Sviluppo Applicazioni Mobili

Federico Matteoni

A.A. 2019/20

Indice

Introduzione

Vincenzo Gervasi, gervasi@di.unipi.it
circe.di.unipi.it/ gervasi/main/
developer.android.com

Modalità d'esame Sviluppo di un'app, proposta dallo studente ma concordata con il docente. Presentazione dell'app con ispezione del codice e domande "teoriche" su aspetti non coperti dal progetto. No compitini.
3 criteri: applicazione mobile, non deve avere senso su applicazione web o su computer. diversità, almeno tre framework presentati. progetto adeguato a esame di 6 CFU.

Capitolo 1

Programmazione Android

1.1 Breve Storia di Android

2007 Telefonini Nokia, Palm, Windows CE e BlackBerry. Tutti **sistemi fortemente proprietari**, spesso con versioni frammentate e di difficile manutenzione. Giravano su una versione di Java portatile ma fortemente limitata, **JavaME**.

Novembre 2007 La **Open Handset Alliance**, formata da vari produttori di telefoni, pubblica la **Open Platform for Mobile Handset**.
Era il 5 Novembre, **7 giorni dopo rilasciano Android**.
Chi c'era dietro, tra le altre: Google, eBay, China Mobile, HTC, Intel, LG, Motorola, NTT DoCoMo, Qualcomm, nVidia, Samsung, Sprint Nextel, Telecom Italia, Telefònica, Texas Instrument, T-Mobile. Ovvero vari produttori di telefoni, di chip, fornitori di servizi e di telefonia.

Android Il 12 Novembre viene rilasciato **Android**
Rilasciato su licenza Apache, **basato su Linux 2.6** e sviluppato su Eclipse, Java e Python. Il kernel **era completo e standard**, non era personalizzato.
Sviluppato da **Android Inc.**, startup californiana nata nel 2003 a Palo Alto, acquistata da Google nel 2005 e brevetti registrati nel 2007. Lo sviluppo è avvenuto in gran segreto, brevetti registrati all'ultimo così da non destare sospetti a Microsoft e Apple. Fondata da **Andy Rubin**.

Adesso Dal 2007 sono state rilasciate numerose **versioni**, dai *codename* ispirati a nomi di dolciumi in ordine alfabetico... fino ad Android Q. Adesso parleremo principalmente di software, ma non bisogna dimenticare il lato **hardware**: potenza di calcolo, efficienza della batteria, sensori e schermi. I produttori, inoltre, hanno **poco interesse ad aggiornare i telefoni vecchi**: il principale problema è che per ogni modello e ogni aggiornamento bisogna far omologare e convalidare la parte telefonica, quindi servono mesi di test e tanti soldi. Per cui è meglio **spingere gli utenti a comprarne di nuovi** $\Rightarrow$ ***frammentazione***.

Software Ogni versione è (*quasi*) sempre **pienamente compatibile con le precedenti**: i cambiamenti nelle API sono identificati da un **API Level**.
Le applicazioni possono quindi dichiarare:

API Level minimo di cui hanno bisogno per funzionare

API Level targe per cui sono state scritte

API Level massimo oltre il quale non funzionano più (pessima idea, sconsigliato, obsoleto e ignorato già da Android 2.0.1)

I vincoli vengono verificati dal market e dalle procedure di aggiornamento del S.O.

Rispetto iOS, i quali dispositivi vengono (*quasi*) sempre aggiornati alla versione più recente, Android tende a diffondere gli aggiornamenti più lentamente: l'Android più recente è sempre una nicchia.

Supporto Google cerca di supportare più o meno all'infinito le vecchie versioni del S.O. con le **librerie di compatibilità** (`libcompat`).

Codice che le applicazioni possono includere nel loro "eseguibile"

Simula le funzioni delle versioni più recenti sulle versioni più vecchie

Inoltre, parte delle funzioni del S.O. sono incorporate nei **Google Play Services**, libreria aggiornabile dal market.
Un **grosso ostacolo** è la **customizzazione** (skinning) del sistema.

1.2 Ant e Gradle

Ant antiquato
Gradle più moderno.

Gradle Sistema di build avanzato, configurabile. Distribuito nel senso di risorse per lo sviluppo sparse in rete tramite URL. Fill-in del manifest (manifest contiene metadati per il s.o.), gradle genera e mantiene aggiornato il manifest.

`compileSdkVersion` per fill-in manfest e `buildToolsVersion` per scaricare tools se non presenti.
`Lint` analisi statica di codice per warnings e errori sintattici della scrittura del codice.

1.3 Architettura Android Studio/Gradle

IntelliJ con plugin: android plugin. android designer, android gradle adapter.
Si appoggia all'android SDK e a Gradle (tool separato, con plugin android e anch'esso collegato all'SDK)
Inoltre c'è il progetto, con `.properties` con config per l'ambiente di sviluppo (come dove si trova il compilatore ecc.) e `build.gradle`.

Capitolo 2

Architettura Android

2.1 Struttura

Kernel Linux Kernel Linux standard, senza fare dei derivati. Adattamenti per telefono eseguiti tramite moduli del kernel. Come display driver, driver per la tastiera keypad, driver camera, wifi, flash, audio, driver binder (IPC) che cura inter-process communication (diversamente da socket e FIFO, che non andavano bene per Android. Su android non ci sono solo file, ma oggetti con metodi, e FIFO/socket adatte per trasferire flussi di byte. Binder fa comunicare processi in termini object-oriented). Power management driver.

Librerie Librerie `.so`, che fanno tantissime cose. Tra cui surface manager (equivalente dei window system X o wayland), OpenGL ES per la grafica 3D, FreeType, SSL (HTTPS e Secure Socket Layer), WebKit, SQLite, libc (scanf, strlen...).

Android Runtime In aggiunta alle librerie, Macchina Virtuale che esegue codice applicazioni Android (Dalvik), insieme alle core librearies (garbage collector, heap, memoria...)

Separazione tra mondo Java e mondo del codice eseguibile ARM Da qui in giù c'è Java, sopra è C.

Application Framework S.O. rappresentato da oggetti nello heap. Librerie: package manager, telefony manager, activity manager, window manager, location manager (GPS), notification manager

Applicazioni Tra cui servizi interni: home, contatti, telefono, browser... Firmate a chiave asimmetrica. Platform key per applicazioni che usano funzioni critiche, chiave che firma il kernel.

2.2 Dalvik

2.3 ART

Android RunTime