ALMA MATER STUDORIUM · UNIVERSITÀ DI BOLOGNA

FACOLTÀ DI SCIENZE MATEMATICHE, FISICHE E NATURALI
Corso di Laurea in Informatica

CUSTOMIZZAZIONE DI ANDROID

Tesi di Laurea in Architettura degli Elaborati

Relatore:
Prof. VITTORIO GHINI

Presentata da:
MARCO BOEMO

I Sessione
Anno Accademico 2011-2012

Indice

Introduzione **1**

1 Android **3**
1.1 Cos’è Android? . . . 3
1.2 Breve storia . . . 5
1.3 Il Sistema Operativo . . . 9
1.3.1 Application Layer . . . 10
1.3.2 Application Framework Layer . . . 10
1.3.3 Libraries Layer . . . 12
1.3.4 Android Runtime Layer . . . 14
1.3.5 Linux Kernel Layer . . . 15
1.4 Le applicazioni . . . 16

2 Kernel **19**
2.1 Cos’è un kernel? . . . 19
2.2 Kernel Linux . . . 20
2.2.1 Kernel monolitici . . . 21
2.3 Ricompilazione di un kernel . . . 22
2.4 Kernel Android . . . 23
2.4.1 Custom Kernel . . . 23

3 Ambiente di sviluppo **27**
3.1 Android Software Development Kit (SDK) . . . 28
3.1.1 Android Virtual Device . . . 29
3.1.2 Il tool android . . . 30

3.1.3 L'emulatore di Android 31
3.1.4 Android Debug Bridge 33
3.2 Eclipse e Android Development Tools 34
3.3 Android Native Development Kit 35
3.4 Samsung Galaxy S 35
3.4.1 Samsung Kies e Odin 36

4 Customizzazione di Android passo per passo 39
4.1 Compilazione 39
4.1.1 Compilazione del kernel per l'emulatore 39
4.1.2 Compilazione del kernel per il Galaxy S 41
4.2 Installazione 42
4.2.1 Installazione del kernel sull'emulatore 42
4.2.2 Installazione del kernel sul Galaxy S 43
4.3 Problemi riscontrati 44
4.4 Test del kernel sull'emulatore 46
4.4.1 Una semplice applicazione Android 51

Conclusioni 53

A Ricompilazione del kernel Linux 55
A.1 Preparativi 55
A.2 Compilazione e installazione 56
A.3 Problemi e soluzioni 60

B Android Virtual Device 63
B.1 Creazione di un AVD con AVD Manager 63
B.2 Creazione di un AVD dalla riga di comando 65
B.3 Elenco delle opzioni di emulazione hardware 68

C Il tool android 71

D L'emulatore di Android 73
D.1 Elenco dei comandi emulator 73
D.2 La console dell'emulatore 82

D.2.1 Reindirizzamento delle porte 83
D.2.2 Emulazione della geolocalizzazione 83
D.2.3 Caratteristiche dell'alimentazione del dispositivo 84
D.2.4 Emulazione degli eventi hardware 85
D.2.5 Emulazione della rete 85
D.2.6 Emulazione degli SMS 87
D.2.7 Emulazione della telefonia 87
D.2.8 Stato della Virtual Machine 89
D.2.9 La finestra dell'emulatore 89
D.2.10 Chiusura di un'istanza dell'emulatore 90
D.2.11 Limitazioni dell'emulatore 90

E Android Debug Bridge **91**
E.1 Utilizzo dei comandi adb 91
E.2 Interrogazione per istanze dell'emulatore/dispositivo 91
E.3 Dirigere comandi ad una specifica istanza dell'emulatore/dispositivo . 92
E.4 Elenco dei comandi adb 93
E.5 Utilizzo dei comandi shell 96
E.6 Comandi shell 96
E.7 Comando logcat 97

F Installazione del plugin ADT su Eclipse **101**

G Guida all'utilizzo di Odin **103**
G.1 Flashing di un custom firmware 103
G.2 Flashing di un custom kernel 107

Bibliografia **111**

Introduzione

L’introduzione dei cosiddetti “smartphone” ha cambiato radicalmente la concezione di telefono cellulare, rivoluzionando completamente il modo in cui le persone interagiscono con questi ultimi.

Inizialmente concepiti come “semplici” mezzi di comunicazione, oggi sono divenuti congegni dalle molteplici funzioni che incorporano molti altri dispositivi tecnologici come fotocamere, navigatori satellitari, lettori multimediali. Questi dispositivi mobili danno la possibilità alle persone di avere accesso a tutte le informazioni di cui hanno bisogno, in qualunque posto ed in qualunque momento. Grazie alla loro enorme versatilità ed ubiquità, e alla loro ampia dotazione di strumenti, come microfoni, fotocamere, touchscreen, sistemi di geolocalizzazione e sensori di ogni tipo, gli smartphone stanno diventando a tutti gli effetti delle estensioni delle nostre percezioni.

Il mercato degli smartphone è caratterizzato da una fortissima domanda che coinvolge in maniera indiscriminata tutte le classi di consumatori. Sviluppare applicazioni per dispositivi mobili è indubbiamente un’importante opportunità in quanto permette di posizionarsi nel lato dell’offerta nel mercato sopracitato.

Android è una tra le tante realtà che si sono venute a creare durante l’evoluzione dei dispositivi mobili. La sua natura aperta priva di barriere lo ha portato rapidamente ad essere tra le piattaforme mobili più apprezzate attualmente.

La parola “open” ruota intorno ad Android in tutti i suoi aspetti, inclusa la sua piattaforma di sviluppo, molto potente e allo stesso tempo estremamente semplice da utilizzare. Gli sviluppatori hanno libero accesso a tutti gli strumenti usati dai creatori stessi di Android, pertanto il potenziale delle applicazioni non è soggetto a limitazioni di alcun tipo, visto anche che non viene fatta discriminazione tra software nativo e di terze parti. “Essere” uno sviluppatore non richiede certificazioni particolari, inoltre le applicazioni non vengono sottoposte a procedure di approvazione.

L’obiettivo di questa tesi è quello di studiare l’architettura di Android e vedere cosa è possibile “customizzare”, ovvero personalizzare a proprio piacimento in base ad opportune esigente, ad ogni livello del suo stack. In particolare ci si soffermerà sul suo livello più basso, ovvero il kernel, e si tenterà di modificarlo e personalizzarlo per poi installarlo su uno smartphone.

Nel primo capitolo verrà presentato il sistema operativo Android: dopo un breve accenno sulla sua storia, verranno elencate le sue varie versioni e verrà descritta in maniera dettagliata la sua architettura.

Il secondo capitolo sarà incentrato sul kernel: dopo una prima panoramica sui vari tipi di kernel, il capitolo si incentrerà sul kernel Linux e sull’importanza dell’installazione di un kernel ricompilato sulla propria macchina. Verranno infine descritti i principali custom kernel di Android installabili sul Samsung Galaxy S.

Il terzo capitolo sarà incentrato sull’ambiente di sviluppo: verranno presentati i vari strumenti forniti dall’Android SDK, il plugin ADT per Eclipse e l’Android NDK. Infine sarà illustrato lo smartphone utilizzato per i test di “personalizzazione” di Android.

Nel quarto capitolo sarà presentata la guida per la customizzazione di Android, partendo dalla compilazione dei suoi file sorgente, per poi passare all’installazione del kernel sull’emulatore e sul Samsung Galaxy S, e ai vari test fatti su di essi. Verranno inoltre discussi i vari problemi riscontrati.

Infine troveremo le appendici A, B, C, D, E, F, G che, rispettivamente, tratteranno nel dettaglio: la ricompilazione del kernel Linux, l’Android Virtual Device, il tool android, l’emulatore di Andorid, l’Android Debug Bridge, l’installazione del plugin ADT su Eclipse, e l’utilizzo di Odin.

Capitolo 1

Android

1.1 Cos'è Android?

Quando si parla di Android ci si riferisce ad un sistema operativo open source per dispositivi mobili, ad una piattaforma di sviluppo open source ed inoltre all'insieme di dispositivi e applicazioni basati su Android, concetto schematizzato in figura 1.1. Ecco come viene descritto da uno dei suoi creatori, Andy Rubin[1]:

> *"Android is the first truly open and comprehensive platform for mobile devices. It includes an operating system, user-interface and applications -- all of the software to run a mobile phone but without the proprietary obstacles that have hindered mobile innovation"*[2]

Infatti tutti gli aspetti del sistema operativo Android, da quelli più superficiali a quelli più critici possono essere modificati liberamente. Questo significa che è consentito ai produttori di dispositivi mobili di adattare il sistema operativo ai propri smartphone[3], per esempio rimpiazzando l'interfaccia grafica standard con una fornita da essi oppure andando semplicemente a sostituire il sistema di gestione delle chiamate con

[1] Andy Rubin è uno dei co-fondatori e CEO della società che ha sviluppato Android. Adesso è vicepresidente della sezione ingegneria a Google. Per la sua dichiarazione su Android, vedere il blog ufficiale di Google [1].

[2] "Android è la prima piattaforma per dispositivi mobili realmente aperta e completa. Include un sistema operativo, un'interfaccia utente e le applicazioni -- tutto il software che fa funzionare un telefono ma senza gli ostacoli proprietari che hanno frenato l'innovazione del mondo mobile."

[3] Con il termine smartphone si definisce un telefono cellulare che abbia anche le funzioni e le potenzialità di un computer palmare in grado di funzionare con un sistema operativo.

un altro ritenuto migliore. Quanto detto vale sia per gli utenti finali che hanno la piena facoltà di personalizzare in ogni aspetto il prodotto acquistato che per gli sviluppatori che hanno la possibilità di rilasciare applicazioni in grado di modificare qualunque caratteristica del sistema.

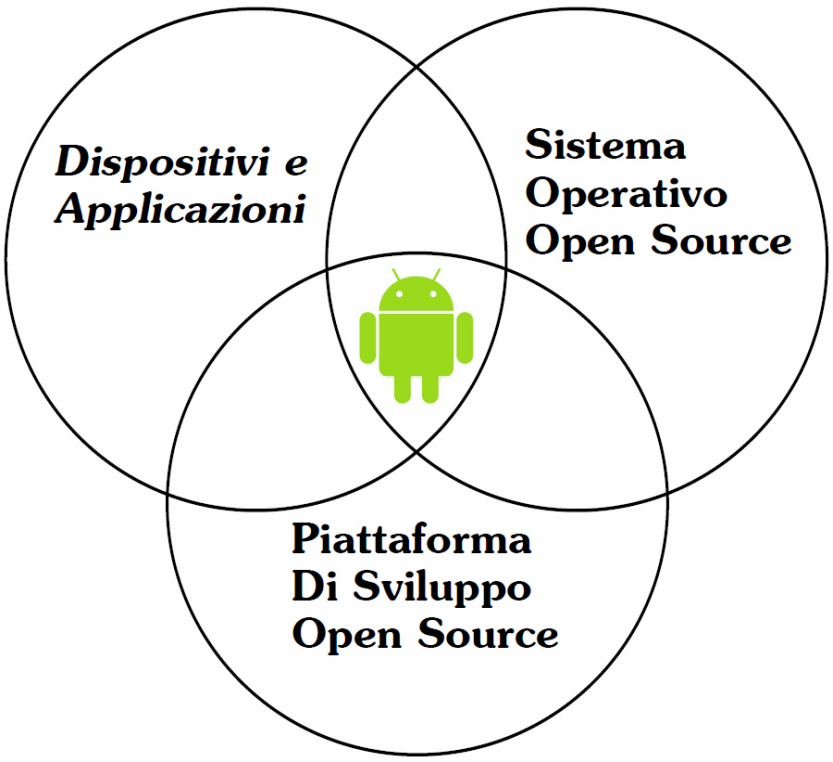

Figura 1.1: I tre domini di Android.

> *"What really makes Android compelling is its open philosophy, which ensures that you can fix any deficiencies in user interface or native application design by writing an extension or replacement. Android provides you, as a developer, with the opportunity to create mobile phone interfaces and applications designed to look, feel, and function exactly as you imagine them."*[4]
>
> Reto Meier[5]

[4] "Ciò che rende Android davvero irresistibile è la sua filosofia aperta, che permette di compensare ogni carenza nell'interfaccia grafica o nel sistema di progettazione di applicazioni native attraverso l'attuazione di estensioni o sostituzioni. Android da agli sviluppatori l'opportunità di creare interfacce e applicazioni per telefoni cellulari progettati per essere proprio come essi le immaginano."

[5] Reto Meier è uno sviluppatore Android, Advocate di Google ed autore del libro Professional Android 4 Development [2].

1.2 Breve storia

Il sistema operativo fu inizialmente sviluppato da Android Inc. che venne acquisita nel 2005 da Google. I fondatori di Android Inc., Andy Rubin, Rich Miner, Nick Sears e Chris White, passarono alle dipendenze di Google e svilupparono una piattaforma basata sul kernel Linux.

Il 5 novembre 2007 venne annunciata la costituzione dell'Open Handset Alliance (OHA)[6], la quale lo stesso giorno presentò Android, basato sulla versione 2.6 del kernel Linux. Pochi giorni dopo, l'OHA rilasciò una versione di anteprima del Software Development Kit (SDK) di Android, molto prima dell'arrivo di una qualsiasi forma di hardware in grado di supportare il nuovo sistema operativo di Google.

Il 23 settembre 2008, contemporaneamente all'aggiornamento della SDK alla versione 1.0, venne annunciato il primo dispositivo Android: il T-Mobile G1, prodotto dalla società taiwanese HTC e commercializzato dal carrier telefonico T-Mobile.

Da quel momento la piattaforma Android ebbe una crescita incredibile, subendo numerosi aggiornamenti in poco tempo.

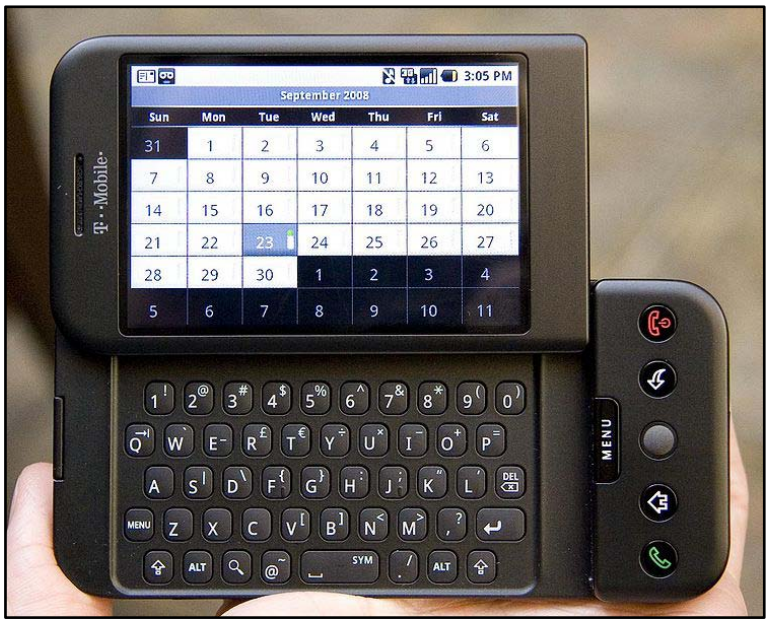

Figura 1.2: Il T-Mobile G1.

[6]La Open Handset Alliance [3] è un insieme di società operanti nell'ecosistema della telefonia mobile costituito da operatori di telefonia mobile, produttori di dispositivi, produttori di software, produttori di semiconduttori e compagnie per la commercializzazione. Dei più dei 70 membri se ne elencano alcuni tra i più rilevanti: Google, Motorola, HTC, Sony, Intel, T-Mobile, Samsung, Nvidia, Qualcomm.

Una delle caratteristiche più evidenti di Android, come vedremo in seguito nel dettaglio, è il fatto che le sue diverse versioni sono indicate a livello ufficiale con un numero di versione secondo gli standard informatici, ma che alla fine queste vengono di preferenza distinte per il proprio "codename", tradizionalmente ispirato alla pasticceria e rigorosamente in ordine alfabetico [4].

Versione 1.0 (Apple Pie), Settembre 2008

- prima versione del sistema operativo

Versione 1.1 (Banana Bread), Febbraio 2009

- risolve numerosi problemi e aggiunge nuove API

Versione 1.5 (Cupcake), Aprile 2009

- introduzione della tastiera virtuale
- possibilità di aggiungere widget e cartelle alla schermata Home
- maggior integrazione con i servizi Google

Versione 1.6 (Donut), Settembre 2009

- aggiornamento della funzione di ricerca vocale
- miglioramenti all'Android Market
- aggiunta di un sintetizzatore vocale e delle gestures
- possibilità di effettuare ricerche direttamente dalla schermata Home

Versione 2.0 e 2.1 (Eclair), Ottobre 2009 e Gennaio 2010

- supporto di risoluzioni multiple
- nuova interfaccia grafica
- supporto al multi-touch
- supporto ad HTML 5
- supporto al Bluetooth 2.1
- aggiunte funzionalità per la fotocamera: zoom digitale, scene mode, effetto sui colori, bilanciamento del bianco, macro focus

Version	Codename	API Level	Distribution
1.5	Cupcake	3	0.3%
1.6	Donut	4	0.6%
2.1	Eclair	7	5.2%
2.2	Froyo	8	19.1%
2.3 - 2.3.2	Gingerbread	9	0.4%
2.3.3 - 2.3.7	Gingerbread	10	64.6%
3.1	Honeycomb	12	0.7%
3.2	Honeycomb	13	2.0%
4.0 - 4.0.2	Ice Cream Sandwich	14	0.4%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	6.7%

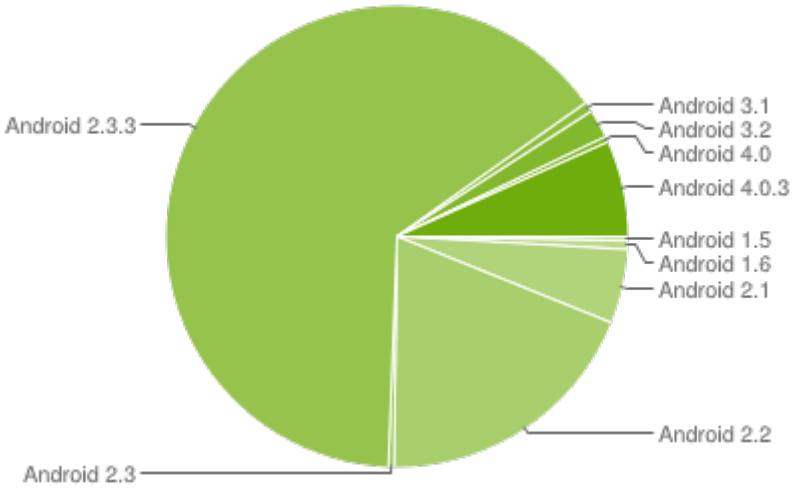

Figura 1.3: Utilizzo delle differenti versioni di Android a Giugno 2012 [5].

Versione 2.2 (Froyo), Maggio 2010

- ottimizzazione delle prestazioni
- implementazione del compilatore Dalvik JIT
- integrazione del motore JavaScript V8 di Google Chrome nel browser di sistema
- supporto ad Adobe Flash 10.1
- possibilità di installare applicazioni sulla memoria SD
- possibilità di inviare contenuti dal browser del PC direttamente al dispositivo
- tethering USB e Wi-Fi hotspot
- miglioramenti all'Android Market

Versione 2.3 (Gingerbread), Dicembre 2010

- nuova interfaccia grafica
- introduzione del garbage collector[7] concorrente
- miglioramenti relativi allo sviluppo di giochi
- supporto a più tipi di sensori, come giroscopio e barometro
- supporto nativo al SIP VoIP[8] e alla tecnologia NFC[9]
- migliore gestione energetica

Versione 3.0 (Gingerbread), Febbraio 2011

- versione ottimizzata per tablet
- nuova interfaccia grafica
- aggiunta la System Bar[10] e la Action Bar[11]
- browser multi-tab
- supporto per le periferiche USB (flash drive, gamepad)
- supporto per processori multi-core
- possibilità di criptare i dati personali

Versione 4.0 (Ice Cream Sandwich), Ottobre 2011

- interfaccia grafica completamente riprogettata
- miglioramenti alla API
- app "Contatti" con integrazione con i social network
- miglioramenti alla fotocamera: zoom durante la ripresa video, ritardo di scatto nullo, modalità panoramica
- dettatura in tempo reale
- supporto all'accelerazione hardware
- possibilità di accedere alle applicazioni dalla schermata di sblocco

[7]Per garbage collection si intende una modalità automatica di gestione della memoria, mediante la quale un sistema operativo, o un compilatore e un modulo di run-time, liberano le porzioni di memoria che non dovranno più essere successivamente utilizzate dalle applicazioni. In altre parole, il garbage collector annoterà le aree di memoria che non sono più referenziate, cioè allocate da un processo attivo, e le libererà automaticamente.

[8]Tecnologia che rende possibile effettuare una conversazione telefonica sfruttando una connessione Internet o una qualsiasi altra rete dedicata a commutazione di pacchetto che utilizzi il protocollo IP senza connessione per il trasporto dati.

[9]Near Field Communication (NFC) è una tecnologia che fornisce connettività wireless bidirezionale a corto raggio, fino ad un massimo di 10 cm.

[10]Barra che permette un accesso veloce alle notifiche, allo stato e ai pulsanti di navigazione.

[11]Barra che permette l'accesso ad opzioni contestuali, di navigazione, dei widget o di altri tipi di contenuti.

- Face Unlock[12]
- Android Beam[13]
- Wi-Fi Direct[14]

Figura 1.4: Alcuni screenshot della versione Ice Cream Sandwich di Android.

1.3 Il Sistema Operativo

Il sistema operativo Android è basato su kernel Linux e consiste in una struttura formata da vari livelli o layer, ognuno di essi fornisce al livello superiore un'astrazione del sistema sottostante. La figura 1.5 raffigura il cosiddetto Software Stack di Android.

[12]Funzione che permette di sbloccare il dispositivo tramite un software di riconoscimento facciale.
[13]Scambio di dati tramite NFC.
[14]Standard che permette ai dispositivi Wi-Fi di connettersi gli uni agli altri senza bisogno di un punto di accesso wireless.

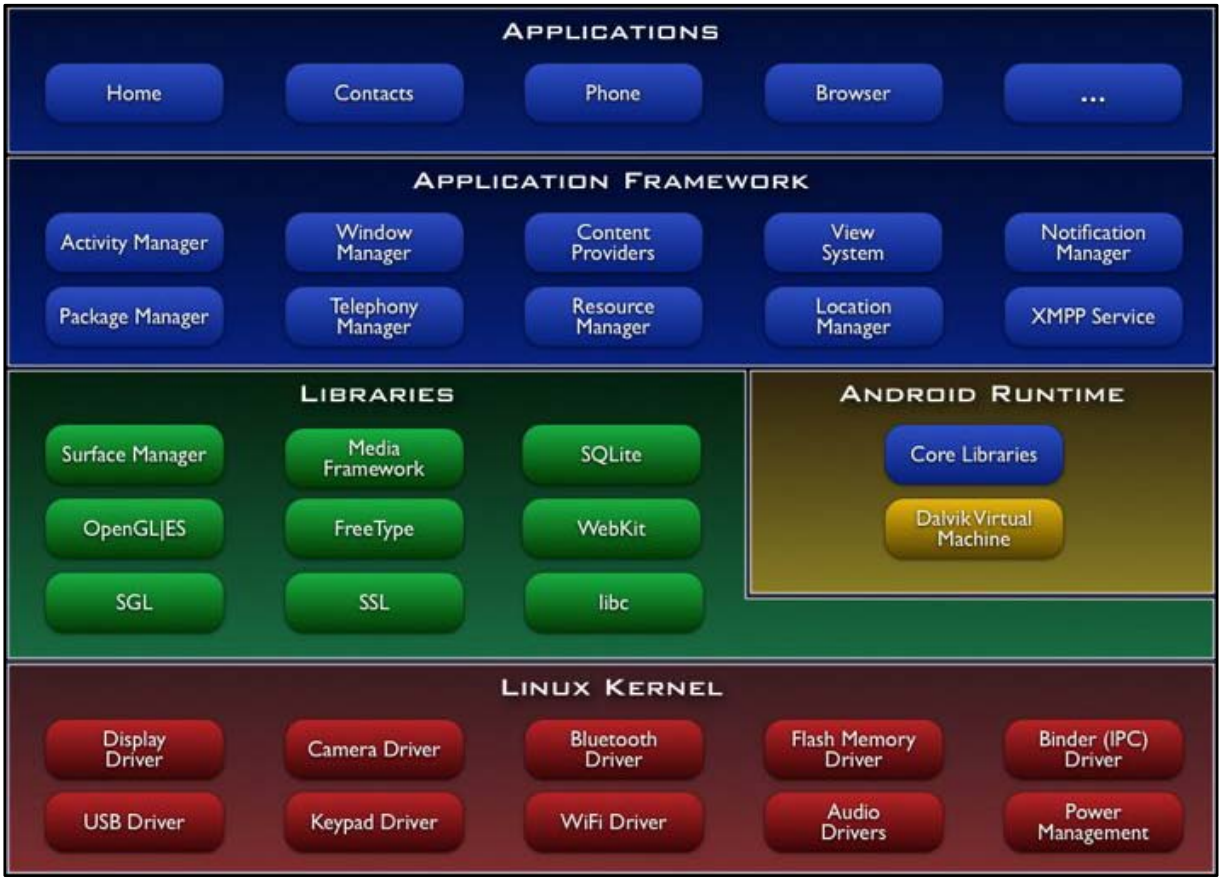

Figura 1.5: Lo stack di Android [6].

1.3.1 Applications Layer

Il livello più alto dello stack è costituito dalle applicazioni: non soltanto quelle native come per esempio il sistema di gestione dei contatti, l'applicazione per l'invio di SMS, il calendario, ecc…, ma anche quelle provenienti da altre fonti. Android non differenzia le applicazioni di terze parti da quelle già incluse nel telefono, infatti garantisce gli stessi privilegi a entrambe le categorie.

1.3.2 Application Framework Layer

L'architettura di Android incoraggia il riutilizzo di componenti rendendo possibile la condivisione di servizi tra più applicazioni. In questo modo l'Application Framework permette agli sviluppatori di concentrarsi nella risoluzione di problemi non ancora affrontati avendo sempre a propria disposizione il lavoro già svolto da altri.

Questo framework è basato su classi Java che però non vengono eseguite in un classico ambiente Java. Infatti è stato possibile evitare gli oneri derivanti dall'adozione della Java Virtual Machine (JVM) realizzando una macchina virtuale ad hoc denominata Dalvik Virtual Machine (DVM); essa verrà analizzata nel paragrafo 1.3.4.

Activity Manager

L'Activity Manager gestisce tutto il ciclo di vita delle activity. Le activity sono entità associate ad una schermata; rappresentano quindi l'interfaccia verso l'utente. Il compito dell'Activity Manager è quello di gestire le varie activity sul display del terminale e di organizzarle in uno stack in base all'ordine di visualizzazione sullo schermo.

Window Manager

Il Window Manager gestisce le finestre relative a differenti applicazioni; astrae alcuni servizi del Surface Manager dello strato Libraries.

Content Providers

I Content Providers gestiscono la condivisione di informazioni tra i vari processi attivi. Il suo utilizzo è simile a quello di un repository comune nel quale i vari processi possono leggere e scrivere informazioni.

View System

La View System gestisce l'insieme delle viste utilizzate nella costruzione dell'interfaccia verso l'utente (bottoni, griglie, text boxes, ecc…).

Package Manager

Il Package Manager gestisce i processi di installazione e rimozione delle applicazioni dal sistema. Ogni applicazione Android è un pacchetto APK che racchiude tutti i dati e le informazioni relative ad essa.

Resource Manager

Il Resource Manager è il modulo deputato alla gestione delle informazioni relative ad una applicazione (file di configurazione, file di definizione dei layout, immagini utilizzate, ecc…).

Telephony Manager

Il Telephony Manager gestisce l'interazione con le funzioni tipiche di un cellulare.

Location Manager

Il Location Manager è il modulo che mette a disposizione dello sviluppatore una serie di API che si occupano della localizzazione. Esistono due provider per la localizzazione: GPS e NETWORK. GPS utilizza i satelliti geostazionari per il posizionamento geografico, ha bisogno però della vista del cielo e risente delle cattive condizioni atmosferiche. NETWORK utilizza punti dei quali si conosce la posizione geografica, come ad esempio celle GSM oppure reti wireless geolocalizzate (Hot Spot).

Notification Manager

Il Notification Manager è il modulo che mette a disposizione una serie di meccanismi utilizzabili dalle applicazioni per notificare eventi al dispositivo e che intraprenderà delle particolari azioni in conseguenza della notifica ricevuta. Le notifiche all'utente possono avvenire in vari modi: attraverso un'icona nella barra di notifica, attraverso il LED del dispositivo, attraverso l'attivazione della retroilluminazione del display, attraverso la riproduzione di suoni, attraverso la generazioni di vibrazioni.

1.3.3 Libraries Layer

Android include una serie di librerie C/C++ che vengono usate da vari componenti del sistema. Attraverso l'Application Framework gli sviluppatori hanno accesso ai servizi forniti da queste librerie. Di seguito viene riportata una lista con alcune di esse.

Surface Manager

È il modulo che gestisce le View, cioè i componenti di un'interfaccia grafica. Funziona praticamente come uno scheduler per la gestione della visualizzazione delle interfacce grafiche e previene eventuali problemi di accavallamento scoordinato sul display. È un componente di vitale importanza per un terminale che fa delle interfacce grafiche un punto di forza del suo funzionamento.

Media Framework

È il componente in grado di gestire i contenuti multimediali (Codec per l'acquisizione e riproduzione di contenuti audio e video).

SQLite

Si tratta del Database Management System (DBMS) utilizzato da Android. SQLite è un DBMS relazionale piccolo ed efficiente messo a disposizione dello sviluppatore per la persistenza dei dati nelle varie applicazioni sviluppate.

OpenGL | SE

È una libreria grafica, versione light della libreria OpenGL per terminali mobili, che permette di utilizzare grafica 3D.

FreeType

Il sistema di gestione dei font è affidato al motore FreeType. È stato scelto perché particolarmente leggero, efficiente, personalizzabile e portabile, ma allo stesso tempo capace di produrre immagini di alta qualità. Tutto questo mantenendo un approccio indipendente dal tipo di file da visualizzare.

SGL (Scalable Graphics Library)

È la libreria scritta in C++ che permette di gestire la grafica 2D.

WebKit

È un moderno web browser engine che sta alla base del browser di Android, il quale può essere utilizzato da qualunque applicazione sotto forma di finestra browser. Da sottolineare che non si tratta di un browser web, quindi dovrà essere integrato nelle diverse applicazioni.

SSL

Si tratta della famosa libreria per la gestione del protocollo di sicurezza crittografico Secure Socket Layer, un servizio fondamentale che garantisce un certo livello di sicurezza per tutte le comunicazioni effettuate tramite il dispositivo.

Libc

Un'implementazione di derivazione Berkeley Software Distribution (BSD) della libreria di sistema standard C, ottimizzata per dispositivi basati su versioni embedded di Linux.

1.3.4 Android Runtime Layer

Ciò che distingue il sistema operativo Android da un'implementazione mobile di Linux è il runtime, che è formato dalle cosiddette Core Libraries e dalla Dalvik Virtual Machine (DVM). Le Core Libraries includono buona parte delle funzionalità fornite dalle librerie standard di Java a cui sono state aggiunte librerie specifiche di Android.

Dalvik Virtual Machine

Come accennato nel paragrafo 1.3.2 le applicazioni Android non vengono eseguite all'interno di una Java Virtual Machine, esse girano sulla Dalvik Virtual Machine che è una particolare macchina virtuale progettata appositamente per dispositivi mobili. Essa garantisce una certa indipendenza del software dalle varie architetture hardware possibili.

Un'essenziale caratteristica della Dalvik Virtual Machine è quella di riuscire a gestire in maniera molto parsimoniosa ed efficiente le poche risorse messe a disposizione dai dispositivi mobili.

Questo risultato è stato raggiunto grazie a numerosi sforzi e accorgimenti, come per esempio la rimozione di numerose librerie Java non necessarie ai dispositivi mobili o l'utilizzo di un particolare tipo di Bytecode, diverso dal Bytecode Java.

L'adozione di un'architettura Register-Based10 per la DVM, in contrapposizione alla natura delle JVM che sono stack machines, costituisce un altro esempio di ottimizzazione.

La DVM è stata concepita per poter essere eseguita in più istanze contemporaneamente sullo stesso dispositivo, ogni applicazione viene associata ad un processo che gira all'interno di una DVM ad esso dedicata. Dato che le varie DVM sono isolate, un eventuale malfunzionamento di un'applicazione non influenza le altre né mette in pericolo la stabilità del sistema operativo stesso.

Un'altra delle molte differenze della Dalvik Virtual Machine rispetto alla classica realtà Java sta nei file eseguibili: la DVM non esegue file `.class` ma file `.dex` (dalvik

executable). I file `.class` vengono convertiti in file `.dex` che sono ottimizzati per avere dimensioni sensibilmente minori.

Tra gli ultimi aggiornamenti che sono stati fatti alla DVM troviamo l'introduzione della compilazione Just In Time (JIT) che offre consistenti incrementi prestazionali.

La DVM si appoggia al kernel Linux per funzionalità quali la gestione di thread e la gestione della memoria a basso livello.

1.3.5 Linux Kernel Layer

Alla base dello stack Android troviamo un kernel Linux nella versione 2.6. La scelta di una simile configurazione è nata dalla necessità di disporre di un vero e proprio sistema operativo che fornisse gli strumenti di basso livello per la virtualizzazione dell'hardware sottostante attraverso l'utilizzo di diversi driver. A differenza di un kernel Linux standard, per Android sono stati aggiunti ulteriori moduli come:

Binder (IPC) Driver

Un driver dedicato che permette a processi di fornire servizio ad altri processi attraverso un insieme di API di più alto livello rispetto a quelle presenti su un kernel Linux standard: ciò permette la comunicazione tra processi con un costo computazionale minore e un relativo minore consumo di batteria.

Low Memory Killer

Un sistema che si preoccupa di terminare i processi liberando così spazio nella memoria centrale per soddisfare le richieste di un altro processo. La terminazione dei processi avviene secondo un sistema di ranking che assegna dei punteggi ai processi; i processi che verranno terminati saranno quelli con punteggio più alto. Ad esempio, un processo che controlla la UI (User Interface) di un'applicazione visibile sarà sicuramente più basso di quello relativo ad un''applicazione non visibile sullo schermo. Il processo init non può essere terminato.

Android Debug Bridge

Uno strumento che permette di gestire in maniera versatile un'istanza dell'emulatore o eventualmente di un dispositivo reale.

RAM Console e Log devices

Per agevolare il debugging, Android fornisce la capacità di memorizzare i messaggi di log generati dal kernel in un buffer RAM. È disponibile inoltre un modulo separato che permette ai processi utente di leggere e scrivere messaggi di log.

Ashmem

Un sistema di memoria condiviso anonimo (Anonimous Shared Memory) che definisce interfacce che consentono ai processi di condividere zone di memoria attraverso un nome. Il vantaggio di Ashmem rispetto ai sistemi Linux che fanno uso della memoria condivisa, è che viene fornito al kernel uno strumento per recuperare dei blocchi di memoria non utilizzati.

Power Management

Una sezione progettata per permettere alla CPU di adattare il proprio funzionamento al fine di non consumare energia se nessuna applicazione o servizio ne fa richiesta.

Il fatto che Android usa un kernel di derivazione Linux non si traduce nel fatto che Android sia a tutti gli effetti una distribuzione GNU/Linux. Il sistema operativo di Google infatti non possiede un X Window System (il gestore grafico standard per i sistemi Unix) e non include tutto l'insieme delle librerie standard GNU.

1.4 Le applicazioni

Uno strumento che permette l'incontro tra la domanda e l'offerta di programmi per Android è il Market, il servizio che da la possibilità agli utenti di reperire applicazioni e agli sviluppatori di renderle disponibili.

Bisogna sottolineare che agli sviluppatori non viene imposto di avvalersi di questa facoltà, essi possono scegliere liberamente il canale attraverso il quale distribuire i propri programmi. Gli utenti di conseguenza hanno a disposizione diverse fonti da cui attingere.

L'Android Market nasce per limitare i problemi derivanti dalla notevole quantità di dispositivi e versioni del sistema operativo. Infatti la natura aperta del sistema compor-

ta una frammentazione dovuta alle svariate possibili personalizzazioni del prodotto.

Gli sviluppatori sono tenuti a indicare il range di dispositivi con cui le loro applicazioni sono compatibili, specificando per esempio la versione minima del firmware richiesta dall'applicazione piuttosto che le risoluzioni per le quali essa è stata ottimizzata. In questo modo il Market rende accessibili agli utenti soltanto le applicazioni adatte ai loro smartphone, rendendo la ricerca di applicazioni una procedura più semplice e sicura, e allo stesso tempo fa sì che gli sviluppatori non vengano investiti da lamentele causate da incompatibilità.

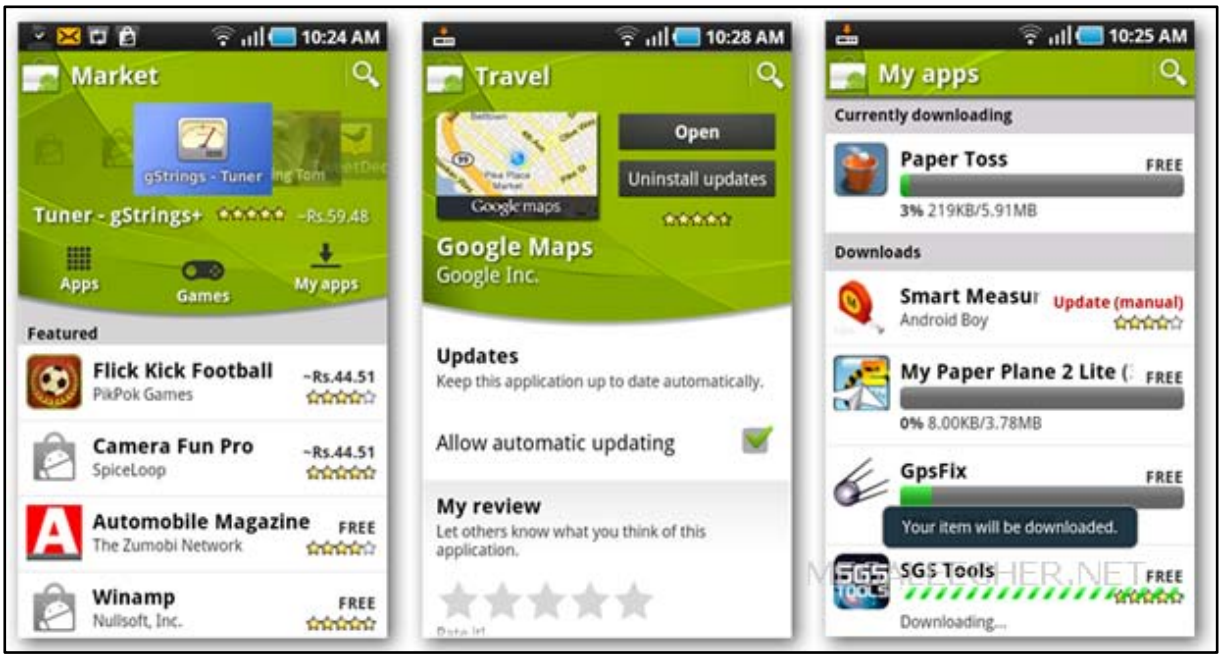

Figura 1.6: L'Android Market.

Il Market opera quindi da punto di riferimento senza però esercitare alcun controllo sulle applicazioni né sulle loro possibili caratteristiche; esso si limita alla gestione delle transazioni economiche tra acquirenti e venditori. Nel momento dell'installazione viene chiesto all'utente di garantire all'applicazione eventuali permessi, come per esempio l'accesso ad internet, allo storage del dispositivo oppure il permesso di effettuare chiamate.

L'introduzione di nuove applicazioni nel Market ed il loro aggiornamento non devono passare attraverso nessun processo di approvazione; queste operazioni possono avvenire quindi in maniera immediata e lo sviluppatore ha il pieno controllo su tutte le operazioni che riguardano il software di cui lui è autore.

Dato che non c'è nessuna entità che misura la qualità dei prodotti presenti nel Market, è stato introdotto un sistema di rating che permette agli utenti di valutare le applicazioni fornendo informazioni preziose per la comunità.

Le ricerche effettuate nel Market vengono ordinate in base alle valutazioni e al numero di download e di conseguenza agli sviluppatori non è lasciata altra scelta che rilasciare software di buona qualità, dato che le risultanti valutazioni negative corredate dagli analoghi commenti possono facilmente compromettere il futuro di un'applicazione.

Il Market inoltre offre un buon sistema di tutela del consumatore: quest'ultimo infatti può chiedere il rimborso della somma pagata entro un certo limite di tempo dal momento dell'acquisto dell'applicazione.

Un recente aggiornamento dell'Android Market consiste nella possibilità di esplorare il catalogo delle applicazioni attraverso una pagina web [7] raggiungibile anche dal proprio PC. Questo facilita notevolmente il processo decisionale dell'utente dato che la pagina web in questione offre una notevole quantità di informazioni utili alla scelta delle applicazioni, tra cui le valutazioni ed i commenti, link esterni, video, immagini. Inoltre è possibile effettuare l'eventuale pagamento e l'installazione delle applicazioni direttamente dal proprio PC, attraverso un sistema che collega il proprio smartphone all'account Google da cui si sta accedendo all'Android Market.

Capitolo 2

Kernel

2.1 Cos’è un kernel?

Il kernel [8] costituisce il nucleo di un sistema operativo e fornisce tutte le funzioni essenziali per il sistema, in particolare la gestione della memoria, delle risorse del sistema e delle periferiche, assegnandole di volta in volta ai processi in esecuzione. La controparte del kernel è la shell, ovvero l’interfaccia utente del sistema, la parte più esterna. I programmi chiedono le risorse al kernel attraverso delle chiamate (system call) e non possono accedere direttamente all’hardware. Il kernel si occupa quindi di gestire il tempo processore, le comunicazioni e la memoria distribuendole ai processi in corso a seconda delle priorità (scheduling) realizzando così il multitasking.

L’accesso diretto all’hardware può essere anche molto complesso, quindi i kernel usualmente implementano uno o più tipi di astrazione dall’hardware, il cosiddetto Hardware Abstraction Layer. Queste astrazioni servono a “nascondere” la complessità e a fornire un’interfaccia pulita ed uniforme all’hardware sottostante, in modo da semplificare il lavoro degli sviluppatori.

I kernel si possono classificare in base al grado di astrazione dell’hardware in quattro categorie:

- *kernel monolitici*, che implementano direttamente una completa astrazione dell’hardware sottostante;

- *microkernel*, che forniscono un insieme ristretto e semplice di astrazione dell'hardware e usano software (chiamati device driver o server) per fornire maggiori funzionalità;
- *kernel ibridi (o microkernel modificati)*, che si differenziano dai microkernel puri per l'implementazione di alcune funzioni aggiuntive al fine di incrementare le prestazioni;
- *esokernel*, che rimuovono tutte le limitazioni legate all'astrazione dell'hardware e si limitano a garantire l'accesso concorrente allo stesso, permettendo alle singole applicazioni di implementare autonomamente le tradizionali astrazioni del sistema operativo per mezzo di speciali librerie.

2.2 Kernel Linux

Il kernel Linux [9] è un software libero distribuito con licenza GNU General Public License; è stato creato nel 1991 da Linus Torvalds. Integrato con il Sistema GNU, sviluppato da Richard Stallman, ha dato vita al sistema operativo GNU/Linux (chiamato comunemente con il solo nome Linux).

Il kernel Linux ha una struttura monolitica; è considerata da alcuni obsoleta a differenza della più moderna architettura a microkernel. Sebbene oggi il kernel possa essere compilato in modo da avere un'immagine binaria ridotta al minimo e i driver caricabili da moduli esterni, l'architettura originaria è chiaramente visibile: tutti i driver infatti devono avere una parte eseguita in kernel mode, anche quelli per cui ciò non sarebbe affatto necessario (ad esempio i driver dei file system).

Linux è un kernel che supporta il multitasking ed è multi utente. Ciò permette che diversi utenti (con privilegi differenziati) possano eseguire sullo stesso sistema diversi processi software in simultanea. Attualmente Linux supporta gran parte dell'hardware disponibile per PC e supporta un numero enorme di architetture (tra cui SPARC, PowerPC e le più moderne CPU a 64 bit).

Dato che il codice sorgente di Linux è disponibile a tutti, è ampiamente personalizzabile, al punto da rendere possibile, in fase di compilazione, l'esclusione di codice non strettamente indispensabile. La flessibilità di questo kernel lo rende adatto a tutte quelle tecnologie embedded emergenti e anche nei centri di calcolo distribuito (cluster

Beowulf) fino ad essere incorporato in alcuni videoregistratori digitali e nei telefoni cellulari.

2.2.1 Kernel monolotici

Come è stato detto nel paragrafo precedente, il kernel Linux è un kernel monolitico. L'approccio monolitico definisce un'interfaccia virtuale di alto livello sull'hardware, con un set di primitive o chiamate di sistema per implementare servizi di sistema operativo, come gestione dei processi, multitasking e gestione della memoria, in diversi moduli che girano in modalità supervisore.

Anche se ogni modulo che serve queste operazioni è separato dal resto, l'integrazione del codice è molto stretta e difficile da fare in maniera corretta e, siccome tutti i moduli operano nello stesso spazio, un bug in uno di essi può bloccare l'intero sistema. Tuttavia, quando l'implementazione è completa e sicura, la stretta integrazione interna dei componenti rende un buon kernel monolitico estremamente efficiente.

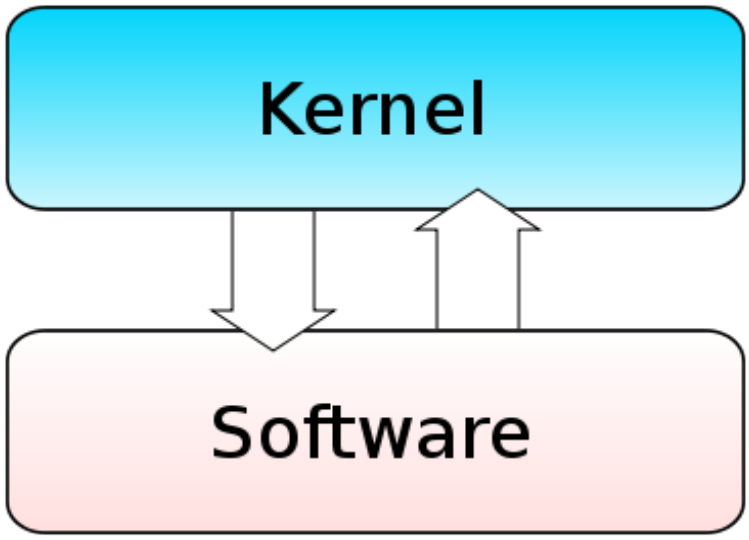

Figura 2.1: Rappresentazione grafica di un kernel monolitico.

Il più grosso svantaggio dei kernel monolitici è, tuttavia, che non è possibile aggiungere un nuovo dispositivo hardware senza aggiungere il relativo modulo al kernel, operazione che richiede la ricompilazione del kernel. In alternativa è possibile compilare un kernel con tutti i moduli di supporto all'hardware, al costo di aumentarne molto

la dimensione. Tuttavia i kernel monolitici più moderni come il kernel Linux e FreeBSD possono caricare moduli in fase di esecuzione, se sono previsti in fase di compilazione, permettendo così l'estensione del kernel quando richiesto, mantenendo al contempo le dimensioni del codice nello spazio del kernel al minimo indispensabile.

2.3 Ricompilazione di un kernel

Spesso risulta vantaggioso installare sulla propria macchina un kernel "personalizzato", un kernel cioè ricompilato, anziché uno di quelli precompilati forniti dalla distribuzione GNU/Linux installata. Ciò può portare alcuni vantaggi, come:

- avere una configurazione diversa e più adatta alla propria macchina;
- gestire hardware particolare o gestire conflitti hardware con kernel preconfezionati;
- usare opzioni del kernel che non sono supportate dai kernel preconfezionati (per esempio il supporto per la memoria alta);
- ottimizzare il kernel rimuovendo i driver inutili in modo da velocizzare l'avvio del sistema;
- avere un sistema operativo leggermente più reattivo grazie all'ottimizzazione basata sul tipo di processore;
- usare un kernel aggiornato o di sviluppo;
- creare un kernel monolitico al posto di uno modulare[15].

Non è necessario compilare un kernel nei casi in cui l'hardware non funzioni alla perfezione o le periferiche non vengano completamente riconosciute. A volte per far riconoscere correttamente al sistema una data periferica basta caricare i moduli necessari con le dovute opzioni. È utile ricompilare il kernel solo se tali moduli non sono presenti o se si è certi che i driver della periferica sono presenti solo in una versione diversa da quella attualmente in uso. Inoltre, l'aumento di prestazioni tende a essere irrilevante, soprattutto su computer già veloci.

[15]Per kernel modulare si intende un'estensione del kernel monolitico, con la capacità di caricare/scaricare parti di codice (moduli) secondo necessità e richieste. È quello utilizzato da tutte le distribuzioni in fase di installazione di Linux su una macchina.

È bene tenere presente che compilare un nuovo kernel significa, nella sostanza, cambiare sistema operativo, in quanto esso ne costituisce il motore; inoltre è richiesta un buona conoscenza del proprio hardware.

Nell'appendice A vengono descritti in maniera dettagliata i passi da svolgere per compilare ed installare un kernel Linux sulla propria macchina.

2.4 Kernel Android

Come descritto nel paragrafo 1.3.5, il kernel Android è sostanzialmente un kernel Linux nella versione 2.6. La scelta di una simile configurazione è nata dalla necessità di disporre di un vero e proprio sistema operativo che fornisse gli strumenti di basso livello per la virtualizzazione dell'hardware sottostante attraverso l'utilizzo di diversi driver.

L'installazione di un kernel "personalizzato" può portare a numerosi vantaggi in termini di prestazioni e funzionalità. Nel prossimo paragrafo verranno descritti i principali custom kernel[16] compatibili con il Samsung Galaxy S.

2.4.1 Custom Kernel

CF-Root

Questo kernel, dotato dei permessi di root, è per i principianti e per quelli che vogliono mantenere il kernel il più possibile simile all'originale.

CF-Root [10] prende il kernel da un firmware originale Samsung (nella versione Gingerbread) e aggiunge:

- root: permette di ottenere dei permessi di alto livello sull'intero sistema operativo, in modo che l'utente ne diventi completamente l'amministratore (superuser). Il vantaggio principale dei privilegi di root è quello di poter accedere in scrittura (quindi creazione, modifica e cancellazione) al contenuto di tutte le cartelle del sistema operativo, altrimenti inaccessibili nelle configurazioni di base. Con questa azione sarà possibile eliminare le applicazioni stock, modificare o so-

[16]Per custom kernel si intende un kernel modificato per adattarlo alla esigenze del singolo utente.

stituire i file di sistema, lavorare il file di framework per cambiare la grafica del dispositivo, utilizzare programmi con funzioni particolari, ed altro ancora;
- BusyBox: permette di avere sul proprio Android comandi Linux di base;
- ClockWorkMod Recovery: si tratta di una recovery che permette di installare firmware, applicazioni e altro attraverso la scheda SD, di effettuare il backup completo e il ripristino della memoria interna, di effettuare il wipe (pulizia) della memoria interna e della partizione di cache, di montare/smontare le partizioni di sistema, oppure formattarle.

Semaphore

Semaphore è forse il kernel che ha avuto più successo tra gli appassionati di modding[17]. Il suo vero punto di forza sono la completezza e l'ottimizzazione volta a cercare le massime prestazioni.

Questo kernel è disponibile in due versioni: una versione [11] ricompilata partendo dall'ultima versione dei sorgenti di Android Gingerbread e basata sul kernel CF-Root, e una versione [12] basata su Android Ice Cream Sandwich. Dal sito ufficiale di Semaphore è possibile scaricare e vedere le caratteristiche principali delle due versioni del kernel.

Midnight

Il kernel Midnight, più che alle prestazioni, si preoccupa di ottimizzare al massimo i consumi della batteria. Anche questo kernel è disponibile in due versioni: una versione [13] basata sui sorgenti di Gingerbread e una versione [14] basata sui sorgenti di Ice Cream Sandwich.

SpeedMod

Si tratta di un kernel basato sui sorgenti di Gingerbread, che gode di un'ottima stabilità e velocità, e che fa della gestione dei consumi il suo punto di forza. Le caratteristiche principali sono [15]:
- gestione dei colori: neutrale, più freddi, più caldi;
- lag fix filesystem ext4;

[17] Modding è un'espressione gergale che deriva dal verbo inglese "modify". Modding si riferisce all'atto di modificare hardware, software, o potenzialmente qualsiasi altra cosa, per svolgere una funzione non originariamente concepita o destinata dal progettista.

- root e ClockWorkMod Recovery;
- 364MB di RAM disponibile all'avvio;
- tweak per la CPU;
- kernel impostato a 300Hz;
- supporto Voodoo Sound;
- TinyRCU:
- logcat debugging disabilitato;
- miglioramenti su volume audio e microfono.

Galaxian

Questo kernel è basato sui sorgenti di Andorid Gingerbread rilasciati da Samsung e quindi può essere installato su tutti i tipi di firmware senza alcun problema. Come la maggior parte dei kernel modificati supporta sia il filesystem ext4 che RFS (Remote File Sharing). Le caratteristiche principali sono [16]:

- root e ClockWorkMod Recovery;
- gamma dei colori modificata con la luminosità del display al minimo;
- Voodoo Sound e Voodoo Color;
- miglioramenti nella sensibilità del touch;
- vari tweak per facilitare l'overclocking della CPU;
- 344MB di RAM disponibile all'avvio;
- modulo MDNIE prelevato dal Galaxy Tab per colori più vivi e nitidi;
- gestione della luminosità più precisa;
- supporto BNL, gestibile con l'applicazione BLN Control;
- driver OLED riscritti;
- barra di progressione animata durante il boot.

Altri kernel di rilievo sono: **Voodo** [17], **TalonDev** [18] e **FuguMod** [19] basati sui sorgenti di Gingerbread; **Devil** [20] e **Icy Glitch** [21] basati sui sorgenti di Ice Cream Sandwich.

Capitolo 3

Ambiente di sviluppo

Uno dei grandi punti di forza di Android è la sua piattaforma di sviluppo open source, a cui vanno attribuiti buona parte dei meriti del successo raggiunto fin'ora.

I membri della Open Handset Alliance sono convinti che il miglior modo per offrire software di qualità ai consumatori è rendere semplice per gli sviluppatori scriverlo.

L'idea su cui è basata la piattaforma di sviluppo di Android è quella di non limitare in nessun modo le potenzialità degli sviluppatori offrendo loro gli stessi mezzi usati dai creatori di Android stesso. Infatti le applicazioni di terze parti vengono considerate allo stesso livello di quelle native:

> *"your apps are not second class citizens, they are at the same level as any other app that ships with the phone"…"you got the ability to leverage other people's work to enrich your own app, or to become the source for other people to use so your app can be part of someone else's"*[18]
>
> Reto Meier [2]

Inoltre Android permette di utilizzare parti di applicazioni all'interno di altre non solo attraverso il classico riutilizzo di codice ma soprattutto grazie ad un efficace meccanismo basato sugli Intent Filters. Gli Intent Filters sono un metodo per esporre al re-

[18]Le vostre applicazioni non sono cittadini di seconda classe, esse sono allo stesso livello di qualsiasi altra applicazione che viene fornita con il telefono"…"Avete la possibilità di sfruttare il lavoro di altri per arricchire la vostra applicazione, o di essere una vera e propria fonte, in modo che la vostra applicazione può essere parte di quella di qualcun altro."

sto del sistema le azioni che un'applicazione può compiere in modo da poter essere sfruttate da qualunque applicazione. Essi offrono numerosi benefici tra cui:

- massimizzare il riutilizzo e la modularità dei componenti. Le applicazioni possono specializzarsi su servizi singoli e in caso di necessità interagire tra loro per fornire un servizio migliore;
- adattare al meglio le applicazioni alle esigenze dell'utente. Ad esempio l'applicazione relativa alla fotocamera, che ha una funzione per permettere la condivisione delle immagini, può chiedere all'utente di selezionare l'applicazione da usare per portare a termine tale operazione. Il software che richiede il servizio può tranquillamente non essere a conoscenza delle applicazioni correntemente installate sul dispositivo, sarà il meccanismo degli Intent Filters a individuare i programmi che potranno accogliere la richiesta.

3.1 Android Software Development Kit (SDK)

Il Software Development Kit[19] di Android mette a disposizione numerosi strumenti considerati nel seguito.

Application Programming Interface (API).

Il nucleo della SDK è costituito dalle API[20] che sono librerie messe a servizio degli sviluppatori e che sono esattamente le stesse usate da Google per creare le applicazioni native.

Android Virtual Device Manager

Un Android Virtual Device (AVD) consiste in un vero e proprio emulatore di dispositivi Android, utile per poter sviluppare applicazioni anche senza essere in possesso di uno smartphone oppure per simulare una configurazione hardware neutrale. È possibi-

[19]Software Development Kit è un termine che in italiano si può tradurre come "pacchetto di sviluppo per applicazioni" e sta ad indicare un insieme di strumenti per lo sviluppo e la documentazione di software. [22]

[20]Le Application Programming Interface, sono ogni insieme di procedure disponibili al programmatore, di solito raggruppate a formare un set di strumenti specifici per l'espletamento di un determinato compito all'interno di un certo programma. La finalità è ottenere un'astrazione, di solito tra l'hardware e il programmatore o tra software a basso e quello ad alto livello semplificando così il lavoro di programmazione. Le API permettono infatti di evitare ai programmatori di riscrivere ogni volta tutte le funzioni necessarie al programma dal nulla, ovvero dal basso livello, rientrando quindi nel più vasto concetto di riuso di codice. [23]

le creare dispositivi virtuali scegliendo la grandezza e densità del display, la versione del sistema operativo installata e altri parametri.

Dalvik Debug Monitor Server (DDMS)

È uno strumento essenziale per il debugging che include una serie di funzionalità per monitorare l'esecuzione delle applicazioni su dispositivi reali o emulatori.

Documentazione

La documentazione inclusa nella SDK è molto dettagliata e offre informazioni precise su ogni singolo package e classe.

Tutorial

Sono guide per la realizzazione di applicazioni basilari, utili per gli sviluppatori alle prime armi.

Codice di esempio/dimostrativo

I tutorial sono accompagnati da applicazioni già pronte che possono venire usate come modello per costruirne altre.

3.1.1 Gestione dei dispositivi virtuali

Un Android Virtual Device (AVD) [24] è una configurazione per un emulatore che permette di plasmare un vero e proprio dispositivo attraverso la definizione di opzioni hardware e software per essere emulate dall'emulatore Android. Un AVD è composto da:

- un profilo hardware: definisce le caratteristiche hardware del dispositivo virtuale. Ad esempio, è possibile definire la quantità di memoria che ha, se il dispositivo ha una fotocamera, se utilizza una tastiera fisica QWERTY o un pad alfanumerico, e così via;
- una mappatura di un'immagine del sistema: è possibile definire quale versione della piattaforma Android verrà eseguita sul dispositivo virtuale. È possibile scegliere una versione della piattaforma standard di Android o l'immagine del sistema fornita con un add-on dell'SDK;

- altre opzioni: è possibile specificare la skin (aspetto grafico) dell'emulatore che si desidera utilizzare con l'AVD, consentendo di controllare le dimensioni dello schermo, l'aspetto, e così via. È inoltre possibile specificare la scheda SD emulata da utilizzare con l'AVD;
- un'area di memorizzazione ad hoc sulla macchina di sviluppo: i dati utenti del dispositivo (applicazioni installate, impostazioni, e così via) e la scheda SD emulata vengono memorizzate in questa area.

È possibile creare tanti AVD quanti se ne ha bisogno, in base ai tipi di dispositivi che si desidera plasmare. Per testare a fondo un'applicazione, occorre creare un AVD per ogni configurazione del dispositivo (ad esempio, differenti dimensioni dello schermo e diverse versioni della piattaforma) con la quale l'applicazione è compatibile, e testare l'applicazione su ciascuno di essi.

Il modo più semplice per creare un AVD è quello di utilizzare l'interfaccia grafica del AVD Manager. È possibile avviare questo strumento sia da Eclipse cliccando su **Window -> AVD Manager**, sia dalla riga di comando richiamando il tool `android` con l'opzione `avd`. In alternativa è possibile creare un AVD dalla riga di comando attraverso l'uso del tool `android`.

Nell'appendice B viene descritto in maniera dettagliata come creare un AVD sia utilizzando l'AVD Manager [25] sia utilizzando la riga di comando e il tool `android` [26].

3.1.2 Il tool android

Il tool `android` [27] è un importante strumento di sviluppo che consente di:

- creare, eliminare e visualizzare Android Virtual Device;
- creare e aggiornare progetti Android [28];
- aggiornare l'Android SDK con nuove piattaforme, add-on e documentazione.

È possibile trovare il tool `android` in `<sdk-directory>/tools/`.

Nell'appendice C vengono descritti in maniera dettagliata i vari comandi `android` e il loro significato e utilizzo [27].

3.1.3 L’emulatore di Android

L’Android SDK include un emulatore di un dispositivo mobile, un dispositivo mobile virtuale che viene eseguito sulla propria macchina. L’emulatore consente di sviluppare e testare applicazioni Android senza l’utilizzo di un dispositivo fisico.

L’emulatore di Android [29] simula tutte le funzioni hardware e software di un dispositivo mobile tipico, con la differenza che non può effettuare chiamate telefoniche reali. Esso fornisce una serie di pulsanti di navigazione e di controllo, che si possono “premere” con il mouse o la tastiera per generare eventi per le applicazioni. Esso prevede, inoltre, uno schermo in cui vengono visualizzate le applicazioni Android attive.

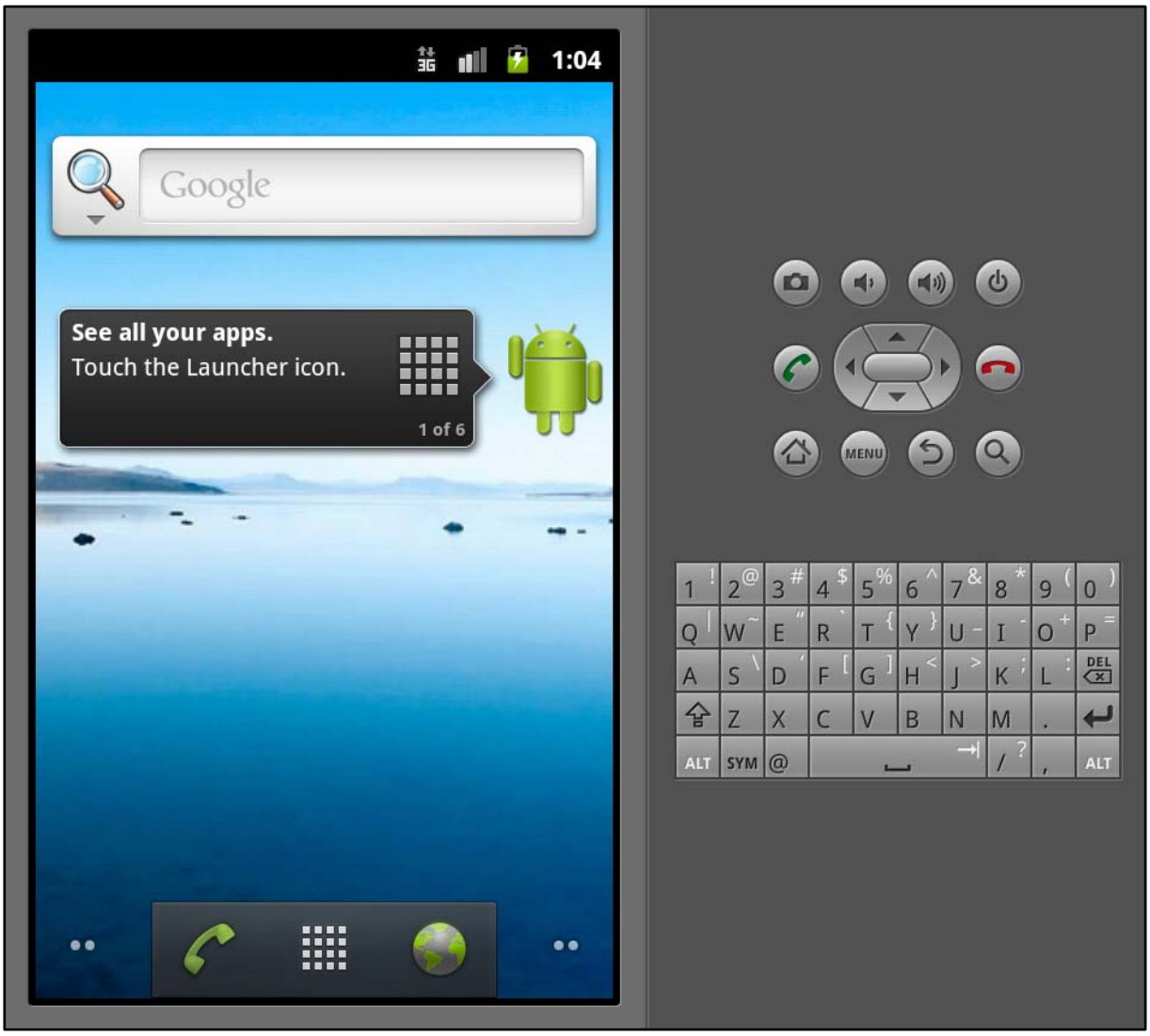

Figura 3.1: L’emulatore di Andorid.

Per testare le applicazioni più facilmente, l'emulatore utilizza configurazioni Android Virtual Device (AVD). Gli AVD consentono di definire alcuni aspetti hardware del dispositivo emulato e permettono di creare diverse configurazioni per testare molte piattaforme Android e permutazioni hardware. Una volta che un'applicazione è in esecuzione sull'emulatore, è possibile utilizzare i servizi della piattaforma Android per richiamare altre applicazioni, accedere alla rete, riprodurre file audio e video, archiviare e recuperare dati, mandare notifiche all'utente, ed eseguire il rendering delle transizioni grafiche e dei temi.

L'emulatore include anche una varietà di funzionalità di debug, come una console da cui è possibile registrare l'output del kernel, simulare gli interrupt di un'applicazione (come la ricezione di SMS o di telefonate), e simulare gli effetti di latenza e caduta di segnale sulla rete dati.

Le immagini del sistema Android disponibili tramite l'Android SDK Manager contengono il codice per il kernel Linux di Android, le librerie native, la Dalvik VM, e i vari pacchetti Android (come il framework di Android e le applicazioni preinstallate). L'emulatore fornisce la traduzione binaria dinamica del codice macchina del dispositivo al sistema operativo e all'architettura del processore della propria macchina.

L'emulatore di Android supporta molte caratteristiche hardware che possono essere presenti sui dispositivi mobili, tra cui:

- una CPU ARMv5 e la corrispondente unità di gestione della memoria (MMU);
- un display LCD a 16-bit;
- una o più tastiere (una tastiera Qwerty e i pulsanti del telefono associati);
- un chip audio con funzioni di ingresso e di uscita;
- un modem GSM, tra cui una scheda SIM simulata;
- partizioni di memoria flash (emulate attraverso i file di immagine del disco sulla propria macchina);
- una fotocamera, utilizzando una webcam collegata alla propria macchina;
- sensori come l'accelerometro, utilizzando i dati da un dispositivo Android collegato tramite USB.

Sia quando si avvia l'emulatore, che in fase di esecuzione, è possibile utilizzare una varietà di comandi e opzioni per controllare il suo comportamento.

Nell'appendice D vengono descritti in maniera dettagliata i vari comandi dell'emulatore [30] e della sua console [29], e il loro significato e utilizzo.

3.1.4 Android Debug Bridge

Android Debug Bridge (adb) [31] è un versatile strumento a riga di comando che permette di comunicare con un'istanza dell'emulatore o con un dispositivo Android collegato. Si tratta di un programma client-server che comprende tre componenti:

- un client, che viene eseguito sulla macchina. È possibile richiamare un client da una shell mediante l'emissione di un comando adb. Altri strumenti di Android come il plugin ADT e DDMS possono creare client adb;
- un server, che viene eseguito come un processo in background sulla macchina. Il server gestisce la comunicazione tra il client e il demone adb in esecuzione su un emulatore o un dispositivo;
- un demone, che viene eseguito come un processo in background su ogni istanza dell'emulatore o del dispositivo.

Quando si avvia un client adb, il client controlla in primo luogo se vi è un processo server adb già in esecuzione. Se non c'è, avvia il processo server. Quando il server si avvia, si lega alla porta TCP locale 5037 e attende i comandi inviati dai client adb; tutti i client adb utilizzano la porta 5037 per comunicare con il server adb.

Il server imposta quindi le connessioni a tutte le istanze dell'emulatore/dispositivo in esecuzione. Localizza le istanze del'emulatore/dispositivo scandendo le porte dispari in un range da 5555 a 5587 (che è l'intervallo usato dagli emulatori/dispositivi). Se il server trova un demone adb, imposta una connessione a quella porta. Si noti che ogni istanza dell'emulatore/dispositivo acquisisce una paio di porte in sequenza: una porta con numero pari per le connessioni di console e una porta con numero dispari per le connessioni adb. Per esempio:

Emulatore 1, console: 5554
Emulatore 1, adb: 5555
Emulatore 2, console: 5556
Emulatore 2, adb: 5557

Come mostrato, l'istanza dell'emulatore collegata ad adb sulla porta 5555, è la stessa istanza la cui console comunica con la porta 5554.

Una volta che il server ha impostato le connessioni a tutte le istanze dell'emulatore, è possibile utilizzare i comandi adb per controllare e accedere a tali istanze. Poiché il server gestisce le connessioni a istanze dell'emulatore/dispositivo e gestisce i comandi

da più client adb, è possibile controllare qualsiasi istanza dell'emulatore/dispositivo da qualsiasi client (o da uno script).

È possibile trovare lo strumento `adb` in `<sdk-directory>/platform-tools/`.

Nell'appendice E vengono descritti in maniera dettagliata i vari comandi `adb` e il loro significato e utilizzo [31].

3.2 Eclipse e Android Development Tools

Android Development Tools (ADT) [32] è un plugin per Eclipse[21] che è stato progettato per fornire un potente ambiente integrato in cui costruire le applicazioni Android.

ADT estende le capacità di Eclipse e consente di configurare rapidamente nuovi progetti Android, creare un'interfaccia utente dell'applicazione, aggiungere i pacchetti basati sulle Framework API di Android, eseguire il debug delle applicazioni utilizzando gli strumenti dell'Android SDK, e persino esportare file `.apk` al fine di distribuire l'applicazione.

Molti degli strumenti dell'Android SDK che è possibile avviare o eseguire dalla riga di comando sono integrati nei menù e nelle prospettive di Eclipse, o come parte dei processi in background eseguiti da ADT. Tra questi abbiamo:

- Traceview: consente di profilare l'esecuzione del programma;
- Hierarchy Viewer: consente di visualizzare la vista gerarchica dell'applicazione per trovare le inefficienze;
- Perfect Pixel: consente di esaminare da vicino l'interfaccia utente per aiutare con la progettazione e la realizzazione;
- android: fornisce l'accesso all'Android SDK Manager e all'AVD Manager;
- DDMS: fornisce funzionalità di debug, tra cui cattura dello schermo, informazioni su thread e heap, e logcat;
- adb: fornisce l'accesso a un dispositivo dalla propria macchina;
- ProGuard: permette l'offuscamento, la riduzione e l'ottimizzazione del codice.

Nell'appendice F vengono descritti i passi per installare il plugin ADT su Eclipse [34].

[21] Eclipse è un ambiente di sviluppo integrato (IDE) multi-linguaggio e multipiattaforma. [33]

3.3 Android Native Development Kit

L'Android Native Development Kit (NDK) [35] è un set di strumenti che consente di incorporare i componenti che fanno uso di codice nativo nelle applicazioni Android.

Le applicazioni Android vengono eseguite nella macchina virtuale Dalvik. L'NDK consente di implementare parti delle applicazioni utilizzando linguaggi a codice nativo quali C e C++. Questo può fornire benefici per certe classi di applicazioni, in forma di riutilizzo di codice esistente e in alcuni casi di una maggiore velocità.
L'NDK fornisce:

- un insieme di strumenti e file utilizzati per generare librerie di codice nativo da sorgenti C e C++;
- un modo per incorporare le relative librerie native in un pacchetto applicazione (`.apk`) che può essere distribuito su dispositivi Android;
- un insieme di librerie native che saranno supportate in tutte le versioni future della piattaforma Android, a partire da Android 1.5;
- documentazione, esempi e tutorial.

3.4 Samsung Galaxy S

Il dispositivo utilizzato per eseguire i test di customizzazione di Android è un Samsung Galaxy S (GT-i9000). Si tratta di uno smartphone prodotto da Samsung annunciato nel mese di marzo 2010.

Esteticamente si presenta piuttosto simile al rivale Apple iPhone, ma il dispositivo coreano ha un'identità tutta sua, a partire dal sistema operativo Android 2.1 (aggiornabile alla verisone 2.3.3 o anche al nuovo arrivato Android 4.0 Ice Cream Sandwich se si ricorre a firmware modificati) ed al display full-touch capacitivo da 4" Super AMOLED con una risoluzione di 800x480 pixels, 233ppi e 16 milioni di colori. Lo smartphone è inoltre dotato di un processore ARM Cortex A8 da 1GHz, 512MB di RAM, memoria interna da 8GB espandibile con microSD fino a 32GB, e fotocamera da 5Mpx con autofocus che permette di girare video in alta definizione fino a 720p.

Figura 3.2: Il Samsung Galaxy S.

3.1.1 Samsung Kies e Odin

Samsung Kies è un software scaricabile dal sito ufficiale della Samsung [36] e che permette di fare molteplici cose:

- creare playlist musicali;
- sincronizzare musica, foto e video;
- sincronizzare i contatti con quelli di Outlook, Google o Yahoo;
- acquistare e sincronizzare applicazioni;
- salvare video e foto scattate;
- aggiornare il firmware del dispositivo;
- eseguire il backup e il ripristino dei dati.

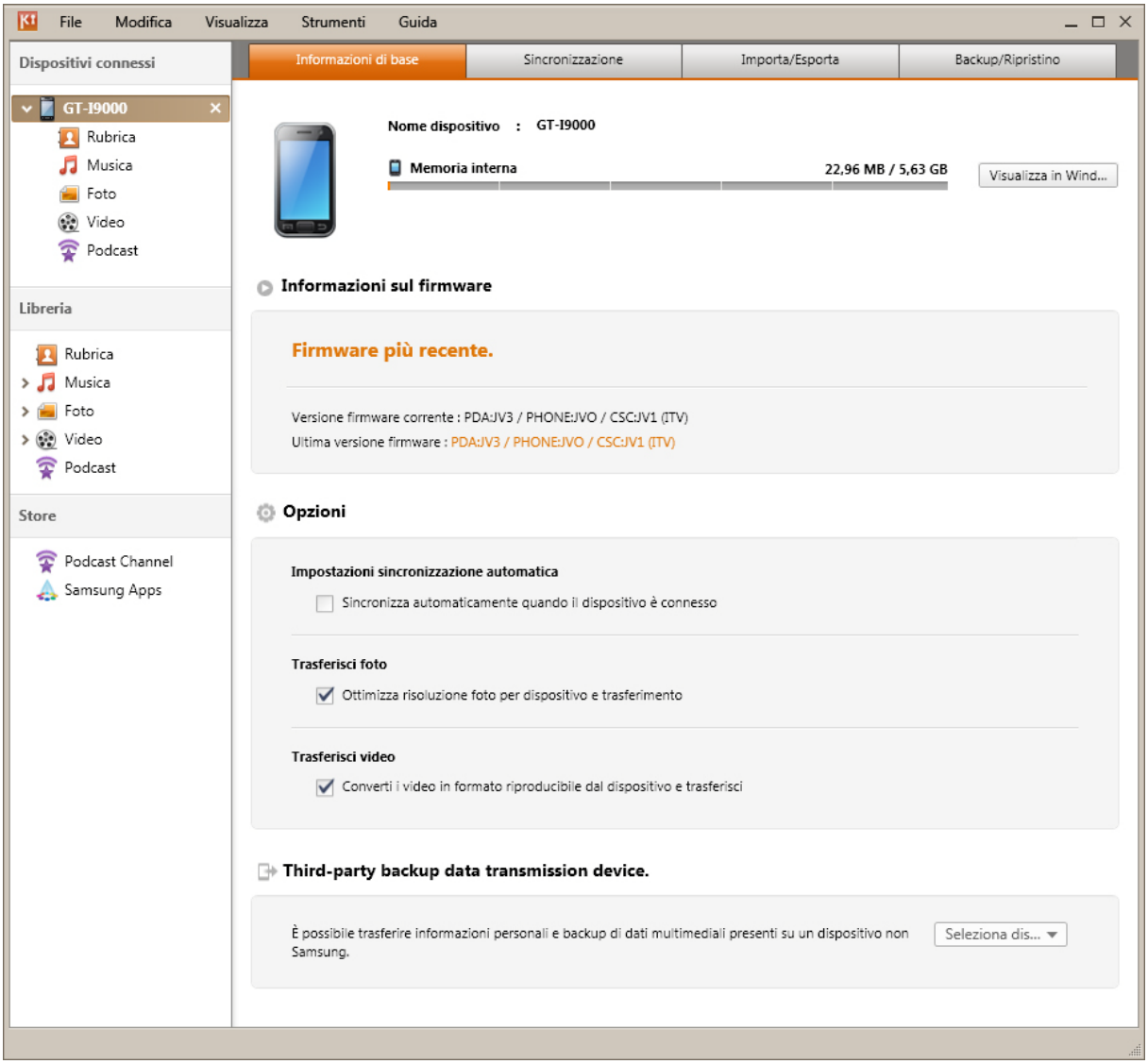

Figura 3.3: Samsung Kies.

Kies permette solamente l'aggiornamento dei firmware ufficiali. Per poter installare firmware e kernel personalizzati (ma anche ufficiali), è necessario l'utilizzo di un software differente: Odin. Si tratta di un programma che permette di flashare[22] firmware, kernel, modem, bootloader ed altro ancora su dispositivi Samsung.

[22]È il processo si sovrascrittura dei dati esistenti sui moduli ROM presenti in dispositivi elettronici con dei nuovi dati.

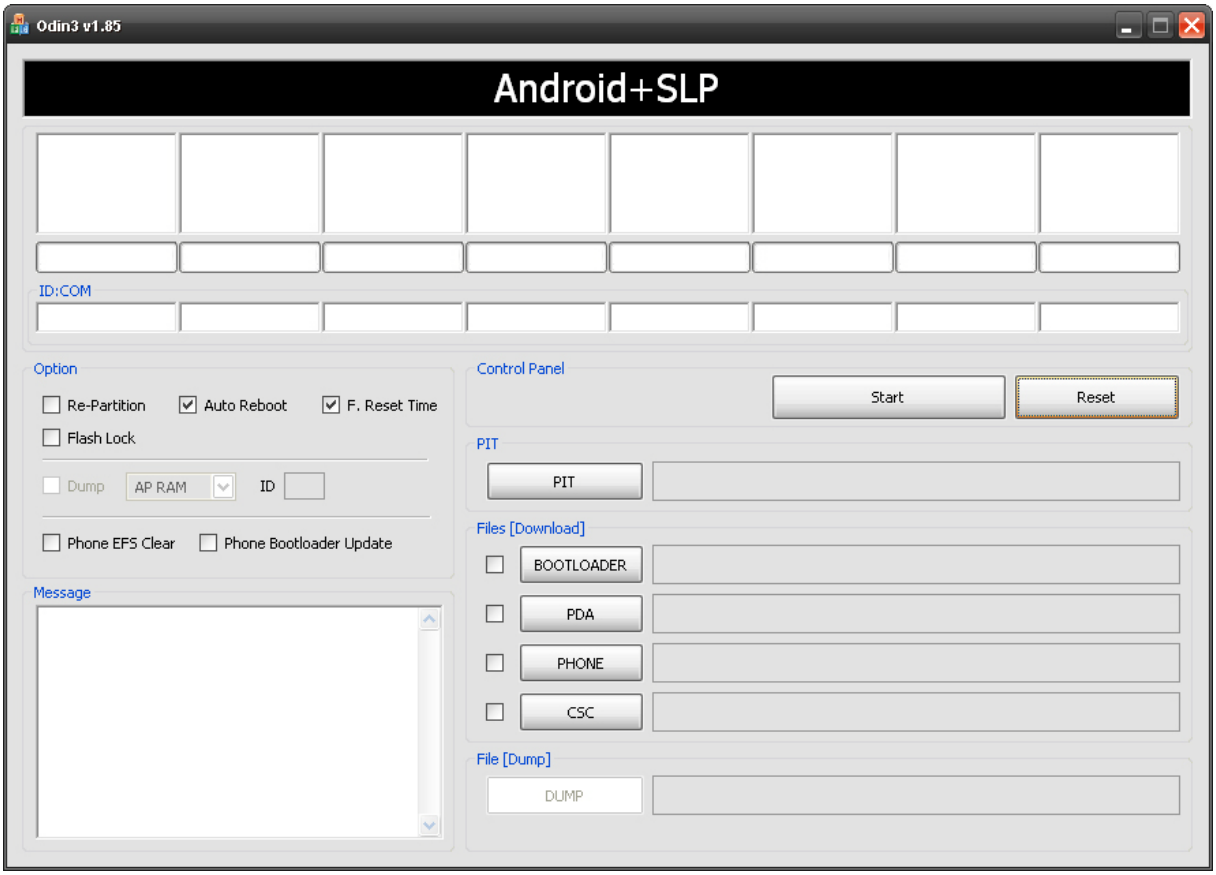

Figura 3.4: Odin.

Nell'appendice G vengono descritti i passi per installare un custom firmware [37] e un custom kernel [38] sul Galaxy S attraverso l'uso di Odin.

Capitolo 4

Customizzazione di Android passo per passo

In questo capitolo verrà descritto come customizzare (personalizzare) Android, partendo dalla compilazione dei suoi file sorgente, per poi passare all'installazione del kernel sull'emulatore e sul Samsung Galaxy S, e ai vari test fatti su di essi.

4.1 Compilazione

4.1.1 Compilazione del kernel per l'emulatore

Per compilare un kernel Android è innanzitutto necessario disporre di una macchina con installato Mac OS o una distribuzione Linux (ad esempio Ubuntu). La compilazione sotto Windows non è attualmente supportata.

Installazione delle dipendenze

Per prima cosa occorre installare il Sun JDK. Per scaricarlo è necessario aggiungere il repository appropriato e indicare al sistema quale JDK deve essere utilizzato. Per Android Gingerbread e le versioni successive, si utilizza Java 6:

```
sudo add-apt-repository "deb http://archive.canonical.com/
  lucid partner"
```

```
sudo apt-get update
sudo apt-get install sun-java6-jdk
```

Quindi si passa all'installazione dei pacchetti necessari:

```
sudo apt-get install git-core gnupg flex bison gperf build-
  essential zip curl zlib1g-dev libc6-dev lib32ncurses5-dev
  ia32-libs x11proto-core-dev libx11-dev lib32readline5-dev
  lib32z-dev libgl1-mesa-dev g++-multilib mingw32 tofrodos
  python-markdown libxml2-utils xsltproc valgrind libsdl-
  dev libesd0-dev libwxgtk2.6-dev libglade2-dev
```

Infine si scarica l'Andorid NDK dal sito web di Android Developers [35] e lo si scompatta nella home directory. L'Andorid NDK verrà utilizzato come compilatore.

Download dei sorgenti del kernel Andorid

Per ottenere i sorgenti del kernel, digitare dal terminale:

```
git clone https://android.googlesource.com/kernel/goldfish-
  .git
cd goldfish
git checkout –t origin/android-goldfish-2.6.29 –b goldfish
```

Compilazione del kernel

Digitare il comando (1) se si vogliono utilizzare le configurazioni del kernel di default, oppure il comando (2) se si vuole personalizzare la configurazione:

(1) `make ARCH=arm goldfish_defconfig`

(2) `make ARCH=arm gconfig`

Il comando per compilare il kernel è:

```
make ARCH=arm SUBARCH=arm CROSS_COMPILE=~/<ndk-directory>/
  toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86
  /bin/arm-linux-androideabi- -j4
```

[23]

Se la compilazione si conclude senza errori, verrà visualizzato il messaggio:

```
Kernel:  arch/arm/boot/zImage is ready
```

L'immagine del kernel è stata creata correttamente ed è possibile utilizzarla per lanciare l'emulatore.

[23] Il comando make è in grado di gestire attività parallele con l'argomento –jN: è comune utilizzare un numero di attività N che sia 1 o 2 volte il numero di thread hardware della macchina utilizzata per la compilazione. Ad esempio su una macchina dual-E5520 (2 CPU, 4 core per CPU, 2 thread per core), le più veloci compilazioni sono realizzate con comandi tra make -j16 e make -j32.

4.1.2 Compilazione del kernel per il Galaxy S

Il procedimento per compilare il kernel Android per il Galaxy S è del tutto identico a quello per l'emulatore, salvo per quanto riguarda il download dei sorgenti del kernel. Essi vanno infatti scaricati dal sito web del Samsung Open Source Release Center (SOSRC) [39] e scompattati nella home directory.

Quindi aprire il terminale e digitare il comando (1) se si vogliono utilizzare le configurazioni del kernel di default, oppure il comando (2) se si vuole personalizzare la configurazione:

(1) `make ARCH=arm aries_eur_defconfig`

(2) `make ARCH=arm gconfig`

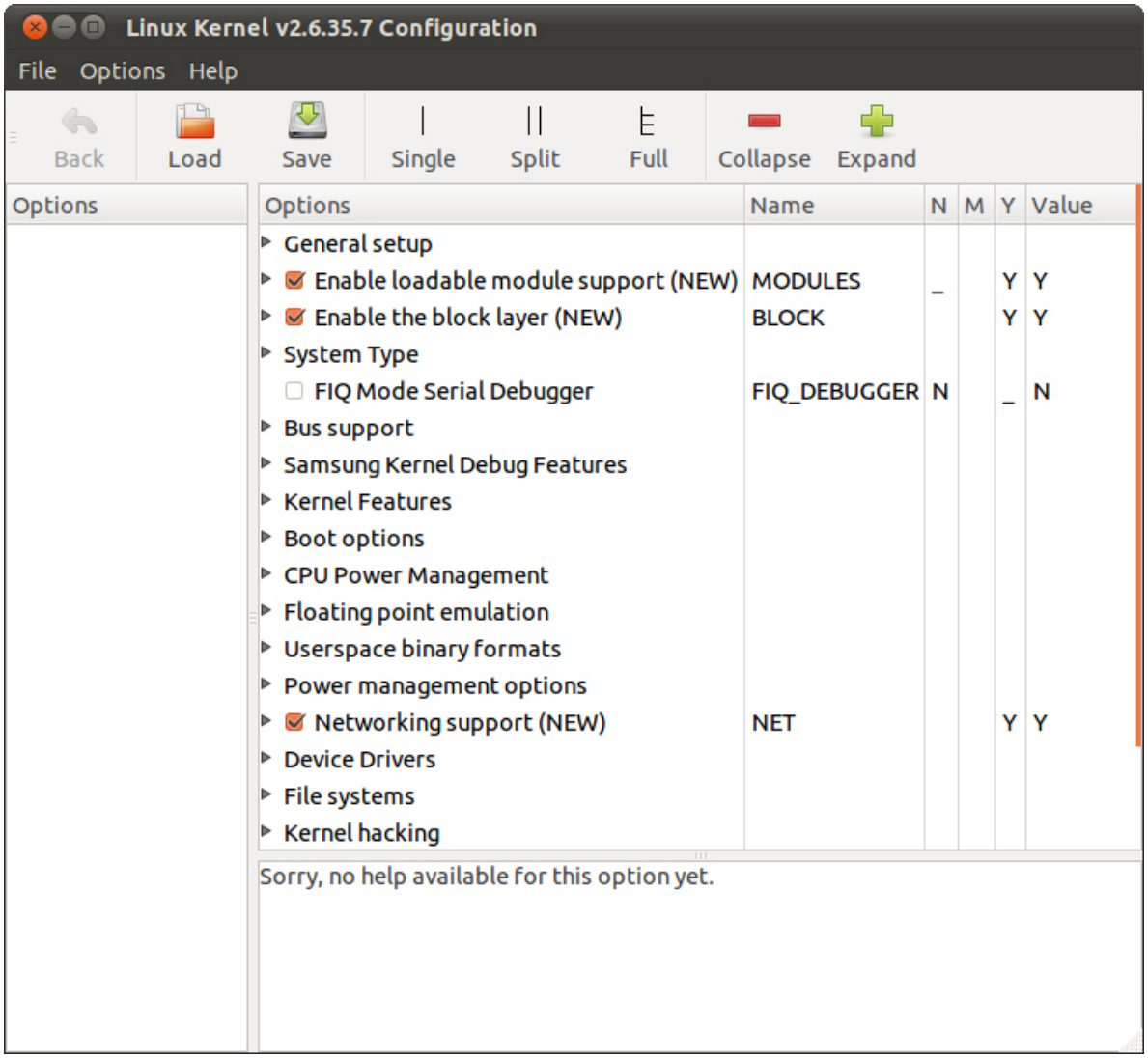

Figura 4.1: La schermata di configurazione del kernel.

Il comando per compilare il kernel è:

```
make ARCH=arm SUBARCH=arm CROSS_COMPILE=~/<ndk-directory>/
  toolchains/arm-linux-androideabi-4.4.3/prebuilt/linux-x86
  /bin/arm-linux-androideabi- -j4
```

Una volta terminata la compilazione, verranno visualizzati i seguenti messaggi:

```
Kernel:  arch/arm/boot/zImage is ready
LD [M]   crypto/ansi_cprng.ko
LD [M]   drivers/scsi/scsi_wait_scan.ko
LD [M]   drivers/net/wireless/bcm4329/dhd.ko
LD [M]   drivers/misc/vibetonz/vibrator.ko
LD [M]   drivers/misc/fm_si4709/Si4709_driver.ko
LD [M]   drivers/bluetooth/bthid/bthid.ko
```

Oltre all'immagine del kernel, troveremo alcuni file con estensione `.ko`: si tratta dei moduli del kernel.

4.2 Installazione

4.2.1 Installazione del kernel sull'emulatore

Per prima cosa occorre scaricare l'Andorid SDK[24] dal sito web di Android Developers [40] e scompattare l'archivio nella home directory.

Quindi si crea un Android Virtual Device (AVD) utilizzando l'AVD Manager oppure la riga di comando e il tool `android` (vedi appendice B).

Si lancia, infine, l'emulatore specificando l'AVD e l'immagine del kernel:

```
emulator -avd my_avd -kernel <kernel-directory>/arch/arm/
  boot/zImage
```

Per un elenco dei comandi dell'emulatore, si veda l'appendice D.

[24] L'Android SDK separa le diverse parti del SDK in pacchetti scaricabili separatamente utilizzando l'Android SDK Manager. Il pacchetto iniziale del SDK include solo il *SDK Tools*. Per sviluppare un'applicazione Android, è necessario scaricare almeno una piattaforma Android e gli ultimi *SDK Platform-tools*. È possibile aggiornare e installare i pacchetti SDK in qualsiasi momento utilizzando il l'Android SDK Manager. Per lanciare l'Android SDK Manager, aprire un terminale, navigare fino alla directory `tools/` ed eseguire il comando `andorid sdk`.

Figura 4.2: La schermata di caricamento di Android.

4.2.2 Installazione del kernel sul Galaxy S

Per installare il kernel sul Galaxy S si utilizza il programma Odin (vedi paragrafo G.2 dell'appendice G). Odin è disponibile solo per Windows; quindi occorre copiare l'immagine del kernel su una macchina differente.

Odin richiede che il file relativo al kernel sia un archivio tar. Quindi, prima di passare su una macchina Windows, occorre comprimere l'immagine del kernel:

```
cd <kernel-directory>/arch/arm/boot/
tar -cvf custom-kernel.tar zImage
```

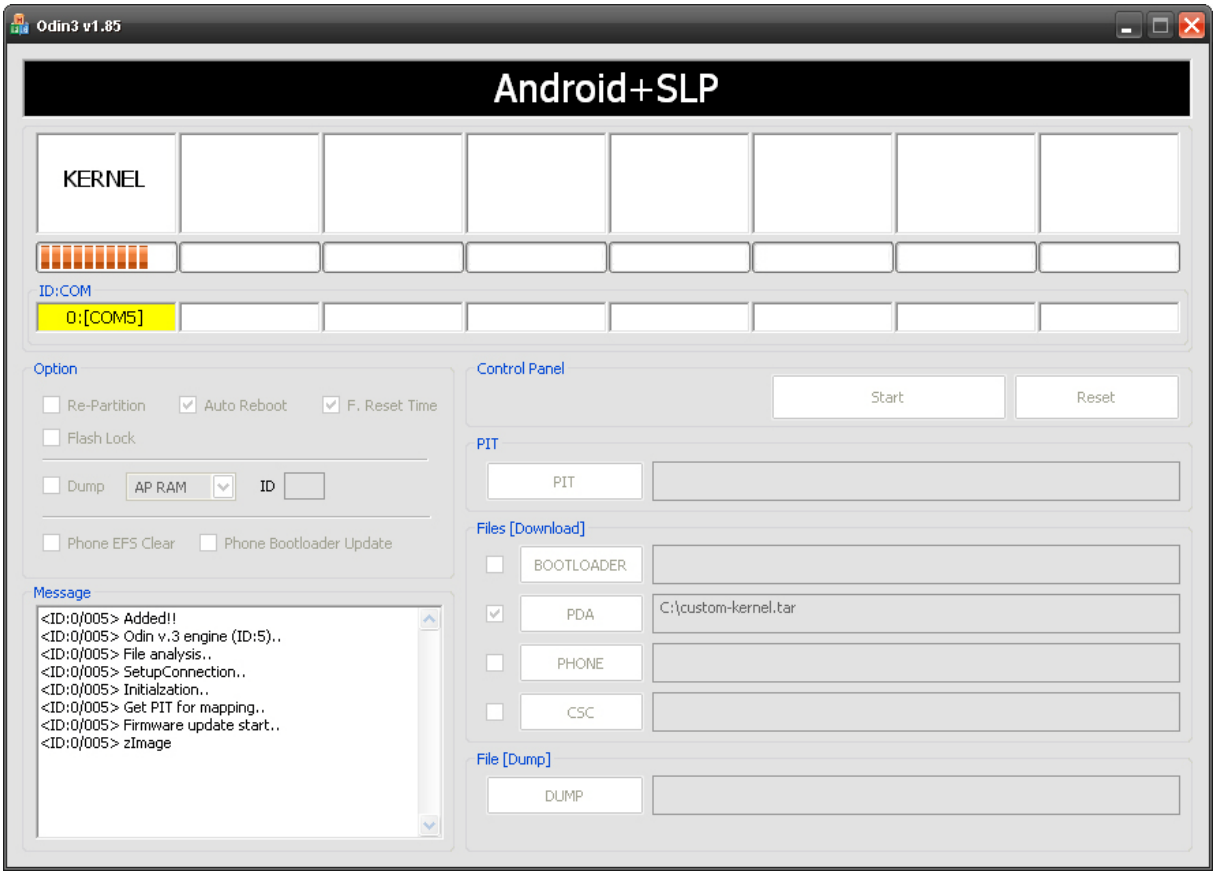

Figura 4.3: La schermata di Odin durante il flashing del kernel.

Una volta aver copiato l'archivio su una macchina Windows, avviare Odin e flashare il kernel sul Galaxy S.

4.3 Problemi riscontrati

Sebbene siano stati seguiti alla lettera i passi descritti all'interno del file README (presente nell'archivio dei file sorgente) per compilare il kernel del Galaxy S, l'installazione non è andata a buon fine. Infatti, dopo aver eseguito il flashing dell'immagine del kernel con Odin e avviato lo smartphone per testarne il funzionamento, questo è entrato in loop al boot: una volta acceso, veniva visualizzato il logo Samsung (vedi figura 4.4) per qualche secondo e poi si riavviava.

Figura 4.4: La schermata di avvio del Galaxy S.

Diverse sono state le strade intraprese per capire quale fosse il problema e per cercare di risolverlo:

- ricompilazione dei sorgenti con specifiche opzioni di debug in modo tale da avere uno strumento di rilevazione degli errori all'avvio;
- download (sempre dal SOSRC) e compilazione di sorgenti differenti;
- ricompilazione dei sorgenti includendo nell'immagine del kernel i moduli dello smartphone [41];
- ricompilazione dei sorgenti utilizzando un differente compilatore;
- flashing dell'immagine del kernel originale[25] per l'emulatore;

[25]Si tratta dell'immagine del kernel che viene creata quando si scarica dall'Android SDK Manager una piattaforma Android. È possibile trovare tale immagine in `<sdk-directory>/platforms/<android-version>/` sotto il nome di `kernel-qemu`.

- flashing dell'immagine del kernel ricompilata per l'emulatore;
- utilizzo dell'Android Debug Bridge come strumento di debug.

Nessuna di queste strade ha però portato alla risoluzione del problema. La compilazione di differenti sorgenti e l'utilizzo di differenti configurazioni per il kernel (sia quelle di default, che quelle "spulciando" tra le opzioni), hanno portato sempre al loop all'avvio dello smartphone. Soltanto la strada della compilazione dei moduli prima della compilazione del kernel ha portato ad un risultato differente (sebbene inutile): dopo l'avvio, lo smartphone rimaneva bloccato sulla schermata del logo Samsung.

Anche gli strumenti di debug non hanno fornito una risposta al problema: tali strumenti, infatti, non erano in grado di rilevare il dispositivo e quindi non veniva visualizzato alcun log. Questi strumenti non sono risultati totalmente inutili: infatti l'assenza di log e l'impossibilità di rilevare il dispositivo può significare che il Galaxy S di sblocca all'avvio in una sorta di schermata di bios e non è in grado di avviare il kernel.

Un altro tentativo è stato quello di avviare l'emulatore con i kernel compilati per il Galaxy S. Anche qui nessun esito positivo: l'emulatore rimaneva bloccato in una schermata nera senza mai raggiunge la schermata di caricamento di Android (rappresentata in figura 4.2). Gli strumenti di debug dell'emulatore e la sua console non sono stati di aiuto per rilevare il problema in quanto l'emulatore non era in grado di avviarsi.

Anche la strada opposta, ovvero il flashing di immagini del kernel per l'emulatore sullo smartphone, non ha portato ad una soluzione: dispositivo bloccato anche qui sulla schermata del logo Samsung.

4.4 Test del kernel sull'emulatore

Non essendo stato possibile testare il kernel ricompilato sul Galaxy S, si è lavorato sull'emulatore. In particolare sono stati effettuati diversi test su due emulatori lanciati contemporaneamente: uno (*pippo*) con kernel originale e uno (*pluto*) con un kernel ricompilato. Su entrambi gli emulatori si è utilizzata come piattaforma Android la versione Gingerbread 2.3.3.

La figura 4.6 mostra come effettivamente sui due emulatori ci sono i due diversi kernel. Alla voce *Versione kernel* troviamo infatti due distinti valori: nell'emulatore

pluto la dicitura “`bomo@bomo-1101HA #1`” nella seconda riga, fa riferimento al terminale dal quale è stato compilato il kernel.

Figura 4.5: La schermata Home dei due emulatori.

I testi che sono stati effettuati sono:

- verifica delle funzionalità principali del sistema operativo;
- avvio delle applicazioni installate di default;
- installazione e avvio dell’applicazione *MyApp* da me realizzata (vedi paragrafo 4.4.1);
- utilizzo della console dell’emulatore per testare i vari comandi (vedi paragrafo D.2 dell’appendice D);
- utilizzo della console dell’emulatore per simulare chiamate e invio di SMS tra i due emulatori (vedi figure 4.7, 4.8, 4.9);
- lettura della console di debug per analizzare ciò che accade all’avvio del kernel;
- utilizzo dell’Android Debug Bridge per caricare/scaricare file sulla/dalla scheda SD virtuale.

Figura 4.6: La schermata *Info sul telefono* dei due emulatori.

L'emulatore inoltra automaticamente le chiamate vocali simulate ed gli SMS da un emulatore all'altro. Per inviare una chiamata vocale o un SMS, è sufficiente utilizzare l'applicazione dialer (*Phone*) o l'applicazione SMS (*Messaging*), rispettivamente, da uno degli emulatori.
Per avviare una chiamata vocale simulata occorre:

- avviare l'applicazione dialer sull'emulatore di origine;
- come numero da comporre, immettere il numero della porta di console dell'emulatore che si desidera chiamare. È possibile determinare il numero della porta di console dell'emulatore di destinazione, controllando il titolo della finestra, dove il numero di porta della console è riportato come *porta:nome_avd*;
- premere il tasto "Chiama". Una nuova chiamata in entrata verrà visualizzata nell'emulatore di destinazione.

Per inviare un SMS il procedimento è analogo: avviare l'applicazione SMS, specificare il numero della porta di console dell'emulatore di destinazione come indirizzo del SMS, inserire il testo del messaggio, e inviare il messaggio. Il messaggio verrà recapitato all'emulatore di destinazione.

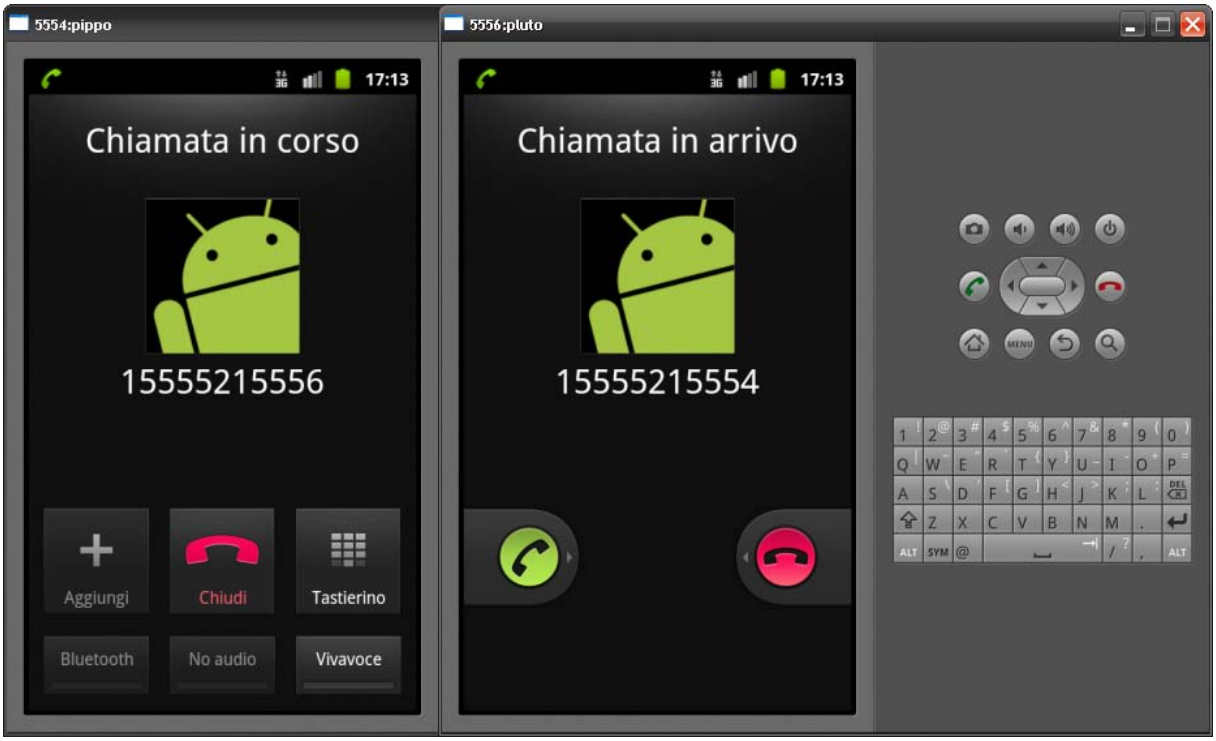

Figura 4.7: La simulazione di una chiamata tra i due emulatori.

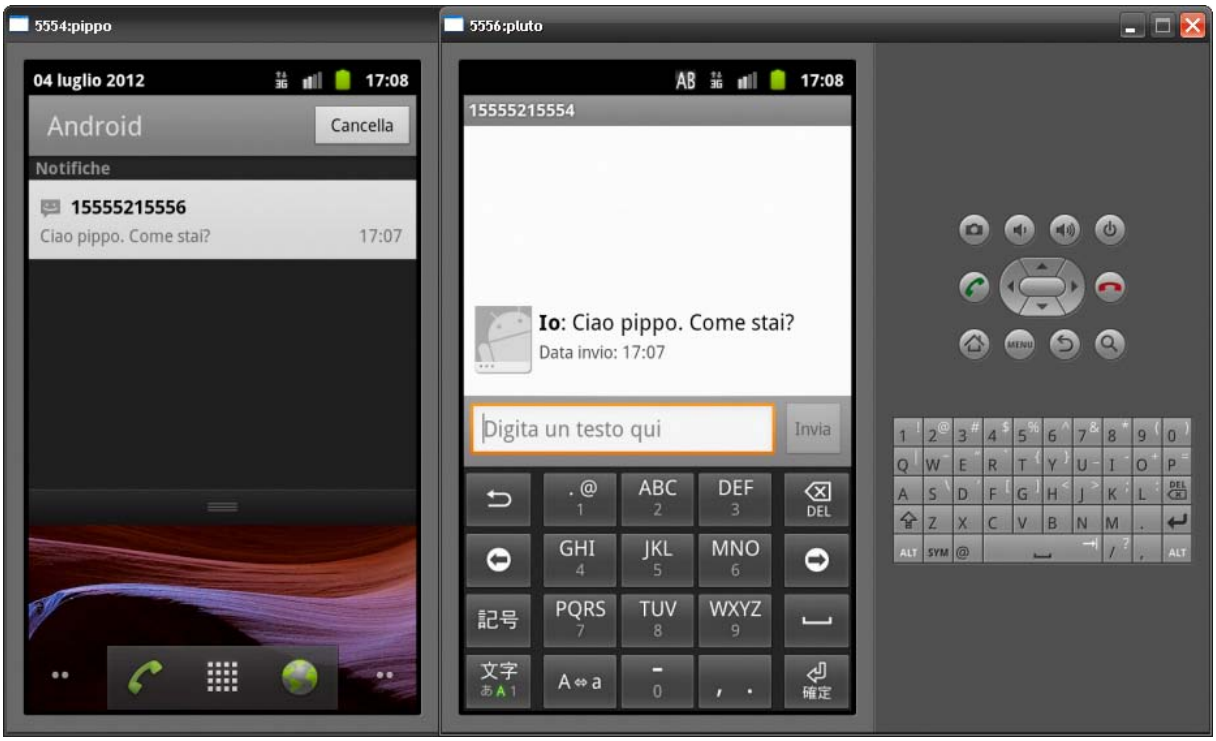

Figura 4.8: La simulazione dell'invio di SMS tra i due emulatori.

Figura 4.9: La simulazione dell'invio di SMS tra i due emulatori.

Figura 4.10: A sinistra l'applicazione *Browser* in esecuzione.
A destra tutte le applicazioni del menù di Android.

4.4.1 Una semplice applicazione Android

Per testare al meglio il kernel ricompilato, si è deciso di realizzare una semplice applicazione Android. Questa applicazione, una volta avviata, stampa un semplice messaggio testuale (vedi figura 4.11).

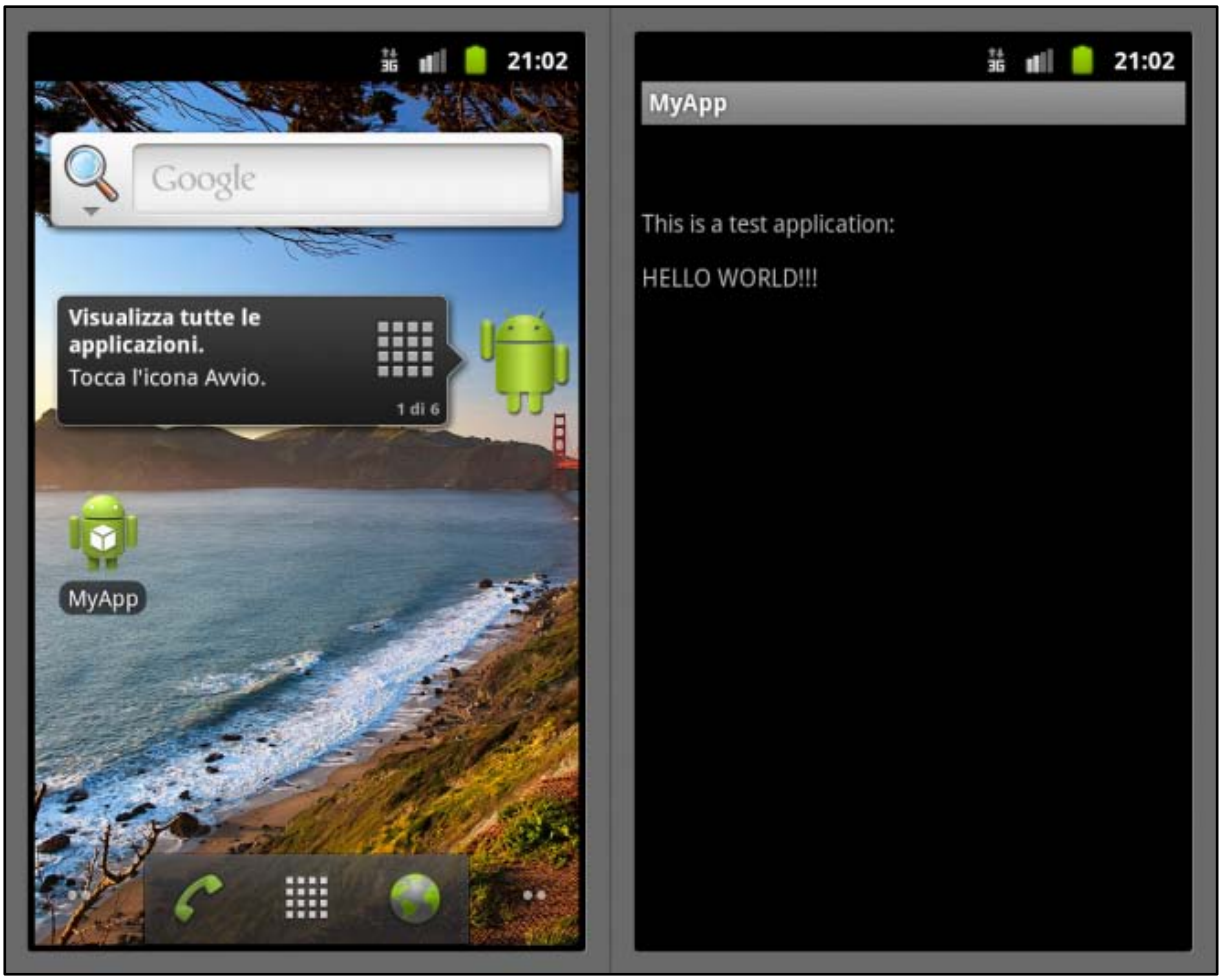

Figura 4.11: A sinistra la schermata Home con l'icona dell'applicazione *MyApp*. A destra l'esecuzione dell'applicazione *MyApp*.

La realizzazione e l'installazione dell'applicazione sull'emulatore, è stata fatta attraverso il programma Eclipse e il suo plugin ADT (vedi paragrafo 3.2).

La guida proposta sul sito web di Android Developers [42] descrive nel dettaglio tutti i passi da seguire, partendo dalla creazione del progetto e alla scrittura del codice

su Eclipse, per poi passare alla creazione del file di installazione `.apk` e alla sua installazione sull'emulatore.

Di seguito viene mostrato il codice Java dell'applicazione *MyApp*:

```
1   package com.example.myapp;
2
3   import android.app.Activity;
4   import android.os.Bundle;
5   import android.widget.TextView;
6
7   public class MyAppActivity extends Activity {
8       /** Called when the activity is first created. */
9       @Override
10      public void onCreate(Bundle savedInstanceState) {
11          super.onCreate(savedInstanceState);
12          TextView tv = new TextView(this);
13          tv.setText("\n\n\nThis is a test application:" +
                       "\n\n" + "HELLO WORLD!!!");
14          setContentView(tv);
15      }
16  }
```

Conclusioni

Nel primo capitolo è stato presentato il sistema operativo Android: dopo un breve accenno sulla sua storia, sono state elencate le sue varie versioni ed è stata descritta in maniera dettagliata la sua architettura.

Il secondo capitolo è stato incentrato sul kernel: dopo una prima panoramica sui vari tipi di kernel, il capitolo si è incentrato sul kernel Linux e sull'importanza dell'installazione di un kernel ricompilato sulla propria macchina. Sono stati infine descritti i principali custom kernel di Android installabili sul Samsung Galaxy S.

Il terzo capitolo è stato incentrato sull'ambiente di sviluppo: sono stati presentati i vari strumenti forniti dall'Android SDK, il plugin ADT per Eclipse e l'Android NDK. Infine è stato illustrato lo smartphone utilizzato per i test di "personalizzazione" di Android.

Nel quarto capitolo è stata presentata la guida per la customizzazione di Android, partendo dalla compilazione dei suoi file sorgente, per poi passare all'installazione del kernel sull'emulatore e sul Samsung Galaxy S, e ai vari test fatti su di essi. Sono stati inoltre discussi i vari problemi riscontrati.

Infine abbiamo visto le appendici A, B, C, D, E, F, G che, rispettivamente, hanno trattato nel dettaglio: la ricompilazione del kernel Linux, l'Android Virtual Device, il tool android, l'emulatore di Andorid, l'Android Debug Bridge, l'installazione del plugin ADT su Eclipse, e l'utilizzo di Odin.

L'obiettivo iniziale di questa tesi è stato quello di studiare l'architettura di Android e vedere cosa era possibile customizzare ad ogni livello del suo stack. In particolare ci si è soffermati sul suo livello più basso, ovvero il kernel. Sebbene non si è riusciti ad installare un kernel ricompilato sullo smartphone, questa tesi ho prodotto come risultato finale una dettagliata guida per tutti coloro che vorranno in futuro cimentarsi nella customizzazione e nello sviluppo di Android. Risulta infatti chiaro come Android,

sebbene sia un sistema operativo giovane ed abbia imponenti concorrenti come Apple iOS e Windows Phone, abbia già raggiunto una forte popolarità nel mercato odierno e, grazie al crescente mercato degli smartphone e delle applicazioni, sempre maggiore sarà lo sviluppo e il miglioramento di questo sistema operativo non solo da parte di Google, ma anche da parte degli utenti che, in base alle loro esigenze, vorranno ottenere il massimo.

La sempre più crescente popolarità di Android e il fatto che sia un sistema operativo open source, rende questo sistema operativo un campo fertile per numerosi e sempre più avanzati sviluppi futuri. Un primo possibile sviluppo risiede nell'analizzare quali hardware dei dispositivi mobili (accelerometro, sistema di geolocalizzazione, bluetooth, ecc…) dispongono di driver proprietari e quali invece open source. In questo modo sarà possibile capire cosa effettivamente sia possibile personalizzare a livello del kernel.

Un altro possibile sviluppo futuro risiede nel kernel dual boot, ovvero nella possibilità di installare su uno stesso dispositivo Android due differenti kernel. Questo permetterebbe agli utenti di avviare il proprio dispositivo scegliendo il kernel che preferisce in base alle proprie esigenze.

Appendice A

Ricompilazione del kernel Linux

La seguente guida [43] mostra come ricompilare e installare un kernel Linux su una macchina con installato il sistema operativo Ubuntu nella versione 11.04 (Natty Narwhal).

A.1 Preparativi

Installazione delle dipendenze

Il processo di compilazione richiede l'installazione dei pacchetti *build-essential*, *bin86*, *fakeroot*, *kernel-package*. Inoltre, se si desidera usare un'interfaccia grafica per configurare il kernel, è necessario installare il pacchetto *libglade2-dev*, se si usa Ubuntu, oppure il pacchetto *libncurses5-dev*, se si vuole avere un'interfaccia grafica direttamente dentro il terminale e spostarsi tra i menu usando la sola tastiera (utile per computer poco potenti o senza server grafico).

Download dell'archivio dei sorgenti del kernel

Se si desidera usare l'ultima versione stabile del kernel Linux ufficiale (la cosiddetta versione *vanilla*), scaricare il pacchetto dall'indirizzo *http://www.kernel.org/*.

In alternativa se si desidera usare la versione del kernel Linux modificata dagli sviluppatori di Ubuntu, è possibile prelevarla dai repository [44], installando il pacchetto

linux-source. Questa versione ha il vantaggio di contenere driver e componenti aggiuntivi che gli sviluppatori di Ubuntu hanno ritenuto di integrare nel kernel, ma per contro potrebbe non essere sempre aggiornata rispetto all'ultima versione del kernel Linux ufficiale.

Estrazione dei sorgenti

Sebbene non sia strettamente necessario, la policy Debian consiglia di usare la directory `/usr/src/` per compilare il kernel. Il gruppo proprietario della directory `/usr/src/` è il gruppo *src*. Per inserire il proprio utente nel gruppo *src* è sufficiente digitare il seguente comando in una finestra di terminale, sostituendo alla dicitura *nomeutente* il proprio nome utente:

```
sudo adduser nomeutente src
```

La modifica verrà applicata all'accesso successivo.

Qualora il gruppo *src* non avesse più i diritti di scrittura sulla directory `/usr/src/`, questi dovranno essere ripristinati con il seguente comando:

```
sudo chmod -R g+wr /usr/src
```

Scompattare e spostare i sorgenti nella directory di destinazione:

```
tar -xjf linux-[VERSIONE_SCARICATA].tar.bz2
mv linux-[VERSIONE_SCARICATA] /usr/src/
```

A.2 Compilazione e installazione

Entrare nella directory dei sorgenti del kernel appena scaricati:

```
cd /usr/src/linux-[VERSIONE_SCARICATA]
```

Applicazione delle patch

A questo punto, se necessario, sarà possibile applicare una o più patch al kernel. Le patch, detto in maniera semplice, sono dei file che modificano le funzioni originali del kernel adattandolo a un uso specifico: bassa latenza (soft real-time), hard real-time (risposte agli interrupt RT in qualche microsecondo), desktop molto reattivo, ecc.
Tra le più diffuse si possono trovare quelle di:

- Con Kolivas (per kernel vanilla fino al 2.6.22) [45]
- Alan Cox [46]

- Andrew Morton [47]
- Ingo Molnar [48]

Copiare la patch nella directory dei sorgenti del kernel e digitare il seguente comando:

```
bzcat [NOME_DELLA_PATCH].bz2 | patch -p1 --dry-run
```

L'opzione `--dry-run` permette di simulare l'applicazione della patch senza modificare realmente i file. Se non vengono visualizzati errori, si può procedere con l'applicazione della stessa rieseguendo il comando precedente senza l'opzione `--dry-run`. Se il procedimento non dovesse andare a buon fine, sarà necessario verificare di aver scaricato la versione corretta della patch e ripetere il procedimento.

Configurazione

Per personalizzare la configurazione, ci sono diverse modalità:

- grafica su Ubuntu (usando le librerie gtk)
  ```
  make gconfig
  ```
- pseudo-grafica nel terminale (usando le librerie ncurses)
  ```
  make menuconfig
  ```
- testuale nel terminale
  ```
  make config
  ```

È possibile cercare le varie opzioni di configurazione di ogni versione del kernel sul sito web: *http://kernel.xc.net/*.

L'errata configurazione di un semplice modulo può compromettere l'intero sistema. Per cercare di ridurre al minimo gli errori, usare il comando `lshw` per visualizzare il proprio hardware.

È possibile importare la configurazione di un altro kernel e partire da questa per effettuare ulteriori personalizzazioni. I file di configurazione per i kernel attualmente installati nel sistema si trovano nella directory `/boot/` e sono chiamati `config-VERSIONE_DEL_KERNEL`, ma è possibile importare un qualunque file di configurazione valido.

Lo stesso risultato è possibile raggiungerlo copiando a mano un file di configurazione per il kernel all'interno della directory con i sorgenti e rinominando il file in `.config`, dopodiché si avvia il programma di configurazione; oppure se non si vuole usare l'interfaccia grafica, `make oldconfig` importa direttamente la configurazione

del kernel attualmente in esecuzione sul computer, e con un'interfaccia a linea di comando propone una ad una tutte le opzioni presenti nel nuovo sorgente.

Alla fine la configurazione viene salvata all'interno del file `.config` presente nella directory del kernel. È consigliato farne una copia, se si è soddisfatti del lavoro svolto.

Impostazioni consigliate

In *General Setup* attivare:

- Support for paging of anonymous memory (swap)
  - Support for prefetching swapped memory

In *Processor type and features*:

- in *Processor family* scegliere il modello del processore
- attivare *Preemption Model -> Voluntary Kernel Preemption* (Desktop)
- in *High Memory Support* scegliere:
  - *off* se si possiede meno di 1GB di RAM
  - *1GB Low Memory Support* se si possiede 1GB di RAM
  - *4GB* se si possiede più di 1GB di RAM
- in *Timer frequency* abilitare: *1000 Hz*

In *Block layer -> IO Schedulers* lasciare solo *CFQ I/O scheduler*

In *Kernel hacking* deselezionare *Kernel debugging*

In *Device Drivers -> Character devices* selezionare *Virtual terminal*

Compilazione

Finita la fase di personalizzazione, si passa alla compilazione.

Nel caso in cui si possiede un sistema multiprocessore è possibile impostare la variabile CONCURRENCY_LEVEL affinché vengano eseguite più compilazioni in parallelo. Impostate tale variabile secondo l'uscita del comando:

```
cat /proc/cpuinfo | grep processor
```

Ad esempio per un processore con 2 core è possibile impostare tale valore a 2:

```
export CONCURRENCY_LEVEL=2
```

Il comando per compilare il kernel è:

```
make-kpkg --rootcmd fakeroot --initrd kernel_image ker-
  nel_headers modules_image
```

L'opzione `--rootcmd` è necessaria perché alcuni passaggi nella creazione del pacchetto richiedono un ambiente di root, che fakeroot simula.

L'opzione `--initrd` serve a creare un file immagine di una partizione speciale (initial ram disk), che all'avvio del sistema verrà caricata nella ram, montata temporaneamente come directory root e usata per caricare i moduli strettamente necessari per l'avvio del sistema, prima di venire sostituita dalla partizione contenente il sistema reale. L'immagine initrd è usata soprattutto nei kernel ufficiali, che devono supportare un gran numero di chipset, file system, dischi fissi, ecc…, e perciò caricano tutti questi moduli nell'initrd. Serve anche nel caso si desideri avere un bootsplash. Perché l'initrd funzioni, è necessario che l'opzione *RAM disk support* nella sezione *Block Device* della configurazione del kernel sia abilitata. Per poter fare a meno di questa opzione è necessario, nella sezione *Device Drivers*, compilare staticamente nel kernel sia il modulo relativo al chipset della propria scheda madre, sia il modulo relativo al filesystem usato nella partizione di root (in questo modo l'avvio risulterà anche un po' più veloce).

L'opzione `--append-to-version`, facoltativa, serve a definire un suffisso personalizzato per il proprio kernel. Si può scrivere ciò che si vuole, basta che sia scritto in minuscolo, senza punti e senza caratteri extra.

I target specificano quali pacchetti deb verranno creati:

- il target `kernel_image` compila il kernel e tutti i moduli scelti nella configurazione, e ne crea un pacchetto `.deb`;
- il target `kernel_headers` crea un pacchetto `.deb` con una serie di file e script che consentono di compilare moduli esterni in un secondo momento. Non è necessario, in quanto gli stessi sorgenti del kernel servono già allo scopo, ma se si desidera eliminare in seguito i sorgenti del kernel (magari per liberare spazio su disco) è bene specificare anche questo target;
- il target `modules_image` compila i sorgenti dei moduli esterni scaricati a parte e posizionati in `/usr/src/modules/` (come i moduli per i driver video proprietari). Se non ce ne fossero, non serve aggiungere questo parametro.

Installazione

Entrare nella directory superiore digitando il comando

```
cd ..
```

e installare i pacchetti appena creati (`modules-image` sarà presente solo se c'erano dei moduli in `/usr/src/modules/`). Da terminale digitare:

```
sudo dpkg -i linux-headers-[VERSIONE_DEL_KERNEL].deb
sudo dpkg -i linux-image-[VERSIONE_DEL_KERNEL].deb
sudo dpkg -i modules-image-[VERSIONE_DEL_KERNEL].deb
```

Il boot loader GRUB verrà aggiornato automaticamente. Riavviare il computer e quindi scegliere il nuovo kernel installato. Se ci sono problemi, riavviare la macchina e utilizzare il vecchio kernel.

A.3 Problemi e soluzioni

Kernel panic

Se a video subito dopo GRUB compare un messaggio di kernel panic e non si ha utilizzato l'opzione `--initrd`, allora non si ha compilato staticamente (dando M invece di Y) i moduli necessari per il kernel. Se invece si ha utilizzato l'opzione `--initrd`, si ha escluso (dando N invece di Y) dei moduli necessari per il kernel.

Esempio di messaggio di kernel panic:

```
kernel panic - not syncing: VFS: Unable to mount root fs on
  unknown-block
```

In entrambi i casi precedenti occorre:

- far ripartire la macchina scegliendo un vecchio kernel sicuramente funzionante nel menù di GRUB;
- dare nuovamente `make gconfig` (oppure `menuconfig` o `config`);
- cercare quali moduli si è dimenticato di compilare: il messaggio del kernel panic può essere d'aiuto. I moduli da abilitare assolutamente sono relativi al file system, ai driver del disco e alla sezione i2c in *Device Drivers*. Devono, inoltre, essere presenti i moduli per il bus PCI;
- dare `make kpkg-clean` prima di compilare nuovamente.

Schermata nera

Il kernel è partito, ma invece di KDM/GDM (login grafico) si ha una schermata nera con login testuale. Probabilmente questo è dovuto al fatto che si ha compilato i driver della scheda video che ora, col nuovo kernel, non vengono caricati. Per verificare questa ipotesi bisogna vedere se i driver *vesa* consentono di ottenere il login grafico.

Modificare il file `xorg.conf` con un editor testuale:

```
sudo nano /etc/X11/xorg.conf
```

Nella sezione “Device” commentare la linea “Driver” originale (mettendo un # davanti) e aggiungere la linea:

```
Driver “vesa”
```

Salvare (CTRL+O), chiudere nano (CTRL+X) e riavviare il computer.

Appendice B

Android Virtual Device

B.1 Creazione di un AVD con AVD Manager

L'Android Virtual Device Manager è un'interfaccia utente di facile utilizzo per gestire le configurazioni degli AVD (Android Virtual Device). Un AVD è una configurazione del dispositivo per l'emulatore di Android che consente di modellare diverse configurazioni di dispositivi Android.

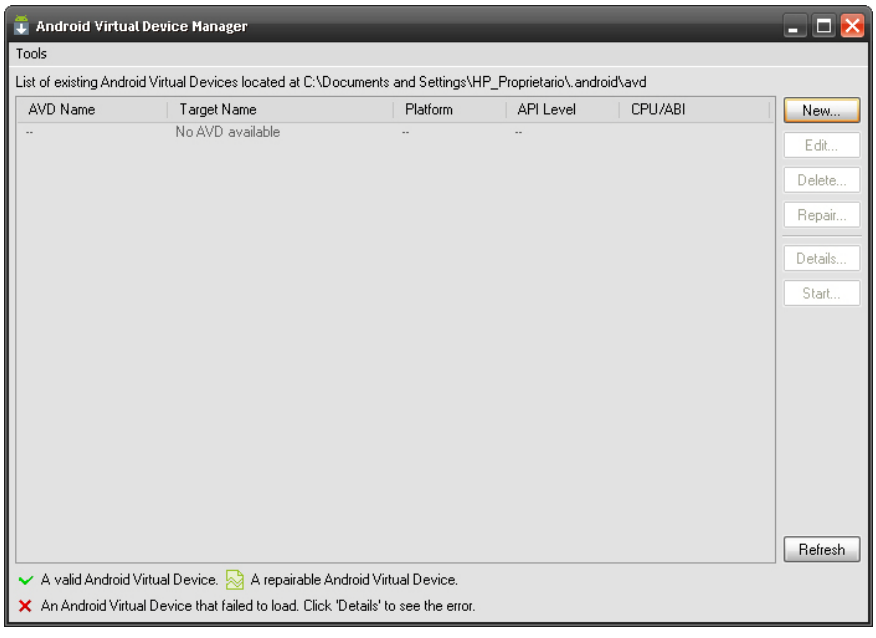

Figura B.1: L'Android Virtual Device Manager.

Quando si avvia l'AVD Manager in Eclipse o si esegue il tool `android` dalla riga di comando, si vedrà l'AVD Manager come mostrato in figura B.1.

Dalla schermata principale, è possibile creare, eliminare, riparare e avviare AVD, così come vedere i dettagli di ogni AVD. È possibile creare tanti AVD quanti se ne desidera testare.

Per creare un AVD:

1. avviare l'AVD Manager:
   - in Eclipse: selezionare **Window -> AVD Manager**, oppure cliccare sull'icona di AVD Manager nella barra degli strumenti di Eclipse;
   - in altri IDE: navigare fino alla directory `tools/` dell'SDK ed eseguire il tool `android` con argomento `avd`;
2. nel pannello *Virtual Devices*, è possibile vedere un elenco degli AVD esistenti. Cliccare su **New** per creare un nuovo AVD. Apparirà la finestra di dialogo *Create New AVD* (figura B.2);

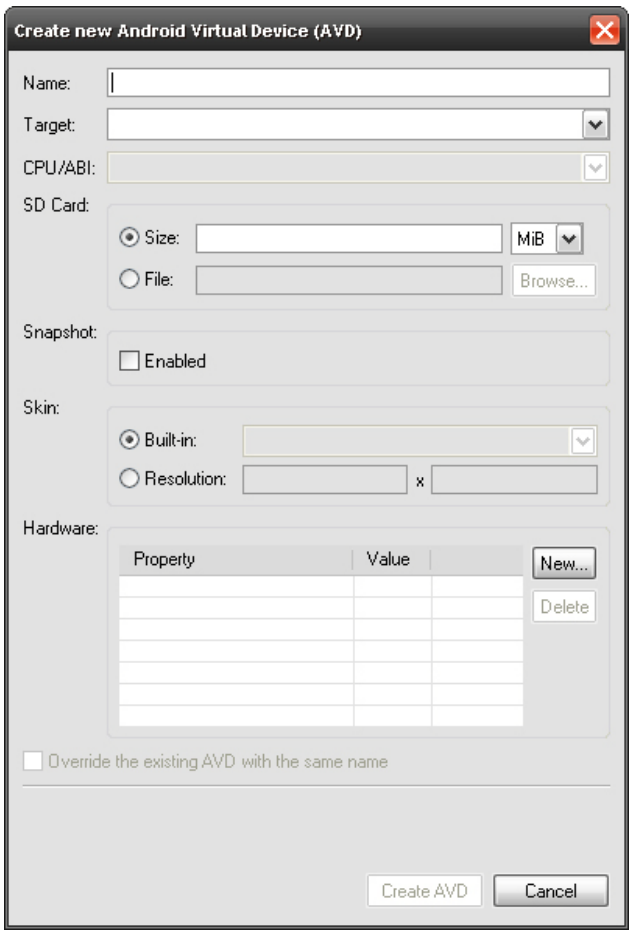

Figura B.2: La finestra di dialogo *Create New AVD*.

3. compilare i dettagli per l'AVD: inserire un nome, una piattaforma di destinazione, una dimensione per la scheda SD e una skin (HVGA è di default). È inoltre possibile aggiungere funzionalità hardware specifiche del dispositivo emulato facendo clic sul pulsante **New...** e selezionando la funzionalità.
4. cliccare su **Create AVD**.

L'AVD è ora pronto ed è possibile chiudere l'AVD Manager, creare altri AVD, o lanciare un emulatore con l'AVD appena creato, selezionando un dispositivo e cliccando su **Start**.

B.2 Creazione di un AVD dalla riga di comando

Il tool `android` consente di gestire gli AVD dalla riga di comando.

Per creare un AVD occorre prima di tutto aprire un terminale e posizionarsi nella directory `<sdk-directory>/tools/`. Si esegue quindi il comando `android create avd` , con opzioni che specificano un nome per il nuovo AVD e l'immagine del sistema che si desidera eseguire sull'emulatore quando viene invocato l'AVD. È possibile specificare altre opzioni dalla riga di comando, come la dimensione della scheda SD emulata, la skin dell'emulatore, o un percorso personalizzato per i file dei dati utente. Ecco la riga di comando di utilizzo per la creazione di un AVD:

```
android create avd -n <name> -t <targetID> [-<option>
  <value>] …
```

È possibile utilizzare qualsiasi nome che si desidera per l'AVD, ma poiché è probabile che si creino molteplici AVD, si dovrebbe scegliere un nome che permetta di riconoscere le caratteristiche generali offerti dal AVD. Il target ID è un numero intero assegnato dal tool `android`. Il target ID non è derivato dal nome dell'immagine del sistema, dalla versione, o dal livello API, o da un altro attributo, quindi è necessario eseguire il comando `android list targets` per elencare il target ID di ogni immagine del sistema. Si dovrebbe fare questo prima di eseguire il comando `android create avd`.

Per generare un elenco dei target delle immagini di sistema, si utilizza questo comando:

```
android list targets
```

Il tool `android` esegue la scansione delle cartelle `<sdk-directory>/platforms/` e `<sdk-directory>/add-ons/` alla ricerca di immagini di sistema valide e genera quindi l'elenco dei target. Ecco un esempio di output del comando:

```
Available Andorid targets:
id: 1 or "android-3"
      Name: Android 1.5
      Type: Platform
      API level: 3
      Revision: 4
      Skins: QVGA-L, HVGA-L, HVGA (default), HVGA-P, QVGA-P
 id: 2 or "android-4"
      Name: Android 1.6
      Type: Platform
      API level: 4
      Revision: 3
      Skins: QVGA, HVGA (default), WVGA800, WVGA854
 id: 3 or "android-7"
      Name: Android 2.1-update1
      Type: Platform
      API livel: 7
      Revision: 2
      Skins: QVGA, WQVGA400, HVGA (default), WVGA854,
        WQVGA432, WVGA800
 id: 4 or "android-8"
      Name: Android 2.2
      Type: Platform
      API livel: 8
      Revision: 2
      Skins: WQVGA400, QVGA, WVGA854, HVGA (default),
        WVGA800, WQVGA432
 id: 5 or "android-9"
      Name: Android 2.3
      Type: Platform
      API livel: 9
      Revision: 1
      Skins: HVGA (default), WVGA800, WQVGA432, QVGA,
        WVGA854, WQVGA400
```

Dopo aver selezionato il target che si desidera utilizzare e preso nota del suo ID, utilizzare il comando `android create avd` per creare l'AVD, fornendo il target ID come argomento `-t`. Ecco un esempio che crea un AVD con nome "my_android1.6" e target ID "2" (l'immagine del sistema di Android 1.6 nella lista sopra):

```
android create avd –n my_android1.6 -t 2
```

Se il target selezionato è un'immagine del sistema Android (`Type: Platform`), il tool `android` chiederà se si desidera creare un profilo hardware personalizzato.

```
Android 1.6 is a basic Android platform.
Do you wish to create a custom hardware profile [no]
```

Se si desidera impostare opzioni di emulazione hardware personalizzate per l'AVD, digitare "yes" e impostare i valori in base alle esigenze. Se si desidera utilizzare le opzioni di emulazione hardware predefinite per l'AVD, basta premere il tasto invio (il valore di default è "no"). Il tool `android` creerà l'AVD con il nome e l'immagine del sistema richiesto, con le opzioni specificate.

Durante il test di un'applicazione, è consigliabile testare un'applicazione in diversi AVD, utilizzando diverse configurazioni dello schermo (diverse combinazioni di dimensioni e densità). Inoltre, è necessario impostare gli AVD in modo da eseguirli ad una dimensione fisica che si avvicina al dispositivo reale.

Per impostare gli AVD ad una specifica risoluzione o densità, occorre:

1. utilizzare il comando `create avd` per creare un nuovo AVD, specificando l'opzione `--skin` con un valore che faccia riferimento al nome di una skin di default (ad esempio "WVGA800") o ad una risoluzione della skin personalizzata (ad esempio 240x432). Ecco un esempio:
   ```
   android create avd -n <nome> -t <targetID> --skin WVGA800
   ```
2. per specificare una densità personalizzato per la skin, rispondere "yes" quando viene chiesto se si desidera creare un profilo hardware personalizzato per il nuovo AVD;
3. continuare con le varie impostazioni del profilo finché il tool non chiede di specificare "Abstracted LCD density" (hw.lcd.density). Inserire un valore appropriato, come "120" per uno schermo a bassa densità, "160" per uno a media densità, o "240" uno ad alta densità;
4. impostare le altre opzioni hardware e completare la creazione dell'AVD.

L'AVD è ora pronto ed è possibile lanciare, tramite la riga di comando, un emulatore con l'AVD appena creato (vedi paragrafo D.2 dell'appendice D).

B.3 Elenco delle opzioni di emulazioni hardware

La seguente tabella elenca le opzioni hardware che si possono impostare al momento della creazione di un nuovo AVD e che vengono memorizzate nel file di configurazione dell'AVD (il file `config.ini` nella directory locale dell'AVD).

Caratteristica	**Descrizione**	**Proprietà**
Device ram size	La quantità di RAM fisica sul dispositivo, in megabyte. Il valore di default è "96".	hw.ramSize
Touch-screen support	Se c'è un touch screen nel dispositivo. Il valore di default è "yes".	hw.touchScreen
Trackball support	Se c'è un trackball nel dispositivo. Il valore di default è "yes".	hw.trackBall
Keyboard support	Se il dispositivo ha una tastiera QWERTY. Il valore di default è "yes".	hw.keyboard
DPad support	Se il dispositivo ha pulsanti DPad. Il valore di default è "yes".	hw.dPad
GSM modem support	Se c'è un modem GSM nel dispositivo. Il valore di default è "yes".	hw.gsmModem
Camera support	Se il dispositivo ha una fotocamera. Il valore di default è "yes".	hw.camera
Maximum horizontal camera pixels	Il valore di default è "640".	hw.camera.maxHorizontalPixels
Maximum vertical camera pixels	Il valore di default è "480".	hw.camera.maxVerticalPixels
GPS support	Se c'è GPS nel dispositivo. Il valore di default è "yes".	hw.gps
Audio recording support	Se il dispositivo può registrare audio. Il valore di default è "yes".	hw.audioInput
Audio playback support	Se il dispositivo può riprodurre audio. Il valore di default è "yes".	hw.audioOutput

Caratteristica	Descrizione	Proprietà
Battery support	Se il dispositivo può essere eseguito su una batteria. Il valore di default è “yes”.	hw.battery
Accelerometer	Se c’è un accelerometro nel dispositivo. Il valore di default è “yes”.	hw.accelerometer
SD Card support	Se il dispositivo supporta l’inserimento/rimozione di schede SD virtuali. Il valore di default è “yes”.	hw.sdCard
Cache partition support	Se si usa una partizione cache sul dispositivo. Il valore di default è “yes”.	disk.cachePartition
Cache partition size	Il valore di default è “66MB”.	disk.cachePartition.size
Abstracted LCD density	Imposta la densità utilizzata dallo schermo dell’AVD. Il valore di default è “160”.	hw.lcd.density

Appendice C

Il tool android

L'utilizzo del tool `android` dalla riga di comando è il seguente:

```
android [global options] action [action options]
```

Le opzioni globali possibili sono:

- `-s`: silent mode (vengono stampati solo gli errori)
- `-h`: usage help
- `-v`: verbose mode (vengono stampati gli errori, gli avvisi e i messaggi informativi)

La seguente tabella elenca le azioni (`action`) e le opzioni (`action option`) del tool `android`, e spiega il loro significato e utilizzo.

Azione	**Opzione**	**Descrizione**
`avd`	Nessua	Avvia l'AVD Manager.
`sdk`	Nessuna	Avvia l'SDK Manger.
`create avd`	`-n <name>`	Il nome dell'AVD.
	`-t <targetID>`	Target ID dell'immagine del sistema da utilizzare con il nuovo AVD. Per ottenere un elenco dei target disponibili, utilizzare il comando: `android list targets`

Azione	**Opzione**	**Descrizione**

	`-c <path>|<size>[K|M]`	Il percorso all'immagine della scheda SD da utilizzare con questo AVD o la dimensione di una nuova immagine della scheda SD da creare per questo AVD. Ad esempio:
		`-c path/to/sdcard`
		oppure
		`-c 1000M`

	`-f`	Forza la creazione dell'AVD.
	`-p <path>`	Percorso alla locazione in cui creare la directory per i file di questo AVD.
	`-s <name>|<width>-<height>`	La skin da utilizzare per questo AVD, identificato da un nome o dalle dimensioni. Il tool `android` analizza la skin corrispondente in base al nome o alla dimensione nella directory `skins/` del target riferito nell'argomento `-t <targetID>`. Per esempio: `-s HVGA-L`
`delede avd`	`-n <name>`	Il nome dell'AVD da eliminare.
`move avd`	`-n <name>`	Il nome dell'AVD da spostare.
	`-p <path>`	Percorso alla locazione in cui creare la directory per i file di questo AVD.
	`-r <new-name>`	Il nuovo nome dell'AVD se lo si desidera rinominare.
`update avd`	`-n <name>`	Il nome dell'AVD da aggiornare.

Appendice D

L’emulatore di Android

Per utilizzare l’emulatore, è necessario creare una o più configurazioni AVD. In ogni configurazione, è necessario specificare una piattaforma Android da eseguire nell’emulatore, una serie di opzioni hardware e la skin dell’emulatore che si desidera utilizzare. Poi, quando si avvia l’emulatore, è necessario specificare la configurazione AVD che si desidera caricare.

Ogni AVD funziona come un dispositivo indipendente, con la sua area di memorizzazione privata per i dati utente, per la scheda SD, e così via. Quando si avvia l’emulatore con una configurazione AVD, esso carica automaticamente i dati utente e i dati della scheda SD dalla directory AVD. Per default, l’emulatore memorizza i dati utente, i dati delle schede SD e la cache nella directory di AVD.

D.1 Elenco dei comandi emulator

Come descritto nell’appendice B, il modo più semplice per avviare l’emulatore è quello di utilizzare l’Android Virtual Device Manager. È possibile specificare una serie di opzioni per controllare l’aspetto e il comportamento dell’emulatore al momento del suo lancio. Tuttavia, queste opzioni possono essere specificate solo dalla riga di comando attraverso l’uso del comando `emulator`. Ecco la sintassi del comando:

```
emulator -avd <avd_name> [-<option> [<value>]]…
  [-<qemu args>]
```

La seguente tabella elenca tutti i comandi `emulator` e spiega il loro significato e utilizzo.

Categoria	Comando	Descrizione
AVD	`-avd <avd_name>` o `@<avd_name>`	**Obbligatorio**. Specifica l'AVD da caricare per questa istanza dell'emulatore. È necessario creare una configurazione AVD prima di lanciare l'emulatore.
Disk Image	`-cache <filepath>`	Utilizza `<filepath>` come immagine della partizione di cache di lavoro. Un percorso assoluto o relativo alla directory di lavoro corrente. Se non è specificato nessun file di cache, il comportamento predefinito dell'emulatore è quello di utilizzare un file temporaneo.
	`-data <filepath>`	Utilizza `<filepath>` come immagine del disco dei dati utente di lavoro. Opzionalmente, è possibile specificare un percorso relativo alla directory di lavoro corrente. Se `-data` non viene utilizzato, l'emulatore cerca un file chiamato `userdata-qemu.img` nella zona di memoria dell'AVD utilizzato.
	`-initdata <filepath>`	Quando si azzera l'immagine dei dati utente (attraverso `-wipe-data`), copia il contenuto di questo file nella nuova immagine del disco dei dati utente. Di default, l'emulatore copia `<system>/userdata.img`. Opzionalmente, è possibile specificare un percorso relativo alla directory di lavoro corrente.
	`-wipe-data`	Ripristina l'immagine del disco corrente dei dati utenti (cioè il file specificato da `-datadir` e `-data`, o il file di default). L'emulatore cancella tutti i dati dal file immagine dei dati utente, quindi copia il contenuto del file del dato `-inidata` nel file dell'immagine prima dei iniziare.

Categoria	Comando	Descrizione
	`-nocache`	Avvia l’emulatore senza una partizione di cache.
	`-ramdisk <filepath>`	Utilizza `<filepath>` come immagine della ramdisk. Il valore di default è `<system>/ramdisk.img`. Opzionalmente, è possibile specificare un percorso relativo alla directory di lavoro corrente.
	`-sdcard <filepath>`	Utilizza `<filepath>` come immagine della scheda SD. Il valore di default è `<system>/sdcard.img`. Opzionalmente, è possibile specificare un percorso relativo alla directory di lavoro corrente.
Debug	`-verbose`	Abilita un output dettagliato. Equivalente a `-debug-init`. È possibile definire le opzioni di output predefinite, utilizzate dalle istanze dell’emulatore, nella variabile di ambiente Android ANDROID_VERBOSE. Per fare ciò, si definiscono le opzioni che si vogliono utilizzare in un elenco separato da virgole, specificando solo la radice di ogni opzione: `-debug-<tags>`. Ecco un esempio che mostra ANDROID_VERBOSE definita con le opzioni `-debug-init` e `-debug-modem`: `ANDROID_VERBOSE=init,modem`
	`-debug <tags>`	Abilita/disabilita i messaggi di debug per il tag di debug specificato. `<tags>` è un elenco separato da spazi/virgole/colonne di nomi di componenti di debug.
	`-debug-<tag>`	Abilita/disabilita i messaggi di debug per il tag di debug specificato.

Categoria	Comando	Descrizione
	`-debug-no-<tag>`	Disattiva i messaggi di debug per il tag di debug specificato.
	`-logcat <logtags>`	Abilita l'output di logcat con i tag dati. Se la variabile di ambiente ANDROID_ LOG_TAGS è definita e non vuota, il suo valore sarà utilizzato per abilitare l'output di logcat per impostazione di default.
	`-shell`	Crea una console shell di root sul terminale corrente. È possibile utilizzare anche se il demone adb nel sistema emulato è rotto. Premendo `CTRL+C` dalla shell, viene fermato l'emulatore invece che la shell.
	`-shell-serial <device>`	Abilita la shell di root (come in `-shell`) e specifica il dispositivo QEMU da utilizzare per la comunicazione con la shell. `<device>` deve essere un dispositivo di tipo QEMU device type. Vedi la documentazione [49] per "`-serial dev`" per un elenco dei tipi di dispositivo. Ecco alcuni esempi: • `-shell-serial stdio` è identico a `-shell` • `-shell-serial tcp::4444, server, nowait` consente di comunicare con una shell sulla porta TCP 4444 • `-shell-serial fdpair:3:6` consente ad un processo padre di comunicare con una shell utilizzando fds 3 (in) e 6 (out) • `-shell-serial fdpair:0:1` utilizza il normale stdin e stdout fds.
	`-show-kernel <name>`	Visualizza i messaggi del kernel.
	`-trace <name>`	Abilita il profiling del codice (premere F9 per iniziare), scritto in un file specificato.
Media	`-audio <backend>`	Utilizza il backend audio specificato.

Categoria	Comando	Descrizione
	`-audio-in <backend>`	Utilizza il backend audio-input specificato.
	`-audio-out <backend>`	Utilizza il backend audio-output specificato.
	`-noaudio`	Disabilita il supporto audio nell’istanza dell’emulatore corrente.
	`-radio <device>`	Reindirizza l’interfaccia radio modem a un dispositivo host.
	`-useaudio`	Abilita il supporto audio nell’istanza dell’emulatore corrente. Abilitato per default.
Network	`-dns-server <servers>`	Utilizza i server DNS specificati. Il valore di `<servers>` deve essere un elenco separato da virgole di un massimo di 4 nomi di server DNS o indirizzi IP.

	`-http-proxy <proxy>`	Effettua tutte le connessioni TCP attraverso un proxy HTTP/HTTPS specifico.
		Il valere di `<proxy>` può essere uno dei seguenti:
		`http://<server>:<port>`
		`http://<username>:<password>@<server>:<port>`
		Il prefisso `http://` può essere omesso. Se il comando `-http-proxy <proxy>` non viene fornito, l’emulatore cerca la variabile di ambiente `http_proxy` e automaticamente utilizza qualsiasi valore corrispondente al formato `<proxy>` descritto in precedenza.

	`-netdelay <delay>`	Imposta l’emulazione della latenza di rete a `<delay>`. Il valore di default è `none`.
	`-netspeed <speed>`	Imposta l’emulazione della velocità di rete a `<speed>`. Il valore di default è `full`.

Categoria	Comando	Descrizione
	`-netfast`	Scorcaitoia per `-netspeed full -netdelay none`
	`-port <port>`	Imposta il numero di porta della console per questa istanza dell'emulatore a `<port>`. Il numero di porta della console deve essere un numero intero pari tra 5554 e 5584, inclusi. `<port>+1` deve essere libero e sarà riservato ad ADB.

	`-report-console <socket>`	Riporta la porta della console assegnata per questa istanza dell'emulatore ad una terza parte remota prima di avviare l'emulazione.
		
		`<socket>` deve utilizzare uno di questi formati:
		`tcp:<port>[,server][,max=<seconds>]`
		`unix:<port>[,server][,max=<seconds>]`

Sistema	`-cpu-delay <delay>`	Rallenta la velocità della CPU emulata di `<delay>`. I valori supportati per `<delay>` sono numeri interi tra 0 e 1000.
	`-gps <device>`	Reindirizza il GPS NMEA al dispositivo. Utilizzare questo commando per emulare un GPS compatibile con NMEA collegato ad un dispositivo esterno o a un socket. Il formato di `<device>` deve essere una specificazione QEMU del dispositivo seriale. Vedi la documentazione [49] per "`-serial` *`dev`*".
	`-qemu`	Passa argomenti al software emulatore qemu. Quando si utilizza questa opzione, assicurarsi che sia l'ultima opzione specificata, dal momento che tutte le opzioni successive sono interpretate come opzioni specifiche qemu.

Categoria	Comando	Descrizione

	`-qemu -enable-kvm`	Abilita l’accelerazione KVM della macchina virtuale dell’emulatore.
		
		Questa opzione è efficace solo quando il sistema è impostato per utilizzare l’accelerazione KVM della macchina virtuale. È possibile specificare una dimensione della memoria (`-m <size>`) per la macchina virtuale, che deve corrispondere alla dimensione della memoria dell’emulatore:
		`-qemu -m 512 -enable-kvm`
		`-qemu -m 1024 -enable-kvm`

	`-qemu -h`	Visualizza l’help di qemu.
	`-nojni`	Disabilita i controlli JNI nella Dalvik runtime.
	`-gpu on`	Attiva l’accelerazione grafica per l’emulatore. Questa opzione è disponibile solo per gli emulatori che utilizzano un’immagine di sistema con API Level 15, revision 3 e superiori.
	`-radio <device>`	Reindirizza la modalità radio al dispositivo specificato. Il format di `<device>` deve essere una specificazione QEMU del dispositivo seriale. Vedi la documentazione [49] per “`-serial dev`”.
	`-timezone <timezone>`	Imposta il fuso orario per il dispositivo emulato a `<timezone>`, invece che al fuso orario dell’host. `<timezone>` deve essere specificato nel formato zoneinfo. Per esempio: “America/Los_Angeles” “Europe/Paris”
	`-version`	Visualizza il numero di versione dell’emulatore.

Categoria	Comando	Descrizione
UI	`-dpi-device <dpi>`	Ridimensiona la risoluzione dell'emulatore in modo che corrisponda alle dimensioni dello schermo di un dispositivo fisico. Il valore di default è 165.
	`-no-boot-anim`	Disabilita l'animazione di boot durante l'avvio dell'emulatore. Disabilitare l'animazione di boot può velocizzare il tempo di avvio per l'emulatore.
	`-no-window`	Disabilita la visualizzazione della finestra grafica dell'emulatore.
	`-scale <scale>`	Ridimensiona la finestra dell'emulatore. `<scale>` è un numero tra 0.1 and 3 che rappresenta il fattore di scala desiderato. È inoltre possibile specificare la scala come un valore DPI se si aggiunge il suffisso "dpi" al valore della scala. Un valore "auto" dice all'emulatore di selezionare la migliore dimensione della finestra.
	`-raw-keys`	Disabilita la tastiera Unicode reverse-mapping.
	`-noskin`	Non viene utilizzata alcuna skin dell'emulatore.
	`-keyset <file>`	Utilizza il file keyset specificato invece di quello predefinito. Il file keyset definisce l'elenco delle combinazioni di tasti tra l'emulatore e la tastiera dell'host.
	`-onion <image>`	Utilizza l'immagine overlay sullo schermo. Nessun supporto per JPEG. Solo PNG è supportao.
	`-onion-alpha <percent>`	Specifica il valore della traslucenza della skin (in percentuale). Il valore di default è 50.

Categoria	Comando	Descrizione
	`-onion-rotation <position>`	Specifica la rotazione della skin. `<position>` deve essere uno dei valori 0, 1, 2, 3.
	`-skin <skinID>`	Queste opzioni dell’emulatore sono deprecate. Utilizzare gli AVD per impostare le opzioni della skin, piuttosto che utilizzare questa opzione dell’emulatore. L’utilizzo di questa opzione può provocare inaspettati e in alcuni casi fuorvianti risultati, poiché la densità con cui si rende la skin può non essere definita. Gli AVD consentono di associare ogni skin con una densità predefinita e non tener conto di quella predefinita se necessario.
	`-skindir <dir>`	
Help	`-help`	Stampa un elenco di tutte le opzioni dell’emulatore.
	`-help-all`	Stampa l’help per tutte le opzioni di avvio.
	`-help-<option>`	Stampa l’help per una specifica opzione di avvio.
	`-help-debug-tags`	Stampa un elenco di tutti i tag per `-debug <tags>`.
	`-help-disk-images`	Stampa l’help per l’utilizzo delle immagini del disco dell’emulatore.
	`-help-environment`	Stampa l’help per le variabili di ambiente dell’emulatore.
	`-help-keys`	Stampa la mappatura corrente delle tasti.
	`-help-keyset-file`	Stampa l’help per definire un file di mappatura delle chiavi personalizzato.
	`-help-virtual-device`	Stampa l’help per l’utilizzo dell’Android Virtual Device.

È possibile eseguire un'applicazione su una singola istanza dell'emulatore o, a seconda delle esigenze, è possibile avviare istanze multiple dell'emulatore (ognuno con la propria configurazione AVD e area di memorizzazione per i dati utente, per la scheda SD, e così via) ed eseguire un'applicazione in più di un dispositivo emulato.

Per avviare un'istanza dell'emulatore dalla riga di comando, navigare fino alla directory `tools/` dell'SDK e digitare il seguente comando:

```
emulator -avd <avd_name> [<option>]
```

Questo inizializza l'emulatore, carica una configurazione AVD e visualizza la finestra dell'emulatore.

Per interrompere un'istanza dell'emulatore, è sufficiente chiudere la finestra dell'emulatore.

D.2 La console dell'emulatore

Ogni istanza dell'emulatore in esecuzione fornisce una console che permette di interrogare e controllare l'ambiente del dispositivo emulato. Ad esempio, è possibile utilizzare la console per gestire il reindirizzamento delle porte, le caratteristiche della rete, ed gli eventi di telefonia mentre un'applicazione è in esecuzione sull'emulatore.

Per connettersi in qualsiasi momento alla console di una qualsiasi istanza dell'emulatore in esecuzione, utilizzare questo comando:

```
telnet localhost <console-port>
```

Un'istanza dell'emulatore occupa una coppia di porte adiacenti: una porta di console ed una porta adb. I numeri di porta differiscono di 1, con la porta adb avente il numero più alto di porta. La console della prima istanza dell'emulatore in esecuzione su una determinata macchina utilizza la porta di console 5554 e la porta adb 5555. Istanze successive utilizzano numeri di porta incrementati di due: ad esempio, 5556/5557, 5558/5559, e così via. È possibile eseguire fino a 16 istanze simultanee dell'emulatore.

Per connettersi alla console dell'emulatore, è necessario specificare una porta di console valida. Se più istanze dell'emulatore sono in esecuzione, è necessario determinare la porta di console dell'istanza dell'emulatore che si desidera connettersi. È possibile trovare la porta di console dell'istanza elencata nel titolo della finestra dell'istanza.

In alternativa, è possibile utilizzare il comando `adb devices`, che stampa un elenco delle istanze dell'emulatore in esecuzione e i loro numeri di porta di console (vedi

paragrafo E.2 dell’appendice E). Si noti che l’emulatore rimane in ascolto per connessioni su porte in un range da 5554 a 5587 e accetta connessioni solo da localhost.

Una volta collegati alla console, è possibile digitare il comando `help [command]` per visualizzare un elenco dei comandi della console.

Per uscire dalla sessione della console, utilizzare il comando `quit` o `exit` .

D.2.1 Reindirizzamento delle porte

È possibile utilizzare la console per aggiungere e rimuovere il reindirizzamento delle porte, mentre l’emulatore è in esecuzione. Dopo essersi connessi alla console, è possibile gestire il reindirizzamento di porta inserendo il seguente comando:

```
redir <list|add|del>
```

Il comando `redir` supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	**Descrizione**
`list`	Elenca il reindirizzamento di porta corrente.
`add <protocol>:<host-port>: <guest-port>`	Aggiunge un reindirizzamento di porta. • `<protocol>` deve essere “tcp” o “udp” • `<host-port>` è il numero di porta da aprire sull’host • `<guest-port>` è il numero di porta per instradare i dati sull’emulatore/dispositivo
`del <protocol>:<host-port>`	Elimina un reindirizzamento di porta. Il significato di `<protocol>` e `<protocol>` sono elencati nella riga precedente.

D.2.2 Emulazione della geolocalizzazione

È possibile utilizzare la console per impostare la posizione geografica segnalata alle applicazioni in esecuzione all’interno di un emulatore. Utilizzare il comando `geo` per inviare un semplice fix GPS all’emulatore, con o senza formattazione NMEA 1083:

```
geo <fix|nmea>
```

È possibile eseguire il comando `geo` non appena un'istanza dell'emulatore è in esecuzione.

Il comando `geo` supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	**Descrizione**
`fix <longitude> <latitude> [<altitude>]`	Invia un semplice fix GPS all'istanza dell'emulatore. Specificare longitudine e latitudine in gradi decimali. Specificare l'altitudine in metri.
`nmea <sentence>`	Invia una frase NMEA 0183 al dispositivo emulato, come se fosse inviata da un modem GPS emulato. `<sentence>` deve essere con "$GPtcp". Solo le frasi "$GPGGA" e "$GPRCM" sono attualmente supportate.

D.2.3 Caratteristiche dell'alimentazione del dispositivo

Il comando `power` controlla lo stato di alimentazione riportato dall'emulatore per le applicazioni. La sintassi per questo comando è la seguente:

```
power <display|ac|status|present|health|capacity>
```

Il comando `power` supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	**Descrizione**
`display`	Visualizza lo stato della batteria e di carica.
`ac <on|off>`	Imposta lo stato di carica AC a on o off.
`status <unknown|charging| discharging|not-charging|full>`	Cambia lo stato della batteria come specificato.
`present <true|false>`	Imposta lo stato di presenza della batteria.
`health <unknown|good|overheat| dead|overvoltage|failure>`	Imposta lo stato di salute della batteria.
`power capacity <percent>`	Imposta lo stato di capacità rimanente della batteria (0-100).

D.2.4 Emulazione degli eventi hardware

Il comando `event` invia eventi hardware per l’emulatore. La sintassi per questo comando è la seguente:

```
event <send|types|codes|text>
```

Il comando `event` supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	**Descrizione**
`send <type>:<code>:<value> […]`	Invia uno o più eventi al kernel di Android. È possibile utilizzare nomi testuali o numeri interi per `<type>` e `<value>`.
`types`	Elenca tutti gli alias della stringa `<type>` supportati dai sottocomandi di `event`.
`codes <type>`	Elenca tutti gli alias della stringa `<codes>` supportati dai sottocomandi di `event` per lo specificato `<type>`.
`event text <message>`	Simula i tasti premuti da inviare la specificata stringa di caratteri come un messaggio. Il messaggio deve essere una stringa UTF-8.

D.2.5 Emulazione della rete

È possibile utilizzare la console per controllare lo stato della rete, il ritardo corrente e le caratteristiche della velocità. Per fare ciò, occorre utilizzare il comando `network status`.

L’emulatore permette di simulare diversi livelli di latenza della rete, in modo da poter testare l’applicazione in un ambiente più tipico delle condizioni concrete in cui verrà eseguito. È possibile impostare un livello o un intervallo di latenza all’avvio dell’emulatore o è possibile utilizzare la console per modificare la latenza, mentre l’applicazione è in esecuzione nell’emulatore.

Per impostare la latenza all’avvio dell'emulatore, occorre utilizzare il comando `emulator` con l’opzione `-netdelay` e con un valore `<delay>` supportato, come indicato nella tabella sottostante. Ecco alcuni esempi:

```
emulator -netdelay gprs
emulator -netdelay 40 100
```

Per apportare modifiche al ritardo della rete mentre l'emulatore è in esecuzione, occorre invece collegarsi alla console e utilizzare il comando `netdelay` con un valore `<delay>` supportato:

```
network delay gprs
```

Il formato di `<delay>` è uno dei seguenti (i numeri sono millisecondi):

Valore	**Descrizione**
`gprs`	GPRS (min 150, max 550)
`edge`	EDGE/EGPRS (min 85, max 400)
`umts`	UMTS/3G (min 35, max 200)
`none`	Nessuna latenza (min 0, max 0)
`<num>`	Emula un'esatta latenza (in millisecondi).
`<min>:<max>`	Emula un specifico intervallo di latenza (in millisecondi).

L'emulatore consente inoltre di simulare varie velocità di trasferimento della rete. È possibile impostare una velocità o un intervallo di trasferimento all'avvio dell'emulatore oppure è possibile utilizzare la console per modificare la velocità, mentre un'applicazione è in esecuzione nell'emulatore.

Per impostare la velocità della rete all'avvio dell'emulatore, occorre utilizzare il comando `emulator` con l'opzione `-netspeed` e con un valore `<speed>` supportato, come indicato nella tabella sottostante. Ecco alcuni esempi:

```
emulator -netspeed gsm
emulator -netspeed 14.4 80
```

Per apportare modifiche alla velocità della rete, mentre l'emulatore è in esecuzione, occorre invece collegarsi alla console e utilizzare il comando `netspeed` con un valore `<speed>` supportato:

```
network speed 14.4 80
```

Il formato di `<speed>` è uno dei seguenti (i numeri sono kilobit/sec):

Valore	**Descrizione**
`gsm`	GSM (up: 14.4, down: 14.4)
`hscsd`	HSCSD (up: 14.4, down: 43.2)
`gprs`	GPRS (up: 40.0, down: 80.0)
`edge`	EDGE/EGPRS (up: 118.4, down: 236.8)
`umts`	UMTS/3G (up: 128.0, down: 1920.0)
`hsdpa`	HSDPA (up: 348.0, down: 14400.0)
`full`	Nessun limite (up: 0.0, down: 0.0)
`<num>`	Imposta un’esatta velocità utilizzata sia per l’upload che per il download.
`<up>:<down>`	Imposta esatte velocità per l’upload e il download.

D.2.6 Emulazione degli SMS

La console dell’emulatore di Android consente di generare un messaggio SMS e dirigerlo ad un’istanza dell’emulatore. Una volta connesso ad un’istanza dell’emulatore, è possibile generare un SMS in arrivo emulato utilizzando il seguente comando:

```
sms send <senderPhoneNumber> <textmessage>
```

dove `<senderPhoneNumber>` contiene una stringa numerica arbitraria.

La console inoltra il messaggio SMS al framework di Android, che passa attraverso un’applicazione che gestisce quel tipo di messaggio.

D.2.7 Emulazione della telefonia

L’emulatore di Android include il suo proprio modem GSM emulato che permette di simulare le funzioni di telefonia nell’emulatore. Ad esempio, è possibile simulare le chiamate telefoniche in entrata, di stabilire connessioni dati e di porvi fine.

Il sistema Android gestisce le chiamate simulate esattamente come sarebbe per le chiamate reali. L'emulatore non supporta l'audio delle chiamate.

È possibile utilizzare il comando `gsm` per accedere alle funzioni di telefonia dell'emulatore. La sintassi per questo comando è la seguente:

```
gsm <call|accept|busy|cancel|data|hold|list|voice|status>
```

Il comando `gsm` supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	**Descrizione**
`call <phonenumber>`	Simula una chiamata in entrata da `<phonenumber>`.
`accept <phonenumber>`	Accetta una chiamata in entrata da `<phonenumber>` e cambia lo stato della chiamata in "active". È possibile cambiare lo stato di una chiamata a "active" solo se il suo stato corrente è "waiting" o "held".
`busy <phonenumber>`	Chiude una chiamata in uscita a `<phonenumber>` e cambia lo stato della chiamata in "busy". È possibile cambiare lo stato di una chiamata a "busy" solo se il suo stato corrente è "waiting".
`cancel <phonenumber>`	Termina una chiamata in entrata o in uscita da/a `<phonenumber>`.
`data <state>`	Cambia lo stato della connessione dati GPRS a `<state>`. Valori `<state>` supportati sono: • `unregistered`: nessuna rete disponibile; • `home`: sulla rete locale, non roaming; • `roaming`: sulla rete roaming; • `searching`: ricerca reti; • `denied`: solo chiamate di emergenza; • `off`: come `unregistered`; • `on`: come `home`.
`hold`	Cambia lo stato di una chiamata in "held". È possibile cambiare lo stato di una chiamata a "held" solo se il suo stato corrente è "active" o "waiting".
`list`	Elenca tutte le chiamate in entrata e in uscita e il loro stato.

Sottocomando	Descrizione
`voice <state>`	Cambia lo stato della connessione voce GPRS a `<state>`. Valori `<state>` supportati sono: • `unregistered`: nessuna rete disponibile; • `home`: sulla rete locale, non roaming; • `roaming`: sulla rete roaming; • `searching`: ricerca reti; • `denied`: solo chiamate di emergenza; • `off`: come `unregistered`; • `on`: come `home`.
`status`	Riporta lo stato voce/dati GMS corrente. I valori sono quelli descritti per i comandi `voice` e `data`.

D.2.8 Stato della Virtual Machine

È possibile utilizzare il comando `vm` per controllare la Virtual Machine su un’istanza dell’emulatore. La sintassi per questo comando è la seguente:

```
vm <start|stop|status>
```

Il comando `vm` supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	Descrizione
`start`	Avvia la Virtual Machine sull’istanza.
`stop`	Arresta la Virtual Machine sull’istanza.
`status`	Visualizza lo stato corrente della Virtual Machine (in esecuzione o ferma).

D.2.9 La finestra dell’emulatore

È possibile utilizzare il comando `window` per gestire la finestra dell’emulatore. La sintassi per questo comando è la seguente:

```
window <scale>
```

Il comando `window` supporta i sottocomandi elencati nella seguente tabella.

Sottocomando	**Descrizione**
`scale <scale>`	Ridimensiona la finestra dell'emulatore. `<scale>` è un numero tra 0.1 and 3 che rappresenta il fattore di scala desiderato. È inoltre possibile specificare la scala come un valore DPI se si aggiunge il suffisso "dpi" al valore della scala. Un valore "auto" dice all'emulatore di selezionare la migliore dimensione della finestra.

D.2.10 Chiusura di un'istanza dell'emulatore

È possibile interrompere un'istanza dell'emulatore tramite la console, utilizzando il comando `kill`.

D.2.11 Limitazioni dell'emulatore

Le limitazioni funzionali dell'emulatore comprendono:

- nessun supporto per effettuare o ricevere telefonate reali. È comunque possibile simulare chiamate (effettuate e ricevute) attraverso la console dell'emulatore;
- nessun supporto per le connessioni USB;
- nessun supporto per le cuffie collegate al dispositivo;
- nessun supporto per determinare lo stato di connessione della rete;
- nessun supporto per determinare il livello di carica della batteria e lo stato di carica AC;
- nessun supporto per determinare l'inserimento/espulsione di una scheda SD;
- nessun supporto per il Bluetooth.

Appendice E

Android Debug Bridge

E.1 Utilizzo dei comandi adb

È possibile eseguire i comandi `adb` dalla riga di comando o da uno script. L'utilizzo è il seguente:

```
adb [-d|-e|-s <serialNumber>] <command>
```

Quando si esegue un comando, il programma invoca un client adb. Il client non è specificamente associato a qualsiasi istanza dell'emulatore, quindi se più emulatori o dispositivi sono in esecuzione, è necessario utilizzare l'opzione `-d` per specificare l'istanza target a cui il comando deve essere diretto.

E.2 Interrogazione per istanze dell'emulatore/dispositivo

Prima di eseguire comandi `adb`, è utile sapere quali istanze dell'emulatore/dispositivo sono collegati al server adb. È possibile generare un elenco degli emulatori/dispositivi collegati tramite il comando `devices`:

```
adb devices
```

In risposta, adb stampa queste informazioni di stato per ogni istanza:

- numero di serie: una stringa creata da adb per identificare univocamente un'istanza dell'emulatore/dispositivo tramite il suo numero di porta di console. Il

formato del numero di serie è `<type>-<consolePort>` . Ecco un esempio di numero di serie: `emulator-5554`

- stato: lo stato della connessione dell'istanza. Due stati sono supportati:
  - `offline`: l'istanza non è collegata a adb o non risponde;
  - `device`: l'istanza è collegata al server adb. Si noti che questo stato non implica che il sistema Android è completamente avviato e operativo, dal momento che l'istanza si connette a adb mentre il sistema è ancora in avvio. Tuttavia, dopo boot-up, questo è lo stato normale di funzionamento di un'istanza dell'emulatore/dispositivo.

L'output per ogni istanza viene formattata in questo modo:

```
[SerialNumber] [state]
```

Ecco un esempio che mostra il comando `devices` e il suo output:

```
$ adb devices
List of devices attached
emulator-5554 device
emulator-5556 device
emulator-5558 device
```

Se non c'è un emulatore/dispositivo in esecuzione, adb restituisce `no device`.

E.3 Dirigere comandi ad una specifica istanza dell'emulatore/dispositivo

Se più istanze dell'emulatore/dispositivo sono in esecuzione, è necessario specificare un'istanza di destinazione per l'esecuzione dei comandi `adb`. Per farlo occorre utilizzare l'opzione `-s` nei comandi. L'utilizzo per l'opzione `-s` è:

```
adb -s <serialNumber> <command>
```

Come si vede, è necessario specificare l'istanza di destinazione per un comando usando il suo numero di serie assegnato. È possibile utilizzare il comando `devices` per ottenere i numeri di serie delle istanze degli emulatori/ dispositivi in esecuzione.
Ecco un esempio che mostra l'utilizzo dell'opzione `-s`:

```
adb -s emulator-5556 install helloWorld.apk
```

Si noti che, se si esegue un comando senza specificare un'istanza dell'emulatore/dispositivo di destinazione utilizzando `-s`, adb genera un errore.

E.4 Elenco dei comandi adb

La seguente tabella elenca tutti i comandi `adb` supportati e spiega il loro significato e utilizzo.

Categoria	Comando	Descrizione
Opzioni	`-d`	Indirizza un commando adb all'unico dispositivo USB collegato. Restituisce un errore se più di un dispositivo USB è collegato.
	`-e`	Indirizza un commando adb all'unica istanza dell'emulatore in esecuzione. Restituisce un errore se più di un'istanza dell'emulatore è in esecuzione.
	`-s <serialNumber>`	Indirizza un commando adb a una specifica istanza dell'emulatore/dispositivo, identificata dal numero di serie assegnatole (ad esempio "emulator-5556"). Se non specificato, adb genera un errore.
Generale	`devices`	Stampa un elenco di tutte le istanze dell'emulatore/dispositivo collegate.
	`help`	Stampa un elenco dei comandi adb supportati.
	`version`	Stampa il numero di versione di adb.
Data	`install <path-to-apk>`	Indirizza un'applicazione Android (specificata attraverso un percorso a un file `.apk`) al file dei dati di un emulatore/dispositivo.
	`pull <remote> <local>`	Copia un file specifico da un'istanza dell'emulatore/dispositivo alla propria macchina.

Categoria	Comando	Descrizione
	`push <local> <remote>`	Copia un file specifico dalla propria macchina ad un'istanza dell'emulatore/dispositivo.
Debug	`logcat [<option>] [<filter-specs>]`	Stampa sullo schermo il log dei dati.
	`bugreport`	Stampa sullo schermo i dati `dumpsys`, `dumpstate` e `logcat`, ai fini della segnalazione dei bug.

	`jdwp`	Stampa un elenco dei processi JDWP disponibili su un dato dispositivo.
		
		È possibile utilizzare il comando `forward jdwp:<pid>` per connettersi ad un specifico processo JDWP.
		Per esempio:
		`adb forward tcp:8000 jdwp:472`
		`jdb -attach localhost:8000`

Porte e Networking	`forward <local> <remote>`	Inoltra connessioni socket da una specifica porta locale a una specifica porta remota sull'istanza dell'emulatore/dispositivo. Le porte possono utilizzare questi schemi: • `tcp:<portnum>` • `local:<UNIX domain socket name>` • `dev:<character device name>` • `jdwp:<pid>`
	`ppp <tty> [parm]…`	Esegue PPP attraverso l'USB. • `<tty>` - il tty per lo stream PPP. Per esempio: `dev:/dev/omap_csmi_ttyl` • `[parm]…` - zero o più opzioni PPP/PPPD, come `defaultroute`, `local`, `notty`, ecc. Si noti che non occorre avviare automaticamente una connessione PPP.
Scripting	`get-serialno`	Stampa la stringa relativa al numero di serie dell'istanza adb.

Categoria	Comando	Descrizione
	`get-state`	Stampa lo stato adb di un'istanza dell'emulatore/dispositivo.

	`wait-for-device`	Blocca l'esecuzione fin quando il dispositivo è online, cioè fin quando lo stato dell'istanza è `device`.
		
		È possibile anteporre questo commando ad altri comandi adb, in tal caso adb attenderà fin quando l'istanza dell'emulatore/dispositivo è connessa prima di eseguire gli altri comandi. Ecco un esempio:
		`adb wait-for-device shell getprop`
		
		Si noti che questo commando non causa che adb debba attendere fin quando l'intero sistema sia completamente avviato. Per questo motivo, non lo si dovrebbe anteporre ad altri comandi che richiedano un sistema completamente avviato. Per esempio, `install` richiede il gestore dei pacchetti Android, che è disponibile solo dopo che il sistema è completamente avviato. Un comando come
		`adb wait-for-device install <app>.apk`
		avrebbe eseguito il comando `install` non appena l'istanza dell'emulatore o del dispositivo fosse collegata al server adb, ma prima che il sistema Android fosse completamente avviato, cosicché avrebbe generato un errore.

Server	`star-server`	Controlla se il processo server adb è in esecuzione e lo avvia se non lo è.
	`kill-server`	Termina il processo server adb.
Shell	`shell`	Avvia una shell remota nell'istanza target dell'emulatore/dispositivo.
	`shell [<shellCommand>]`	Esegue un comando shell nell'istanza target dell'emulatore/dispositivo e poi esce dalla shell remota.

E.5 Utilizzo dei comandi shell

Adb fornisce una shell che è possibile utilizzare per eseguire una serie di comandi su un emulatore o un dispositivo. I comandi binari sono memorizzati nel file system dell'emulatore o del dispositivo, in questa posizione:

```
/system/bin/…
```

È possibile utilizzare il comando `shell` per eseguire comandi, con o senza accesso alla shell remota adb sull'emulatore/dispositivo.

Per eseguire un comando senza accedere alla shell remota, utilizzare il comando `shell` in questo modo:

```
adb [-d|-e|-s {<serialNumber>}] shell <shellCommand>
```

Per accedere ad una shell remota su un emulatore/dispositivo, utilizzare il comando `shell` in questo modo:

```
adb [-d|-e|-s {<serialNumber>}] shell
```

Quando si è pronti ad uscire dalla shell remota, utilizzare `CTRL+D` o `exit` per terminare la sessione di shell.

E.6 Comandi shell

La seguente tabella elenca alcuni dei comandi `shell` adb disponibili. Per un elenco completo dei comandi e dei programmi, avviare un'istanza dell'emulatore e utilizzare il comando `adb -help`.

Comando Shell	**Descrizione**
`dumpsys`	Stampa i dati di sistema sullo schermo.
`dumpstate`	Stampa lo stato ad un file.
`logcat [<option>]…[<filter-specs>]`	Abilita il radio logging e stampa l'output sullo schermo.
`dmesg`	Stampa i messaggi di debug del kernel sullo schermo.
`start`	Avvia (riavvia) un'istanza dell'emulatore/dispositivo.

Comando Shell	Descrizione
`stop`	Ferma l'esecuzione di un'istanza dell'emulatore/dispositivo.

E.7 Comando logcat

Il sistema di log di Android fornisce un meccanismo per raccogliere e visualizzare l'output di debug del sistema. I log di varie applicazioni e di parti del sistema sono raccolti in una serie di buffer circolari che possono essere visualizzati e filtrati dal comando `logcat`.

È possibile utilizzare il comando `logcat` per visualizzare e seguire i contenuti dei log dei buffer del sistema. L'utilizzo generale è:

```
[adb] logcat [<option>]…[<filter-spec>]…
```

È possibile utilizzare il comando `logcat` dalla propria macchina o da una shell adb remota in un'istanza dell'emulatore/dispositivo. Per visualizzare l'output del log dalla propria macchina si utilizza:

```
$ adb logcat
```

mentre da una shell adb remota si utilizza:

```
# logcat
```

Ogni messaggio di log di Android ha un *tag* e una *priorità* ad esso associati. Il tag di un messaggio di log è una breve stringa che indica il componente di sistema da cui il messaggio ha origine (ad esempio, "View" per il sistema di visualizzazione). La priorità è uno dei valori dei seguenti caratteri, ordinati dal più basso al più alto livello di priorità:

- `V`: verbose (priorità più bassa)
- `D`: debug
- `I`: info (priorità di default)
- `W`: avviso
- `E`: errore
- `F`: fatal
- `S`: silent (massima priorità, su cui nulla è mai stampato)

È possibile ottenere un elenco dei tag utilizzati nel sistema, insieme alle priorità, eseguendo `logcat` e osservando le prime due colonne di ogni messaggio, dato come `<priority>/<tag>`.

Ecco un esempio di output logcat che mostra che il messaggio si riferisce al livello di priorità "I" e tag "ActivityManager":

```
I/ActivityManager (585): Starting activity: Intent {action=
  android.intent.action…}
```

Per ridurre l'output del log a un livello gestibile, è possibile limitare l'output del log utilizzando *espressioni filtro*. Le espressioni filtro consentono di indicare al sistema le combinazioni tag-priorità che sono di interesse. Il sistema sopprime altri messaggi per i tag specificati.

Un'espressione filtro segue questo formato `tag:priority…` , dove `tag` indica il tag di interesse e `priority` indica il livello minimo di priorità da segnalare per quel tag. I messaggi per quel tag alla specifica o superiore priorità, vengono scritti nel log. È possibile specificare un numero qualsiasi di specifiche `tag:priority` in una sola espressione filtro. La serie di specifiche è delimitato da uno spazio. L'output predefinito è di mostrare tutti i messaggi di log con priorità Info (`*:I`).

Ecco un esempio di un'espressione filtro che elimina tutti i messaggi di log ad eccezione di quelli con il tag "ActivityManager", con la priorità "info" o superiore, e tutti i messaggi di log con tag "MyApp", con priorità "debug" o superiore:

```
adb logcat ActivityManager:I MyApp:D *:S
```

La specifica `*:S` imposta il livello di priorità per tutti i tag a "silent", garantendo in tal modo che solo i messaggi di log con "View" e "MyApp" vengano visualizzati. L'utilizzo di `*:S` è un ottimo modo per garantire che l'output dei log venga limitato ai filtri che sono stati esplicitamente specificati.

La seguente espressione visualizza tutti i messaggi di log con livello di priorità "warning" e superiore, su tutti i tag:

```
adb logcat *:W
```

I messaggi di log contengono una serie di campi di metadati, oltre al tag e la priorità. È possibile modificare il formato di output per i messaggi in modo da visualizzare uno specifico campo di metadati. Per fare ciò, si utilizza l'opzione `–v` e si specifica uno dei formati di output supportati elencati di seguito:

- `brief`: visualizza la priorità/tag e il PID del processo che genera il messaggio (formato di default);
- `process`: visualizza solamente il PID;

- `tag`: visualizza solamente la priorità/tag;
- `raw`: visualizza il messaggio di log grezzo, senza altri campi di metadati.
- `time`: visualizza la data, l'ora invocazione, la priorità/tag e PID del processo che genera il messaggio;
- `threadtime`: visualizza la data, l'ora invocazione, la priorità/tag, e il PID e il TID del thread che genera il messaggio;
- `long`: visualizza tutti i campi dei metadati e separa i messaggi con alcune righe vuote.

Quando si avvia `logcat`, è possibile specificare il formato di output desiderato utilizzando l'opzione `–v`:

```
[adb] logcat [-v <format>]
```

Il sistema di log di Android mantiene più buffer circolari per i messaggi di log, e non tutti i messaggi di log vengono inviati al buffer circolare di default. Per visualizzare i messaggi di log aggiuntivi, è possibile avviare `logcat` con l'opzione `-b`, per richiedere la visualizzazione di un buffer circolare alternativo. L'uso dell'opzione `-b` è:

```
[adb] logcat [-b <buffer>]
```

È possibile visualizzare uno di questi buffer alternativi:

- `radio`: visualizza il buffer che contiene messaggi relativi a radio/telefonia;
- `events`: visualizza il buffer che contiene messaggi relativi ad eventi;
- `main`: visualizza il buffer di log principale (default).

La seguente tabella elenca le opzioni del comando `logcat` e spiega il loro significato e utilizzo.

Opzione	**Descrizione**
`-b <buffer>`	Carica un buffer di log alternativo per la visualizzazione, come `event` o `radio`. Il buffer `main` è usato di default.
`-c`	Cancella l'intero log ed esce.
`-d`	Stampa il log sullo schermo ed esce.
`-f <filename>`	Scrive l'output del messaggio di log in `<filename>`. Il valore di default è `stdout`.

Opzione	Descrizione
`-g`	Stampa la dimensione del buffer di log specificato ed esce.
`-n <count>`	Imposta il numero massimo di log ruotati a `<count>`. Il valore di default è 4. Richiede l'opzione `-r`.
`-r <kbytes>`	Ruota il file di log ogni `<kbytes>` di output. Il valore di default è 16. Richiede l'opzione `-f`.
`-s`	Imposta il `filter spec` di default a silent.
`-v <format>`	Imposta il formato di output per i messaggi di log. Il formato di default è `brief`.

Appendice F

Installazione del plugin ADT su Eclipse

Per prima cosa occorre avere scaricato [50] e installato sulla propria macchina una versione di Eclipse. Si consiglia di utilizzare la versione “Eclipse Classic”. In alternativa si può utilizzare la versione Java o RCP di Eclipse.
Avviare quindi Eclipse e seguire i seguenti passi:

1. dal menù a tendina selezionare **Help -> Install New Software**;
2. cliccare su **Add** nell’angolo in alto a destra;
3. nella finestra di dialogo *Add Repository*, inserire “ADT Plugin” nel campo *Name* e l’URL “https://dl-ssl.google.com/android/eclipse/” nel campo *Location*;
4. cliccare su **OK**.
   Nota: in caso di problemi nell’acquisizione del plugin, provare ad usare “http” invece di “https” nell’URL del campo *Location*;
5. nella finestra di dialogo *Available Software*, selezionare il checkbox accanto a *Developer Tools* e cliccare su **Next**;
6. nella finestra successiva verrà visualizzato un elenco degli strumenti da scaricare. Cliccare su **Next**;
7. leggere e accettare gli accordi di licenza, quindi cliccare su **Finish**.
   Nota: se si riceve un avviso di sicurezza dicendo che l'autenticità e la validità del software non può essere stabilita, cliccare su **OK**;
8. al termine dell’installazione, riavviare Eclipse.

Dopo aver installato ADT e riavviato Eclipse, è necessario specificare il percorso della directory dell'SDK di Android:

1. dal menù a tendina selezionare **Window -> Preferences**;
2. selezionare **Android** dal pannello di sinistra;
3. cliccare su **Browse** dal campo *SDK Location* e individuare la directory relativa all'SDK;
4. cliccare su **Apply**, quindi **OK**.

Appendice G

Guida all'utilizzo di Odin

Odin è un software che permette di flashare firmware, kernel, modem, bootloader ed altro ancora su dispositivi Samsung. Nel seguito verranno descritti i passi per installare un custom firmware (paragrafo G.1) e un custom kernel (paragrafo G.2) sul Galaxy S.

G.1 Flashing di un custom firmware

Le indicazioni descritte nel seguito si riferiscono a come installare un custom firmware costituito da tre pacchetti:

- PDA: contiene il sistema operativo;
- PHONE: contiene il modulo telefonico;
- CSC: contiente le personalizzazioni del firmware in base alla localizzazione del terminale e brand operatore.

Ecco come procedere:

1. assicurarsi di aver installato sulla propria macchina i driver USB del Galaxy S;
2. effettuare il backup dei propri dati tramite *Samsung Kies* [36], dal momento che questa procedura cancellerà tutti i dati presenti sul dispositivo;
3. scaricare Odin [51];

4. scaricare il file PIT 512 (è il file che permette di ripartizionare il telefono) [51], scompattarlo e copiare il file con estensione `.pit` in `C:\`;
5. scaricare il firmware che ci interessa. Solitamente si tratta di un file compresso che contiene i tre pacchetti PDA, PHONE e CSC. Scompattarlo e copiare questi 3 file con estensione `.tar` o `.tar.md5` in `C:\`;
6. effettuare, per sicurezza, un reset completo del telefono[26];
7. spegnere il dispositivo ed estrarre la SIM e la eventuale microSD;
8. mettere il Galaxy S in Download Mode[27]. Sullo schermo verrà visualizzata la seguente immagine:

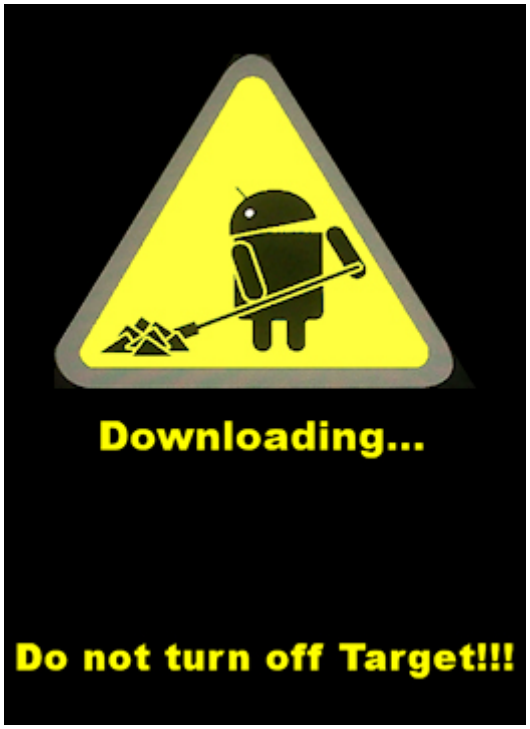

Figura G.1: La schermata di Download Mode.

9. avviare Odin e collegare il dispositivo alla porta USB. Se il terminale è riconosciuto correttamente da Odin, sarà possibile vedere nel primo campo ID:COM il nome della porta COM su sfondo giallo (il numero della porta COM è indifferente). Se così non fosse provare i seguenti tentativi in sequenza:

[26] Entrare nelle impostazioni di sistema di Android e cliccare su **Privacy**. Quindi cliccare su **Ripristina dati di fabbrica** e confermare l'operazione. La procedura avrà inizio e il telefono si riavvierà.

[27] Da dispositivo spento, premere contemporaneamente (rispettando la sequenza) il tasto VOLUME GIÙ + il tasto centrale HOME + il tasto POWER e rilasciare dopo 3 secondi circa.

- chiudere e riavviare Odin;
- scollegare e ricollegare il cavo USB;
- provare a cambiare porta USB;
- ripetere tutta la procedura dal punto 8 in poi;
- provate a reinstallare i driver USB del Galaxy S.

10. configurare Odin per il flash del Galaxy S con il firmware scaricato:
    - attivare la casella **Re-Partition** e lasciare inalterate le altre;
    - cliccare sul pulsante **PIT** e scegliere il file di PIT scaricato in precedenza;
    - cliccare sul pulsante **PDA** e scegliere il file PDA con estensione `.tar` o `.tar.md5` precedentemente copiato in `C:\`;
    - cliccare sul pulsante **PHONE** e scegliere il file PHONE con estensione `.tar` o `.tar.md5` precedentemente copiato in `C:\`;
    - cliccare sul pulsante **CSC** e scegliete il file CSC con estensione `.tar` o `.tar.md5` precedentemente copiato in `C:\`.

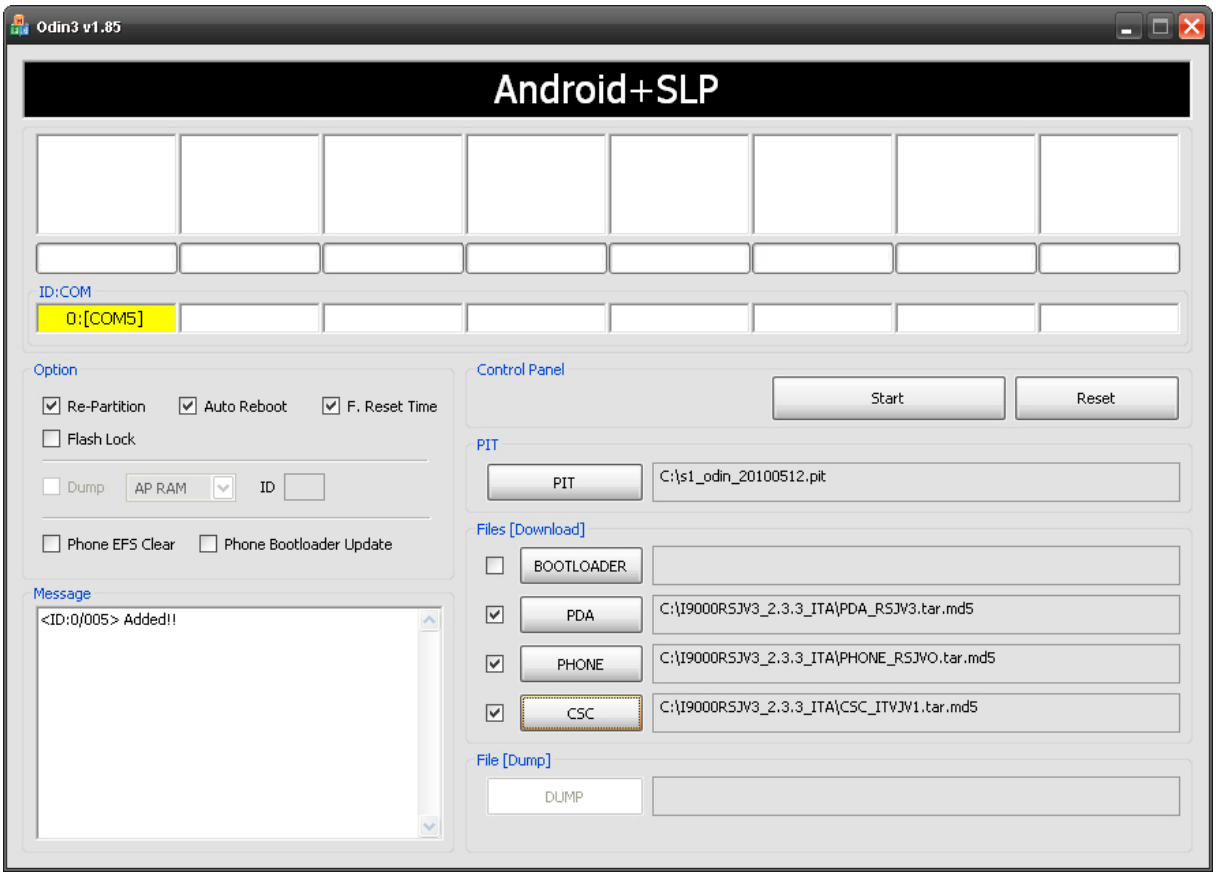

Figura G.2: La schermata di configurazione di Odin.

11. cliccare sul pulsante **Start**. Verranno effettuate delle operazioni che è possibile vedere nel campo *Message*. Sul telefono comparirà sotto al triangolo giallo una barra azzurra che indica lo stato di avanzamento. Alla fine della procedura il terminale verrà riavviato e sarà possibile staccare il cavo USB.

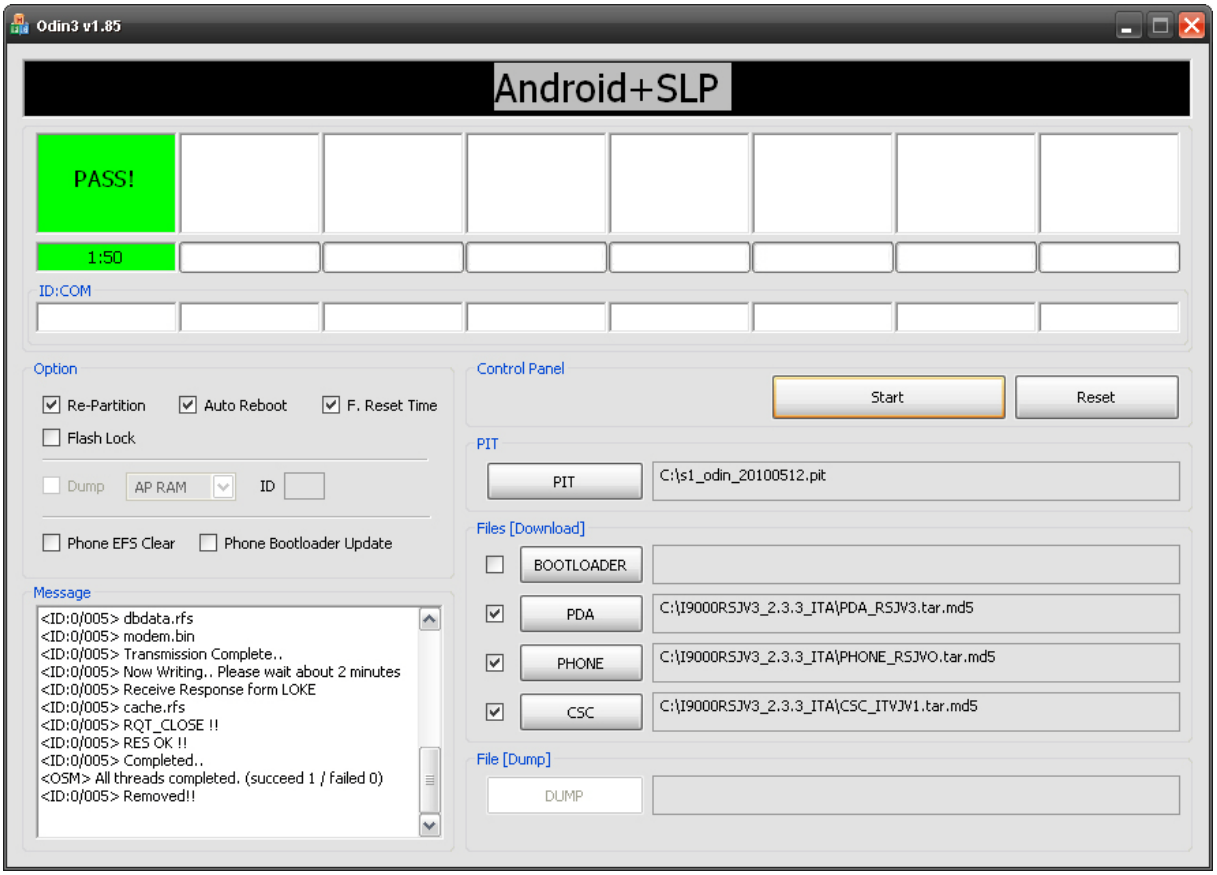

Figura G.3: La schermata di Odin a procedura terminata.

Se Odin dovesse rimanere bloccato a lungo su *file analysis* o su *Downloading… do not turn off target* staccare il terminale e ripetere la procedura a partire dal punto 10. In caso di esito ancora negativo ripartire dal punto 8.

Se anche il terminale risulta bloccato provare a tenere premuto per 10 secondi il pulsante di accensione o a staccare la batteria e ripetere dal punto 6.

G.2 Flashing di un custom kernel

Le indicazioni descritte nel seguito si riferiscono a come installare un custom kernel. Ecco come procedere:

1. assicurarsi di aver installato sulla propria macchina i driver USB del Galaxy S;
2. scaricare Odin [51];
3. scaricare il kernel che ci interessa. Si tratta di un file compresso con estensione `.tar` che contiene l'immagine del kernel. Copiare questo file in `C:\`;
4. spegnere il dispositivo e mettere il Galaxy S in Download Mode. Sullo schermo verrà visualizzata la seguente immagine:

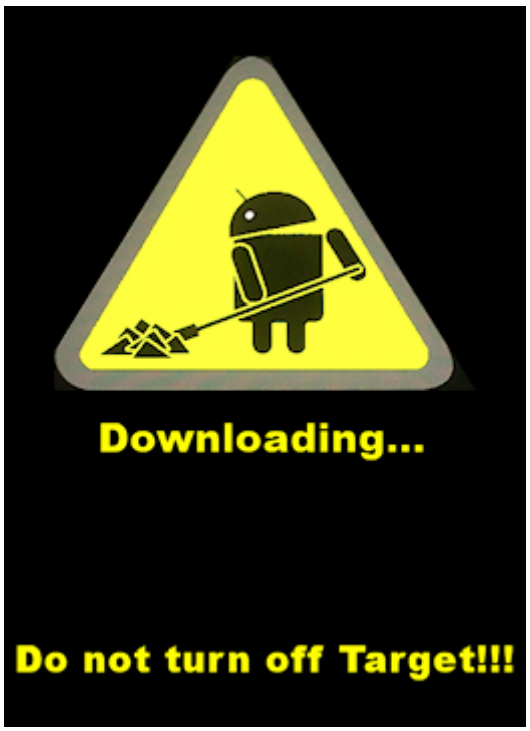

Figura G.4: La schermata di Download Mode.

5. avviare Odin e collegare il dispositivo alla porta USB. Se il terminale è riconosciuto correttamente da Odin, sarà possibile vedere nel primo campo ID:COM il nome della porta COM su sfondo giallo (il numero della porta COM è indifferente). Se così non fosse provare i seguenti tentativi in sequenza:
   - chiudere e riavviare Odin;
   - scollegare e ricollegare il cavo USB;

- provare a cambiare porta USB;
- ripetere tutta la procedura dal punto 4 in poi;
- provate a reinstallare i driver USB del Galaxy S.

6. configurare Odin per il flash del Galaxy S con il kernel scaricato:
   - cliccare sul pulsante **PDA** e scegliere il file del kernel con estensione `.tar` precedentemente copiato in `C:\`;

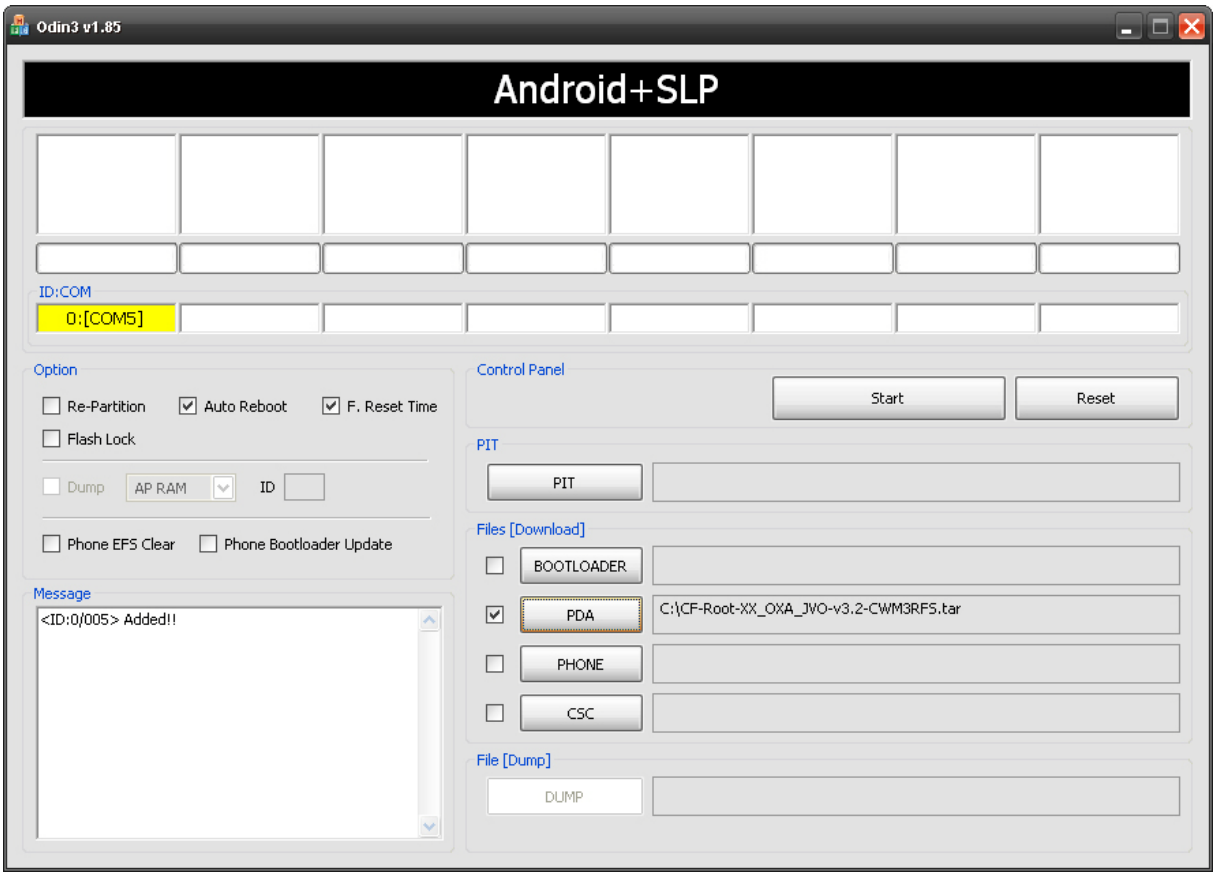

Figura G.5: La schermata di configurazione di Odin.

7. cliccare sul pulsante **Start**. Verranno effettuate delle operazioni che è possibile vedere nel campo *Message*. Sul telefono comparirà sotto al triangolo giallo una barra azzurra che indica lo stato di avanzamento. Alla fine della procedura il terminale verrà riavviato e sarà possibile staccare il cavo USB.

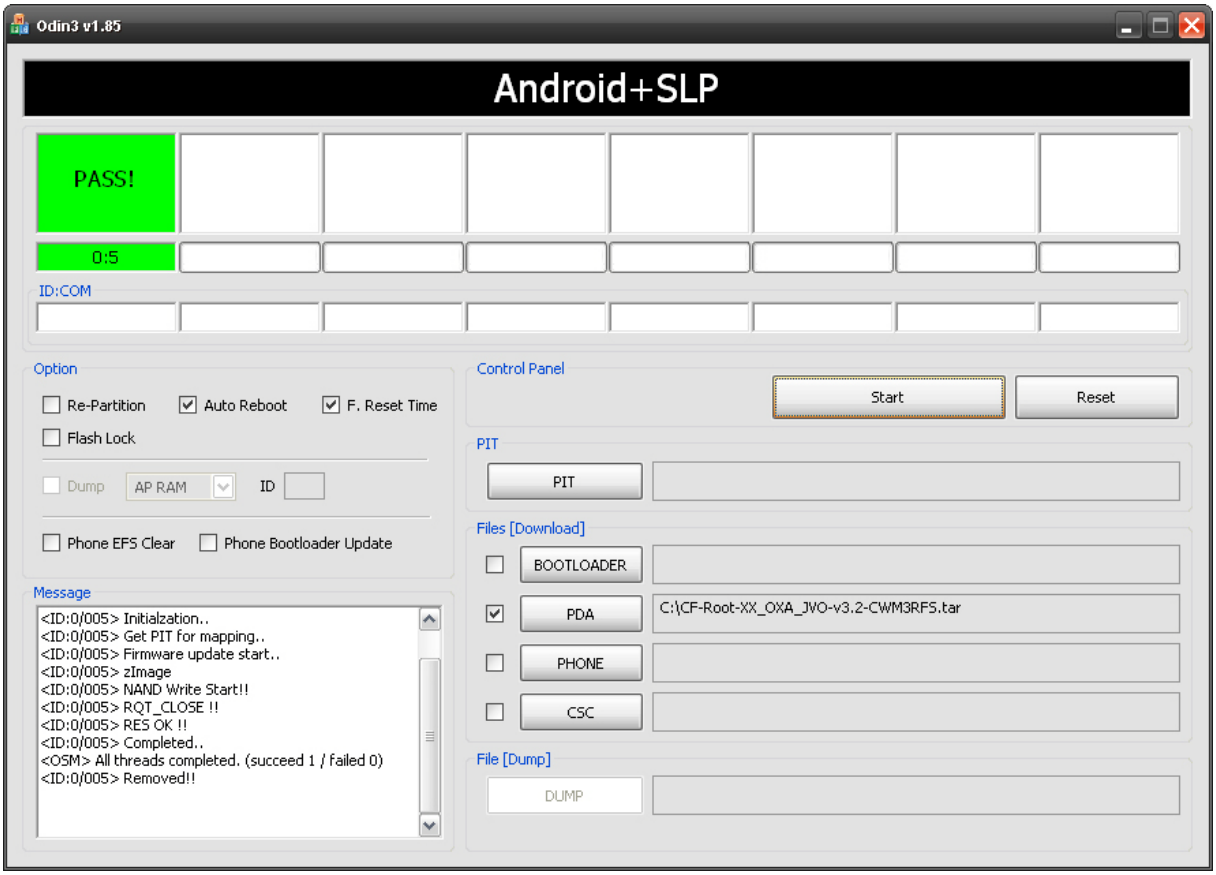

Figura G.6: La schermata di Odin a procedura terminata.

Se Odin dovesse rimanere bloccato a lungo su *file analysis* o su *Downloading… do not turn off target* staccare il terminale e ripetere la procedura a partire dal punto 6. In caso di esito ancora negativo ripartire dal punto 4.

Se anche il terminale risulta bloccato provare a tenere premuto per 10 secondi il pulsante di accensione o a staccare la batteria e ripetere dal punto 6.

Bibliografia

[1] Official Google Blog, Where’s my Gphone? http://googleblog.blogspot.com-2007/11/wheres-my-gphone.html

[2] Google+, Reto Meier. https://plus.google.com/111169963967137030210/posts

[3] Open Handset Alliance. http://www.openhandsetalliance.com/

[4] Wikipedia, Android. http://it.wikipedia.org/wiki/Android

[5] Android Developers, Dashboards. http://developer.android.com/about/dashboards/index.html

[6] Android Developers, What is Android? http://developer.android.com/guide/basics/what-is-android.html

[7] Google Play. https://play.google.com/store

[8] Wikipedia, Kernel. http://it.wikipedia.org/wiki/Kernel

[9] Wikipedia, Linux (kernel). http://it.wikipedia.org/wiki/Kernel_Linux

[10] xda-developers, Kernel CF-Root. http://forum.xda-developers.com/showthread.php?t=788108

[11] Semaphore, Semaphore GB. http://www.semaphore.gr/homepage/semaphore-gb

[12] Semaphore, Semaphore ICS. http://www.semaphore.gr/homepage/semaphore-ics

[13] xda-developers, Kernel Midnight GB. http://forum.xda-developers.com/showthread.php?t=1199140

[14] xda-developers, Kernel Midnight ICS. http://forum.xda-developers.com/showthread.php?t=1551410&highlight=kernel

[15] xda-developers, Kernel SpeedMod. http://forum.xda-developers.com/showthread.php?t=1044519

[16] xda-developers, Kernel Galaxian. http://forum.xda-developers.com/showthread.php?t=1137595

[17] xda-developers, Kernel VooDoo. http://forum.xda-developers.com/showthread.php?p=16806490#post16806490

[18] xda-developers, Kernel TalonDev. http://forum.xda-developers.com/showthread.php?t=1106075

[19] xda-developers, Kernel FuguMod. http://forum.xda-developers.com/showthread.php?t=812836

[20] xda-developers, Kernel Devil. http://forum.xda-developers.com/showthread.php?t=1445214

[21] xda-developers, Kernel Icy Glitch. http://forum.xda-developers.com/showthread.php?t=1459475

[22] Wikipedia, Software development kit. http://it.wikipedia.org/wiki/Software_development_kit

[23] Wikipedia, Application programming interface. http://it.wikipedia.org/wiki/Application_programming_interface

[24] Android Developers, Android Virtual Device. http://developer.android.com/tools/devices/index.html

[25] Android Developers, Managing AVDs with AVD Manager. http://developer.android.com/tools/devices/managing-avds.html

[26] Android Developers, Managing AVDs from the Command Line. http://developer.android.com/tools/devices/managing-avds-cmdline.html

[27] Android Developers, android. http://developer.android.com/tools/help/android.html

[28] Android Developers, Managing Projects from the Command Line. http://developer.android.com/tools/projects/projects-cmdline.html

[29] Android Developers, Using the Android Emulator. http://developer.android.com/tools/devices/emulator.html

[30] Android Developers, Android Emulator. http://developer.android.com/tools/help/emulator.html

[31] Android Developers, Android Debug Bridge. http://developer.android.com/tools/help/adb.html

[32] Android Developers, Android Development Tools. http://developer.android.com/tools/help/adt.html

[33] Eclipse. http://www.eclipse.org/

[34] Android Developers, ADT Plugin. http://developer.android.com/sdk/installing/installing-adt.html

[35] Android Developers, What is the NDK?. http://developer.android.com/tools/sdk/ndk/index.html

[36] Kies Samsung. http://www.samsung.com/it/support/usefulsoftware/KIES/JSP

[37] AndroidGalaxys.net, Aggiornare i firmware con Odin tramite PDA, MODEM, CSC. http://www.androidgalaxys.net/firmware-samsung-galaxy-s/aggiornare-firmware-con-odin-PDA-MODEM-CSC/

[38] AndroidGalaxys.net, Aggiornare i firmware con Odin tramite PDA. http://www.androidgalaxys.net/firmware-samsung-galaxy-s/aggiornare-firmware-con-odin-PDA/

[39] Samsung Open Source Release Center. http://opensource.samsung.com/

[40] Android Developers, Android SDK. http://developer.android.com/sdk/index.html

[41] xda-developers, Android Dev. How-To Guide: Compiling the Android/Linux kernel for the Epic Touch 4G. http://forum.xda-developers.com/archive/index.php/t-1442870.html

[42] Android Developers, Building Your First App. http://developer.android.com/training/basics/firstapp/index.html

[43] Wiki di ubuntu-it, Amministrazione Sistema / Compilazione Kernel. http://wiki.ubuntu-it.org/AmministrazioneSistema/CompilazioneKernel

[44] Wiki di ubuntu-it, Repository. http://wiki.ubuntu-it.org/Repository

[45] The Linux Kernel Archives, Con Kolivas. http://www.kernel.org/pub/linux/kernel/people/ck/patches/2.6/

[46] The Linux Kernel Archives, Alan Cox. http://www.kernel.org/pub/linux/kernel/people/alan/

[47] The Linux Kernel Archives, Andrew Morton.
http://www.kernel.org/pub/linux/kernel/people/akpm/

[48] The Linux Kernel Archives, Ingo Molnar.
http://www.kernel.org/pub/linux/kernel/projects/rt/

[49] QEMU Emulator User Documentation.
http://wiki.qemu.org/download/qemu-doc.html

[49] Ecplise Downloads. The. http://www.eclipse.org/downloads/

[51] AndroidGalaxys.net, Pagina di download.
http://www.androidgalaxys.net/DWL/odin/