

2 Systemaufbau

In diesem Kapitel beschäftigen wir uns mit der Systemarchitektur von Android, die in Abbildung 2-1 skizziert ist.

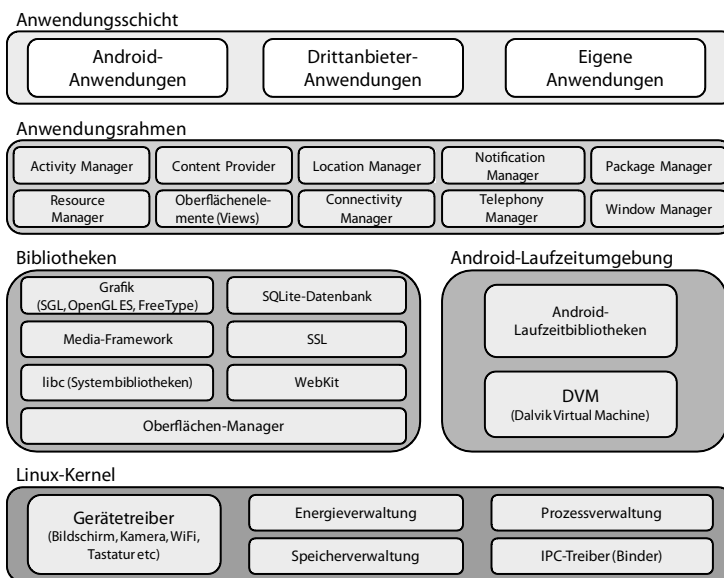


Abb. 2-1
Die Android-Systemarchitektur

2.1 Architekturübersicht

Linux Basis von Android ist ein Linux-Kernel (aktuell 2.6). Dieser stellt die erforderlichen Hardwaretreiber zur Verfügung und bildet eine bewährte Betriebssystemgrundlage.

Android-Laufzeitumgebung Kern der Laufzeitumgebung bildet die *Dalvik Virtual Machine* (DVM). Wird eine Android-Anwendung gestartet, so läuft sie in einem eigenen Betriebssystemprozess. Aber nicht nur das. Die DVM ist so klein und performant, dass Android jeder Anwendung darüber hinaus noch eine eigene DVM spendiert. Dies kostet

zwar extra Ressourcen, gibt aber erheblichen Auftrieb in puncto Sicherheit und Verfügbarkeit, da sich die Anwendungen keinen gemeinsamen Speicher teilen und ein sterbender Prozess nur eine Anwendung mit in die Tiefe reit.

Die Anwendungen selbst werden in Java geschrieben und auch zur Entwicklungszeit von einem normalen Java-Compiler in Java-Bytecode bersetzt. Die Transformation des Java-Bytecode in DVM-kompatiblen *.dex-Code bernimmt das dx-Tool, welches im Lieferumfang des Android Development Kit enthalten ist. Es kommt immer dann zum Einsatz, wenn eine lauffertige Anwendung im Dalvik-Bytecode erzeugt werden soll. Dank des Eclipse-Plug-ins bekommt man davon jedoch als Entwickler meist nichts mit.

Standardbibliotheken Die Anwendungsschicht und der Anwendungsrahmen von Android greifen auf eine Menge von Basisbibliotheken zu, die in den folgenden Abschnitten detailliert beschrieben werden. Diese C/C++-Bibliotheken stellen alle zum Betrieb von Android-Anwendungen erforderlichen Funktionalitten (Datenbank, 3D-Grafikbibliotheken, Webzugriff, Multimedia-Verwaltung, Oberflchengestaltung etc.) bereit. Die Standardbibliotheken sind fester Bestandteil des Systems und knnen von Anwendungsentwicklern nicht gendert werden.

Programmierschnittstelle/Anwendungsrahmen Der Anwendungsrahmen ist die fr Android-Entwickler interessanteste Schicht des Systems. Android stellt verschiedene Programmierschnittstellen bereit, die eine Kommunikation zwischen einzelnen Anwendungen sowie zwischen Endanwender und Anwendung realisieren. Der Umgang mit diesen Manager-Komponenten wird uns im weiteren Verlauf des Buches noch beschftigen.

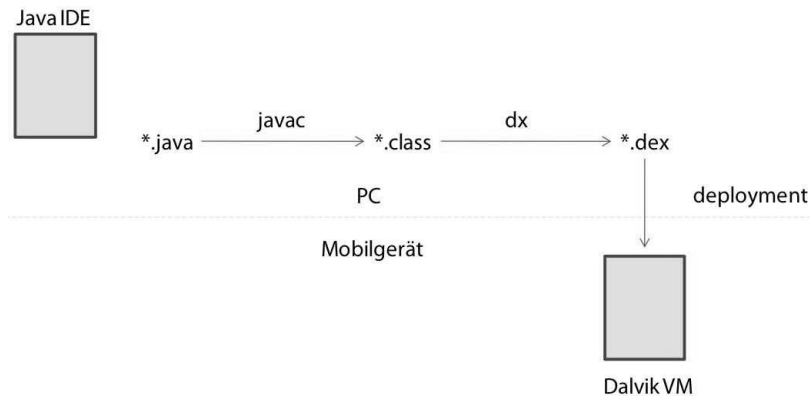
Anwendungsschicht Auf dieser Ebene des Systems befinden sich die Android-Anwendungen. Dabei kann es sich um Eigenentwicklungen oder die von Google mitgelieferten Standardanwendungen handeln. Innerhalb der Anwendungsschicht findet die Kommunikation zwischen Mensch und Maschine sowie Interaktionen zwischen Anwendungen statt. Jede Anwendung bedient sich dabei der darunterliegenden Programmierschnittstelle.

2.2 Die Dalvik Virtual Machine

Die Dalvik Virtual Machine wurde von einem Google-Mitarbeiter namens Dan Bornstein entwickelt. Benannt ist sie nach dem isländischen Ort Dalvík, in dem Verwandte von Bornstein lebten. Sie stellt das Herzstück der Android-Laufzeitumgebung dar und basiert auf der quelloffenen JVM *Apache Harmony*, wurde aber in Aufbau und Funktionsumfang an die Anforderungen mobiler Endgeräte angepasst. Wir werden an dieser Stelle lediglich die Besonderheiten der VM darstellen. Für weiterführende Recherchen zur Dalvik VM sei ein Blick in [2] empfohlen.

Wie unser Eingangsbeispiel bereits gezeigt hat, lässt sich Android komplett in Java programmieren. Dies hat den großen Vorteil, dass vorhandenes Wissen genutzt und vorhandene Entwicklungsumgebungen weiterverwendet werden können. Es stellt sich nun die Frage, wie der Java-Code Schritt für Schritt in ablauffähigen Code transformiert wird und worin die Unterschiede zwischen DVM und JVM liegen. Wir wollen dabei aber nicht zu tief in technische Details abtauchen, da das den Rahmen dieses Buches sprengen würde.

Das Schaubild in Abb. 2-2 skizziert den Weg des Java-Codes von der Erstellung bis zur Ausführung im Android-Endgerät. Oberhalb der gestrichelten Linie findet die Entwicklung in der IDE auf dem PC statt. Der Pfeil nach unten deutet den Deployment-Prozess auf das Mobilgerät an.



JVM == DVM?

Vom Code zum Programm

Abb. 2-2

Von *.java zu *.dex

Unterscheiden sich die »klassischen« JVMs, insbesondere die in der J2ME eingesetzten, von der Dalvik VM? Und wenn ja, inwieweit?

DVM != JVM

Zunächst mal hat Sun bei der Entwicklung der JVM für J2ME wenig Wert auf echte Neuerungen gelegt. J2ME bietet nur eine Untermenge der Java-Klassen des SDK. Einige Bestandteile der Sprache, wie z.B. *Reflection*, wurden weggelassen, da sie relativ viele Ressour-

DVM nutzt Register.

cen verbrauchen. Im Grunde wurde nur abgespeckt, um Java auf mobilen Endgeräten mit wenig Speicher und langsamen Prozessoren zum Laufen zu bringen. Klassische JVMs nutzen in ihrer virtuellen Prozessorarchitektur nicht aus, dass Mikroprozessoren Register haben. Register sind Zwischenspeicher direkt im Mikroprozessor, die Berechnungen über mehrere Zwischenergebnisse extrem schnell machen. Google hat mit der DVM eine Java-Laufzeitumgebung geschaffen, die Registermaschinencode verarbeitet, also die Möglichkeiten moderner Prozessoren gut ausnutzt. Hinzu kommt, dass im Mobilfunkbereich Prozessoren der britischen Firma ARM Ltd. sehr verbreitet sind. Diese registerbasierten Prozessoren sind dank ihrer speziellen RISC-Architektur sehr sparsam und zugleich schnell. Die DVM ist sehr gut an diese Art von Prozessoren angepasst und läuft darauf ebenfalls schnell und ressourcenschonend. Aufgrund dieser Tatsache kann es sich Android leisten, pro Anwendung bzw. pro Prozess eine DVM zu starten. Dies ist ein sehr großer Vorteil gegenüber J2ME, insbesondere in Bezug auf die Sicherheit der auf dem Android-Gerät gespeicherten Daten, da der Zugriff auf Dateiressourcen innerhalb der VM nur mit Aufwand zu realisieren ist. Da aber sogar normalerweise pro Android-Anwendung ein eigener Betriebssystem-User verwendet wird (ein Prozess, ein User, eine DVM), sind gespeicherte Daten zum einen über die Berechtigungen des Betriebssystems geschützt und zum anderen über die Sandbox, in der die Anwendung innerhalb der VM ausgeführt wird. Es ist daher nicht möglich, unerlaubt aus einer Android-Anwendung heraus auf die gespeicherten Daten einer anderen Anwendung zuzugreifen.

Eine DVM pro Anwendung

Kein unerlaubter Zugriff

Google hat nun einen klugen lizenzrechtlichen Trick angewandt. Für Android steht ein Großteil der API der Java Standard Edition (J2SE) zur Verfügung. Dadurch gewinnt es deutlich an Attraktivität gegenüber der stark eingeschränkten Java Mobile Edition (J2ME) von Sun. Die der DVM zugrunde liegende Standard-JVM *Apache Harmony* wird unter der Apache License vertrieben, so dass Änderungen am Code der JVM von den Geräteherstellern nicht im Quellcode ausgeliefert werden müssen.

Apache Harmony JVM

dx-Tool erzeugt Bytecode.

Die DVM selbst wird explizit nicht als Java-VM dargestellt, da der Name »Java« von Sun geschützt ist. Da sie auch keinen Java-Bytecode verarbeitet, sondern Android-eigenen DEX-Bytecode, fällt sie nicht unter die Lizenz von Sun. Der DEX-Code wird durch den Cross-Compiler, das *dx-Tool* im Android-SDK, erzeugt. Hier liegt der ganze Trick: Man programmiert in Java, erzeugt wie gewohnt mit Hilfe des Java-Compilers des Java-SDKs von Sun den Bytecode in Form von *.class*-Dateien und wendet darauf dann das *dx-Tool* an. Dieses liefert DEX-Bytecode-Dateien mit der Endung *.dex*, die zu einer fertigen Anwendung zusammengepackt werden. Das Ergebnis ist schließlich eine *.apk*-Datei,

die fertige Anwendung. Da die Programmierschnittstelle der J2SE von Sun bisher nicht patentrechtlich geschützt ist, liegt auch hier aktuell keine Verletzung bestehender Rechte vor.

2.3 Standardbibliotheken

Die Kernfunktionalität von Android wird über C/C++-Standardbibliotheken bereitgestellt, die von der Anwendungsschicht genutzt werden. Die folgenden Abschnitte stellen einige dieser Bibliotheken kurz vor. Im Praxisteil des Buches werden wir uns etwas intensiver damit befassen.

*Standardbibliotheken
in C*

LibWebCore Android stellt eine auf der quelloffenen Bibliothek *WebKit* (www.webkit.org) basierende Webbrowser-Umgebung zur Verfügung. WebKit ist Grundlage vieler Browser auf Mobiltelefonen und wird z.B. in Nokias Symbian-S60-Betriebssystem, in Apples iPhone oder aber im Google-Chrome-Browser eingesetzt.

Browser-Grundlage

SQLite Als Datenbanksystem kommt das im mobilen Bereich bewährte *SQLite* (www.sqlite.org) zum Einsatz, welches uns ein längeres Kapitel wert ist (siehe Kap. 11).

Leichte Datenbank

Media Framework Das Android Media Framework basiert auf dem quelloffenen Multimedia-Subsystem *OpenCORE* der Firma PacketVideo. Diese Bibliothek ist für die Darstellung und Verarbeitung der gängigen Multimediaformate auf dem Gerät verantwortlich. Für die Grafikdarstellung und -verarbeitung werden die Bibliotheken *SGL* (2D) und *OpenGL 1.0* (3D) genutzt.

Medien darstellen

2.4 Der Anwendungsrahmen

Als *Anwendungsrahmen* bezeichnet man eine Schicht im Android-Systemaufbau, die den Unterbau für die Anwendungen bildet. Der Anwendungsrahmen enthält die Android-spezifischen Klassen und abstrahiert die zugrunde liegende Hardware. Wir werden hier nicht die einzelnen Komponenten und Manager-Klassen vorstellen, da wir die wichtigsten im Verlauf des Buchs noch kennenlernen werden bzw. sie verwenden, ohne es unbedingt zu merken. Wenn wir zum Beispiel in Kapitel 13 über den Lebenszyklus von Prozessen reden, dann spielt der Activity Manager eine große Rolle, da er den Lebenszyklus der Activities verwaltet und sich eine Anwendung und damit ein Prozess (unter

*Unterbau für
Anwendungen*

anderem) aus Activities zusammensetzt. Uns interessiert aber nicht das »Wie«, sondern die Konsequenzen für die Anwendungsentwicklung, die sich aus der Funktionsweise des Anwendungsrahmens ergibt. Wir gehen daher einfach einen Abschnitt weiter und abstrahieren den Anwendungsrahmen in »Komponenten«.

2.5 Android-Komponenten

Anwendungen
bestehen aus
Komponenten.

Wir werden im weiteren Verlauf oft von »*Android-Komponenten*« oder schlicht »*Komponenten*« sprechen. Dies hat einen ganz bestimmten Grund: Die Sprechweise soll immer wieder bewusst machen, dass es sich bei Android um eine sehr moderne Plattform für komponentenbasierte Anwendungen handelt. Ziel der Softwareentwicklung unter Android soll es sein, nicht jedesmal das Rad neu zu erfinden. Ausgehend von einigen vorinstallierten Standardanwendungen lassen sich Anwendungen entwickeln, die Teile der Standardanwendungen verwenden. Daraus entstehen wieder neue Anwendungen, die ihrerseits vielleicht wieder zum Teil von anderen Anwendungen genutzt werden.

Ein Beispiel: Wer ein Android-Mobiltelefon kauft, findet darauf die vorinstallierten Anwendungen »*Contacts*« und »*Phone*«. Wenn Sie selbst Anwendungen für Android schreiben, können Sie sich aus der Datenbank der Anwendung »*Contacts*« eine Telefonnummer holen und sie in der Activity der Anwendung »*Phone*« wählen lassen. Die Anwendungen sind »offen«. Sie können über das Berechtigungssystem anderen Anwendungen erlauben, einige ihrer Komponenten zu verwenden. Was sind nun diese Komponenten?

Als Komponenten werden wir in Zukunft die folgenden Begriffe bezeichnen, die in späteren Kapiteln wesentlich detaillierter beschrieben werden:

Activity Anwendungen, die mit dem Anwender interagieren, brauchen mindestens eine Activity, um eine Oberfläche darzustellen. Activities sind sichtbar und können miteinander zu einer komplexeren Anwendung verknüpft werden. Sie kümmern sich um die Darstellung von Daten und nehmen Anwendereingaben entgegen. Sie sind jedoch Komponenten einer Anwendung, die mehr machen als die reine Darstellung von Daten und Formularen. Genaueres erfahren wir in Kapitel 5.

View

Service Nicht jeder Teil einer Anwendung braucht eine Oberfläche. Wenn wir Musik im Hintergrund abspielen wollen, können wir die Bedienung des Players einer Activity überlassen und das Abspielen der

Controller

Musik durch einen Service erledigen lassen, auch wenn die Bedienoberfläche schon geschlossen wurde. Ein Service erledigt Hintergrundprozesse und wird in Kapitel 8 näher erklärt.

Content Provider Nicht jede, aber viele Anwendungen bieten die Möglichkeit, Daten zu laden oder zu speichern. Ein Content Provider verwaltet Daten und abstrahiert die darunterliegende Persistenzschicht. Er kann über Berechtigungen seine Daten einer bestimmten Anwendung oder auch vielen Anwendungen zur Verfügung stellen. Er hat eine definierte Schnittstelle und wird darüber lose an Anwendungen gekoppelt. Wir beschäftigen uns in Kapitel 12 mit Content Providern.

Model

Broadcast Receiver Durch die komponentenbasierte Anwendungsentwicklung unter Android ist es notwendig, zwischen Betriebssystem und Anwendungen zu kommunizieren. Auch die Kommunikation zwischen Anwendungen ist möglich. Intents (»Absichtserklärungen«) als Objekte zur Nachrichtenübermittlung haben wir schon kurz in Listing 1.9 kennengelernt und werden das Thema später noch vertiefen. Broadcast Receiver lauschen jedoch als Komponente auf Broadcast Intents, die auf Systemebene verschickt werden und z.B. über Störungen der Netzwerkverbindung informieren oder über einen schwachen Akku. Mehr dazu erfahren wir in Kapitel 9.

Verbindung zum System

2.6 Die Klasse Context

Die Klassen Activity und Service sind von der abstrakten Klasse `android.content.Context` abgeleitet. Context gehört zur Android-Plattform und bildet eine Schnittstelle für Activities und Services zur Laufzeitumgebung. Über Methoden der Klasse Context lassen sich allgemeine Informationen über die Anwendungen erhalten und Methoden auf Anwendungsebene aufrufen. Damit ist zum Beispiel das Starten eines Service gemeint oder das Schreiben in eine Datei.

Schnittstelle zur Laufzeitumgebung

In der Anwendungsentwicklung verwendet man oft Methoden der Klasse Context, wenn man Activities oder Services implementiert. Der Begriff »Context« wird dabei immer wieder verwendet, wenn man die Umgebung der Anwendung meint. Zu dieser Umgebung gehören unter anderem

- der Classloader,
- das Dateisystem,
- die Berechtigungen der Anwendung,

- die verfügbaren Datenbanken,
- die anwendungseigenen Bibliotheken,
- die anwendungseigenen Ressourcen (Bilder etc.),
- die Bildschirmhintergrund,
- der Zugriff auf andere Komponenten der Anwendung
- etc.

Context via this

Man spricht in diesem Zusammenhang meist von dem Context, wenn man das `this`-Attribut eines Service oder einer Activity meint. Aber wie man an der Auflistung sieht, ist Context ein recht komplexer Begriff und steht für die Verbindung der Anwendung zu ihrer Laufzeitumgebung (und darüber hinaus, wie wir später sehen werden!).

Der Context einer Anwendung wird uns noch oft begegnen, und es ist wichtig, sich diesen Begriff zu merken und ein Gefühl für seine Bedeutung zu entwickeln.