. Grund ist, dass der Bytecode durch ART erheblich maschinennher als Dalvik formuliert wird und somit weniger Rechenaufwand zur Ausfhrung vonnten ist. Dies bedeutet, dass die Ressourcen des Gerts effizienter genutzt werden knnen und die gefhlte Performance ansteigt.

ART und Dalvik sind kompatiblel zueinander, das bedeutet Apps, die fr Dalvik implementiert wurden, laufen auch auf ART. Die ART benutzt dabei denselben Input Bytecode wie Dalvik, geliefert mit Standard .dex-Dateien als Teil der Android Application Package-Dateien (APK-Dateien), wobei die .odex-Dateien durch Executable and Linkable Format (ELF) Executables ersetzt werden. Da die Kompilierung nicht bei jeder Ausfhrung einer Applikation durchgefhrt werden muss, wird die Prozessorauslastung verringert und dadurch die Akkulaufzeit verbessert.

D. Neuheiten in Android Runtime[10][11]

1) Verbesserte Garbage Collection: GC kann die Performance einer App beeinflussen, also abgehackte Bilder, langsame Reaktion auf Benutzerinteraktionen oder andere Probleme. Verbesserungen sind unter anderem, dass weniger Ausfhrungen des Garbace Collectors gemacht werde und whrend der GC-Pause trotzdem Anwendungen parallel laufen knnen. Auerdem wird der GC krzer ausgefhrt fr kurzlebige Objekte.

2) Verbesserungen fr Entwicklung und Debugging: Fr Dalvik wurde das Tool Traceview als Profiler verwendet, also zur Analyse des Laufzeitverhaltens von Software. Dieses gibt zwar sinnvolle Informationen, jedoch wird bei jedem Aufruf durch das Tool der Overhead grer, wodurch die Laufzeit beeinflusst wird. Bei Traceview verwendet die Debug class um Informationen im Code zu protokollieren. Die log-Dateien knnen geladen werden und in 2 unterschiedlichen Panels angezeigt werden:

Timeline Panel (Abbildung 7): zeigt wann jeder

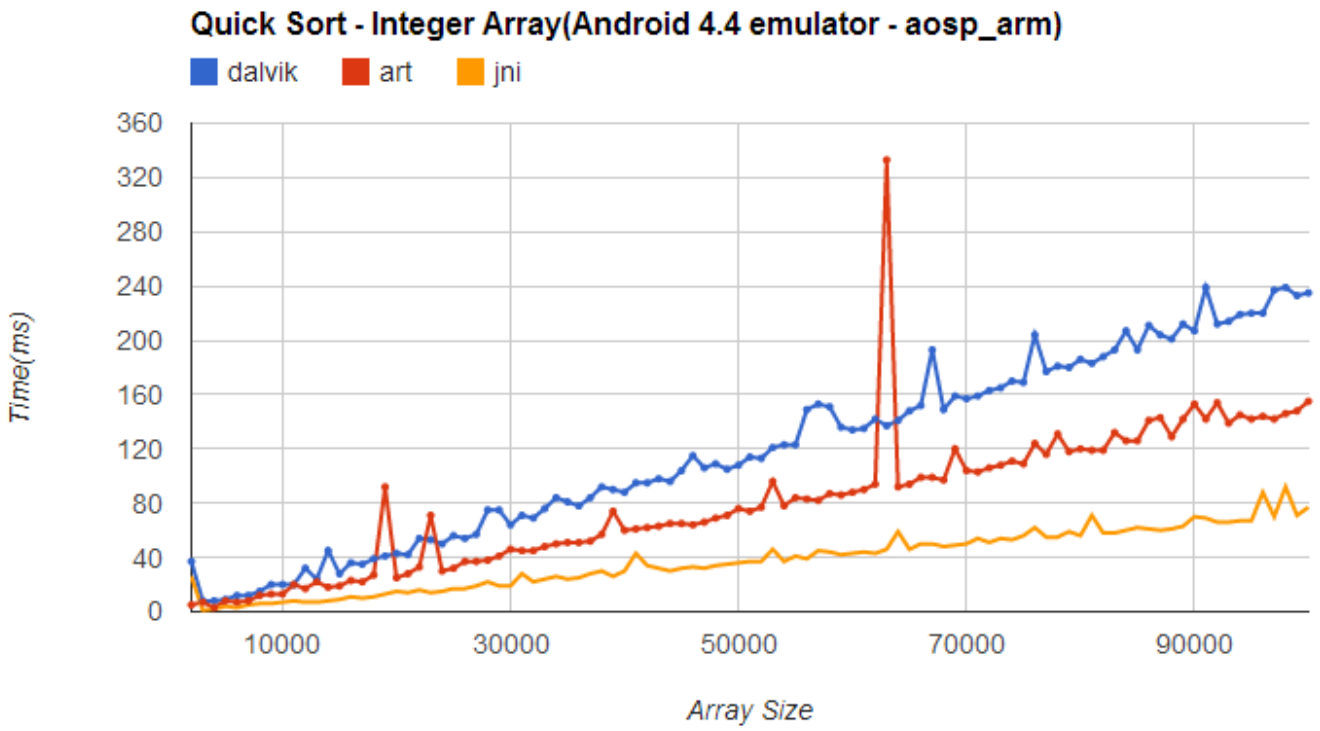


Fig. 6. Dalvik VM ART Vergleich[9]

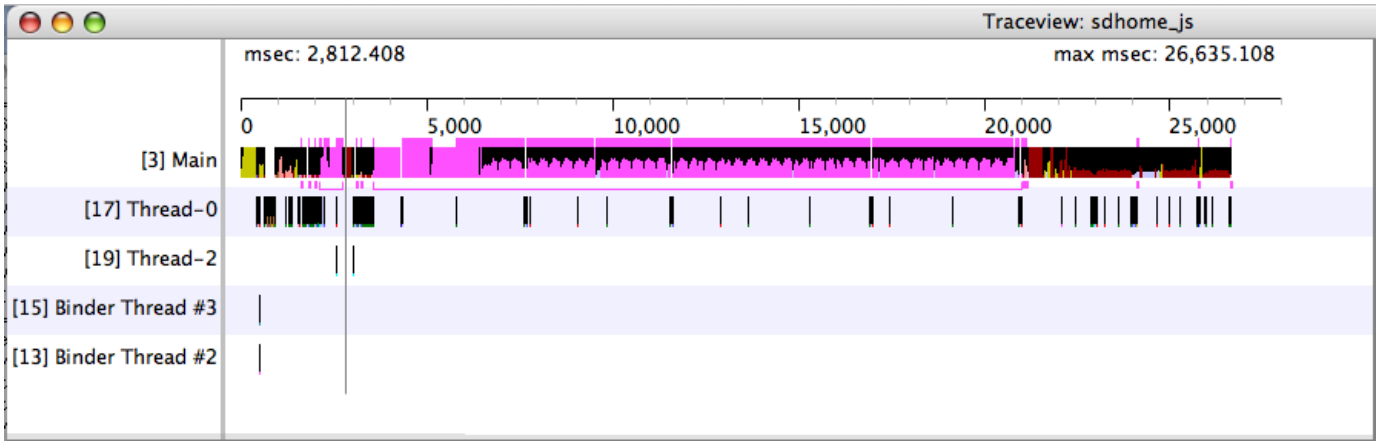


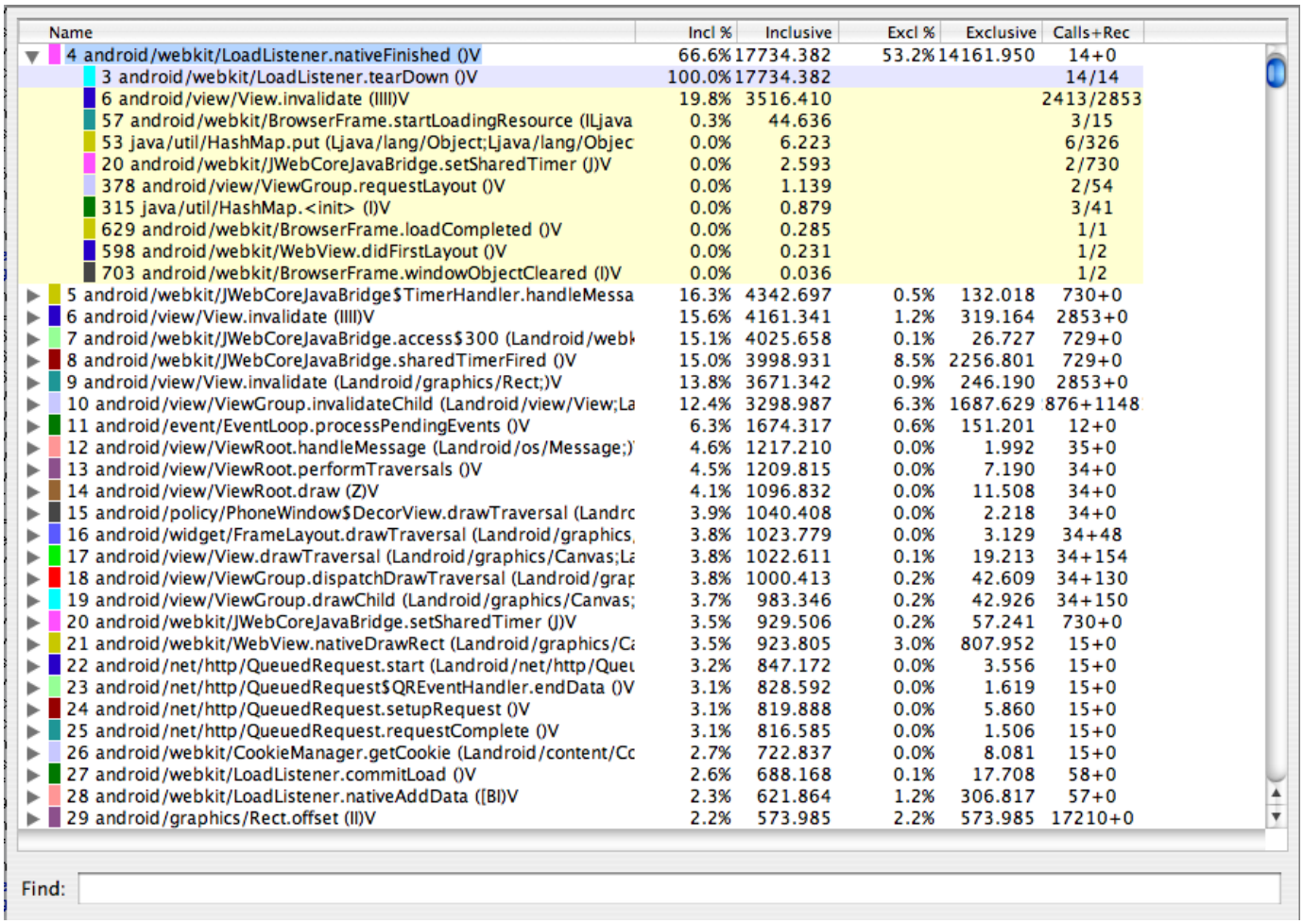
Fig. 7. Traceview Timeline[11]

Thread und jede Methode startet und stoppt.

Profile Panel (Abbildung 8): stellt einen berblick dar von Ereignissen innerhalb der Methoden.

ART untersttzt einen geeigneten Profiler der diese Einschrnkungen nicht hat. Dadurch gibt es eine detaillierte Sicht ber die Ausfhrung der App ohne eine erhebliche Verlangsamung.

3) Mehr Debugging Features: Man kann anzeigen welche Locks im Stack gehalten werden und zu dem Thread springen, der den Lock hlt. Es ist auerdem mglich nachzusehen, wie viele Instanzen einer Klasse existieren und welche Referenzen diese haben. Zustzlich gibt es sogenannte method-exit-events, die anzeigen, welche Werte bestimmte Methoden zurckliefern, und man kann Watchpoints auf Members setzen und die Ausfhrung unterbrechen falls auf diese zugegriffen wird bzw. diese modifiziert werden.



Name	Incl %	Inclusive	Excl %	Exclusive	Calls+Rec
4 android/webkit/LoadListener.nativeFinished ()V	66.6%	17734.382	53.2%	14161.950	14+0
3 android/webkit/LoadListener.tearDown ()V	100.0%	17734.382			14/14
6 android/view/View.invalidate (IIII)V	19.8%	3516.410			2413/2853
57 android/webkit/BrowserFrame.startLoadingResource (ILjava	0.3%	44.636			3/15
53 java/util/HashMap.put (Ljava/lang/Object;Ljava/lang/Objec	0.0%	6.223			6/326
20 android/webkit/JWebCoreJavaBridge.setSharedTimer (J)V	0.0%	2.593			2/730
378 android/view/ViewGroup.requestLayout ()V	0.0%	1.139			2/54
315 java/util/HashMap.<init> (I)V	0.0%	0.879			3/41
629 android/webkit/BrowserFrame.loadCompleted ()V	0.0%	0.285			1/1
598 android/webkit/WebView.didFirstLayout ()V	0.0%	0.231			1/2
703 android/webkit/BrowserFrame.windowObjectCleared (I)V	0.0%	0.036			1/2
5 android/webkit/JWebCoreJavaBridge$TimerHandler.handleMessa	16.3%	4342.697	0.5%	132.018	730+0
6 android/view/View.invalidate (IIII)V	15.6%	4161.341	1.2%	319.164	2853+0
7 android/webkit/JWebCoreJavaBridge.access$300 (Landroid/webk	15.1%	4025.658	0.1%	26.727	729+0
8 android/webkit/JWebCoreJavaBridge.sharedTimerFired ()V	15.0%	3998.931	8.5%	2256.801	729+0
9 android/view/View.invalidate (Landroid/graphics/Rect;)V	13.8%	3671.342	0.9%	246.190	2853+0
10 android/view/ViewGroup.invalidateChild (Landroid/view/View;La	12.4%	3298.987	6.3%	1687.629	876+1148
11 android/event/EventLoop.processPendingEvents ()V	6.3%	1674.317	0.6%	151.201	12+0
12 android/view/ViewRoot.handleMessage (Landroid/os/Message;)	4.6%	1217.210	0.0%	1.992	35+0
13 android/view/ViewRoot.performTraversals ()V	4.5%	1209.815	0.0%	7.190	34+0
14 android/view/ViewRoot.draw (Z)V	4.1%	1096.832	0.0%	11.508	34+0
15 android/policy/PhoneWindow$DecorView.drawTraversal (Landrc	3.9%	1040.408	0.0%	2.218	34+0
16 android/widget/FrameLayout.drawTraversal (Landroid/graphics,	3.8%	1023.779	0.0%	3.129	34+48
17 android/view/View.drawTraversal (Landroid/graphics/Canvas;La	3.8%	1022.611	0.1%	19.213	34+154
18 android/view/ViewGroup.dispatchDrawTraversal (Landroid/grap	3.8%	1000.413	0.2%	42.609	34+130
19 android/view/ViewGroup.drawChild (Landroid/graphics/Canvas;	3.7%	983.346	0.2%	42.926	34+150
20 android/webkit/JWebCoreJavaBridge.setSharedTimer (J)V	3.5%	929.506	0.2%	57.241	730+0
21 android/webkit/WebView.nativeDrawRect (Landroid/graphics/Ca	3.5%	923.805	3.0%	807.952	15+0
22 android/net/http/QueuedRequest.start (Landroid/net/http/Queu	3.2%	847.172	0.0%	3.556	15+0
23 android/net/http/QueuedRequest$QREventHandler.endData ()V	3.1%	828.592	0.0%	1.619	15+0
24 android/net/http/QueuedRequest.setupRequest ()V	3.1%	819.888	0.0%	5.860	15+0
25 android/net/http/QueuedRequest.requestComplete ()V	3.1%	816.585	0.0%	1.506	15+0
26 android/webkit/CookieManager.getCookie (Landroid/content/Cc	2.7%	722.837	0.0%	8.081	15+0
27 android/webkit/LoadListener.commitLoad ()V	2.6%	688.168	0.1%	17.708	58+0
28 android/webkit/LoadListener.nativeAddData ([BI)V	2.3%	621.864	1.2%	306.817	57+0
29 android/graphics/Rect.offset (II)V	2.2%	573.985	2.2%	573.985	17210+0

Find:

Fig. 8. Traceview Profile[11]

4) Verbesserte Diagnose in Exceptions und Crash Reports: ART gibt so viele Details wie mglich falls Exceptions zur Laufzeit auftreten. Zum Beispiel zeigt die java.lang.NullpointerException Informationen darber was die App mit dem Nullpointer versuchte zu tun, wie auf einen Wert zu schreiben oder der Versuch, eine Methode aufzurufen. Ein Beispiel dazu ist in Ausschnitt 2 zu sehen.

```
java.lang.NullPointerException: Attempt to
write to field 'int␣android.
accessibilityservice.AccessibilityService
Info.flags' on a null object reference

java.lang.NullPointerException: Attempt to
invoke virtual method 'java.lang.String
java.lang.Object.toString()' on a null
object reference
```

Ausschnitt 2. Exception

REFERENCES

[1] Android logo. [Online]. Available: http://upload.wikimedia.org/wikipedia/commons/thumb/d/d7/Android_robot.svg/872px-Android_robot.svg.png

[2] Dalvik (software). [Online]. Available: http://en.wikipedia.org/wiki/Dalvik_(software)

[3] Art and dalvik. [Online]. Available: https://source.android.com/devices/tech/dalvik/

[4] D. Bornstein. Dalvik vm internals. [Online]. Available: https://14b1424d-a-62cb3a1a-s-sites.googlegroups.com/site/io/dalvik-vm-internals/2008-05-29-Presentation-Of-Dalvik-VM-Internals.pdf

[5] Android kompiliervorgang. [Online]. Available: https://devmaze.files.wordpress.com/2011/05/image10.png

[6] Just in time kompilierung. [Online]. Available: http://de.wikipedia.org/wiki/Just-in-time-Kompilierung

[7] Android runtime. [Online]. Available: http://en.wikipedia.org/wiki/Android_Runtime

[8] Ahead of time compiler. [Online]. Available: http://de.wikipedia.org/wiki/Ahead-of-time-Compiler

[9] Dalvik vm art vergleich. [Online]. Available:

http://www.droidwiki.de/images/9/9e/Dalvik_VM_ART_Vergleich.png
[10] Introducing art. [Online]. Available: http://source.android.com/devices/tech/dalvik/art.html
[11] Profiling with traceview and dmtracedump. [Online]. Available: http://developer.android.com/tools/debugging/debugging-tracing.html