



Università degli Studi di Padova
Dipartimento di Ingegneria dell'Informazione

Corso di Laurea Triennale in Ingegneria
dell'Informazione

Tesina di laurea triennale

Sviluppo di App per sistema operativo Android

Aspetti generali

Candidato:
Forcolin Fabio
Matricola 1001968

Relatore:
Andrea Pietracaprina

Anno Accademico 2012–2013

Indice

1	Introduzione	1
2	Il sistema operativo mobile Android	3
2.1	Perchè Android	3
2.2	La diffusione di Android nel mondo	5
2.3	L'evoluzione di Android	6
2.4	La grafica di Android	8
3	L'architettura di Android	13
3.1	Linux Kernel	13
3.2	Libraries	14
3.2.1	Android Runtime	15
3.3	Application Framework	16
3.4	Applications	17
4	Le App	19
4.1	Android Graphics Guidelines	20
4.2	Decidere per quali versioni sviluppare	27
4.2.1	Come risolvere i problemi di compatibilità	28
4.3	Gli strumenti necessari	29
5	Struttura di un'App	33
5.1	Da cosa è composta un'app	33
5.2	View e GroupView	35
5.3	Activity	36
5.4	Le azioni: Intent	36
5.5	Ciclo di vita di un'Activity	37
6	Come strutturare l'applicazione	39
6.1	L'Action Bar	40
6.2	Il Navigation Drawer	43
6.3	Le notifiche	43
6.4	I Widget	45

7	Guadagnare con le App	47
7.1	Tecniche remunerative	47
7.2	Pubblicizzare l'app	49
7.3	Controllare le distribuzioni	50
8	Pubblicare la propria App	55
9	Conclusioni	59
	Bibliografia	61

Elenco delle figure

2.1	Le varie forme del robottino di Android	4
2.2	Android ICS	5
2.3	iOS 6	5
2.4	Windows Phone 8	5
2.5	La distribuzione dei sistemi operativi [3].	6
2.6	La distribuzione dei sistemi operativi[34].	7
2.7	L'evoluzione dell'interfaccia nelle varie versioni di Android	8
2.8	Distribuzione delle versioni [2]	9
2.9	Andamento della diffusione nel tempo (Aprile 2013)	9
2.10	Schermata Home	10
2.11	AllApp Screen	11
2.12	Le barre [12]	11
3.1	L'architettura di Android [6]	13
3.2	System e Process Machine	16
4.1	Viber prima della grafica Holo	21
4.2	Viber con grafica Holo	21
4.3	Shazam, prima e dopo la grafica Holo	22
4.4	Android in un telefono HTC con interfaccia Sense	23
4.5	Android in un telefono Samsung con interfaccia Touchwiz	23
4.6	Interfacce uniformate	23
4.7	DPI: Dots Per Inch	24
4.8	Holo Light	25
4.9	Holo Dark	25
4.10	Holo Light con Dark Action Bar	25
4.11	Writing Style	27
4.12	Cartella decompressa SDK	30
4.13	SDK Manager	30
4.14	Bottone JDK	31
4.15	Scelta del S.O.	31
4.16	Finestra Variabili d'Ambiente	32
5.1	Struttura di un'App: AndroidManifest, src, resources	34
5.2	Ciclo di vita di un'Activity [13]	38

6.1	Struttura di una App	40
6.2	Action Bar	40
6.3	Contextual Action Bar	41
6.4	Orari treni	42
6.5	Orari treni: grafica Holo	42
6.6	Navigation Drawer	43
6.7	Gmail: il Navigation Drawer	44
6.8	La struttura di una notifica	44
6.9	Calendario: notifiche e azioni	45
6.10	Widget: informazione, controllo, ibrido	46
6.11	Widget di collezione	46
7.1	Google Play: commenti e voti di una app [16].	49
7.2	I badge di Android	50
7.3	Google Play Developer Console: Paesi di distribuzione [15]	51
7.4	Google Play Developer Console: statistiche [15]	52
8.1	Google Play: gestione app	55
8.2	Google Play: dettagli account	56
8.3	Eclipse: firmare l'app	57

Introduzione

L'utilizzo degli smartphone, i telefonini intelligenti, è un fenomeno che da pochi anni a questa parte è cresciuto esponenzialmente, grazie alle nuove tecnologie che permettono di costruire prodotti sempre più affidabili e completi e ad un costo sempre minore. Questo ha permesso infatti di potersi estendere su una fascia di clienti sempre maggiore: i primi smartphone infatti avevano un prezzo decisamente elevato e quindi non accessibile a tutti, mentre ora se ne può comprare uno senza troppe pretese per poche decine di euro. Si è inoltre passati da telefoni la cui unica funzione era effettuare chiamate, a telefoni in cui non manca nulla: fotocamera, GPS, lettori multimediali e quant'altro. Da un lato, l'introduzione di queste funzioni ha reso il dispositivo più complicato da utilizzare, soprattutto per una clientela non giovane, dall'altro lo ha reso più completo e versatile, creando numerose opportunità agli sviluppatori per inventare e re-inventare applicazioni per svolgere le funzioni più svariate.

Esistono però moltissimi tipi di smartphone che montano sistemi operativi differenti: si va da Android a Apple, da RIM a Windows Phone. Ognuno di questi ha caratteristiche diverse, che lo rende migliore per una certa clientela piuttosto che per un'altra. Tra i tanti sistemi operativi presenti, uno di cui vale la pena parlare è Android. Tutto il mondo Android ruota attorno alla parola "open source": infatti si ha libero accesso a tutti gli strumenti utilizzati dagli sviluppatori stessi di Android e quindi il potenziale delle applicazioni sviluppate da terzi non è soggetto a limitazioni, visto anche che applicazioni native e di terze parti hanno lo stesso peso. Inoltre, chiunque può sviluppare per Android senza dover acquisire software o licenze particolari.

L'idea di questa tesi, che è composta da due parti, è nata in seguito ad un progetto sviluppato insieme ad un collega di studi, Angelo Trevisiol, che tratterà la seconda parte, relativa alla programmazione delle app e un caso di studio specifico: la gestione dei viaggi. Il caso di studio fa riferimento proprio ad un'app sviluppata per partecipare ad un concorso indetto da Samsung. L'app in questione svolge la funzione di diario di viaggio, nel quale annotare viaggi, spese, appunti e foto, per organizzare e rivedere i viaggi effettuati.

L'obiettivo di questa parte della tesi è quello di illustrare gli aspetti generali di Android nell'architettura e nello sviluppo delle famose app.

Nel primo capitolo parleremo dei motivi per cui Android è migliore rispetto

agli altri sistemi operativi e li confronteremo, la sua diffusione nel mondo e la sua evoluzione sia dal punto di vista grafico che del software.

Nel secondo capitolo entreremo più nel dettaglio, analizzando l'architettura di Android: il kernel, le librerie, il framework e tutti gli altri componenti.

Di seguito verrà trattato uno degli argomenti più interessanti: le app. Le analizzeremo dal punto di vista dell'utilità e della grafica, soffermandoci sulle linee guida proposte da Android e vedremo anche come decidere per quali versioni è più conveniente sviluppare la propria applicazione e come crearne una.

Nel capitolo successivo presenteremo la struttura di un'applicazione, descrivendone il ciclo di vita ed i principali elementi che la costituiscono e rendono possibile l'interazione con l'utente e la visualizzazione dei contenuti.

Il quinto capitolo tratta invece i componenti principali della grafica Android, come l'Action Bar e i Widget e, una volta descritto il loro utilizzo, vedremo come questi rispettano perfettamente le linee guida Android ed un esempio del loro utilizzo in app famose.

Gli ultimi due capitoli sono dedicati ad un aspetto più economico delle applicazioni: verranno elencate e analizzate le tecniche remunerative ed i vari modi per aumentare la visibilità della propria applicazione e come pubblicare la propria.

Il sistema operativo mobile Android

2.1 Perchè Android

Il mercato della telefonia mobile ha conosciuto, negli ultimi anni, un notevole sviluppo. In breve tempo si è passati da dispositivi che oggi ci parrebbero enormi e poco pratici, con lo schermo in bianco e nero e l'antenna sporgente, a dei veri e propri mini computer, sempre più curati nel design e nelle prestazioni, i cosiddetti smartphone. Chi oggi vuole acquistare uno di questi telefoni intelligenti, di sicuro si imbatte in un logo molto conosciuto: un simpatico robottino verde. Questo è il simbolo di Android, Figura 2.1, una piattaforma di esecuzione gratuita per dispositivi mobili, creata nel 2003 da Andy Rubin e acquistata da Google nel 2005.

Va fatta da subito una precisazione sul termine piattaforma, spesso abusato: una piattaforma è una base software/hardware sulla quale vengono eseguite o sviluppate applicazioni. È importante quindi distinguere le piattaforme di esecuzione, cioè quelle dedicate all'esecuzione di programmi, comunemente note come sistema operativo, da quelle di sviluppo, cioè quelle utilizzate per sviluppare programmi. I due concetti non sono comunque così distanti: una piattaforma di sviluppo solitamente è anche di esecuzione, ma non necessariamente una piattaforma di esecuzione è anche di sviluppo [37].

Nel mercato sono attualmente disponibili molte piattaforme di esecuzione, come Windows Phone, iOS, Symbian, LiMo, Bada, BlackBerry OS: perché allora molti sviluppatori scelgono proprio Android?

Quando un programmatore decide di sviluppare un'app, deve anche scegliere su quale piattaforma questa potrà essere eseguita. Si ritrova quindi davanti a due strade:

- Scegliere un linguaggio come Java, per Android, che è conosciuto, diffuso e molto utilizzato ma che dà forse troppe libertà.
- Scegliere un linguaggio meno flessibile ma che permetta di concentrarsi di più sul programma e meno sull'interfaccia, in quanto questa sarà gestita dal sistema operativo. Anche qui si arriva ad un bivio: il C# di Windows Phone oppure l'Objective-C di Apple.

Il linguaggio Java, rispetto al C, gestisce automaticamente il ciclo di vita di un'applicazione e anche l'allocazione della memoria, rendendo più semplice lo

sviluppo. Android inoltre è completamente open source, quindi rende possibile reperire, modificare e ridistribuire il codice sorgente adattandolo ad ogni dispositivo e creandone la propria versione personalizzata ed ottimizzata. È anche il sistema operativo mobile più diffuso, ha una gestione di grafica e audio di alta qualità, le applicazioni native e di terze parti sono di uguale importanza e possono essere aggiunte o rimosse senza alcun vincolo.

La scelta di Android non porta tuttavia solo vantaggi dal punto di vista informatico, ma anche dal punto di vista economico. Pubblicare applicazioni sullo store di Android (Google Play), ha un costo di iscrizione di 25€ ed è possibile programmare l'app con tutti i sistemi operativi per desktop più diffusi: Windows, OS X e Linux. Scegliere iOS invece comporta un costo 4 volte superiore per l'iscrizione all'App Store e serve necessariamente un computer Mac per programmare, anche questo molto costoso.



Figura 2.1: Le varie forme del robottino di Android

Vediamo ora le principali differenze che distinguono i diversi sistemi operativi che dominano il mercato. Come vedremo nel prossimo paragrafo, Android è la piattaforma più diffusa, seguita in seconda in posizione da Apple. Il sistema proprietario Apple è sicuramente più user friendly, cioè è più adatto ad una clientela poco abituata ad utilizzare smartphone, ma d'altro canto per poter sincronizzare il telefono con il PC (scambio di foto, musica ...) o ottenere aggiornamenti software, richiede il costante utilizzo di programmi forniti da Apple, come iTunes.

La libertà di poter installare qualsiasi applicazione, anche non ufficiale, rende Android migliore per gli sviluppatori, che possono così provare le loro applicazioni sui terminali senza dover per forza ricorrere ad un emulatore. Poter ottenere il

codice sorgente ha come effetto la possibilità di avere sempre diverse versioni del sistema operativo ottimizzate per ogni tipo di telefono e con funzionalità che la versione originale non aveva. Una famosa distribuzione di Android modificata, detta Custom ROM, è la Cyanogen Mod. L'utilizzo di una custom ROM dà all'utente i privilegi di root, cioè quei permessi che consentono all'utente di fare qualsiasi cosa col telefono, anche accedere o modificare risorse il cui accesso era stato impedito dalla ROM originale. Si può fare qualcosa di simile anche in iOS tramite il cosiddetto "jailbreak", che permette sì di avere nuove funzionalità, ma la licenza del sistema rimane comunque di Apple. Dal punto di vista della flessibilità, iOS è molto meno personalizzabile di Android, il quale lascia maggior voce in capitolo all'utente. Guardando al kernel, invece, entrambe sono basate su kernel Linux.

Symbian, invece, sistema proprietario Nokia fino a poco tempo fa, è stato abbandonato e sostituito da Windows Phone, sviluppato da Microsoft. Quest'ultimo ha introdotto una grande novità: le tiles. Le tiles sono delle tessere che si trovano al posto delle classiche icone e che rendono la ricerca di app e informazioni nel proprio telefono estremamente rapida. La sostanziale differenza tra i due sistemi operativi sta nel fatto che Windows Phone si concentra maggiormente sulla condivisione di informazioni su social network. Per il resto i sistemi sono quasi equivalenti, se non per il fatto che lo store di Android è sicuramente più ricco di app, motivo per il quale molta gente opta per il robottino piuttosto che per Windows Phone.



Figura 2.2: Android ICS



Figura 2.3: iOS 6



Figura 2.4: Windows Phone 8

2.2 La diffusione di Android nel mondo

Dai dati relativi al mese di Aprile 2013 si vede chiaramente in Figura 2.5 che il S.O. più diffuso è Android, installato nel 64.2% dei terminali venduti. In Italia invece lo si può trovare nel 62.5% dei dispositivi. La maggior diffusione di terminali Android si ha in Spagna con il 93% e il minimo in Giappone con il 45%. Quasi sempre in seconda posizione si trova iOS che raggiunge mediamente il 19% con un massimo in Giappone del 49%. Ulteriori dati si possono trovare nella Figura 2.6 in cui è indicata la distribuzione dei sistemi operativi mobili nei maggiori paesi del mondo nell'ultimo trimestre del 2012 ed il primo del 2013.

In questi anni il robottino verde ha guadagnato moltissimi utenti e ultimamente continua ad acquisire mercato grazie ai prodotti di marca Samsung.

Più recente di questi due é Windows Phone che continua a guadagnare terreno grazie ai suoi prezzi contenuti e alle prestazioni comunque di buon livello. iOS invece ha prezzi decisamente più elevati e questo lo sfavorisce, mentre Android realizza telefoni sia di fascia alta, media e bassa, coprendo una clientela maggiore.

Un discorso a parte va fatto per i tablet, nel cui mercato il sistema Apple é presente sul 68% di quelli venduti, mentre ad Android spetta solo il 38%. Il motivo è da ricercarsi sicuramente in una maggiore qualità dei materiali impiegati da Apple, una più lunga presenza sul mercato, ma anche nel fatto che il tablet viene spesso utilizzato per lavoro e il sistema operativo della Apple offre molti più programmi per scrittura e gestione di documenti, a differenza di Android, che ha per lo più software di terze parti spesso non ben sviluppati in questo ambito. Probabilmente è solo questione di tempo dato che Android sta cominciando a inserirsi bene anche in questo settore.

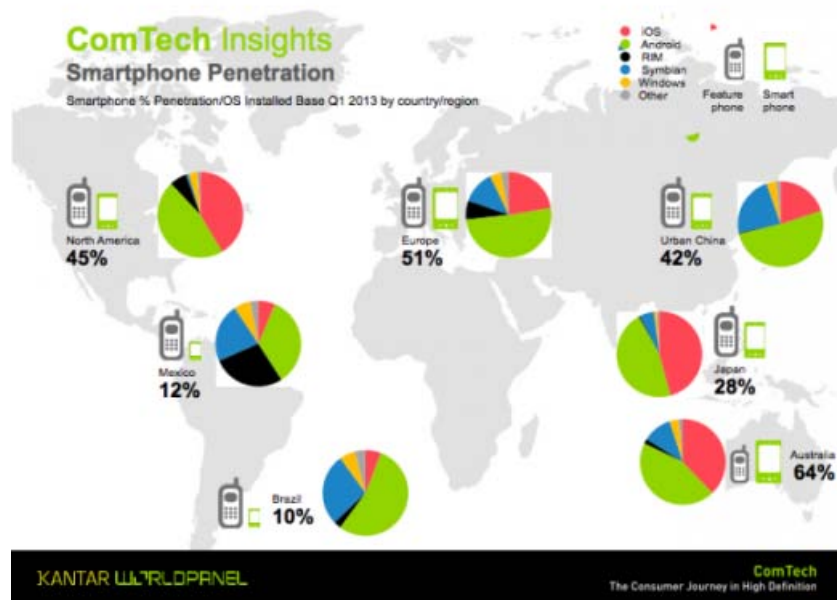


Figura 2.5: La distribuzione dei sistemi operativi [3].

2.3 L'evoluzione di Android

Dal 2008, data della prima uscita su smartphone, ad oggi, Android é migliorato moltissimo e ha subito grandi variazioni dal punto di vista grafico, funzionale e delle prestazioni. Si sono susseguite 39 versioni (contando anche le release minori) e 18 livelli di API, Application User Interface. Le API sono un insieme di funzioni disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per lo svolgimento di un determinato compito all'interno di un certo programma. Spesso con tale termine si intendono le librerie software disponibili

Smartphone OS Sales Share (%)











Germany 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	18.7	16.9	- 1.8
Android	63.4	73.6	10.2
RIM	1.8	0.5	- 1.3
Symbian	8.2	2.5	- 5.7
Windows	6.6	6.1	- 0.5
Other	1.2	0.4	- 0.8
GB 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	30.1	28.7	- 1.4
Android	49.3	58.4	9.1
RIM	15.3	5.1	- 10.2
Symbian	2.1	0.6	- 1.5
Windows	2.9	7.0	4.1
Other	0.2	0.3	0.1
France 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	20.5	21.2	0.7
Android	57.8	63.3	5.5
RIM	7.4	4.0	- 3.4
Symbian	4.3	0.6	- 3.7
Windows	2.8	7.2	4.4
Other	7.2	3.6	- 3.6
Italy 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	22.6	19.9	- 2.7
Android	48.4	62.5	14.1
RIM	3.7	2.5	- 1.2
Symbian	17.7	3.5	- 14.2
Windows	5.9	10.9	5.0
Other	1.6	0.7	- 0.9
Spain 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	5.4	3.2	- 2.2
Android	76.8	93.5	16.7
RIM	7.8	0.2	- 7.6
Symbian	8.1	1.5	- 6.6
Windows	1.8	1.3	- 0.5
Other	0.0	0.3	0.3
USA 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	44.6	43.7	- 0.9
Android	47.9	49.3	1.4
RIM	2.6	0.9	- 1.7
Symbian	0.5	0.2	- 0.3
Windows	3.7	5.6	1.9
Other	0.6	0.3	- 0.3
China 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS		24.6	
Android		69.4	
RIM		0.3	
Symbian		2.9	
Windows		2.0	
Other		0.7	
Australia 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	33.8	31.0	- 2.8
Android	52.9	61.7	8.8
RIM	0.8	0.5	- 0.3
Symbian	6.7	1.1	- 5.6
Windows	3.3	4.1	0.8
Other	2.6	1.5	- 1.1
Japan 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS		49.2	
Android		45.8	
RIM		0.7	
Symbian		0.2	
Windows		0.3	
Other		3.7	
EU5 	3 m/e Mar 2012	3 m/e Mar 2013	% pt. Change
iOS	20.4	19.4	- 1.0
Android	58.1	68.8	10.7
RIM	8.1	2.7	- 5.4
Symbian	7.2	1.6	- 5.6
Windows	4.1	6.5	2.5
Other	2.2	1.1	- 1.2

Figura 2.6: La distribuzione dei sistemi operativi[34].

in un certo linguaggio di programmazione¹. Le release maggiori, cioè che portano grandi cambiamenti, passano da un numero ad un altro (es. da 3 a 4) mentre le release minori, correzioni di piccoli bug e piccole novità, mantengono quasi lo stesso numero di versione (es. da 4.0 a 4.1).

Le versioni di Android prima della 1.0 non avevano un nome identificativo ma solamente un numero di riconoscimento come m3-rc20a e m5-rc15. Per poter essere ricordate più facilmente anche dagli utenti, si decise di dare a queste due versioni dei nomi che potessero ricordare un robottino e furono rispettivamente Astro Boy e Bender. Per le versioni seguenti gli sviluppatori decisero che avrebbero invece utilizzato dei nomi di dolcetti, senza un particolare motivo, ma in ordine alfabetico. La prima di queste versioni è Cupcake, la prima che esce anche in Italia, introduce delle novità che con l'evolversi del sistema operativo verranno mantenute, migliorate e considerate sempre più importanti, come i Widget e la rotazione dello schermo. Donut e Éclair non sono particolarmente interessanti dato che ottimizzano solo in parte le prestazioni di Android ed infatti non sono per nulla diffuse. Froyo introduce importantissime novità nel kernel² grazie alla Dalvik Virtual Machine (di cui parleremo più avanti) e al JIT (Just in Time compiler). Gingerbread porta solamente delle piccole migliorie, mentre con HoneyComb Google si sposta anche nel settore tablet, introducendo la grafica Holo che sarà alla base delle successive versioni per smartphone come Ice Cream Sandwich e Jelly Bean. Quest'ultima versione è decisamente migliore delle precedenti grazie all'implementazione del Project Butter, un'iniziativa di Google per rendere il sistema più fluido. Ad Ottobre dovrebbe uscire l'ultima versione Key Lime Pie di cui non sono ancora state rese note le novità [28].



Figura 2.7: L'evoluzione dell'interfaccia nelle varie versioni di Android

2.4 La grafica di Android

L'app gioca un ruolo fondamentale nel rendere un telefono ricco di funzioni, sempre aggiornato e in costante evoluzione. Per capire come realizzare un'app efficiente dal punto di vista grafico è bene analizzare come si presenta l'interfaccia Android all'utente. L'interfaccia di sistema di Android fornisce il framework³ sul quale viene poi costruita un'applicazione: gli elementi base che costituiscono l'interfaccia sono la schermata Home, le schermate di navigazione e le notifiche.

¹http://it.wikipedia.org/wiki/Application_programming_interface

²Nucleo del sistema operativo

³Struttura logica di supporto su cui un software può essere progettato e realizzato, spesso facilitandone lo sviluppo

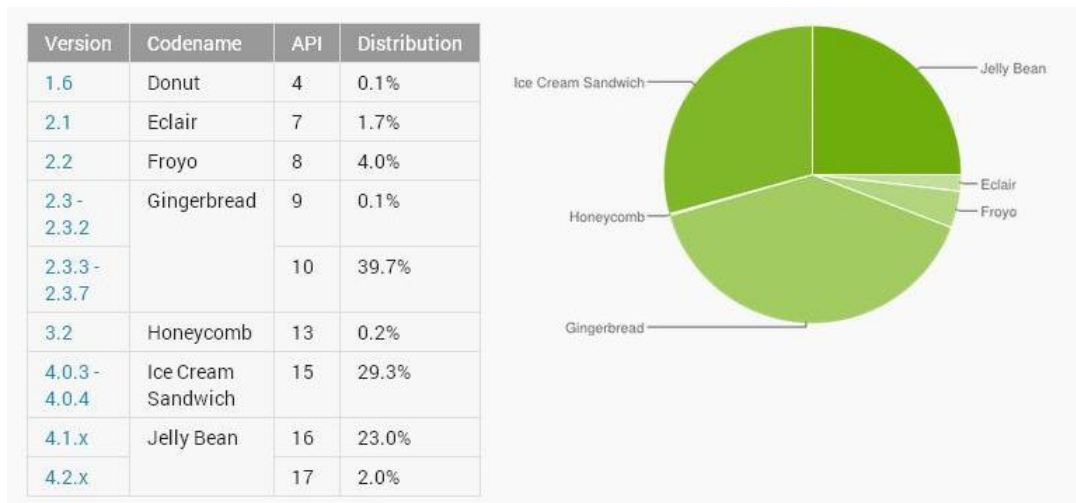


Figura 2.8: Distribuzione delle versioni [2]

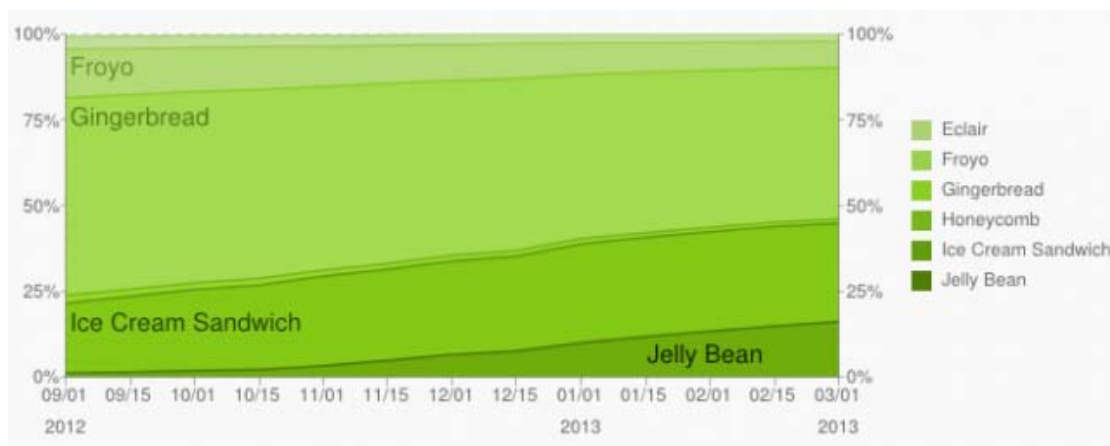


Figura 2.9: Andamento della diffusione nel tempo (Aprile 2013)

La **schermata Home** (Home screen), Figura 2.10, è uno spazio personalizzabile che contiene i collegamenti alle app, alle cartelle e ai widget, cioè quei componenti grafici dell'interfaccia utente di un programma, il cui scopo è di facilitare l'interazione dell'utente con il programma stesso. Se questi elementi sono in grande quantità non possono essere contenuti tutti in una sola schermata e quindi mano a mano che vengono aggiunti si creano nuove schermate (panels), le quali si possono scorrere con il gesto di swipe⁴ verso sinistra o verso destra. Inoltre, nella parte bassa dello schermo è presente anche la barra dei preferiti, nella quale si possono inserire le icone e cartelle delle applicazioni più utilizzate così da avere un accesso più rapido. Al centro di questa barra si trova un pulsante che apre la **AllApp screen**, Figura



Figura 2.10: Schermata Home

2.11, in cui si possono vedere tutte le app ed i widget installati. Per sfogliare documenti recenti o app ancora aperte in multitasking è presente un tastino sulla barra di navigazione (numero 2 in Figura 2.12) che apre il gestore delle applicazioni tramite il quale si possono chiudere le app rimaste aperte o riaprire in velocità un documento letto di recente.

Elementi dell'interfaccia più interessanti sono le **barre di sistema**. Si può vedere un esempio in Figura 2.12. Le barre di sistema sono un'area dedicata alla visualizzazione delle notifiche o alla comunicazione dei messaggi di sistema. L'aspetto che le rende importanti è il fatto che queste barre cambiano in base all'applicazione che si sta eseguendo (a meno che non sia full-screen): saranno quindi disponibili tramite esse le opzioni e quant'altro concerne l'applicazione in esecuzione. Le barre di sistema sono in genere due ed una terza che è la combinazione di esse:

⁴Scorrimento del dito da un lato ad un altro dello schermo, come sfogliare una pagina

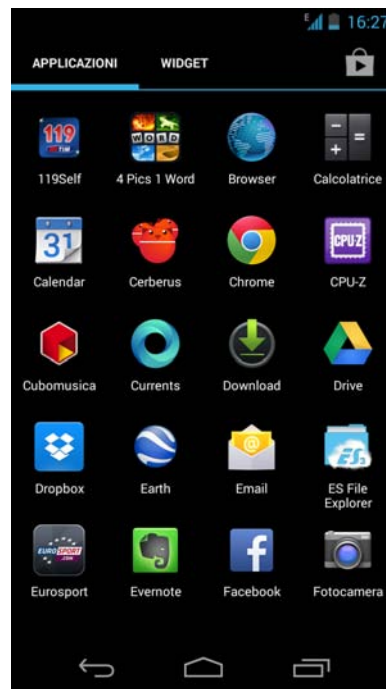


Figura 2.11: AllApp Screen

1. Barra di stato: è posizionata in alto, sulla sinistra si trovano le notifiche delle applicazioni, mentre sulla destra si trovano informazioni relative al telefono, come l'ora, la batteria, la potenza del segnale.
2. Barra di navigazione: disponibile solo nelle versioni successive ad Ice Cream Sandwich e in quei telefoni privi di tasti fisici. È posizionata in basso e in questa barra troviamo i tasti *Indietro*, *Home*, *Recenti*.
3. Barre combinate: solo per i tablet le due precedenti barre vengono fuse in una sola quando il tablet viene posto in posizione orizzontale.

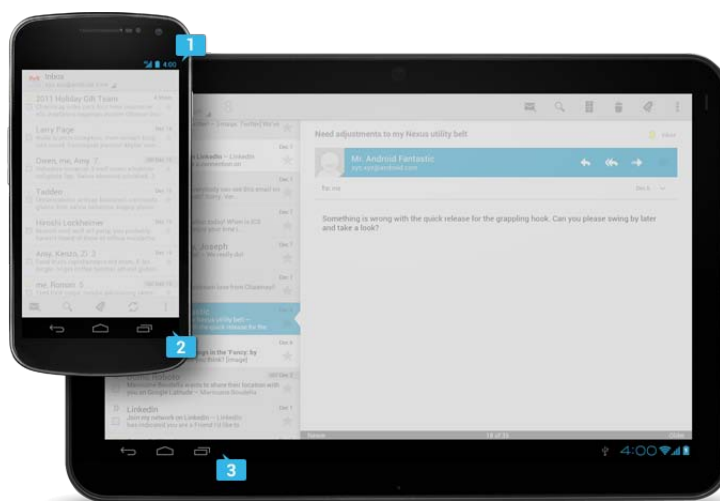


Figura 2.12: Le barre [12]

Tirando la barra delle notifiche verso il basso questa si espande, mostrando ulteriori dettagli riguardo le notifiche. Le notifiche sono brevi messaggi che forniscono informazioni come update, promemoria o altre informazioni importanti ma non abbastanza da interrompere l'attività dell'utente. Toccando una notifica si aprirà l'applicazione corrisponde per poterla visualizzare del tutto [12].

Capitolo 3

L'architettura di Android

Il sistema operativo Android è basato su kernel Linux e consiste in una struttura formata da vari livelli o layer, ognuno dei quali fornisce al livello superiore un'astrazione del sistema sottostante. I layer principali sono quattro, come si può vedere dall'immagine:

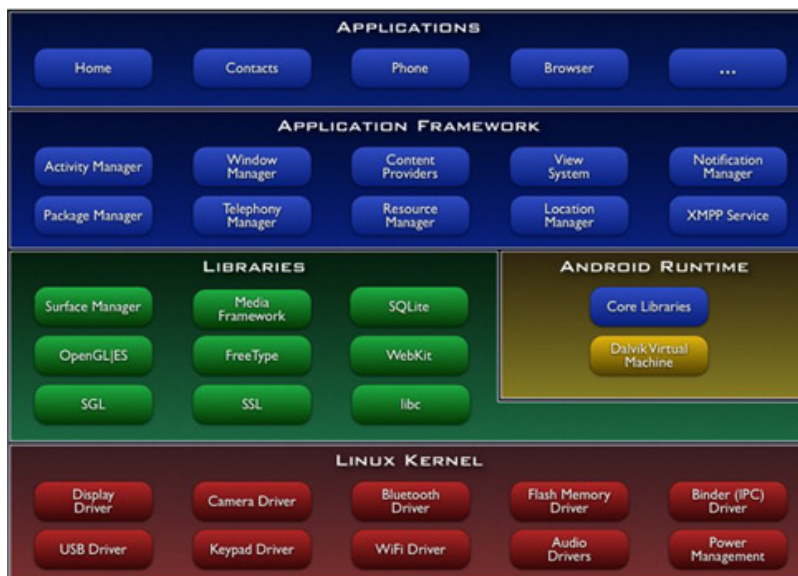


Figura 3.1: L'architettura di Android [6]

- Linux Kernel
- Libraries
- Application Framework
- Applications

3.1 Linux Kernel

Il sistema è basato su Kernel Linux, inizialmente 2.6 poi 3.X, che costituisce il livello di astrazione di tutto l'hardware sottostante: include cioè i driver per la gestione

del WiFi, Bluetooth, GPS, fotocamera, touchscreen, audio, USB driver, eccetera. Ogni produttore di smartphone dovrà modificare in primo luogo questa parte per renderlo compatibile con l'hardware scelto. Anche la creazione di una custom ROM spesso ha bisogno di modifiche a questo livello, in quanto spesso vengono effettuati dei porting¹ delle ultime versioni di Android su dispositivi che non riceverebbero più aggiornamenti ufficiali. L'astrazione hardware permette ai livelli superiori e agli sviluppatori una programmazione ad alto livello senza preoccuparsi del tipo di hardware che monta il telefono. A differenza di un kernel Linux standard, in quello di Android sono stati aggiunti ulteriori moduli per la miglior adattabilità a dispositivi mobili, come:

- Binder (IPC) Driver: driver dedicato che permette la comunicazione tra processi con un costo computazionale minore e quindi un più basso consumo di batteria.
- Low Memory Killer: modulo che si occupa di terminare i processi in modo da liberare spazio nella memoria centrale per soddisfare le richieste di altri processi. La terminazione avviene sulla base di un sistema di ranking che assegna dei punteggi in base all'importanza dei processi. Il processo `init`² non può essere terminato.
- Android Debug Bridge: strumento che permette di gestire in maniera versatile un'istanza dell'emulatore o di un dispositivo reale.
- RAM Console e Log devices: modulo per agevolare il debugging. Android fornisce infatti la possibilità di memorizzare messaggi di log generati dal kernel in un buffer RAM, in modo da poter essere osservati dagli sviluppatori per trovare eventuali bug.
- Ashmem: sistema di memoria condiviso anonimo (Anonymous Shared Memory) che definisce interfacce che permettono ai processi di condividere zone di memoria attraverso un nome.
- Power Management: sezione progettata per permettere alla CPU di adattare il proprio funzionamento per non consumare energia se nessuna applicazione o servizio ne fa richiesta.

3.2 Libraries

Il livello superiore riguarda le librerie C/C++ che vengono utilizzate da vari componenti del sistema. Tra esse troviamo:

- Il Surface Manager: modulo che gestisce le View, cioè i componenti di un'interfaccia grafica.

¹Adattamento del codice del sistema operativo per dispositivi che non lo supporterebbero nativamente

²Primo processo ad essere avviato dal kernel e dal quale dipendono tutti gli altri processi che solo così possono essere attivati

- Il Media Framework: si occupa dei Codec per l'acquisizione e riproduzione di contenuti audio e video.
- SQLite: il Database Management System (DBMS) utilizzato da Android. E' un DBMS relazionale piccolo ed efficiente messo a disposizione dello sviluppatore per la memorizzazione dei dati nelle varie applicazioni sviluppate.
- FreeType: un motore di rendering dei font.
- LibWebCore: un browser-engine basato su WebKit, open source, che può essere integrato in qualunque applicazione sotto forma di finestra browser.
- SGL e OpenGL ES: librerie per gestire rispettivamente grafica 2D e 3D.
- SSL: libreria per il Secure Socket Layer.
- LibC: implementazione della libreria di sistema standard C, ottimizzata per dispositivi basati su versioni embedded di Linux.

3.2.1 Android Runtime

In questo livello, di grande rilievo è l'ambiente di Runtime, costituito dalla Core Library e la Dalvik Virtual Machine: insieme costituiscono l'ambiente di sviluppo per Android. Le Core Libraries includono buona parte delle funzionalità fornite dalle librerie standard di Java a cui sono state aggiunte librerie specifiche di Android. La Dalvik Virtual Machine (DVM) è invece un'evoluzione della Java Virtual Machine sviluppata da Google in cooperazione con altri marchi noti come ad esempio nVidia e Intel, in modo da lavorare su dispositivi poco performanti come i cellulari. Bisogna tuttavia sottolineare che con gli ultimi modelli di smartphone si sono raggiunte delle prestazioni veramente elevate (processori superiori a 2 GHz e memorie RAM da 2 GB), paragonabili a dei notebook di fascia medio-bassa. Vediamo più nel dettaglio come è organizzata la Dalvik Virtual Machine. Per poterlo capire dobbiamo dapprima chiarire il concetto di macchina virtuale: una VM è un software che può essere un sistema operativo, un emulatore o una virtualizzazione hardware completa.

Ci possono essere due tipi di macchine virtuali: **System VM** e **Process VM**. La prima supporta l'esecuzione di un sistema operativo completo mentre la seconda supporta solo l'esecuzione di processi singoli. Rappresentandole con dei layer si troverebbe che la System VM è al di sopra della Process VM, Figura 3.2.

E' utile inoltre chiarire la differenza tra **Stack Based VM** e **Registered-Based VM**. La prima è orientata, come dice il nome, all'uso dello stack mentre la seconda è orientata all'uso dei registri. Se inizialmente, per facilità di implementazione, si utilizzava la Stack Based, ora invece si preferisce la Registered-Based in quanto riduce di molto il numero di istruzioni.

La DVM è una Process Virtual Machine che adotta proprio l'approccio Registered Based, a differenza della JVM che utilizza l'approccio più vecchio, quello Stack Based. E' quindi veloce in esecuzione ma leggermente più lenta nella creazione del bytecode. Ad ogni applicazione viene associato un processo che gira all'interno di una DVM ad esso dedicata, ed il sistema gestisce più macchine virtuali contemporaneamente per

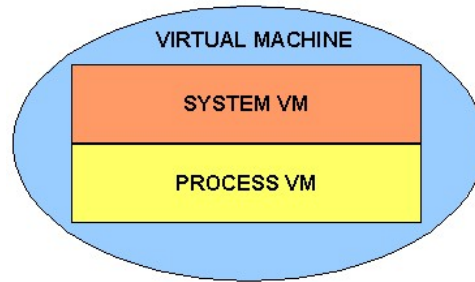


Figura 3.2: System e Process Machine

rendere possibile il multitasking. Il bytecode costituisce l'insieme delle operazioni che la macchina dovrà eseguire e viene scritto una sola volta. Quello della DVM deriva da quello della JVM: viene creato da un file .class un file .dex (Dalvik EXecutable), che non è altro che una modifica del bytecode della JVM a cui vengono tolte le parti ripetute più volte. Da alcune versioni di Android inoltre (in particolare dalla 2.2), la DVM implementa un Just in Time Compiler (JIT), che attraverso il riconoscimento di alcune parti di codice Java, esegue una traduzione in parti di codice nativo C o C++ rendendo l'esecuzione molto più veloce.

Riepilogando, i motivi per cui viene adottata la DVM sono: minor spazio occupato dal bytecode, maggior velocità in esecuzione rispetto ad una Stack Based VM.

Sempre relativamente al kernel, quello di Linux di Android è un sistema multiutente nel quale ogni applicazione è un utente differente. Il sistema infatti crea uno user ID distinto per ogni applicazione e imposta i permessi dei file dell'applicazione stessa in modo che solo quell'ID possa accedervi. Inoltre, ogni applicazione sul telefono viene lanciata in un processo Linux a sé stante all'interno della propria istanza della JVM: questa architettura a sandbox garantisce la stabilità del telefono nel caso in cui qualche applicazione crei problemi. Tutto questo non limita però la possibilità di interazione tra i processi: permette anzi loro di condividere lo stesso user ID e la stessa JVM in modo da preservare la coerenza delle risorse di sistema.

3.3 Application Framework

In questo livello è possibile rintracciare i gestori e le applicazioni di base del sistema. In questo modo gli sviluppatori possono concentrarsi nella risoluzione di problemi non ancora affrontati, avendo sempre a propria disposizione il lavoro già svolto da altri. Tra i vari gestori presenti troviamo:

- Activity Manager: gestisce tutto il ciclo di vita delle Activity. Le Activity sono entità associate ad una schermata dell'applicazione. Il compito dell'Activity Manager è quindi quello di gestire le varie Activity sul display del terminale e di organizzarle in uno stack in base all'ordine di visualizzazione sullo schermo. I concetti di Activity e ciclo di vita dell'Activity saranno esposti in modo più esaustivo successivamente.
- Content Providers: gestiscono la condivisione di informazioni tra i vari processi attivi.

- Window Manager: gestisce le finestre relative a differenti applicazioni.
- Il Telephony Manager: gestisce le funzioni base del telefono quali chiamate ed SMS.
- Resource Manager: gestisce tutte le informazioni relative ad una applicazione (file di configurazione, file di definizione dei layout, immagini utilizzate, ecc).
- Package Manager: gestisce i processi di installazione e rimozione delle applicazioni dal sistema.
- Location Manager: mette a disposizione dello sviluppatore una serie di API che si occupano della localizzazione (tramite GPS, rete cellulare o WiFi).
- System View: gestisce l'insieme degli elementi grafici utilizzati nella costruzione dell'interfaccia verso l'utente (bottoni, griglie, text boxes, ecc...).
- Notification Manager: gestisce le informazioni di notifica tra il dispositivo e l'utente. Le notifiche possono consistere in vari eventi: la comparsa di un'icona nella barra di notifica, l'accensione del LED del dispositivo, l'attivazione della retroilluminazione del display, la riproduzione di suoni o vibrazioni.

3.4 Applications

Al livello più alto risiedono le applicazioni utente. Le funzionalità base del sistema, come per esempio il telefono, il calendario, la rubrica, non sono altro che applicazioni scritte in Java che girano ognuna nella propria DVM. E' bene notare che Android non differenzia le applicazioni di terze parti da quelle già incluse di default, infatti garantisce gli stessi privilegi a entrambe le categorie [4].

Le App

Il termine App non è altro che l'abbreviazione di *Applicazione* e indica un programma pensato appositamente per dispositivi mobili quali smartphone o tablet e che permette agli utenti di avere a disposizione funzioni sempre maggiori: si va dai giochi alla musica, dalle app per interagire nei social network a quelle per la gestione di eventi o documenti.

Quello che la differenzia da un generico programma per computer è solamente il fatto di essere più leggera in termini di spazio occupato e di poter essere eseguita su smartphone, che genericamente hanno risorse hardware limitate rispetto ad un elaboratore.

Dall'apertura nel 2008 del primo store online, le app si sono diffuse sempre di più negli smartphone arrivando a sfiorare le 800mila per Android e superando i 25 miliardi di download, 775mila per iOS e solo 150mila per Windows Phone [31]. In particolare, nello store, il negozio online in cui è possibile sfogliare le applicazioni disponibili, se ne trovano sia gratuite sia a pagamento ed ognuno può caricare la propria e farla conoscere agli altri. Non tutte le app caricate vengono però ritenute interessanti dagli utenti e quindi hanno pochissimi download. Cosa rende allora una app è di successo? Sicuramente queste caratteristiche:

- **Un'icona graficamente carina.** È il biglietto da visita dell'applicazione e quindi è importante che attiri l'attenzione.
- **Un titolo chiaro.** Il titolo deve far capire all'utente a cosa serve l'applicazione, senza che debba a leggersi tutta la descrizione delle sue funzioni.
- **Schermate intuitive.** Utilizzare l'applicazione deve essere facile, poco macchinoso e deve permettere a qualsiasi tipo di utente di utilizzarla agevolmente.
- **Gratuita o con un prezzo conveniente.** Avere prezzi troppo elevati spinge gli acquirenti a cercare applicazioni simili con prezzi più vantaggiosi.
- **Multilingua.** Per ampliare la fetta di mercato è sicuramente utile rendere il proprio prodotto disponibile in più lingue.
- **Riunisce le funzioni di molti oggetti in uno solo.** Risulta infatti più comodo utilizzare un solo oggetto per svolgere varie funzioni. Si pensi ad

esempio ad un'app che realizza un diario di viaggio: in un solo telefono sono racchiusi un'agenda, un blocco note, un diario, delle fotografie ed inoltre rende molto facile e ordinata qualsiasi modifica.

- **Divertente.** Se si tratta di un gioco, questo deve essere coinvolgente e possibilmente deve permettere ai giocatori di sfidarsi tra di loro in rete.

Se l'obiettivo che ci si prefigge però è quello di vendere l'app ad un pubblico sempre più vasto non è sufficiente crearla e pubblicarla sul market dimenticandosene: bisogna pubblicizzarla, per esempio sui social network come Facebook o Twitter, partecipare a concorsi ed eventi informatici attraverso i quali si incontrano nuove idee e ci si fa pubblicità, migliorare costantemente l'app correggendo gli errori o aggiungendo sempre nuove funzioni, portandola quindi ad un livello sempre più alto.

4.1 Android Graphics Guidelines

La progettazione di un'app non si basa solamente sulle caratteristiche sopra elencate, ma richiede anche un grande impegno dal punto di vista dell'interfaccia grafica. L'applicazione non dovrà essere solo bella da vedere ma dovrà cercare di rispettare le linee guida fornite da Android dette **Android Graphics Guidelines**[10]. Poiché il linguaggio Java concedeva troppe libertà al programmatore riguardo all'interfaccia delle app Android, per evitare che le applicazioni risultassero troppo diverse tra loro, a partire dalla versione Ice Cream Sandwich Google decise di introdurre le Android Graphic Guidelines: delle linee guida per sviluppare le app. Infatti nelle versioni di Android precedenti la 4.0, l'aspetto grafico delle app (giochi a parte) non era particolarmente entusiasmante anche a causa della frammentazione di Android; con l'uscita invece di queste linee guida l'interfaccia è ancora molto libera ma è più accattivante (si pensi a Whatsapp) ed è stata uniformata.

Infatti, inizialmente ogni utente sviluppava la grafica a modo suo, creando app tra loro molto diverse graficamente e che quindi mettevano in difficoltà l'utente, il quale doveva adattarsi ogni volta ad uno stile diverso e spesso non molto chiaro. L'utilizzo delle linee guida di Google permette di conferire all'applicazione uno stile semplice ed efficace in cui ogni funzione si trova al posto giusto e in linea con lo stile e il tema del resto del sistema operativo, così sarà più facile che riscontri una grande richiesta di download da parte degli utenti.

Si consideri ad esempio Viber, disponibile per Android, iOS e Windows Phone. L'applicazione consente agli utenti di effettuare chiamate e inviare messaggi gratuitamente a qualunque altro utente abbia installato Viber nel proprio dispositivo e disponga di una connessione 3G o Wi-Fi. Dalla Figura 4.1 e dalla Figura 4.2 possiamo farci un'idea dell'applicazione prima e dopo l'introduzione della grafica Holo e delle Linee Guida di Google.

Le differenze tra le due versioni si notano da subito. La prima delle due immagini potrebbe sembrare quasi appartenente al sistema operativo Apple, con i tasti tondeggianti e la barra in basso che richiama quella delle app di iOS. La seconda invece ricorda inequivocabilmente la grafica Holo di Android, Figura 4.8. L'importanza dell'Action Bar, la barra in alto che contiene i pulsanti di azione e



Figura 4.1: Viber prima della grafica Holo

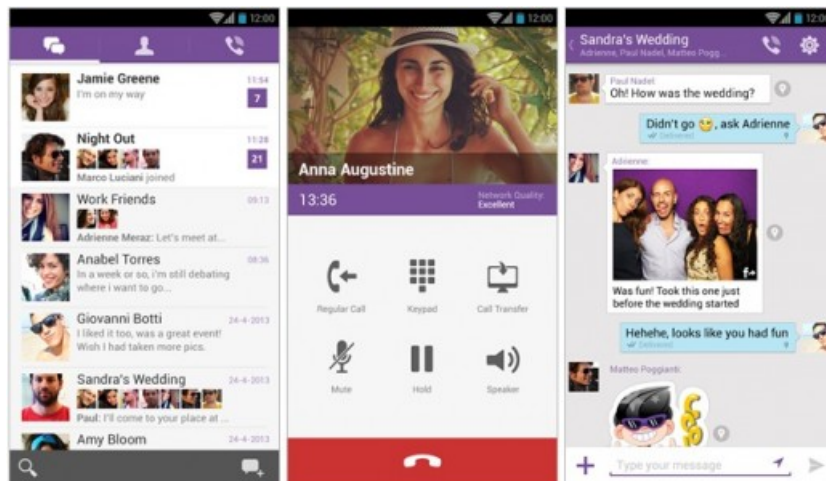


Figura 4.2: Viber con grafica Holo

analizzata nel capitolo 6, è notevole soprattutto nei nuovi dispositivi, i quali spesso non hanno alcun tasto fisico e quindi le operazioni che una volta erano svolte da quei tasti, ora le si devono trovare come tasto virtuale nel display. L'Action Bar infatti permette di rendere molto più pulita ed ordinata la videata, mostrando i tasti più utilizzati (nell'immagine si vedono i tasti Chiama e Opzioni) e nascondendo all'interno di un altro tasto i comandi meno utilizzati.

Un altro esempio è Shazam, recentemente aggiornata alla grafica Holo, vedi Figura 4.3. Anche qui è immediato vedere quanto l'introduzione dell'ActionBar renda

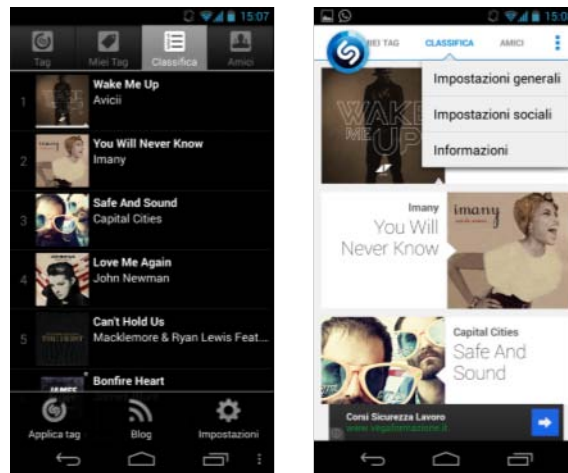


Figura 4.3: Shazam, prima e dopo la grafica Holo

più pulita la videata e ne semplifichi l'utilizzo. Inoltre, le canzoni nella Classifica vengono visualizzate tramite dei riquadri raffiguranti la copertina dell'album e il titolo della canzone in modo totalmente diverso dalla solita ListView.

Non bisogna sottovalutare l'importanza di rendere la grafica delle app uniforme, più omogenea, alla grafica del sistema operativo: una delle maggiori critiche fino a prima dell'uscita di Ice Cream Sandwich era proprio il fatto che le applicazioni avevano un'interfaccia 'a caso'. Infatti ogni sviluppatore decideva a proprio piacimento la disposizione dei tasti e degli elementi grafici. Tutto questo era accompagnato dal fatto che ogni casa produttrice di telefoni, come HTC o Samsung, modificava a proprio piacimento la grafica del sistema operativo. Di fatto quindi un utente quando si ritrovava in mano un dispositivo diverso da quello che utilizzava abitualmente, si doveva chiedere quale fosse il sistema operativo installato: Android quindi non aveva una propria identità (Figura 4.4 e 4.5).

Google allora ha messo dei paletti per limitare le libertà di modifica della grafica: ogni casa produttrice può personalizzarla leggermente, ma in ogni dispositivo si riconosce sempre e facilmente che si tratta di Android. In Figura 4.6 vediamo invece due personalizzazioni diverse ma che comunque lasciano facilmente intuire che il sistema operativo è Android.

Dispositivi e Display. In circolazione esistono un'infinità di dispositivi diversi sia dal punto di vista software che hardware. Va fatto notare infatti che ogni dispositivo, oltre che processori e memorie differenti, ha anche schermi differenti. Il problema di avere diversi schermi, e quindi diverse risoluzioni e densità di pixel, richiede



Figura 4.4: Android in un telefono HTC con interfaccia Sense



Figura 4.5: Android in un telefono Samsung con interfaccia Touchwiz



Figura 4.6: Interfacce uniformate

allora che l'applicazione possa adattare la propria grafica in base al dispositivo che la utilizzerà. Bisogna allora sfruttare al meglio il lavoro fatto sui layout: per i dispositivi con schermi molto grandi si può pensare ad esempio di comporre dei layout multipli per mostrare una maggiore quantità di informazioni nella stessa schermata e facilitare quindi la navigazione. Bisogna creare un'interfaccia che sia abbastanza flessibile per le varie risoluzioni e creare un set di icone che vada bene per tutte le risoluzioni e densità (DPI)¹. Si può vedere un esempio in Figura 4.7. Questo ultimo problema è facilmente risolvibile in quanto basta creare l'icona per l'MDPI, e scalarla, cioè ridimensionarla, per ottenere le icone adatte alle altre risoluzioni. Un altro approccio, forse più semplice, consiste nel creare l'icona per i dispositivi con massima risoluzione e ridimensionarla per le risoluzioni minori [14].



Figura 4.7: DPI: Dots Per Inch

Temi. I temi sono un facile e veloce meccanismo per conferire ad una app o un'activity uno stile elegante ed in armonia con il resto del telefono. Lo stile che si sceglie specifica le proprietà grafiche degli elementi che compongono l'interfaccia utente, come i colori, altezza e dimensione dei caratteri. Per quanto riguarda i caratteri, con ICS è stato introdotto un nuovo set di caratteri denominato Roboto, pensato appositamente per l'interfaccia utente e per gli schermi ad alta risoluzione. Per garantire una certa omogeneità delle app con il sistema operativo, Android fornisce tre temi di sistema che possono essere direttamente applicati all'app quando la si crea e questi sono disponibili però (a meno di support library) da Ice Cream Sandwich in poi:

- Holo Light
- Holo Dark
- Holo Light con Dark Action Bar

Basta scegliere il tema che più combacia con le specifiche e con l'aspetto che vorremmo dare alla nostra app per avere già una buona base di partenza, ma ciò non toglie che si possa creare un'interfaccia completamente personalizzata, magari partendo da un tema predefinito come base. Esistono infatti molte guide che aiutano passo passo l'utente nella creazione di una grafica personale [21].

¹Dots Per Inch, punti per pollice. Esistono già delle dimensioni predefinite: low, medium, high, extrahigh, indicate rispettivamente con L,M,H,XH DPI

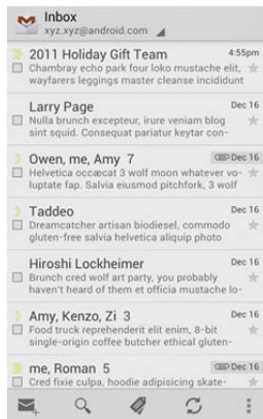


Figura 4.8: Holo Light



Figura 4.9: Holo Dark

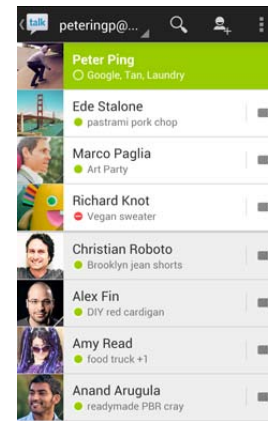


Figura 4.10: Holo Light con Dark Action Bar

Le Icone. Un'icona è un elemento grafico che occupa una piccola porzione di schermo e deve far intuire velocemente lo stato o l'azione che verrà intrapresa alla sua pressione.

Quando si disegnano le icone per la propria app è importante tenere a mente che l'app verrà installata su dispositivi con schermi con densità di pixel diverse, come detto precedentemente. Allora è sufficiente creare icone per le varie densità e poi Android, riconoscendo automaticamente la densità dei pixel dello schermo, caricherà da solo l'icona opportuna. A proposito di icone vale la pena spendere due parole anche riguardo il Launcher che si trova nella AllApp Screen: dato che l'utente può modificare lo sfondo della Home a proprio piacimento sarebbe bene fare in modo che l'icona dell'applicazione fosse visibile con quasi qualsiasi tipo di sfondo.

Le icone dell'**Action Bar** sono dei piccoli bottoni che rappresentano azioni che le persone possono eseguire con l'app. Ognuna di esse dovrebbe rappresentare in modo chiaro ed univoco l'azione che quel tasto rappresenta. Per esempio un tasto con disegnato un bidone rappresenterà l'azione elimina. Esiste una famiglia di icone predefinite, scaricabile dal sito di Android e modificabili, per la grande maggioranza delle azioni che si compiono comunemente durante l'utilizzo di una app, come l'azione di refresh, eliminazione o condivisione. Anche per queste icone Android dà delle linee guida: pittografiche, piatte, non troppo dettagliate, con gli angoli arrotondati o squadrate e magari, per renderle più accattivanti, ruotate di 45°. Inoltre le linee che le compongono dovrebbero avere uno spessore minimo di 2dp.

Le icone dei **menù contestuali** come per esempio le stelline che si trovano nell'app Gmail, devono avere ulteriori caratteristiche rispetto alle precedenti: devono essere sempre piatte e semplici ma questa volta non devono avere dei colori neutri ma brillanti, che facciano risaltare l'icona rispetto a dove è posizionata e che rendano l'idea di cosa vanno ad indicare. Ad esempio un'icona che diventa rossa segnala più facilmente che qualcosa non va come dovrebbe, al contrario di un'icona di colore

verde che rappresenta più facilmente situazioni nella norma.

Le stesse considerazioni valgono anche per le **icone delle notifiche** ma questa volta il colore deve essere necessariamente bianco dato che il colore che farà da sfondo alle icone è sempre nero nel caso delle notifiche.

Per semplificare il lavoro, Google ha reso disponibile nel sito Android Developer delle linee guida per disegnare al meglio icone. Il primo consiglio è quello di utilizzare la grafica vettoriale: così facendo, anche se un'immagine viene ridimensionata non si perde mai in qualità. L'utilizzo della grafica vettoriale non è semplice e quindi va seguito da uno sviluppatore che già sa lavorare con questo tipo di immagini. Per gli utenti invece meno esperti si può utilizzare un piccolo trucco per non perdere la qualità durante il ridimensionamento: ogni volta che si progetta un'immagine basta crearla di dimensioni multiple e maggiori di quella che servirebbe per la risoluzione dello schermo maggiore. Ad esempio per le icone del Launcher, che sono larghe 48, 72, 96, 114 pixel, un'ottima idea è creare l'icona delle dimensioni di 864x864 pixel. Il processo inverso, cioè quello di ingrandire immagini piccole, porterebbe infatti ad avere immagini sgranate e poco accurate.

Per quanto riguarda l'organizzazione delle icone bisognerebbe adottare delle convenzioni per i nomi: si deve cercare di rinominare i file (icone, animazioni...) in modo che ordinandoli alfabeticamente siano divisi anche per tipo. Ad esempio i nomi delle icone vanno preceduti da *ic_*, come *ic_aggiungiSpesa*. Bisogna poi organizzare lo spazio di lavoro in cartelle, divise per densità di pixel degli schermi.

Un ultimo consiglio è quello di rimuovere i tag e i metadata (se non si ha necessità di utilizzarli) dalle immagini per rendere il codice più leggero [17].

Messaggi utente. Qualche volta l'applicazione dovrà mostrare all'utente delle conferme, delle domande o dei messaggi di errore. Sembra una cosa facile, ma porsi nel modo giusto e scrivere le informazioni essenziali in modo chiaro nasconde dietro sé degli accorgimenti particolari che sono ben espressi nelle Android Guidelines [23]. In particolare i messaggi rivolti agli utenti dovranno essere:

1. Brevi. Bisogna essere concisi e precisi, limitandosi ai 30 caratteri (spazi inclusi) e dicendo nulla in più del necessario.
2. Semplici. Bisogna pensare che l'utente non può sempre essere esperto ad utilizzare lo smartphone, quindi sono da evitare termini tecnici e soprattutto cercare di includere più lingue possibili dato che non per forza parlerà la stessa lingua dello sviluppatore.
3. Amichevoli. Parlare direttamente con l'utente dando del tu. Il testo deve essere scritto come lo direbbe una persona parlando. Con il messaggio bisogna dare sicurezza e non mettere dubbi.
4. Scrivere la cosa più importante all'inizio del messaggio. Già le prime due parole devono far capire l'oggetto del messaggio.
5. Descrivere solo il necessario. Inutile spiegare dettagli o cose che un utente normale non potrebbe capire o sono di scarso interesse.
6. Evitare le ripetizioni.

Vediamo ora in Figura 4.11 qualche esempio analizzando la frase tipica che uno sviluppatore alle prime armi potrebbe scrivere e cercando di correggerla.

1. Breve. Wizard iniziale

✗ Troppo formale

Per ottenere ulteriori informazioni invitiamo a leggere il manuale che è stato fornito con il telefono

✓

Consultare il manuale allegato al telefono

2. Facile. Utilizzo del GPS

✗ Confuso

Utilizzo del satellite GPS
Durante la localizzazione, indicare una posizione

✓

GPS
Permetti all'app di rilevare la posizione.

3. Amichevole. Ad esempio se l'applicazione si blocca:

✗ Confuso e ripetitivo. "Mi dispiace" non fa altro che peggiorare la situazione

Mi dispiace!
L'Activity MiaActivity (nell'app MiaApp) non risponde
Forza Chiusura Attendi Segnala

✓ Più corto, più diretto, nessun titolo con le scuse

MiaApp non risponde
Vuoi chiuderla?
Chiudi Attendi Segnala

4. Le notizie più importanti vanno all'inizio.

✗ Notizia importante alla fine

A 77 persone piace questo elemento. E anche a Fabio.

✓ Notizia importante all'inizio

A Fabio e altre 77 persone piace questo elemento.

✗ Attività da compiere per ultima

Premi Avanti per completare l'installazione usando il Wi-Fi

✓ Attività da compiere per prima

Per completare l'installazione usando il Wi-Fi, premere Avanti.

5. Descrivere solo il necessario.

✗ Da una finestra di setup

Accesso in corso...
Il tuo telefono deve comunicare con i server di Google per poter effettuare l'accesso al tuo account. Questo potrebbe richiedere fino a 5 minuti.

✓ Da una finestra di setup

Accesso in corso...
Il telefono sta contattando Google. Potrebbe richiedere fino a 5 minuti

Figura 4.11: Writing Style

4.2 Decidere per quali versioni sviluppare

Uno dei problemi di Android é quello della frammentazione delle versioni: purtroppo non tutti i device vengono aggiornati in tempi brevi o non vengono proprio più aggiornati. Questo complica le cose perché per sviluppare applicazioni complesse o con funzioni particolari servono API di livello alto (dal 16 al 18), ma queste

non sono supportate da versioni vecchie di Android ancora abbastanza diffuse nei cellulari in commercio, e quindi non abilitare tali versioni ridurrebbe il possibile bacino di utenti.

Entrando più nel dettaglio possiamo basarci sui dati forniti da Google per decidere il minimo livello di API che conviene impostare quando viene scritto il programma. Come si vede dalla figura relativa alla diffusione di Android nel Capitolo 2, le versioni più diffuse sono Gingerbread e quelle successive. Sviluppare applicazioni particolarmente complicate richiede un livello alto di API, spesso non compatibile con versioni molto vecchie: includere quindi distribuzioni come Froyo non giova particolarmente al numero di download in quanto i dispositivi su cui potrebbe essere installata l'app sono pochi e si incontrerebbero inoltre molte difficoltà nello sviluppo della stessa. Se l'app però non ha particolari esigenze riguardo il livello di API allora è bene includere tutte le versioni che non creano problemi. Per esempio, se vogliamo sviluppare un'app che gestisca i contatti della rubrica è sconsigliabile includere le versioni precedenti la 2.0 di Android, in quanto con quest'ultima la gestione dei contatti è stata cambiata radicalmente e troverebbe quindi problemi di compatibilità. Analizziamo in modo più approfondito i problemi di incompatibilità e vediamo come risolverli.

4.2.1 Come risolvere i problemi di compatibilità

Fortunatamente per far fronte ai problemi di compatibilità ci vengono in soccorso delle librerie particolari che permettono di integrare nell'app quelle funzioni che creano l'incompatibilità. Un primo aiuto ci viene dato dalla **Support Library** [20]: questo Pacchetto include delle librerie di supporto statiche che possono essere aggiunte all'applicazione con lo scopo di utilizzare API o che non sono disponibili per vecchie versioni di Android o che offrono funzioni totalmente nuove. Lo scopo è quello di semplificare lo sviluppo offrendo più API da includere nell'applicazione così da ridurre i problemi di incompatibilità.

Il Pacchetto di Supporto non include una sola libreria di supporto: ognuna di quelle che include ha un differente livello di API. Per esempio, una libreria richiede API di livello 4 (detta v4) o superiore, mentre un'altra richiede API di livello 13 (detta v13) o superiore. In questo caso la v4 è un sottoinsieme della v13 e v13 include (rispetto alle librerie standard) ulteriore supporto per lavorare con API di tipo v13. La libreria di supporto v4 fornisce l'accesso a diverse classi introdotte con Android 3.0 e successive, qualche versione aggiornata di classi esistenti e anche alcune API che attualmente non esistono nella piattaforma Android. Alcune delle classi più utili e importanti che si trovano nella v4 sono:

- Fragment e Fragment Manager
- Fragment Transaction
- ListFragment
- DialogFragment
- LoaderManager

- AsyncTaskLoader
- CursorLoader

Per ognuna della classi incluse, le API funzionano quasi allo stesso modo di quelle incluse nelle versioni che le hanno di base.

Mancano purtroppo le API per l'ActionBar ma questo non é un grosso problema in quanto basta includere una libreria che ne consente l'integrazione, questa libreria si chiama **ActionBarSherlock** [35]. La libreria utilizzerà direttamente l'Action Bar nativa in quei dispositivi dove è già supportata (Ice Cream Sandwich e successive), mentre ne creerà una equivalente negli altri casi, permettendo quindi di avere l'Action Bar per ogni versione di Android dalla 2.x in poi. Per poter utilizzare questa libreria sono necessari solamente la versione 1.6 del Java Development Kit e la libreria di supporto v4, dato che ActionBarSherlock é un'estensione di quest'ultima. Disporre delle funzionalità di questa libreria è molto facile, soprattutto se l'IDE utilizzato è Eclipse. Basta installare l'ADT plugin e aggiornarlo in modo che la sua versione sia 0.9.7 o superiore e includere ActionBarSherlock come progetto creato da codice già esistente. Chiaramente la cartella da indicare come codice sorgente è quella di ActionBarSherlock. Poi nelle proprietà del proprio progetto, nella sezione Librerie, basta includere la voce ActionBarScherlock. Per una versione più approfondita di come creare una app, come implementare le funzioni della ActionBarScherlok, rimandiamo alla sezione successiva.

4.3 Gli strumenti necessari

Vediamo ora quali sono gli strumenti di cui bisogna dotarsi per poter scrivere la propria applicazione Android. Innanzitutto è necessario dotarsi un pc con sistema operativo Windows, Linux oppure Mac. Riporteremo le istruzioni per chi decide di sviluppare utilizzando un pc Windows, ma nel caso si preferisca una delle altre due scelte non vi è alcun problema dato che si trovano facilmente in Internet guide molto dettagliate. Già da subito specifichiamo che non possedere un dispositivo Android non crea alcun tipo di problema perché esiste un comodo emulatore (con prestazioni però molto inferiori a quelle di un telefono) che sarà disponibile all'interno dell'ambiente di sviluppo dopo aver installato tutti i componenti trattati di seguito.

Il primo passo consiste nell'installare l'SDK. Cos'è l'SDK? É una sigla che sta per Software Development Kit, vale a dire un kit di sviluppo di software. Attraverso questo kit, qualsiasi utente potrà procurarsi il codice sorgente di ogni versione Android e modificarlo o migliorarlo autonomamente. Il codice sorgente è un testo scritto in linguaggio di programmazione che, una volta elaborato, porterà alla realizzazione di un programma eseguibile dal processore. I linguaggi di programmazione alla base della maggior parte delle applicazioni per Android sono Java e C/C++, come precisato nel Capitolo 2.

Installare l'SDK. Per prima cosa bisogna scaricare l'SDK dal sito web <http://developer.android.com/> scegliendo correttamente la versione per il proprio sistema operativo. Lo si trova disponibile sia per Linux che per Windows che per Mac. Scaricato l'SDK per

Windows procediamo ad estrarre l'archivio in una cartella del computer. A questo punto dovremmo trovare una cartella di nome android-sdk-windows come quella di Figura 4.12. Tra i vari file presenti il più importante è sicuramente l'SDK Manager.

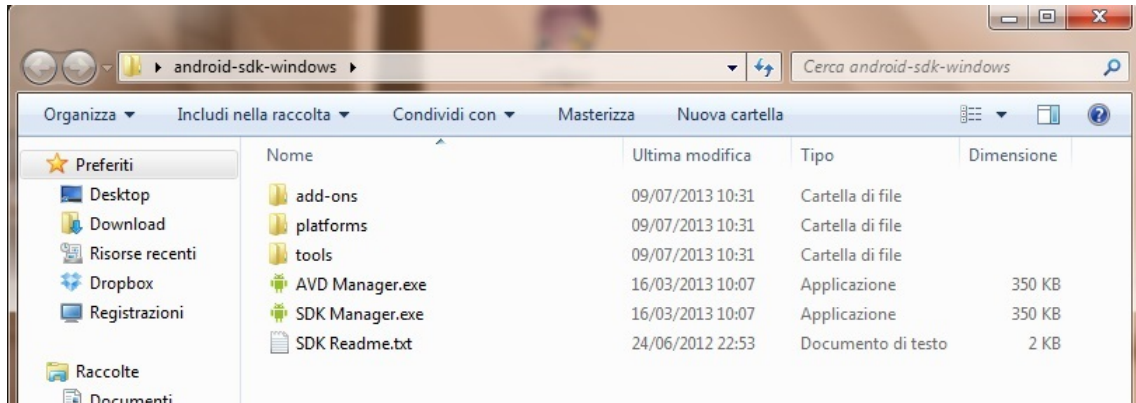


Figura 4.12: Cartella decompressa SDK

Facendo doppio click su di esso si aprirà un finestra in cui sarà possibile installare (e aggiornare) le versioni dell'SDK interessate. Una volta scelte le versioni che

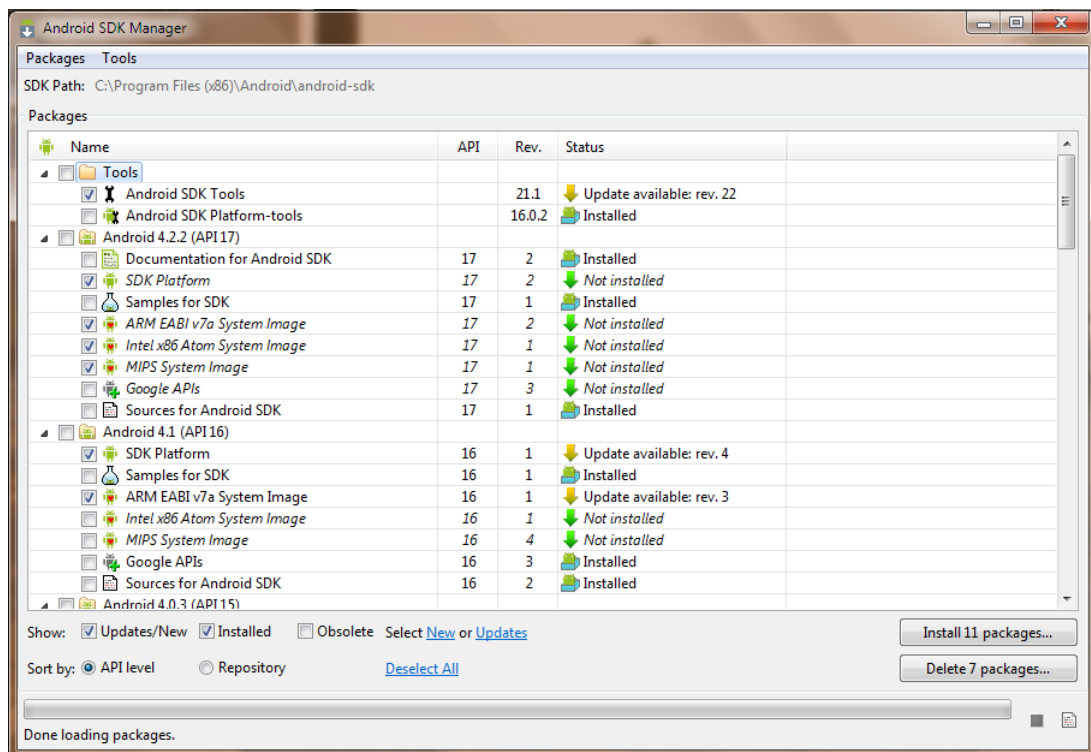


Figura 4.13: SDK Manager

vogliamo installare basta premere Installa e accettare le condizioni d'uso. Nota: l'installazione potrebbe richiedere **molto** tempo, anche delle ore, soprattutto la prima volta. Pertanto è bene evitare di installare versioni che si sa che sicuramente non utilizzeremo. Queste comunque possono essere aggiunte/rimosse in un secondo momento qualora fosse necessario. Completate le operazioni l'SDK è installato correttamente.

Installare il Java Delevolpment Kit. Per procurarsi il JDK basta collegarsi al sito <http://www.oracle.com/technetwork/java/javase/downloads/index.html> e premere sul bottone che porta al download del JDK. Ora basta accettare le condizioni della licenza e scegliere il download corretto per il proprio PC, come si può vedere in Figura 4.14 e 4.15.



Figura 4.14: Bottone JDK

Java SE Development Kit 7u25		
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.		
Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.		
Product / File Description	File Size	Download
Linux ARM v6/v7 Soft Float ABI	65.12 MB	jdk-7u25-linux-arm-sfp.tar.gz
Linux x86	80.38 MB	jdk-7u25-linux-i586.rpm
Linux x86	93.12 MB	jdk-7u25-linux-i586.tar.gz
Linux x64	81.46 MB	jdk-7u25-linux-x64.rpm
Linux x64	91.85 MB	jdk-7u25-linux-x64.tar.gz
Mac OS X x64	144.43 MB	jdk-7u25-macosx-x64.dmg
Solaris x86 (SVR4 package)	136.02 MB	jdk-7u25-solaris-i586.tar.Z
Solaris x86	92.22 MB	jdk-7u25-solaris-i586.tar.Z
Solaris x64 (SVR4 package)	22.77 MB	jdk-7u25-solaris-x64.tar.Z
Solaris x64	15.09 MB	jdk-7u25-solaris-x64.tar.gz
Solaris SPARC (SVR4 package)	136.16 MB	jdk-7u25-solaris-sparc.tar.Z
Solaris SPARC	95.5 MB	jdk-7u25-solaris-sparc.tar.gz
Solaris SPARC 64-bit (SVR4 package)	23.05 MB	jdk-7u25-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	17.67 MB	jdk-7u25-solaris-sparcv9.tar.gz
Windows x86	89.09 MB	jdk-7u25-windows-i586.exe
Windows x64	90.66 MB	jdk-7u25-windows-x64.exe

Figura 4.15: Scelta del S.O.

Finito il download bisogna eseguire il file che é stato scaricato e terminata l'installazione bisogna controllare che l'operazione sia andata a buon fine. Dal prompt dei comandi (*Start* → *cmd*) basta dare in input il comando **java -version** e la versione che appare deve essere la stessa di quella appena installata. Per avviare correttamente gli eseguibili Java bisogna includere la cartella *bin* del JDK nella PATH di Windows. La PATH di Windows include tutti i percorsi che possono essere invocati dalla shell CMD. Il compilatore Java `javac.exe` e il runtime `java.exe` risiedono all'interno della cartella

```
<JAVA_HOME>\bin
```

Bisogna quindi aggiungerla alle variabili PATH. Per poter fare ciò basta:

1. Fare click sul pulsante START e selezionare Pannello di Controllo.
2. Selezionare Sistema e Sicurezza.
3. Fare click su Sistema.
4. Scegliere sulla sinistra la voce Impostazioni di sistema Avanzate e poi Variabili d'Ambiente.
5. Dalla nuova schermata bisogna selezionare la variabile path, fare click sul pulsante modifica e incollare il percorso del JDK/bin. Nel caso di Windows Vista o Windows 7 il percorso da incollare potrebbe esse più o meno simile a questo:

```
C:\Program Files (x86)\Java\jdk1.7.0_03\bin\
```

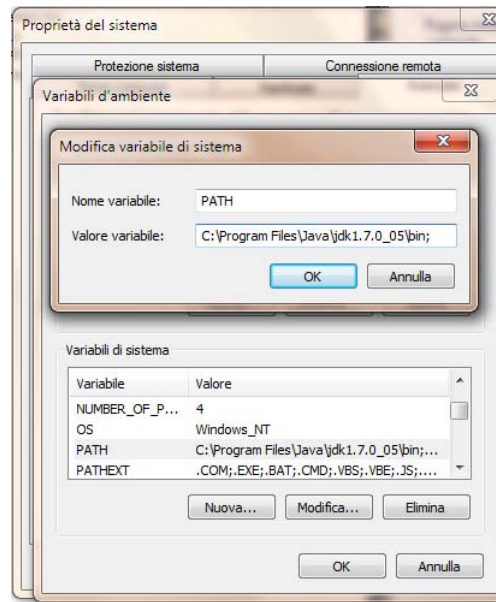


Figura 4.16: Finestra Variabili d'Ambiente

Ambiente di sviluppo. Tutto ciò che è stato scaricato e installato fino ad ora non è sufficiente per poter creare un'applicazione però è comunque fondamentale averlo.

Ci serve ora l'ambiente di sviluppo Java: si consiglia di utilizzare Eclipse (ma vanno bene anche NetBeans e JetBrains), perché è un IDE (Integrated Development Environment) open source. Lo si può scaricare dal sito www.eclipse.org/downloads/. In particolare si potrebbe scaricare la versione Java Developers (Eclipse IDE for Java Developers) che è più leggera rispetto alle altre dato che non contiene funzioni che solitamente per le prime app non vengono utilizzate. Una volta installato abbiamo bisogno di un ultimo componente: l'Android Development Tool (ADT), che permette di integrare in Eclipse i tool per la compilazione, il test, il debug del codice rilasciati con l'Android SDK. L'installazione non è molto intuitiva la prima volta e quindi verrà riportata qui di seguito:

1. In Eclipse scegliere *Help/Install New Software*
2. Nel casella *Work with* inserire: <https://dl-ssl.google.com/android/eclipse/> e premere *Add*
3. Nella finestra che compare inserire un nome che ricordi il plugin, per esempio *ADT Plugin* e premere *Ok*
4. Una volta che Eclipse avrà completato le operazioni necessarie spuntare la checkbox *Developer Tools* e premere *Next*. Dopo qualche minuto potremo accettare le condizioni di licenza e premendo *Finish* il plugin è installato.

Rimane ora un ultimo passo, cioè configurare il plugin. In *Windows/Preferences/Android* bisogna impostare il percorso della cartella in cui abbiamo installato l'SDK di Android (cioè quel percorso in cui abbiamo decompresso l'SDK di Android). Ora ci sono tutti gli strumenti necessari ad un corretto sviluppo delle app.

Capitolo 5

Struttura di un'App

Costruire un'app è un lavoro che richiede una quantità di tempo non indifferente, specialmente se è la prima volta che se ne progetta una. Scrivere il codice presuppone infatti una buona conoscenza di come un'app è strutturata, di come si può interagire con essa e di tutti i componenti che la costituiscono.

5.1 Da cosa è composta un'app

Abbiamo visto fino ad ora gli strumenti necessari per poter programmare in Android e i concetti base per poter stendere la prima versione di un'applicazione. Analizziamola ora dal punto di vista strutturale.

Un'applicazione Android deve avere le seguenti tre componenti fondamentali, visibili anche in Figura 5.1:

- Un file *AndroidManifest.xml* che si trova nella cartella principale e nel quale viene descritto il comportamento e la configurazione dell'applicazione, nonché le risorse di cui richiede l'utilizzo (fotocamera, SD...)
- La cartella *src* che contiene il codice sorgente
- La cartella *res* che contiene le risorse necessarie alla nostra applicazione per un corretto funzionamento: questa cartella può contenere sottocartelle per l'organizzazione delle risorse, come ad esempio *drawable*, *layout*, *anim* e così via

Vanno accennati alcuni piccoli dettagli riguardo la gestione delle risorse.

Android ha un'organizzazione lineare per la cartella *res*. Non saranno quindi riconosciute tutte le sottocartelle oltre il secondo livello: ad es. viene riconosciuta la cartella *layout* ma non verrebbe riconosciuta un'eventuale cartella al suo interno. La cartella *layout* contiene i file XML che definiscono i componenti GUI (Graphic User Interface) utilizzati dall'applicazione. I componenti possono essere sia layout (LinearLayout, TableLayout...) che controlli grafici (Button, EditText...). La GUI può anche essere definita direttamente da codice senza utilizzare l'XML.

Le tre cartelle *drawable-**, rispettivamente **hdpi**, **mdpi**, **ldpi** contengono le risorse grafiche (icone, sfondi..., in formato GIF, PNG, JPEG) per differenti

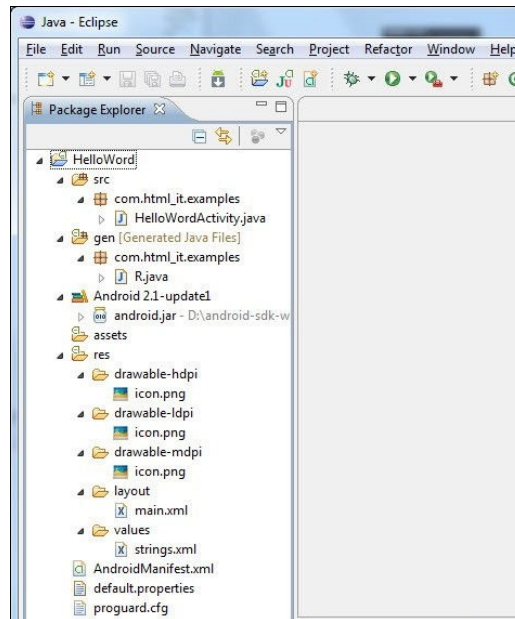


Figura 5.1: Struttura di un'App: AndroidManifest, src, resources

risoluzioni e densità degli schermi, mentre in *anim* vanno inserite le animazioni. Android userà automaticamente le icone della cartella appropriata in base alla risoluzione dello schermo.

La cartella *menù*, come dice il nome stesso, conterrà i file XML che definiscono la struttura dei vari menù (come quello delle opzioni).

L'ultima cartella rimanente, *values*, contiene risorse XML che definiscono variabili e che possono essere stili, stringhe utilizzate dall'applicazione, interi e tante altre tipologie di dati.

Riassumendo, da una parte abbiamo delle risorse XML che definiscono alcune parti dell'applicazione, come le GUI, i parametri e i valori, mentre dall'altra abbiamo il codice Java dell'Activity. Il ponte fra questi due viene gestito dalla classe R contenuta nella cartella **gen** e viene generato ad ogni compilazione del codice o aggiunta di un file XML.

È importante analizzare più in dettaglio il file **AndroidManifest.xml** perché in sua assenza o in caso di errata compilazione, viene compromesso l'utilizzo dell'applicazione. Questo file definisce i contenuti e il comportamento di un'applicazione, viene generato dal plugin ADT automaticamente una volta creato il progetto ed è memorizzato nella cartella principale. Questo file contiene tutte le informazioni che abbiamo specificato nella creazione del progetto, le Activity e i Service dell'applicazione, con i permessi che necessitano per funzionare correttamente. Le Activity verranno trattate in una sezione apposita, quindi daremo più avanti la definizione. I **Service** invece sono delle applicazioni che svolgono delle operazioni autonome e che vengono richiamati dalle attività per fornire alle applicazioni l'accesso all'hardware o a risorse esterne [38].

Come abbiamo visto nei precedenti capitoli, ogni applicazione gira all'interno di un proprio processo Linux, per cui ci sono delle restrizioni ben specifiche: ogni processo non può accedere alla memoria di un altro processo e ad ogni applicazione è assegnato uno specifico identificatore utente. Inoltre i file di un applicativo sono

protetti, cioè non possono essere letti o scritti da altri applicativi. È protetto inoltre l'accesso a particolari funzioni quali la fotocamera, la memoria o i messaggi. Pertanto per utilizzare certe risorse vanno inserite delle richieste nell'AndroidManifest.xml. Quando l'applicazione viene installata, il gestore dei pacchetti concede o non concede i privilegi a seconda di come li abbiamo configurati in questo file e l'utente viene sempre avvisato a quali risorse l'applicazione può accedere.

I principali permessi che si possono richiedere sono:

- READ_CONTACTS: leggere (ma non scrivere) i dati dei contatti dell'utente
- WRITE_CONTACTS: scrivere (ma non leggere) i dati dei contatti dell'utente
- RECEIVE_SMS: monitorare l'arrivo di messaggi SMS
- INTERNET: accedere e utilizzare la connessione Internet
- ACCESS_FINE_LOCATION: utilizzare il GPS
- CAMERA: utilizzo della fotocamera
- WRITE_EXTERNAL_STORAGE : scrivi della microSD esterna

L'AndroidManifest.xml si distingue dagli altri file XML per la presenza nell'intestazione del tag `< manifest >`: questo tag include i nodi che definiscono i componenti dell'applicazione, l'ambiente di sicurezza e tutto ciò che fa parte dell'applicazione.

Va indicata inoltre anche la versione minima di Android che deve avere il telefono per poter eseguire l'applicazione. Compiere una scelta accurata non è affatto semplice dato che si deve tenere conto di diversi fattori che verranno discussi nella prossima sezione.

5.2 View e GroupView

La **view** è sostanzialmente un'area rettangolare nello schermo responsabile del disegno degli elementi grafici e della cattura e gestione degli eventi generati dall'utente. Esistono oltre che alle View, le ViewGroup che non sono altro che delle View ma con altre View all'interno (per l'appunto come suggerisce il nome, gruppi di view). Per gestire agevolmente l'impaginazione dei controlli all'interno di un contenitore di oggetti in Android si usano i Layout. È comodo utilizzare i layout perché per mezzo di essi è più facile posizionare e orientare i controlli che compongono le view. Una view è un po' come un widget: bottoni, etichette, textbox sono tutti controlli e widget utilizzabili per comporre l'interfaccia grafica delle applicazioni. L'interfaccia può essere definita in due modi: interamente da XML, oppure in modo ibrido, cioè dichiarando i componenti nel file XML e poi modificandoli da codice. Il secondo approccio è quello più seguito dato che permette di modificare dinamicamente l'interfaccia [27].

5.3 Activity

Spostiamo ora l'attenzione su un altro elemento altrettanto importante, le Activity. La classe Activity è uno degli elementi centrali della programmazione delle app in Android. Un'Activity è una finestra che contiene l'interfaccia dell'applicazione con lo scopo di gestire l'interazione tra l'utente e l'applicazione stessa [1]. Le applicazioni possono definire un qualsiasi numero di Activity per poter trattare diversi parti del software: ognuna svolge una particolare funzione e deve essere in grado di salvare il proprio stato in modo da poterlo ristabilire successivamente come parte del ciclo di vita dell'applicazione. Per esempio, in una applicazione per la gestione dei viaggi, un'Activity potrebbe rappresentare la schermata che permette di aggiungere le tappe, un'altra permette di visualizzarle, un'altra permette magari di tenere conto delle spese.

Generalmente un'Activity corrisponde ad un'azione specifica che l'utente può fare: è pertanto fondamentale che le Activity vengano lanciate e gestite correttamente. Può essere lanciata e visualizzata una sola Activity alla volta (l'Activity in esecuzione è detta *foreground*). Le Activity che nello stesso momento non si trovano in *foreground* si dicono in *background* e potrebbero essere terminate dal sistema operativo qualora la memoria fosse insufficiente. Un'Activity deve quindi essere pronta ad essere interrotta e lanciata in qualsiasi momento [24].

5.4 Le azioni: Intent

Un Intent è un meccanismo che descrive un'azione specifica, come ad esempio “chiamare a casa” o “inviare un sms”: in Android praticamente ogni cosa passa attraverso un Intent e lo sviluppatore li può utilizzare per riusare o sostituire diversi componenti software.

Esistono Intent predefiniti che possiamo utilizzare qualora la nostra applicazione ne richieda l'utilizzo, per esempio se stiamo creando un'app per la gestione delle mail possiamo usare l'Intent già definito per l'invio delle mail senza dovercelo scrivere, oppure possiamo crearne di personali se non sono già presenti o se quelli già presenti non soddisfano pienamente le specifiche.

Possiamo vedere quindi un Intent come un'azione che possiamo far invocare ad Android: un Intent può essere molto specifico ed essere richiamabile da una specifica Activity oppure può essere generico e disponibile per qualsiasi Activity che rispetta determinati criteri.

Schematizzando, un Intent può essere utilizzato per:

- Trasmettere l'informazione per cui un particolare evento (o azione) si è verificato
- Esplicitare un'intenzione che una Activity o un Service possono sfruttare per eseguire un determinato compito o azione, solitamente con o su un particolare insieme di dati
- Lanciare una particolare Activity o Service

- Supportare l'interazione tra qualsiasi applicazione installata sul dispositivo Android, senza doversi preoccupare di che tipo di applicazione sia o di quale componente software gli Intent facciano parte

Lavorare senza Intent è quasi impossibile e comunque sconsigliato: il suo utilizzo infatti trasforma ogni dispositivo Android in una collezione di componenti indipendente ma facenti parte di un singolo sistema interconnesso [26].

5.5 Ciclo di vita di un'Activity

Nel paragrafo precedente abbiamo introdotto il concetto di Activity e illustrato il suo ruolo fondamentale nella programmazione Android. Tuttavia per assicurare un corretto funzionamento dell'applicazione, capirne il ciclo di vita è fondamentale. Dal momento in cui lanciamo l'applicazione fino al momento in cui essa viene messa in background infatti, l'Activity passa attraverso varie fasi: queste costituiscono il ciclo di vita dell'app. Per poter definire un'Activity, la nostra classe dovrà estendere la super classe Activity, che definisce una serie di metodi per gestire tutti gli eventi che ne governano il ciclo di vita. Tutti i metodi sono illustrati in Figura 5.2:

Il primo metodo che viene eseguito quando un'Activity viene invocata è l'**onCreate()** che definisce gli elementi di base per il funzionamento. Se nell'**onCreate()** si fa utilizzo anche del metodo **setContentView()** è possibile invece definire un layout personalizzato per la propria Activity. Una volta che ne viene portata a termine l'esecuzione, viene eseguito il metodo **onStart()** che si attiva non appena l'Activity diventa visibile all'utente. Non appena invece l'utente inizia l'interazione con l'Activity viene chiamato il metodo **onResume()**. Da questo momento in poi viene eseguita normalmente l'Activity, fin quando non sopraggiunge un altro evento come la chiamata ad un'altra Activity. In questo caso viene invocato il metodo **onPause()** nel quale solitamente si salvano le modifiche ai dati e si bloccano le animazioni per ridurre il carico di lavoro della CPU. Se si richiama l'Activity precedente, questa riprenderà a funzionare dal metodo **onResume()**, mentre se l'activity diventa non più visibile all'utente viene chiamato il metodo **onStop()**, il quale provvede, in mancanza di memoria, ad eliminare le Activity in sospenso non necessarie. Nel caso venga riaperta l'Activity dopo l'**onStop()**, questa dovrà invece ricominciare la sua esecuzione, tramite il metodo **onRestart()**, dal metodo **onStart()**. Se l'app viene terminata (dall'utente o per mancanza di memoria) viene chiamato il metodo **onDestroy()** che chiude tutte le Activity e le app.

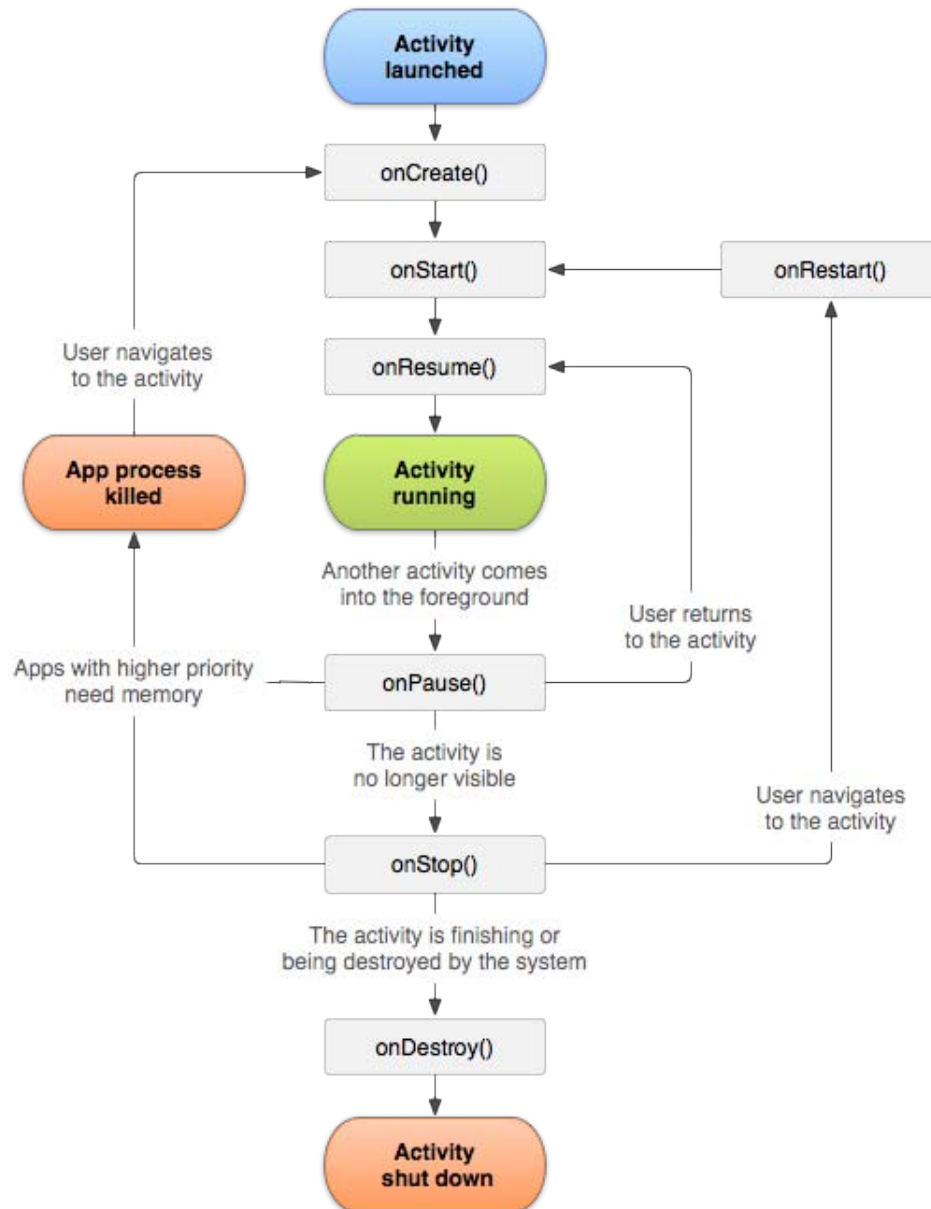


Figura 5.2: Ciclo di vita di un'Activity [13]

Come strutturare l'applicazione

Navigando nel Google Play possiamo notare che ci sono un'infinità di app. Alcune sono simili tra di loro mentre altre svolgono funzioni totalmente diverse. Se allora due app sono completamente diverse, molto probabilmente dovranno essere strutturate in modo diverso. Questo comunque non esclude che app simili sia organizzate diversamente. Senza concentrarci su una specifica applicazione, vediamo come deve essere strutturata un'applicazione in generale, seguendo sempre i consigli di Google. L'applicazione che si andrà a creare dipende molto dai contenuti e dalle funzioni che si vogliono offrire agli utenti. Ci sono in genere tre tipi di organizzazione:

- Applicazioni costituite e gestite da una sola activity in una sola schermata (ad es. la calcolatrice o la fotocamera).
- Applicazioni il cui scopo principale è navigare attraverso diverse activity e diversi livelli (ad es. telefono).
- Applicazioni che combinano un'ampia gamma di view e navigazione attraverso molti livelli (ad es. Gmail, Playstore).

Una tipica applicazione Android consiste in un livello superiore e della schermate (livelli inferiori) per la visualizzazione di ulteriori dettagli ed eventualmente per permettere delle modifiche. Se la gerarchia di navigazione è profonda e complessa si introducono le cosiddette *Category Views* per connettere il livello superiore con i livelli inferiori (Figura 6.1). Il concetto di livello è del tutto identico al concetto di cartelle e sottocartelle.

Livello superiore Il livello superiore è, a meno che non sia presente una splash screen, la prima schermata che si visualizza all'apertura dell'applicazione e quindi bisogna dedicarci particolare attenzione. Bisogna evitare schermate di sola navigazione, dando la possibilità all'utente di raggiungere subito i contenuti, rendendoli così la parte più importante della prima schermata.

Bisogna dare un taglio originale alla propria app utilizzando layout innovativi, belli, facili da utilizzare. Per la gestione dei contenuti dell'app è possibile ricorrere a moltissimi strumenti forniti da Android (ma nessuno vieta di crearne dei propri, anzi!) come per esempio l'Action Bar, le Tab, il Navigation Drawer, le notifiche, i widget e molti altri. Di seguito analizzeremo quelli appena elencati.

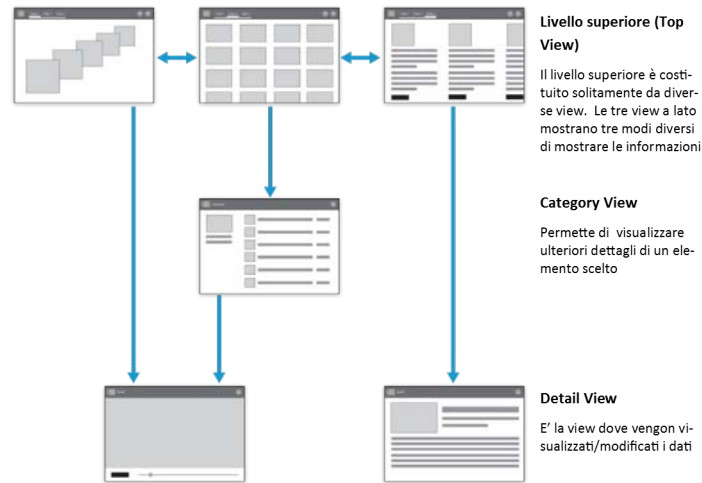


Figura 6.1: Struttura di una App

6.1 L'Action Bar

L'Action Bar è una porzione di schermo nella parte alta ed è sempre presente in ogni app. L'utilizzo dell'Action Bar porta diversi vantaggi:

- Mette in primo piano le azioni che si prevede che un utente possa utilizzare più frequentemente, come la ricerca o la creazione di una nuove voce
- Riduce l'ingombro dato che le funzioni meno utilizzate vengono nascoste da un pulsante (Action Overflow)
- Ha uno spazio per poter inserire nome o logo della propria applicazione



Figura 6.2: Action Bar

Dall'immagine 6.2 si possono capire più facilmente i quattro componenti fondamentali dell'Action Bar:

1. Icone dell'applicazione. Scegliendo un'icona del tutto originale, la propria app sarà riconoscibile dalle altre e avrà un'identità tutta sua.
2. View Control. Serve per potersi spostare tra le varie view agilmente: è di solito un menù a tendina che racchiude le varie view di cui è composta l'app. Se l'app però ha un'unica view o non si vuole implementare questa funzione, si può sostituire il tasto con un titolo che riguarda l'app. Se si vuole si può utilizzare questa parte di barra anche solo per visualizzare un titolo relativo alla schermata visualizzata.

3. Action Buttons. Mostra i pulsanti che vengono più utilizzati nell'applicazione. I pulsanti che per motivi di spazio non riescono a essere visualizzati, vengono 'inseriti' nel tasto più a destra. Tenendo premuto uno di questi tasti se ne può visualizzare il nome.
4. Action Overflow. Racchiude i pulsanti meno utilizzati

Non esiste solamente l'Action Bar superiore ma esistono altre due barre simili, dette Top Bar e Bottom Bar. La prima di queste due permette di navigare tra le varie view, mentre la seconda è del tutto simile all'Action Bar solo che è posizionata in basso [9].

Una funzione che potrebbe essere utile implementare come alternativa graficamente migliore e più funzionale, è la Contextual Action Bar. La CAB è una barra che può essere fatta comparire tenendo premuto un oggetto per poter visualizzare i comandi che possono agire su di esso. Un esempio più esplicativo lo si può vedere dalla Figura 6.3.

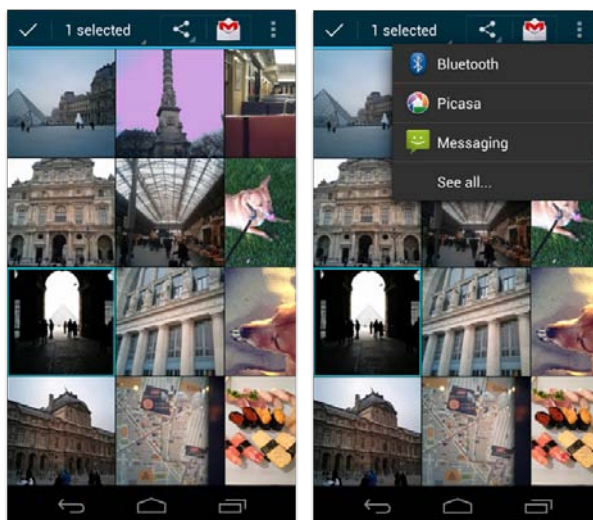


Figura 6.3: Contextual Action Bar

Un esempio che mostra ancora più chiaramente i vantaggi dell'Action Bar è l'app Orari Treni, che ha ricevuto milioni di download, soprattutto tra gli studenti universitari. Guardando la barra di Figura ?? si capisce immediatamente tutto ciò di cui si ha bisogno: il titolo spiega che si stanno visualizzando le soluzioni trovate per il viaggio richiesto, i vari comandi disponibili (Action Buttons), come Aggiorna e Salva, sono ben visibili e posizionati correttamente. Ulteriori opzioni le si trovano nel tasto alla destra degli Action Buttons, dentro l'Action Overflow. L'utilizzo di icone semplici e pittografiche, come consiglia Google, permette, anche se non si è mai utilizzata l'applicazione, di capirne subito il significato. Vediamo che la seconda versione è molto più chiara anche nelle richieste: non ci sono dubbi su dove vadano inserite le stazioni di Partenza e di Arrivo, gli orari, eccetera. Questa versione è anche graficamente più gradevole grazie all'utilizzo di API di livello maggiore e ad un impegno dello sviluppatore che, per visualizzare i viaggi, si è creato un elemento grafico personalizzato.



Figura 6.4: Orari treni

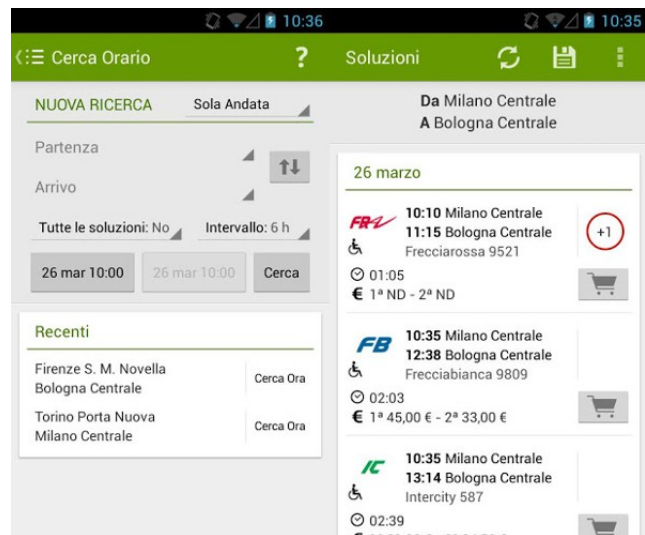


Figura 6.5: Orari treni: grafica Holo

6.2 Il Navigation Drawer

Introdotta con le ultime versioni di ICS, il Navigation Drawer rende la navigazione ancora più semplice. È un pannello che scorre dal margine sinistro dello schermo verso il centro e mostra le principali opzioni di navigazione dell'applicazione. Il Navigation Drawer andrà quindi a coprire parte di ciò che si sta visualizzando ma non coprirà l'Action Bar. Una volta che si sarà aperto del tutto l'Action Bar si modificherà: cambierà il titolo con il nome dell'applicazione e disabiliterà la CAB, lasciando solo l'Overflow Menù. Per chiuderlo ci sono molti modi: toccare

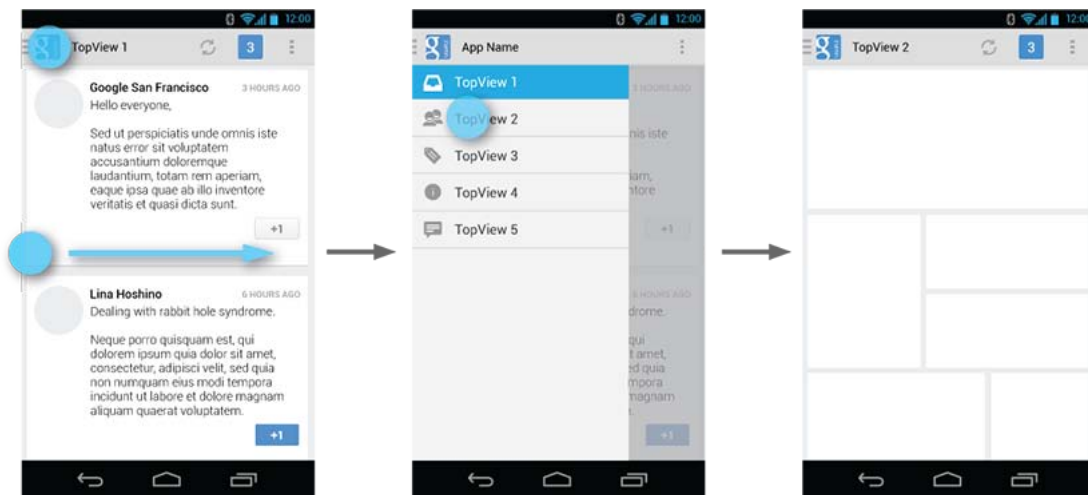


Figura 6.6: Navigation Drawer

lo schermo al di fuori del Navigation Drawer, scegliere una delle voci proposte, richiuderlo facendo lo swipe verso sinistra oppure con il tasto Indietro. L'utilizzo del Navigation Drawer è molto diffuso ed è consigliato in quelle app in cui ci sono molte schede da visualizzare e che quindi starebbero nell'Action Bar [18]. Si pensi ad esempio a Google Music, Google Earth o Gmail. In figura 6.7 possiamo vederne l'utilizzo concreto nell'app Gmail: tutte le schede, come Posta in Arrivo, Posta in uscita, eccetera, non troverebbero spazio nell'Action Bar e non potrebbero essere inserite logicamente nell'Action Overflow perché destinato a contenere solo azioni ed impostazioni.

6.3 Le notifiche

Tra le componenti di cui si può dotare un'app troviamo le notifiche. Il sistema di notifiche permette all'app di tenere l'utente sempre informato riguardo gli eventi che accadono, come l'arrivo di nuovi messaggi in chat o eventi segnati nel calendario. Le notifiche fungono quindi anche da promemoria in quanto l'app tramite le notifiche può ricordare all'utente, per esempio, il suo appuntamento.

Con l'uscita di Jelly Bean le notifiche hanno ricevuti diversi aggiornamenti riguardo la loro struttura. La modifica più utile forse è la possibilità di interagire con le notifiche direttamente dal Notification Drawer, mentre le altre modifiche sono più che altro grafiche: possibilità di ridimensionarle e scegliere quanti detta-

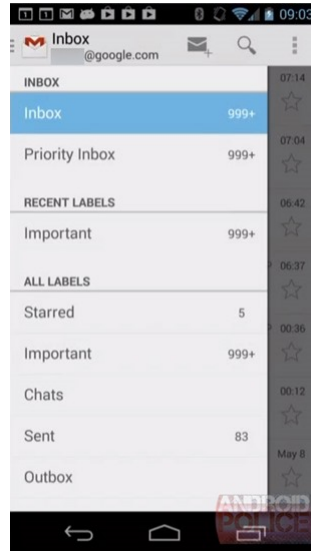


Figura 6.7: Gmail: il Navigation Drawer

gli visualizzare e l'introduzione della priorità, che ordina quindi le notifiche per importanza.

Le notifiche sono composte da quattro componenti principali, Figura 6.8. La prima di queste è un'icona che contiene la foto del mittente o l'icona dell'applicazione che sta generando la notifica. Non possono mancare inoltre un titolo (o nome del mittente) e il messaggio, seguiti dalla data e l'ora della notifica. Un'icona secondaria permette di identificare l'applicazione nel caso venga mostrata la foto del contatto nell'icona principale. Con Jelly Bean le notifiche si possono espandere per poter

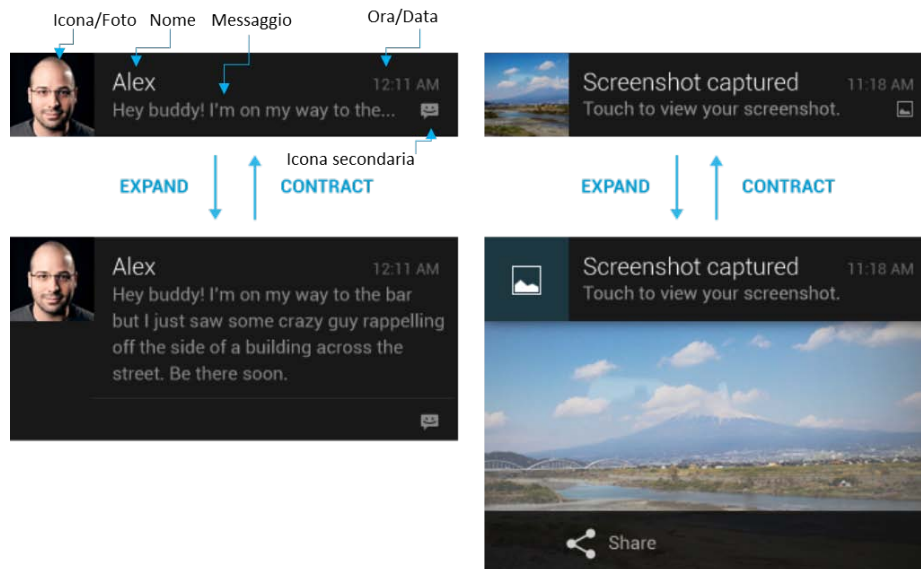


Figura 6.8: La struttura di una notifica

visualizzare maggiori informazioni. Questo può essere utile per poter leggere più righe di un messaggio oppure visualizzare una porzione maggiore di un'immagine; l'utente può anche ingrandire l'immagine con il pinch-to-zoom. Sempre con questa

ultima versione di Android, nelle notifiche sono state introdotte azioni supplementari che vengono mostrate nella parte inferiore del layout della notifica. Azioni tipiche che si possono trovare nelle notifiche del Calendario sono ad esempio Rispondi alla mail oppure Snooze [19].

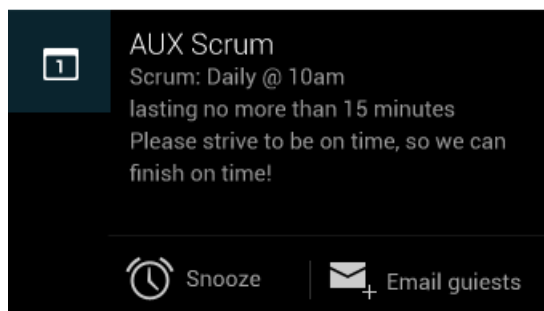


Figura 6.9: Calendario: notifiche e azioni

6.4 I Widget

I widget sono essenziali per quanto riguarda la personalizzazione della schermata Home. Grazie ad essi con un semplice colpo d'occhio si possono ottenere velocemente le informazioni più importanti riguardo un'app direttamente dalla schermata Home. I widget possono essere spostati nelle varie schermate della Home e, se lo supportano, possono essere ridimensionati al fine di visualizzare più o meno informazioni [22]. Esistono diverse tipologie di widget:

- Widget di informazione, Figura 6.10, primo elemento: danno qualche piccola informazione importante per l'utente. Esempi tipici sono i widget per il meteo, l'orologio, i tracker GPS. Solitamente toccando il widget si apre l'app associata che permette di visualizzare il resto delle informazioni.
- Widget di collezione, Figura 6.11, : servono per visualizzare una collezione di elementi dello stesso tipo, come una collezione di immagini, di sms o di mail. Nell'immagine si vede il Widget Calendario, che mostra giorno per giorno nella Home le attività da svolgere.
- Widget di controllo, Figura 6.10, secondo elemento: lo scopo principale di questi widget è racchiudere le operazioni più utilizzate dall'utente e poterle attivare senza dover aprire per forza l'app che le gestisce. Sono sostanzialmente dei controlli remoti per un'app. Esempi tipici sono i widget per la riproduzione della musica, in cui si può velocemente premere Avanti, Indietro, Pausa. Solitamente questi widget non danno informazioni all'utente.
- Widget ibridi, Figura 6.10, terzo elemento: sono combinazioni dei precedenti widget. Ad esempio il riproduttore musicale nominato sopra può essere integrato con una barra che visualizzi il nome del brano in esecuzione.

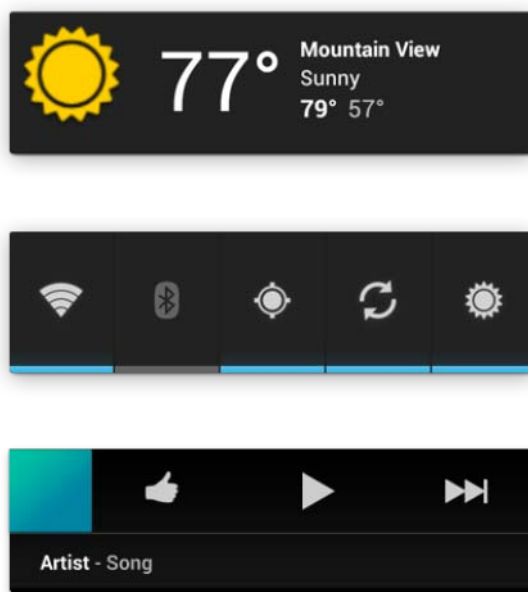


Figura 6.10: Widget: informazione, controllo, ibrido



Figura 6.11: Widget di collezione

Guadagnare con le App

Chi ha pensato di sviluppare un app sicuramente avrà anche pensato alla possibilità di ricavare qualche soldo dalla sua vendita. Ma spiccare nel mercato delle app è una cosa abbastanza difficile, sia perché l'app creata deve essere apprezzata dal pubblico, sia perché ci sono diversi modi di guadagnare con le app, alcuni più efficienti ed altri meno. In questo capitolo illustreremo quindi gli aspetti economici delle app e le tecniche remunerative ad esse legate.

7.1 Tecniche remunerative

Esistono tre possibilità per guadagnare soldi vendendo app, ognuna con i suoi vantaggi ed i suoi svantaggi [30].

App a pagamento: È la modalità più classica e quella che comporta un minore sforzo da parte del programmatore, in quanto, rispetto alle altre modalità, non necessita un ulteriore inserimento di codice nel programma. L'app viene semplicemente messa in vendita nel PlayStore e gli utenti possono scaricarla pagando. App abbastanza semplici hanno un prezzo medio intorno ad 1€ o 2€ mentre altre più elaborate, come i navigatori, possono arrivare anche a 50€. Escludendo questo ultimo caso si nota quindi che, per guadagnare in questo modo, bisogna vendere un gran numero di app. Dal prezzo imposto ricordiamo inoltre che va tolto il 30%, che resta a Google. Questa modalità di guadagno si adatta bene ad applicazioni utilizzate di tanto in tanto da dagli utenti perchè il guadagno è dato dal numero di download e non dalla frequenza di utilizzo. Va precisato inoltre che un utente che vede un'app a pagamento si aspetta che sia di livello professionale, con pochi bug ed un'ottima interfaccia.

Vantaggi: semplice da implementare, guadagno proporzionale al numero di download.

Svantaggi: gli utenti pagano l'app e hanno gli aggiornamenti gratuiti a vita. Si trovano spesso nei market 'pirata' moltissime app a pagamento scaricabili gratuitamente.

Pubblicità in-app: la pubblicità gioca un ruolo fondamentale nella vendita dei prodotti e la stessa affermazione è valida anche nel campo della app: inserendo

banner pubblicitari all'interno delle app è possibile ottenere dei guadagni. Questo modello va adottato solo per quelle app che vengono utilizzate frequentemente perché per avere buoni guadagni occorre realizzare tantissimi click. Bisogna considerare infatti che con un rapporto tra impression (visualizzazioni) e click intorno al 3% e con un guadagno di 0,5 centesimi a click, serve un enorme numero di visualizzazioni anche solo per arrivare ad 1€. Questo guadagno però è giornaliero perché ogni volta che l'utente apre l'app, arrivano l'impression e quindi il click. Inoltre, rispetto ai siti Web, nel campo mobile il rapporto tra impression e click (detto CTR) è leggermente superiore, probabilmente per il maggiore numero di click involontari sul banner se questo è posizionato in modo strategico.

Vantaggi: il guadagno è proporzionale all'effettivo utilizzo dell'app, si continua a guadagnare anche con (relativamente) pochi download purché gli utenti siano molto fidelizzati.

Svantaggi: le app con pubblicità vengono spesso giudicate 'poco professionali' e alcuni utenti non le vedono di buon occhio. Esistono anche dei software che permettono di filtrare la pubblicità, azzerando di fatto i guadagni dello sviluppatore.

Prodotti in-app: questa strategia di marketing è nata di recente e consiste di distribuire gratuitamente l'app con delle funzioni base e poi vendere a pagamento delle funzioni aggiuntive o temporanee. Comprare delle funzionalità aggiuntive significa di fatto sbloccare funzionalità già presenti nell'app, mentre comprare quelle temporanee significa ottenere qualcosa di immediato e spesso 'usa e getta'. Ad esempio il Mighty Eagle di Angry Birds permette di superare il livello in corso dietro il pagamento di un cifra irrisoria. Questo modello si applica bene ai giochi.

Vantaggi: l'utente arriva a spendere molti soldi senza (quasi) accorgersene e tecnicamente il meccanismo è molto ben protetto dalla pirateria.

Svantaggi: la programmazione si complica parecchio.

Donazioni: non è una vera e propria tecnica remunerativa ma permette di ripagarsi un poco gli sforzi fatti per la programmazione. Questa tecnica viene solitamente adottata in quelle app che vengono pubblicate solo per il piacere di fornire strumenti utili alla gente o se non si ha intenzione di venderla o di adottare le precedenti tecniche. Tramite un pulsante, di solito inserito tra le opzioni, è possibile fare una piccola donazione (50 centesimi, 1€. . .) allo sviluppatore come ringraziamento e incoraggiamento allo sviluppo di altre app.

Vantaggi: non è necessario inserire nuovo codice nel programma

Svantaggi: non porta, solitamente, grandi guadagni

Le ricerche dimostrano che gli utenti Android sono più propensi a scaricare App gratuitamente, mentre gli utenti Apple non si fanno problemi a spendere soldi per ottenere l'app che desiderano. Conviene quindi, delle tre tecniche proposte, adottare la terza: non introduce fastidiose pubblicità e se l'app è veramente utile si prospettano grandi guadagni ma d'altro canto ci vuole un ulteriore sforzo in programmazione.

7.2 Pubblicizzare l'app

Scegliere una buona tecnica remunerativa non è abbastanza per avere successo con la vendita di app: bisogna farsi conoscere e farla conoscere. Lo sviluppo del Web è un fattore che ci può essere sicuramente di aiuto, infatti può essere un'ottima idea segnalare l'app ai vari blog che nella rete parlano di Android, pubblicizzarla su Facebook e Twitter, inviare mail agli autori dei blog (i più visitati sono Androidiani e AndroidWorld). Ogni applicazione però verrà sicuramente pubblicata nel Google Play, il negozio online che nel 2012 ha sostituito il vecchio Android Market, in cui è possibile cercare e scaricare le applicazioni di cui abbiamo bisogno. La pubblicazione di una app nel Google Play non è solo un modo per consentire il download, ma è anche un modo per connettere gli utenti e aumentare la visibilità della propria app. Il Google Play infatti è preinstallato in più di 400 milioni di dispositivi in più di 130 paesi e conquista quasi un milione di utenti ogni giorno, i quali personalizzano il proprio telefono con giochi e applicazioni provenienti proprio dal negozio di Google.

Quando un'app è presente nel Google Play, gli utenti possono esprimere una loro opinione a riguardo, votandola e commentandola. Gli utenti votano l'applicazione, da 1 a 5 stelle, dopo aver effettuato il download e possono anche aggiungere un breve commento riguardo l'impressione ricevuta. Quando altri utenti guarderanno l'app per decidere se scaricarla o meno, baseranno la loro decisione sul voto medio ricevuto e la soddisfazione espressa nei commenti. Nella Figura 7.1 si può vedere chiaramente che la maggior parte degli utenti è rimasta soddisfatta: circa 46000 download su 50000 hanno almeno 4 stelle e i commenti sono del tutto positivi e raccomandano il download agli utenti futuri.

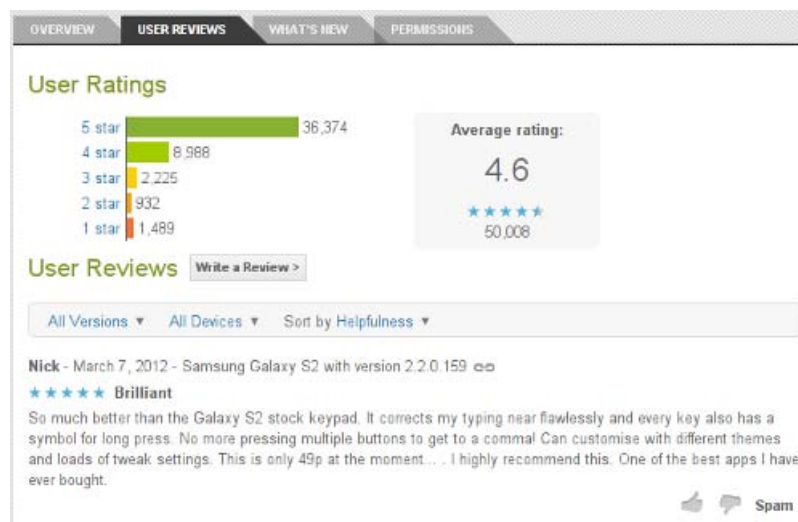


Figura 7.1: Google Play: commenti e voti di una app [16].

È importante, quando si pubblica un'app, scegliere con cura la categoria a cui appartiene: giochi, fotografie, sport e tempo libero sono solo alcune tra più di 30 categorie presenti nel negozio. All'interno di ogni categoria le app sono ordinate in base ai voti, ai commenti, al paese di provenienza e tantissimi altri fattori. Alcune categorie più popolari sono invece curate personalmente dallo personale di Google.

Chi invece non vuole utilizzare le categorie, può utilizzare la funzione di ricerca. Inserendo il nome dell'applicazione o delle parole chiave si possono cercare tutte le applicazioni che contengono quelle parole. È importante allora, durante la fase di pubblicazione, assegnare dei **tag** appropriati alla propria app. I tag sono delle parole chiave da associare all'applicazione e sono utili proprio durante la ricerca: ad esempio in un'app per la gestione dei viaggi, dei tag appropriati potrebbero essere Viaggi, Gestione, Vacanza. L'utente che quindi vuole trovare un'app per le vacanze, scriverà nella barra di ricerche 'Vacanza', e potrà trovare l'app.

Android facilita anche la promozione delle app nelle pagine Web degli utenti. Chi possiede infatti un sito Web o un blog, può fare uso di una funzione disponibile sul sito Android Developers [11] che consente di creare una sorta di badge per promuovere la propria app. Compilati i campi richiesti, il sito genera automaticamente il codice HTML da incorporare nella propria pagina Web [16].

Language: Italiano

Package name: com.viaggio clear

or

Publisher name: Example, Inc.

APP ANDROID SU Google play
 APP ANDROID SU Google play

DISPONIBILE SU Google play
 DISPONIBILE SU Google play

Build my badge

Copy and paste this HTML into your web site:

```
<a href="https://play.google.com/store/apps/details?id=com.viaggio">
  
</a>
```

Try it out:

DISPONIBILE SU Google play

Figura 7.2: I badge di Android

7.3 Controllare le distribuzioni

Controllare la visibilità della propria applicazione potrebbe essere una buona strategia di marketing: si può infatti scegliere, durante la fase di pubblicazione, quali paesi potranno avere accesso a questa app e quali terminali la potranno scaricare. La pubblicazione dell'app avviene istantaneamente, come anche gli aggiornamenti al prezzo di vendita o ai paesi a cui la si vuole rendere accessibile. Infatti da Google Play Developer Console (per capire cos'è, si veda il capitolo Pubblicare la propria App), si possono modificare le impostazioni quante volte si

vogliono, e le modifiche vengono apportate nel giro di qualche ora, senza dover modificare in alcun modo la propria applicazione [15].

Distribuzione geografica. Scegliere attentamente i paesi in cui l'app sarà disponibile è una buona strategia. Infatti, pubblicare app in un paese straniero, sarebbe da fare solo se la propria app è stata tradotta anche nella lingua parlata in quel paese. Ancora una volta Google mette a disposizione uno strumento per poter fare questo: come si vede dalla Figura 7.3, nel Google Play Developer Console si può scegliere sia il paese in cui rendere l'app disponibile, sia il prezzo che si intende far pagare.

Country	Price	Tax	Options
<input type="checkbox"/> Hungary			
<input checked="" type="checkbox"/> Iceland			
<input checked="" type="checkbox"/> India	INR 108.51		
<input type="checkbox"/> Indonesia			
<input checked="" type="checkbox"/> Ireland	EUR 4.00	incl. 0% tax	Hide options
Limit distribution to these carriers:			
<input type="checkbox"/> Vodafone - IE			
<input checked="" type="checkbox"/> Israel	ILS 7.35	incl. 0% tax	
<input checked="" type="checkbox"/> Italy	EUR 1.60	incl. 5% tax	Show options
<input type="checkbox"/> Jamaica			
<input checked="" type="checkbox"/> Japan	JPY 200	incl. 7% tax	Hide options
Limit distribution to these carriers:			
<input type="checkbox"/> NTT DOCOMO			

Figura 7.3: Google Play Developer Console: Paesi di distribuzione [15]

Restrizione sui dispositivi. È possibile, sempre nella stessa schermata, stabilire anche quali dispositivi potranno installare l'app. Non è da confondere con l'argomento trattato nel capitolo 4.2. In quel capitolo si decideva per quale versione è conveniente sviluppare dal punto di vista informatico. Qui, di tutti i dispositivi compatibili con la scelta effettuata durante la programmazione, se ne possono escludere alcuni. Il motivo di escludere alcuni dispositivi potrebbe risiedere, per esempio, in qualche problema hardware riscontrato con un tipo particolare di dispositivo o nel fatto che si è deciso di non supportare certe risoluzioni di schermi e quindi si vogliono escludere i dispositivi che le montano.

Statistiche. Per tenere sotto controllo l'andamento dei download nel tempo, sono disponibili delle statistiche che tengono conto di quante volte l'app viene installata, aggiornata o disinstallata. Si può controllare anche l'andamento secondo specifici fattori: ad esempio rispetto ad una certa lingua, ad un certo dispositivo, versione

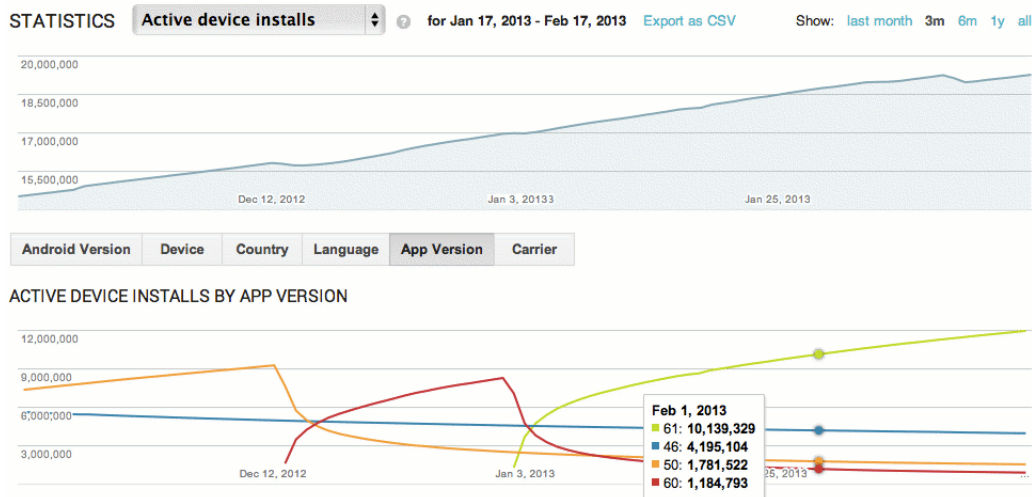


Figura 7.4: Google Play Developer Console: statistiche [15]

o paese. In Figura 7.4 si vede, nella prima immagine, l'andamento nel tempo relativo ai dispositivi che utilizzano l'applicazione, mentre sotto si vede l'andamento relativo alle varie versioni. Si noti che ci sono dei picchi in cui l'app è stata più utilizzata/scaricata, probabilmente dovuti a promozioni o aggiornamenti rilasciati.

Distribuzione avanzata. È sempre utile avere un feedback dagli utenti, specialmente prima del lancio dell'app. Con il Google Play è possibile distribuire delle versioni ancora in fase di sviluppo a piccoli gruppi di tester, magari sparsi nel mondo. Si può cominciare distribuendo, per esempio, ad un piccolo gruppo di "alpha tester"¹ e poi espandersi, con un'app già più completa, verso un gruppo più grande di "beta tester"². Si possono selezionare sia tutti gli utenti di un paese, sia un ristretto numero di essi, magari tramite invito per email.

APK Multipli. Nella maggior parte dei casi è più facile creare un'app che supporti tutte le risoluzioni di schermi e versioni in un singolo file APK. L'estensione APK indica un file Android Package. Questo formato di file, una variante del formato .JAR, è utilizzato per la distribuzione e l'installazione di componenti in dotazione sulla piattaforma per dispositivi mobili Android. Un file APK contiene in genere, al suo interno, le seguenti cartelle e file di cui è stata data una descrizione nei capitoli precedenti: res, AndroidManifest.xml, classes.dex, META-INF. È altamente raccomandato distribuire il singolo APK agli utenti, perché è il modo più facile per controllare e gestire al meglio l'app. Ma potrebbe essere necessario avere degli APK diversi per categorie diverse di dispositivi ed anche questa soluzione è realizzabile grazie al Google Play. È presente infatti un'opzione chiama "APK

¹La versione alfa o alpha, in informatica, individua un software che è in fase di sviluppo, le cui funzionalità non sono ancora state completamente implementate e che presenta sicuramente una lunga serie di bug che dovranno essere risolti. (Fonte: http://it.wikipedia.org/wiki/Versione_alfa)

²La versione beta, in informatica, è una versione di un software non definitiva, ma già testata dagli esperti, che viene messa a disposizione di un numero maggiore di utenti, confidando proprio nelle loro azioni imprevedibili che potrebbero portare alla luce nuovi bug o incompatibilità del software stesso. (Fonte: http://it.wikipedia.org/wiki/Versione_beta)

multipli“ che permette di creare diversi APK con lo stesso nome ma che differiscono, ad esempio, nel formato di compressione delle texture, nelle risoluzioni di schermi supportate o nelle versioni di Android supportate. Gli APK vengono caricati nel negozio online come un singolo APK ma, al momento del download, Android sceglie automaticamente quello più adatto al dispositivo.

Proteggere l'app. La pirateria informatica è molto avanzata e Google mette a disposizione degli sviluppatori due strumenti per proteggere le proprie app.

Il primo strumento è la licenza Google Play, un servizio online che va implementato nell'app. Con questo servizio, l'app interroga periodicamente un server particolare di Google per determinare se l'app ha una licenza corretta. Questo strumento è applicabile sia per applicazioni a pagamento che applicazioni gratuite.

Il secondo strumento è invece un servizio di crittografia per proteggere le app a pagamento. Quando l'app viene scaricata da un dispositivo (con versione Android 4.1 o successive), Google cifra l'app in modo che possa essere eseguita solo dall'utente che l'aveva scaricata e solo sul dispositivo con cui l'aveva scaricata. Questo strumento non ha bisogno di ulteriore sforzo da parte del programmatore, in quanto lo strumento si attiva automaticamente quando l'app viene resa a pagamento.

Publicare la propria App

Una volta terminata l'applicazione e testata per bene, è tempo di pubblicarla. Questa operazione non è particolarmente difficile, ma può creare qualche dubbio per chi è poco esperto e quindi ne parleremo qui di seguito.

Creare l'account per la pubblicazione. La prima cosa da fare è **registrare un account** nel Google Play Developer Console all'indirizzo <https://play.google.com/apps/publish/>. Inserite le proprie generalità, come il nome dello sviluppatore e indirizzo email, basta accettare i termini del contratto. Per poter pubblicare le app bisogna effettuare una pagamento di 25\$ utilizzando il Google Wallet. Se non si è già in possesso di tale account, lo si può creare nel corso della registrazione. Nella Figura 8.1 vediamo come si prospetta la schermata relativa alla gestione delle App e in Figura 8.2 la scheda per la modifica dei dettagli dell'account. Nel caso più persone lavorino

APP NAME	PRICE	ACTIVE / TOTAL INSTALLS	AVG. RATING / TOTAL #	CRASHES & ANRS	LAST UPDATE	STATUS
Animal Translator 1.1	Free	12,078 / 185,410	★ 3.29 / 566		Apr 21, 2010	Published
Earthquake! 3.8	Free	71,426 / 785,829	★ 4.15 / 6,212		Feb 1, 2013	Published

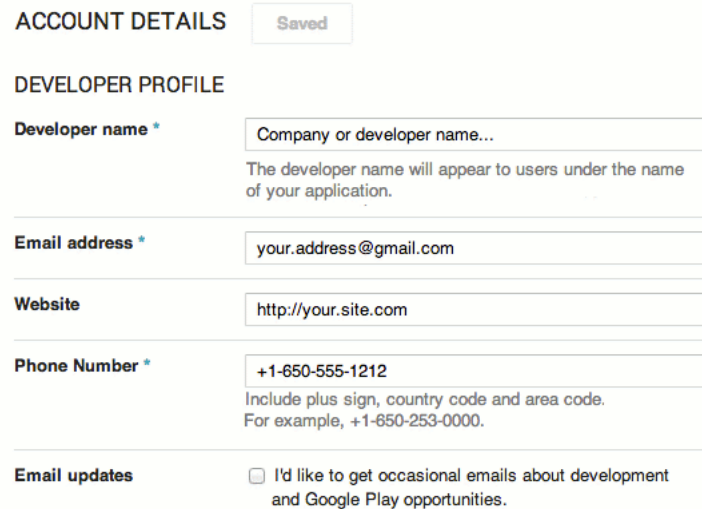
Page 1 of 1

Figura 8.1: Google Play: gestione app

alla stessa applicazione è possibile collegare più account per accedere al Developer Console: uno solo di essi sarà però amministratore, mentre gli altri avranno azioni limitate.

I file necessari per pubblicare un'app. Una volta ottenuto l'account si è pronti per caricare i file necessari alla pubblicazione.

La prima cosa da fare è **firmare** il file **apk** della propria app. Il file apk (che è all'incirca come un file .exe di Windows) viene generato ad ogni compilazione



ACCOUNT DETAILS Saved

DEVELOPER PROFILE

Developer name *
The developer name will appear to users under the name of your application.

Email address *

Website

Phone Number *
Include plus sign, country code and area code.
 For example, +1-650-253-0000.

Email updates I'd like to get occasional emails about development and Google Play opportunities.

Figura 8.2: Google Play: dettagli account

del progetto nella cartella bin del proprio progetto, ma non può essere pubblicato così com'è. Bisogna infatti firmarlo, come succede con le Applet o i progetti Java ME, tramite un nostro certificato. Per ulteriori dettagli su come ottenere il certificato è bene consultare la guida ufficiale di Google all'indirizzo <http://developer.android.com/tools/publishing/app-signing.html> perché è un processo abbastanza lungo. Creato il certificato possiamo firmare l'applicazione tramite un comodo strumento presente nell'ADT di Eclipse: basta fare click con il tasto destro sul progetto, e poi su *AndroidTools* \mapsto *ExportSignedApplicationPackage* (Figura 8.3). Una volta selezionato il progetto e premuto *Next* viene richiesto il **keystore**, una chiave privata per proteggere la propria app, creabile sempre da Eclipse. L'ultimo passo consiste nell'inserire l'alias definito durante la creazione del keystore. Per informazioni più dettagliate è possibile consultare le guide agli indirizzi: <http://mobile.tutsplus.com/tutorials/android/publish-to-android-market/> e <http://developer.android.com/tools/publishing/preparing.html> Ora dobbiamo dotarci di alcuni file per poter caricare l'app:

- Il file apk precedentemente firmato
- Due screenshot dell'applicazione in formato 320x480, 480x800, 480x854, 1280x720, 1280x800 in formato PNG o JPEG 24 bit
- Icona dell'applicazione ad alta risoluzione 512x512, PNG o JPEG 32 bit di dimensione massima 1024kB
- Immagine funzione in formato 1024x500 PNG o JPEG 24bit
- Una descrizione dell'applicazione, possibilmente in più lingue

Ora, dopo aver effettuato il login alla Console per la gestione delle app, basta scegliere l'opzione per aggiungere una nuova app e caricare tutto il materiale richiesto.

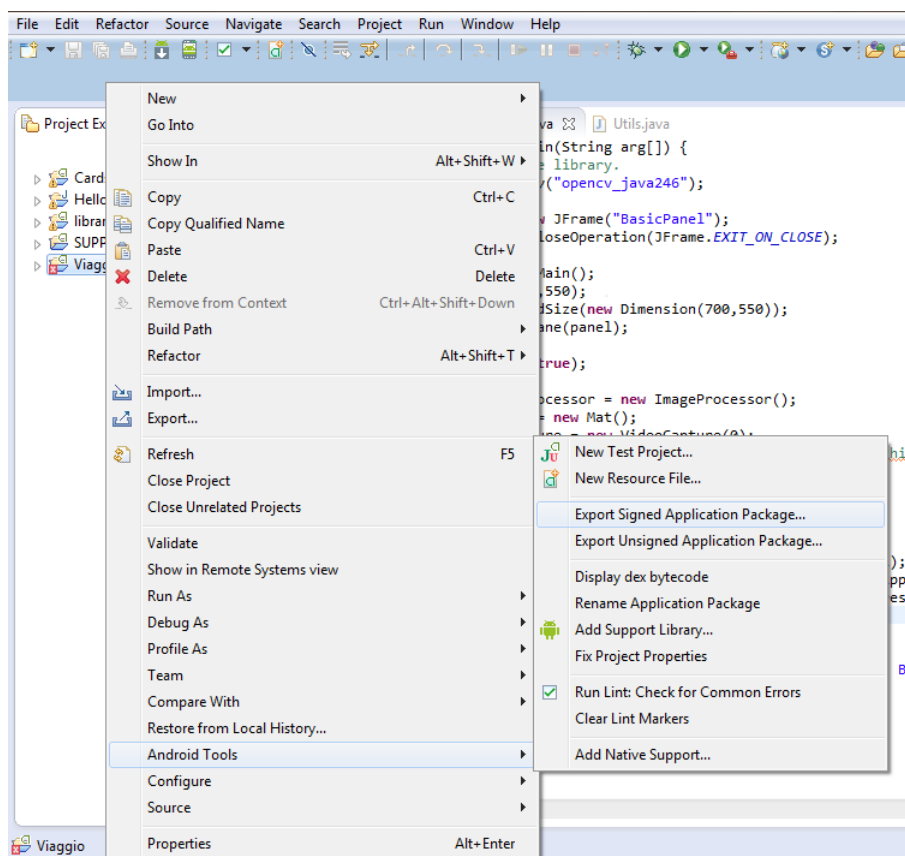


Figura 8.3: Eclipse: firmare l'app

Creare un account per vendere le app. Se si vogliono vendere le proprie app sul Google Play, bisogna creare anche un account speciale, detto Google Wallet Merchant Account. Per fare cioè basta effettuare l'accesso al proprio account Google Play Developer Console creato precedentemente, aprire la scheda Financial Reports, cliccare 'Crea Merchant Account' e compilare i campi richiesti. Verrà inviata una mail che chiede di confermare l'iscrizione. Una volta verificata, tramite il Developer Console possiamo gestire le opzioni di pubblicazione delle app nel Google Play.

Rilasciare aggiornamenti. Può capitare di dover rilasciare correzioni di piccoli bug o nuove funzioni. Per fare questo, oltre alle modifiche che si apportano al codice, bisogna incrementare un valore che si trova nell'AndroidManifest.xml che ne determina la versione. Se questo valore non viene incrementato, non è possibile caricare una nuova applicazione.

```
<manifest xmlns:android="http://schemas.android.com/apk/res/
  android"
    package="xxx.xxxxxx" android:versionCode="1"
    android:versionName="1.0">
```

Il **versionCode** va cambiato obbligatoriamente ad ogni aggiornamento e deve avere un numero sempre maggiore del precedente. Questo servirà al sistema operativo per capire quando è disponibile una versione più nuova di quella installata. Il **versionName** invece può assumere qualsiasi valore. Ad ogni aggiornamento possiamo anche cambiare gli screenshot, le descrizioni, il prezzo, eccetera.

Conclusioni

La tesi tratta gli aspetti generali di Android. In particolare, il primo capitolo illustra i motivi per i quali Android è una scelta migliore rispetto ad altri sistemi operativi per dispositivi mobili come iOS o Windows Phone, mostrando anche la sua evoluzione, da Cupcake fino a KitKat, sia nella grafica che nella diffusione nel mondo. La scelta di Android è infatti maggiormente legata alla possibilità di personalizzazione a qualsiasi livello, da quello grafico e applicativo al livello di kernel: tutto ciò è reso possibile dal fatto che Android è un sistema operativo open source.

Sono stati poi illustrati gli elementi fondamentali che costituiscono l'architettura di Android, come l'Android Runtime, ed è stata dedicata una buona parte del capitolo alla Dalvik Virtual Machine, elemento caratterizzante dei dispositivi che montano il sistema operativo di Google. Potrebbe essere interessante, in una tesi successiva, approfondire di più ogni livello dell'architettura e proporre, per ognuno di essi, una personalizzazione, così da creare la propria custom ROM.

La parte più importante della tesi è la parte successiva, riguardante le app. L'obiettivo è quello di spiegare in modo approfondito cosa sono le app, l'utilizzo delle Android Graphic Guidelines, gli elementi che le costituiscono (Activity, Intent), i problemi di compatibilità tra versioni differenti di Android che sorgono durante lo sviluppo di un'applicazione e gli strumenti più adatti a risolverli, come SupportLibrary e ActionBar Sherlock. Viene poi fatto anche un approfondimento su come strutturare logicamente e graficamente un'applicazione: infatti ogni sviluppatore, fin prima dell'uscita delle AGG, usava uno stile tutto suo, spesso non intuitivo e sempre diverso, creando confusione negli utenti. Grazie a questo capitolo si potranno quindi strutturare in modo semplice ed efficace le proprie applicazioni.

Gli ultimi due capitoli riguardano la pubblicazione e la pubblicizzazione delle app: nello specifico sono state espresse le tecniche remunerative e i modi per pubblicare e dare visibilità alla propria applicazione.

Sono presenti inoltre due brevi sezioni riguardanti l'installazione degli strumenti necessari allo sviluppo delle applicazioni e alla loro pubblicazione.

L'obiettivo della tesi, cioè trattare gli aspetti generali di Android e fornire linee guida e strumenti per lo sviluppo, è stato raggiunto soprattutto prendendo spunto dall'applicazione per la gestione di viaggi sviluppata insieme ad un collega di studi. Nella tesi sono presenti alcuni riferimenti all'applicazione, la quale è oggetto di

studio più approfondito nella tesi del collega Angelo Trevisiol: “Sviluppo di app per il sistema operativo Android. Caso di studio: gestione di viaggi”.

Bibliografia

- [1] AndroidGeek. *Activity*. URL: <http://www.androidgeek.it/tutorials/programmazione-android/capire-le-activity-ed-il-ciclo-di-vita-di-unapplicazione-android/>.
- [2] Androidiani. *Distribuzione Android*. 2013. URL: <http://www.androidiani.com/dispositivi-android/distribuzione-android-di-luglio-ecco-i-dati-ufficiali-182693>.
- [3] AndroidWorld. *Distribuzione sistemi operativi*. URL: <http://www.androidworld.it/2013/04/29/android-in-italia-al-625-e-nel-mondo-al-642-secondo-le-ultime-stime-153115/>.
- [4] Marco Boemo. «Customizzazione di Android». Capitolo 2. Tesi di dott. Università degli studi di Bologna, 2011-2012.
- [5] Fabio Collini. *Versioni differenti di Android, per quali di queste conviene rendere compatibili le proprie applicazioni?* 2011. URL: <http://android.devapp.it/versioni-differenti-di-android-per-quali-di-queste-conviene-rendere-compatibili-le-proprie-applicazioni>.
- [6] Andrea Como. *L'Architettura di Android*. 2011. URL: <http://android.devapp.it/larchitettura-di-android>.
- [7] Antonio Coschignano. *Struttura di un'applicazione Android*. 2010. URL: http://www.simplesoft.it/android/struttura_di_un_applicazione_android.html.
- [8] Elena Giordano. *Ap-passionati di app*. 2011. URL: <http://www.stpauls.it/gio/1215gi/scienzaetecnologia.html>.
- [9] Google. *Action Bar*. URL: <http://developer.android.com/design/patterns/actionbar.html>.
- [10] Google. *Android Developers*. URL: <http://developer.android.com/design/index.html>.
- [11] Google. *Badge*. URL: <http://developer.android.com/distribute/googleplay/promote/badges.html>.
- [12] Google. *Barre di sistema*. URL: <http://developer.android.com/design/get-started/ui-overview.html>.

- [13] Google. *Ciclo di vita di una Activity*. URL: <http://developer.android.com/guide/components/activities.html>.
- [14] Google. *Devices and displays*. URL: <http://developer.android.com/design/style/devices-displays.html>.
- [15] Google. *Distribution*. URL: <http://developer.android.com/distribute/googleplay/about/distribution.html>.
- [16] Google. *Google Play Visibility*. URL: <http://developer.android.com/distribute/googleplay/about/visibility.html>.
- [17] Google. *Iconography*. URL: <http://developer.android.com/design/style/iconography.html>.
- [18] Google. *Navigatio Drawer*. URL: <http://developer.android.com/design/patterns/navigation-drawer.html>.
- [19] Google. *Notifications*. URL: <http://developer.android.com/design/patterns/notifications.html>.
- [20] Google. *Support Library*. 2013. URL: <http://developer.android.com/tools/support-library/index.html>.
- [21] Google. *Themes*. URL: <http://developer.android.com/design/style/themes.html>.
- [22] Google. *Widget*. URL: <http://developer.android.com/design/patterns/widgets.html>.
- [23] Google. *Writing Style*. URL: <http://developer.android.com/design/style/writing.html>.
- [24] Mauro Lecce. *Activity*. URL: <http://www.html.it/pag/19499/le-attivita-activity/>.
- [25] Mauro Lecce. *Guida Android*. 2012. URL: <http://www.html.it/guide/guida-android/>.
- [26] Mauro Lecce. *Intent*. URL: <http://www.html.it/pag/19500/le-azioni-intent/>.
- [27] Mauro Lecce. *View*. URL: <http://www.html.it/pag/19498/la-visualizzazione-view/>.
- [28] Stefano Limberti. *Da CupCake a Key Lime Pie, la storia di Android dagli albori ad oggi riassunta in un'infografica*. 2013. URL: <http://android.hdblog.it/2013/04/11/da-cupcake-a-key-lime-pie-la-storia-di-android-dagli-albori-ad-oggi-riassunta-in-uninfografica/>.
- [29] MegaOverclock. *L'architettura di Android*. 2011. URL: <http://www.megaoverclock.it/blogs/blog1.php/seconda-lezione-l-architettura-di>.
- [30] Mobimentum. *Come si fa a fare soldi con delle app?* 2013. URL: <http://mobimentum.it/2013/04/04/come-si-fa-fare-soldi-app>.
- [31] Adrea Podda. *Google sorpassa Apple per numero di App nello store*. 2013. URL: <http://www.androidplaya.com/play-store/google-sorpassa-apple-per-numero-di-app-nello-store-24791>.

- [32] Redazione Io Programmo. *Android Programming*. 2013.
- [33] WalkOnJob. *Come creare una app di successo*. 2013. URL: <http://crisiesviluppo.manageritalia.it/2013/05/come-creare-una-app-di-successo/>.
- [34] Waracle. *Diffusione dei sistemi operativi*. URL: <http://waracle.net/android-developers-uk-global-smartphone-sales/>.
- [35] Jake Warton. *ActionBarSherlock*. 2012. URL: actionbarscherlock.com.
- [36] Wikipedia. *API*. URL: http://it.wikipedia.org/wiki/Application_programming_interface.
- [37] Wikipedia. *Piattaforma*. URL: http://it.wikipedia.org/wiki/Piattaforma_%28informatica%29.
- [38] Wikipedia. *Service*. URL: <http://it.wikipedia.org/wiki/Android>.