1 Native Apps unter Android

Ein native App ist eine Anwendung, welche spezifisch dafür entwickelt wurde auf einem mobilen Gerät unter einem bestimmten Betriebssystem eingesetzt zu werden. Die Anwendung kann dabei schon vorinstalliert sein oder über einen Store bezogen werden, frei oder auch kostenpflichtig. Auch haben Unternehmen die Möglichkeit eigene interne Stores einzurichten welche nicht öffentlich zugänglich sind. Dies erleichtert die Distribution der Apps innerhalb des Unternehmens. Als Fallbeispiel wird in diesem Dokument Android als Mobilplattform unter die Lupe genommen.

Android ist ein quelloffenes mobiles Betriebssystem, das von der Open Handset Alliance [1] entwickelt wird, wobei Google das Hauptmitglied ist. Die erste offizielle Version wurde am 21. Oktober 2008 veröffentlicht. Die primär zur Entwicklung unter Android vorgesehene Programmiersprache ist Java und wird deshalb im Android-Kontext als native bezeichnet.

1.1 User Experience

Ein wichtiger Punkt in der User Experience stellt das Look-and-Feel einer Applikation dar. Bereits beim ersten Betrachten der Einstiegsseite einer App bildet sich ein Benutzer eine subjektive Meinung anhand des Designs. Ein Benutzer erwartet, dass verschiedene Applikationen ein konsistentes Aussehen haben und sich auch ähnlich bedienen lassen.

Gerade in diesen Punkten gab es in den Anfangsstadien von Android grosse Probleme, da Google die verschiedenen designtechnischen Entscheidungen den Entwicklern überliess. Dadurch entstanden viele - vor allem gratis verfügbare - Apps, die sich vom Aussehen und von der Bedienung signifikant unterschieden. Google hat jedoch mit der neusten Android Version stark nachgebessert und hat ein moderneres Grunddesign geschaffen und bietet dem Entwickler eine Auswahl an Standard-GUI-Elementen zur einfachen Erstellung einer App an, die vom Design und der Bedienung konsistent ist. Google hat ausserdem einen Design Guide [2] und eine Best Practises for User Experience & UI [3] Seite veröffentlicht.

Ein weiterer wichtiger Punkt ist die Integration in das Gesamtsystem. Native Apps haben Zugriff auf eine Vielzahl von Services, welche die User Experience zusätzlich steigern und im Moment in HTML5 nicht zur Verfügung stehen. Die Folgende Auflistung zeigt eine Auswahl der häufigsten verwendeten Services:

- **(Push-) Notifications**. Die obere Bildschirmleiste in Android ist für Benachrichtigungen vorgesehen. Diese kann zudem mit dem Finger heruntergezogen werden, um die Liste der Benachrichtigungen darzustellen und weitere

[1] http://www.openhandsetalliance.com
[2] http://developer.android.com/design/index.html
[3] http://developer.android.com/training/best-ux.html

Aktionen auszulösen. Apps können diesen Service nutzen um eigene Nachrichten zu platzieren.

- **Hintergrund-Dienste**. Bestimmte Apps sind dafür vorgesehen im Hintergrund zu laufen, um ihren Zweck auch dann zu erfüllen, wenn sie nicht gerade auf dem Bildschirm dargestellt werden, z.B. eine Musik-Player App. Auch eine Synchronisation, z.B. mit einem Cloud-Dienst, ist ein Hintergrund laufender Tasks.

- **App-Menü**. Jede App hat ihr eigenes Menü, das durch einen entsprechenden Hardware-Button aufgerufen werden kann. In diesem kann eine App frei ihre Funktionen unterbringen.

1.2 Performance

Im diesem Abschnitt wird kurz die Architektur von Android vorgestellt (siehe Abb. **??**).

Als Basis wird ein Linux-Kernel verwendet. Er übernimmt die Speicher- und Prozessverwaltung und stellt die Treiber und Schnittstellen für den direkten Hardwarezugriff zur Verfügung. Die Laufzeitumgebung (Android Runtime) zur Ausführung von Apps besteht aus der Dalvik Virtual Maschine, welche eine spezielle Implementierung der Java VM ist, und der Java-Standardbibliothek. Zudem gibt es eine grosse Standardbibliothek für Android, die wichtige Funktionen wie das GUI oder Benachrichtigungen anbietet. Obwohl hauptsächlich Java als Programmiersprache eingesetzt wird, sind performancekritische Teile der Bibliothek in C/C++ geschrieben, z.B. Multimedia-Verarbeitung, 3D via OpenGL oder die integrierte SQLite Datenbank.

Obwohl in den letzten Jahren die Geschwindigkeit im Web signifikante Verbesserungen erfahren hat und teilweise auch mit nativen Apps mithalten kann, gibt es trotzdem immer den zusätzlichen Overhead der Browser-Laufzeitumgebung. Die Performance ist natürlich auch immer vom Anwendungszweck der App abhängig. Eine Newsreader-App wird ein Gerät sowohl Nativ als auch HTML5 nie an die Leistungsgrenzen bringen. Ganz anders sieht es mit 3D-Games aus, wo bis auf Ebene Hardware optimiert wird.

In Android kann zudem über das NDK (Native Development Kit) aus Java auf Funktionen die in C/C++ implementiert wurden, zugegriffen werden. Dadurch können performancekritische Teile ausgelagert werden. Durch diesen Faktor ist bei dieser Art von Applikation der native Weg definitiv vorzuziehen.

1.3 Zugriff auf Gerätefunktionen

Native Apps sind so gestaltet, dass sie ohne Umweg Zugriff auf die Gerätefunktionen haben. Für alle vorgesehenen Funktionen gibt es entsprechende APIs, die in der Android Standardbibliothek integriert sind.

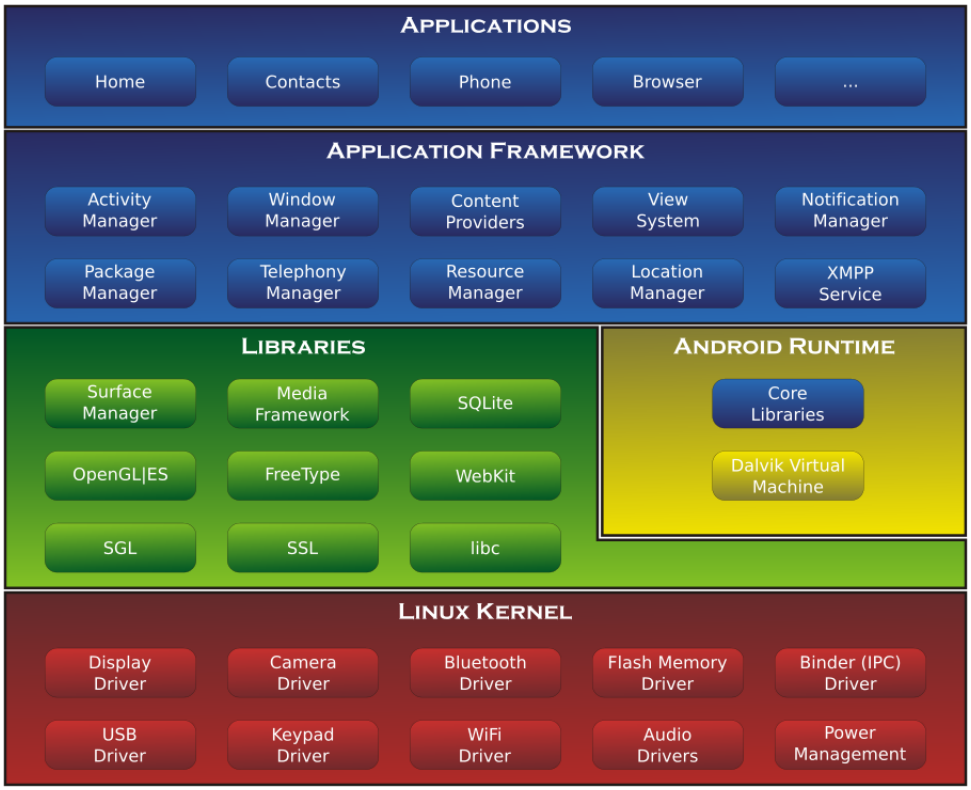

Abb. 1. Systemarchitektur von Android, Quelle: http://commons.wikimedia.org/wiki/File:Android-System-Architecture.svg

1.3.1 GPS Unter Android sind die Schlüssel-Klassen zur Positionsbestimmung im Location-Package [4] zu finden. Kernstück bildet dabei die Klasse *LocationManager*, bei welcher ein App-Entwickler seinen eigenen *LocationListener* registrieren und ab diesem Zeitpunkt von Android Positions-Updates erhalten kann.

Einrichtung eines *LocationListeners*:

[4] http://developer.android.com/reference/android/location/package-summary.html

```java
// Acquire a reference to the system Location Manager
LocationManager manager =
(LocationManager) this.getSystemService(Context.LOCATION_SERVICE);

// Define a listener that responds to location updates
LocationListener listener = new LocationListener() {
    public void onLocationChanged(Location location) {
      // Called when a new location is found
      // by the network location provider.
      makeUseOfNewLocation(location);
    }

    public void onStatusChanged(String provider, int status, Bundle extras) {}
    public void onProviderEnabled(String provider) {}
    public void onProviderDisabled(String provider) {}
  };

// Register the listener with the Location Manager to receive location updates
manager.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0, listener);
```

Auch unter Android kann die Position einmalig angefragt werden:

```java
String provider = LocationManager.GPS_PROVIDER;
Location lastKnownLocation = manager.getLastKnownLocation(provider);
```

Zusätzlich zu dieser Methode kann Android auch Positionsdaten anhand von Wi-Fi Netzwerken oder anhand GSM Cell-IDs liefern, die sich in der Nähe befinden. Diese können durch den Entwickler alternativ oder ergänzend zum GPS genutzt werden. Die Entscheidung wird aber bewusst dem Entwickler überlassen, da die eingesetzte Technik stark vom Anwendungszweck der Anwendung abhängig ist, da u.a. die Akkulaufzeit und die Genauigkeit beeinflusst werden können.

1.3.2 Sensoren Praktisch jedes Smartphone hat heute diverse Sensoren integriert, die bei der Entwicklung einer App genutzt werden können. In Android gibt es drei Arten von Sensoren, die genutzt werden können:

- **Motion Sensors** messen die Beschleunigungs- und Rotationskräfte entlang der drei Raumachsen. In dieser Kategorie befinden sich Gyroskop, Beschleunigungsmesser, Schwerkraftsensor sowie Rotationsvektorsensoren.

- **Environmental sensors** stellen diverse Parameter der Umgebung, beispielsweise Lufttemperator und -druck, Helligkeit, Feuchtigkeit bereit. Diese Kategorie beinhaltet Barometer, Thermometer und Photometer.

- **Position sensors** dienen zur Bestimmung der Geräteposition. Diese Kategorie enthält Orientierungs- und Magnetsensoren.

Welche Sensoren jedoch effektiv genutzt werden können ist vom Gerät abhängig und kann variieren.

Beispielcode zur Abfrage des Magnetfeldsensors:

```
SensorManager manager =
  (SensorManager) getSystemService(Context.SENSOR_SERVICE);
if (manager.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD) != null) {
  // Success! There's a magnetometer.
} else {
  // Failure! No magnetometer.
}
```

1.3.3 Audio- / Videoinput Eine Vielfalt von multimedialen Funktionen gibt es im Media-Package [5] von Android. Die wichtigsten Funktionen werden von der Klasse *MediaRecorder* bereitgestellt.

Aufnahme eines Audio-Signals:

```
mRecorder = new MediaRecorder();
mRecorder.setAudioSource(MediaRecorder.AudioSource.MIC);
mRecorder.setOutputFormat(MediaRecorder.OutputFormat.THREE_GPP);
mRecorder.setOutputFile(mFileName);
mRecorder.setAudioEncoder(MediaRecorder.AudioEncoder.AMR_NB);

mRecorder.prepare();
mRecorder.start();
```

Aufnahme eines Video-Signals mit Live-Vorschaubild:

[5] http://developer.android.com/guide/topics/media/index.html

```
mCamera = Camera.open();
mMediaRecorder = new MediaRecorder();

// Step 1: Unlock and set camera to MediaRecorder
mCamera.unlock();
mMediaRecorder.setCamera(mCamera);

// Step 2: Set sources
mMediaRecorder.setAudioSource(MediaRecorder.AudioSource.CAMCORDER);
mMediaRecorder.setVideoSource(MediaRecorder.VideoSource.CAMERA);

// Step 3: Set a CamcorderProfile (requires API Level 8 or higher)
mMediaRecorder.setProfile(CamcorderProfile.get(CamcorderProfile.QUALITY_HIGH));

// Step 4: Set output file
mMediaRecorder.setOutputFile(mFileName);

// Step 5: Set the preview output
mMediaRecorder.setPreviewDisplay(mPreview.getHolder().getSurface());

// Step 6: Prepare configured MediaRecorder
mMediaRecorder.prepare();

// Step 7: Start capturing video
mMediaRecorder.start();
```

1.4 Einnahmemöglichkeiten

Die Haupteinnahmequelle in der Android-Welt stellt der Google Play Store dar. Über ihn können Apps zentral angeboten werden. Dazu gibt es drei Bezahlmodelle, die direkt in die Plattform integriert sind und genutzt werden können:

- **Direktverkauf**. Eine App kann in Google Play einmalig gekauft werden. Die Preisgestaltung ist dabei dem Entwickler überlassen. Diese hat jedoch einen hohen Einfluss auf den Erfolg der App, besonders da in Google Play viele werbefinanzierte gratis Apps zu finden sind.

- **Eingebettete Werbung**. Bei dieser Methode wird in einer Android App mittels Google Ads ein Werbebanner eingeblendet. Nennenswerte Einnahmen sind aber schwierig, da eine App sehr oft aufgerufen werden und ein Benutzer ein solches Banner auch anklicken müsste.

- **In-App Bezahlung**. Die In-App Bezahlung kann in eine App eingebunden werden, um weitere digitale Inhalte bei Bezahlung für einen Benutzer freizugeben. Häufig genutzt wird dies z.B. bei Games, um zusätzliche Levels oder

Fähigkeiten freizuschalten.

1.5 Cross Plattform Entwicklung

Android ist die einzige Plattform, die Java als Programmiersprache verwendet. Dies hat zwar den Vorteil, dass die gesamte Java Standard Library sowie Libraries von Drittherstellern aus dem bestehenden Java-Umfeld weiterverwendet werden können, jedoch ist es damit nicht möglich für andere Plattformen zu entwickelt. Es ist auch nicht möglich Programme von anderen Plattformen, z.B. iOS, auf einem Android-Gerät auszuführen.

Zusätzlich zum Java-basierten Android SDK gibt es das Android NDK (Native Development Kit). Damit ist es möglich, Apps mit Hilfe von betriebssystemnahen Sprachen wie C/C++ zu entwickeln. Das Problem hierbei ist, dass bei diesem Weg nur wenige API-Funktionen zur Verfügung stehen (keine GUI-Elemente, fehlender Kamerazugriff, etc.) oder nicht stabil sind und bei der nächsten Version wieder ändern können.

Es ist jedoch möglich, gewisse Teile eines Programms, z.B. die Business-Logik, in eine C/C++ Library auszulagern und diese sowohl unter Android als auch iOS zu verwenden. Eine vollständig plattformunabhängige Entwicklung ist jedoch nicht möglich.

1.6 Fragmentierung

Durch die hohe Verbreitung und ständige Weiterentwicklung von Android gibt es derzeit mindestens drei Versionen (Abb. **??**), die bei der Entwicklung berücksichtigt werden sollten. Hinzu kommt, dass jeder Smartphone-Hersteller der Android als Betriebssystem einsetzt, eine eigene Version mit Anpassungen erstellen kann und dadurch ein automatisches Update auf eine neue Version nicht gewährleistet ist.

1.7 Verfügbarkeit von Entwicklerressourcen

Da Android auf Java als Programmiersprache setzt, gibt es im aktuellen Arbeitsmarkt sehr viele Java-Entwickler. Jedoch ist nicht jeder Java-Entwickler automatisch ein Android-Entwickler. Ein Entwickler für Android zeichnet sich durch sein Verständnis der mobilen Plattform aus und kennt sich mit den APIs der Android-Standardbibliothek aus. Deshalb benötigt auch ein Java-Entwickler entsprechende Einarbeitungszeit. Zusätzlich benötigt ein Entwickler für mobile Geräte ein gutes Verständnis des mobilen Ökosystems und von Bedienkonzepten, denn diese unterscheiden sich verglichen mit ihren Desktop-Pendants erheblich.

Wie die Abbildung **??** zeigt, ist die Anzahl an mobilen Entwicklern in den letzten Jahren stark gestiegen und wird in Zukunft voraussichtlich auch nicht abnehmen.

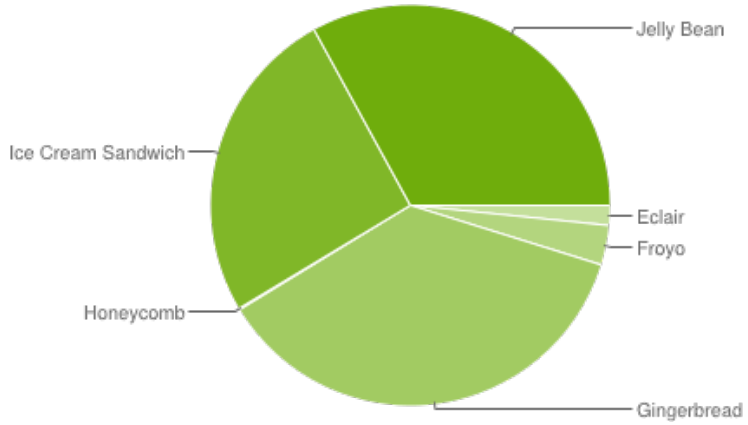

Abb. 2. Android Erhebung vom 3. Juni 2013, Quelle: http://developer.android.com/about/dashboards/index.html#Platform

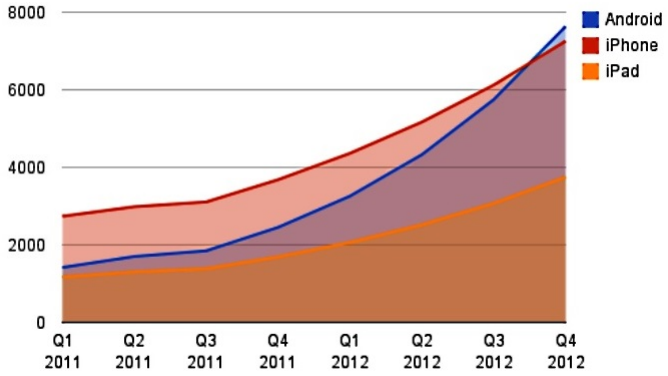

Abb. 3. Wachstumsrate der Anzahl mobiler Entwickler, Quelle: http://e27.co/2012/03/06/fastest-growing-online-job-categories-in-2011-and-whats-next/

1.8 Distribution

Die Distribution einer Android-App erfolgt direkt über den Google Play Store. Eine App kann dort hochgeladen und konfiguriert werden. Dadurch kann die eigene App auch direkt durch eine Suche in Google Play gefunden werden und mit einem Klick direkt auf das Smartphone heruntergeladen und installiert werden.

Bei nativen Apps ist Veröffentlichung in einem Store meist jedoch mit zusätzlichem Aufwand verbunden. Man benötigt einen Entwickleraccount, der z.B. bei Apple etwas kostet, um seine Apps hochzuladen. Nachdem man seine App dann hochgeladen hat, wird die App vom Betreiber des Stores überprüft. Im Falle ei-

ner komplexen App kann dies den Zeitpunkt bis zur effektiven Veröffentlichung stark verzögern. Bugs oder Sicherheitslücken können somit nicht wie bei HTML5 sofort behoben und eine Lösung veröffentlicht werden.

Ein Vorteil dieses Distributionsverfahrens sind jedoch nützliche Statistiken über die Installationen, Deinstallationen und Bewertungen der App. Auch Kommentare von Nutzern sind direkt in Google Play ersichtlich. Dies erleichtert die zukünftige Entwicklung und Verbesserung der App. Bei einer eigenen Verbreitung der App müsste eine solche Feedback-Plattform selber aufgebaut werden.