Istituto di Istruzione Secondaria Superiore

“Alessandro Greppi”

Via dei Mille 27 – 23876 Monticello B.za (LC)

MouseUp: progetto di un’app di sistema

A cura di MATTEO PREDA

Classe 5^IT – articolazione informatica

Anno scolastico 2015/2016

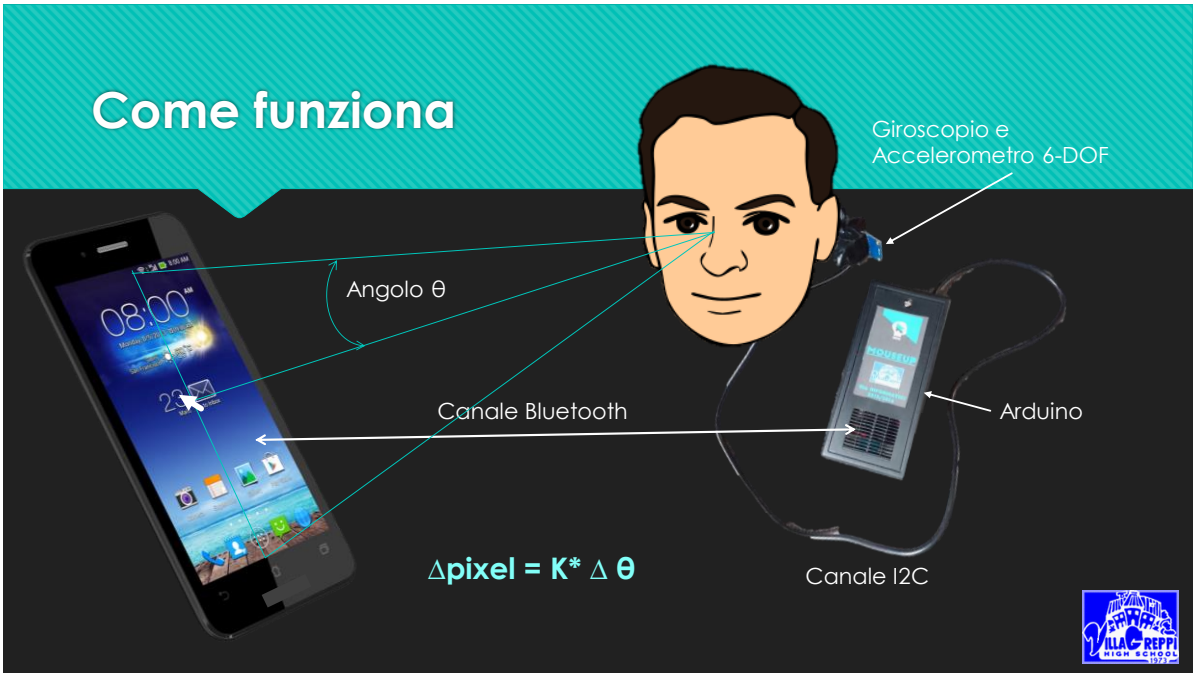
Sommario

1. Introduzione all'idea

La tecnologia è diventata un elemento imprescindibile della nostra vita. I nostri incontri, le nostre decisioni, le nostre emozioni, passano quasi tutte attraverso un telefono o un tablet. Cosa significa questo per tutti coloro che non possiedono l'utilizzo delle mani? Per questo è stato concepito "MouseUp", un'applicazione innovativa che permette alle persone di utilizzare il proprio dispositivo senza l'utilizzo delle dita.

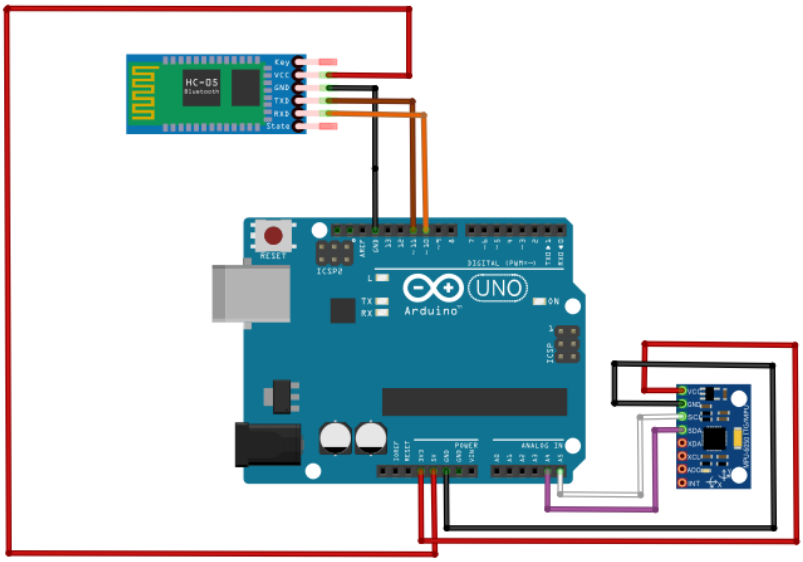
2. Funzionamento

L'applicazione dispone di due diverse modalità di funzionamento: una prima versione prevede lo spostamento di un cursore sullo schermo inclinando il dispositivo, mentre una seconda, utilizzando un apposito auricolare, tramite il movimento della testa. Pronunciando determinate parole chiave è possibile eseguire diverse operazioni, come il click semplice o lungo, lo swipe o lo zoom.
Il prodotto comprende anche la costituzione di una ROM personalizzata in sostituzione al classico sistema operativo Android per garantire il corretto funzionamento dell'applicazione.
La comunicazione tra il microprocessore e il sensore esterno integrante un giroscopio e un accelerometro avviene tramite un canale i2c, un protocollo di comunicazione che permette la gestione di più sensori tramite un unico canale.
L'angolo utilizzato a livello applicativo viene calcolato tramite un opportuno filtraggio del giroscopio. Il valore finale del sensore è dato dall'integrale del valore letto meno il valore precedente. Essendo l'integrale un insieme di infinite funzioni primitive traslate di una costante, si introduce un drift sistematico, eliminabile tramite un filtro passa basso a livello software.



3. Architettura di MouseUp

Il sistema MouseUp si compone di: un microcontrollore Arduino Uno, un modulo Bluetooth HC-05, un sensore integrante un giroscopio e un accelerometro, un supporto auricolare e infine un'applicazione Android (DemoApp).
Il microcontrollore Arduino è collegato allo smartphone tramite una connessione Bluetooth. L'applicazione Android durante l'utilizzo dei sensori esterni rimane in ascolto sui dati provenienti dal microcontrollore e, sulla base di questi, è in grado di muovere un puntatore presente sullo schermo.



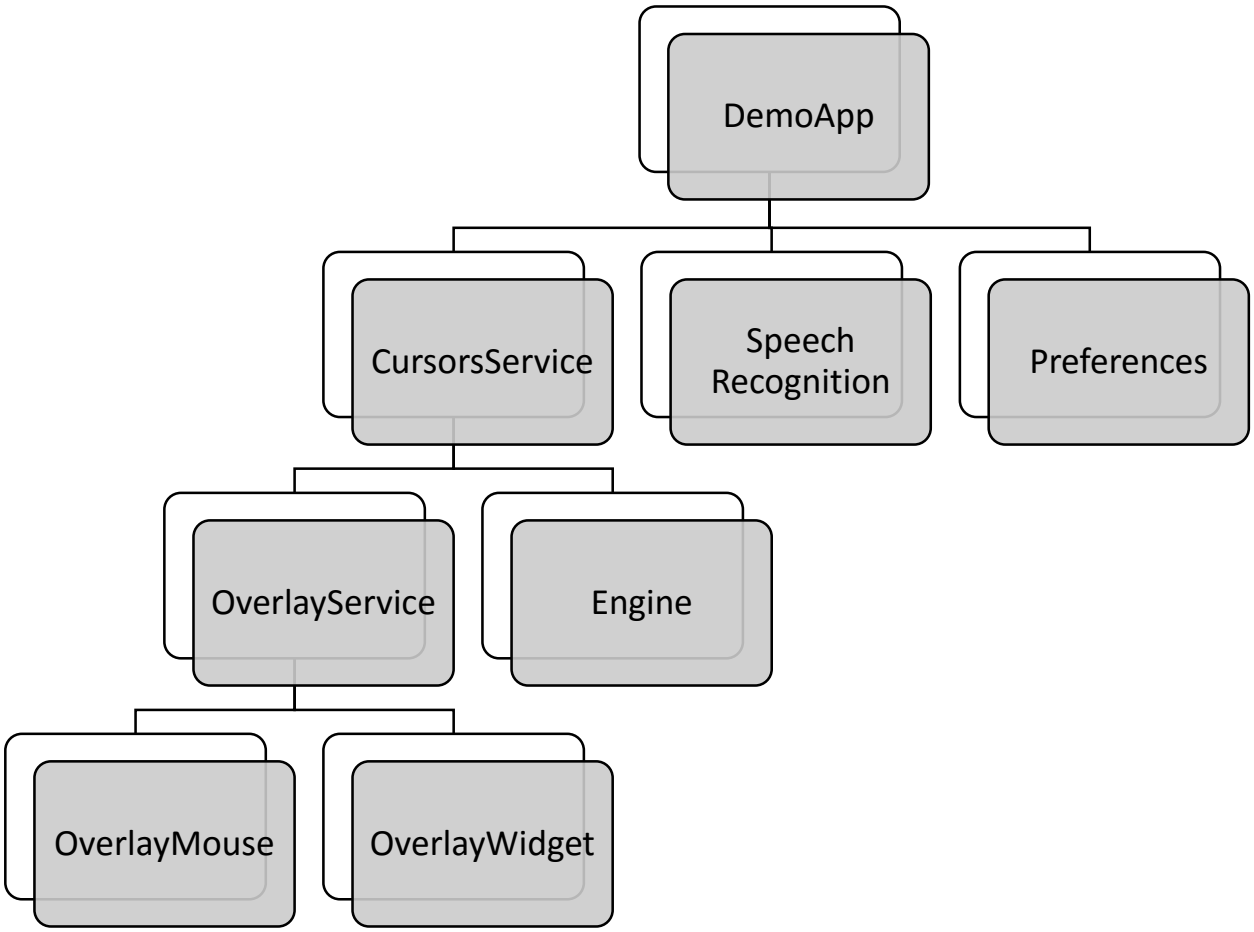
Nella seguente immagine è possibile notare il risultato finale del progetto. All'interno del case nero è contenuta l'intera infrastruttura hardware, eccezion fatta per i sensori. Questi risultano collegati a un auricolare, così da poter percepire ogni minima variazione angolare della testa. L'intero circuito è alimentato da una batteria da 9v.

4. Architettura software

Per ridurre la complessità strutturale, l'applicazione è stata suddivisa in diversi servizi. In particolare significativi sono:

1. Il servizio per la visualizzazione del mouse e l'iniezione di eventi touch;
2. Il servizio per l'elaborazione dei dati provenienti dai sensori interni;
3. Il servizio per l'elaborazione dei dati provenienti dai sensori esterni con l'integrazione di funzioni per la gestione della connessione Bluetooth;
4. Il servizio per il riconoscimento vocale
5. Il servizio in ascolto sul menù di impostazioni tramite un listener.

L'interazione tra questi viene gestita tramite i messaggi di broadcast. Per esempio, quando il servizio vocale riconosce una determinata parola chiave genera un messaggio che viene catturato dal servizio attivo ed interpretato di conseguenza. In questo modo ciascun servizio rimane indipendente dagli altri, eseguendo determinate funzioni solo alla ricezione di un messaggio di broadcast.



L'applicazione DemoApp per funzionare richiede dei seguenti permessi:

1. <**uses-permission android:name="android.permission.SYSTEM_ALERT_WINDOW"** />
   Permette a un'applicazione di creare delle finestre del tipo "*TYPE_SYSTEM_ALERT*", visualizzate sopra ogni applicazione.
2. <**uses-permission android:name="android.permission.BLUETOOTH"**/>
   Permette a un'applicazione di collegarsi a un dispositivo accoppiato tramite bluetooth.

3. <**uses-permission android:name="android.permission.READ_PHONE_STATE"**/>
   <**uses-permission android:name="android.permission.RECORD_AUDIO"** />

Permettono a un'applicazione di accedere alle periferiche audio, potendo così invocare per esempio un servizio di riconoscimento vocale.

4. <**uses-permission android:name="android.permission.INJECT_EVENTS"**/>
   Permette a un'applicazione di iniettare eventi touch ad altre applicazioni. Richiede dei permessi di sistema.

5. I servizi

Un servizio è una componente di un'applicazione in grado di eseguire una lunga serie di operazioni in background limitando i consumi.
Per utilizzare un servizio è necessario svolgere due semplici operazioni:

- creare una classe Java che estenda Service o un suo derivato;
- registrarlo all'interno del file manifest.xml attraverso il nodo <service>:

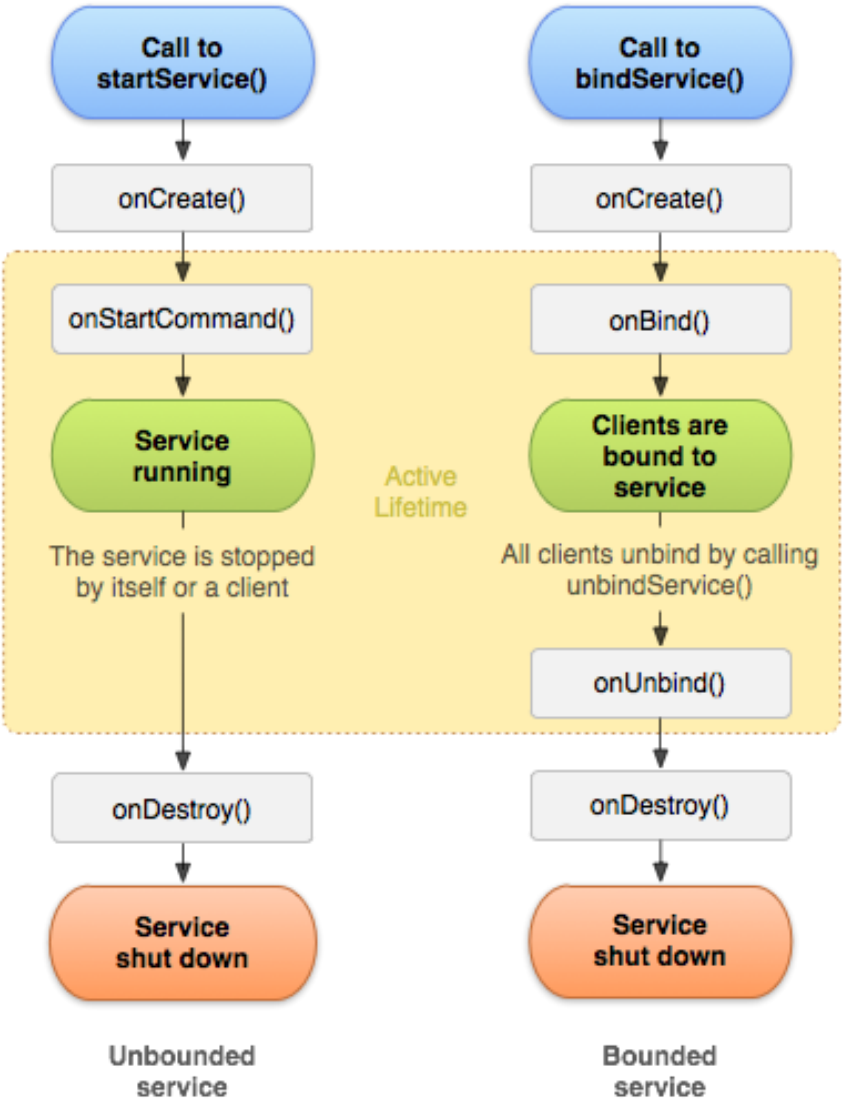
  <service android:name=".SpeechRecognizerService" />

  dove l'attributo android:name specifica il nome della classe che implementa il servizio.

A questo punto è opportuno fare una precisazione. Creare un servizio non equivale a creare un nuovo processo. Ciascuna classe Service di default viene eseguita all'interno del Thread principale dell'applicazione. Per questo motivo, un servizio non può neanche essere considerato come un nuovo thread.
Gli oggetti Service sono classificabili in due tipologie, dipendentemente dal modo in cui vengono avviati e dalle funzioni di callback implementate:

- i servizi Started: vengono avviati tramite il metodo startService(). La loro particolarità è quella di essere eseguiti in background indefinitamente. Non offrono interazioni con il chiamante e proseguono finché non vengono interrotti mediante il metodo stopService(). Sono da prediligere per operazioni con una finalità indipendente dallo stato delle altre applicazioni. Si potrebbero occupare quindi di aggiornamenti dati in background o di scaricare file o immagini.
- i servizi Bound: implementano una modalità client-server. Vengono interrotti nel momento in cui non vi sono più client ad essi collegati, svolgendo un ruolo di supporto ad altre applicazioni.

La differenza tra le due tipologie si riflette anche sul ciclo di vita:



Nell'immagine sono rappresentate le fasi attraversate da un Service Started (sulla sinistra) e da uno Bound (sulla destra). Entrambi i cicli di vita iniziano e terminano con i metodi di callback onCreate() e onDestroy(). Le differenze si concentrano nella fase in cui il Service viene attivato. Mentre l'avvio di un service Started viene notificato per mezzo della funzione onStartCommand(), l'inizio e la fine della connessione con un service bound viene segnalato dai metodi onBind() e onUnbind().

6. Introduzione al problema dei permessi

L'iniezione di eventi touch provenienti da una componente di un'applicazione, per questioni di sicurezza, è confinata dal sistema operativo solo all'interno della stessa applicazione. Questo significa che, se installato su un tradizionale telefono, il primo servizio potrebbe iniettare eventi solo all'interno della stessa MouseUp, non ottenendo di fatto alcun vantaggio per l'utente. La soluzione a questo problema la si trova nella firma digitale. Quando Google rilascia una nuova versione di Android, i diversi costruttori di dispositivi, come ad esempio Samsung, LG, Sony, etc., che dispongono di un'opportuna licenza generano delle chiavi per firmare digitalmente le rispettive applicazioni, così da poterle rendere applicazioni di sistema. Questa è la spiegazione per cui all'acquisto, telefoni diversi, pur disponendo della stessa versione di Android, dispongono di applicazioni diverse. Un'applicazione di sistema è capace di interagire con il sistema operativo con privilegi maggiori, tra i quali la possibilità di iniettare ad altre applicazioni eventi touch. L'accesso a queste chiavi è tuttavia impossibile ad uno sviluppatore di terze parti, dal momento che il costruttore di una ROM Android non renderà mai pubbliche le proprie chiavi di cifratura per ovvi motivi di sicurezza. Per questo motivo per creare un'applicazione di sistema con privilegi elevati (platform nel nostro caso) è stato necessario costruire una ROM "custom" a partire dal codice sorgente open source di Android. Nel nostro caso la versione di Android utilizzata è stata ottenuta a partire dal codice sorgente della distribuzione rilasciata da Cyanogenmod.

7. Cyanogenmod

Cyanogenmod è un firmware personalizzato distribuito gratuitamente per una serie di dispositivi Android. Basata su un progetto open source, è progettata per aumentare le performance e l'affidabilità delle ROM Android rilasciate da compagnie ai livelli di Google o T-Mobile, in particolare tramite miglioramenti allo schedulatore del kernel e all'introduzione di profili di overclock. Inoltre, offre un'ampia varietà di funzionalità che non sono attualmente comprese nelle sopracitate versioni di Android.

Cyanogenmod genera e pubblica giornalmente nuove ROM sulla base degli ultimi sorgenti disponibili che prendono il nome di "nightly builds", conosciute in modo

informale sotto il nome “nightlies”. Sebbene queste siano utilizzate per ricevere feedback da parte degli utenti sulle più recenti correzioni, riscontrabili dal file changelog, le “nightly builds” possono contenere bug, per questo vengono ritenute instabili. Mensilmente invece vengono pubblicate le versioni “stable”, le quali, pur contenenti dei minimi bug, sono considerate potenzialmente vendibili e ad uso quotidiano. L’installazione di una custom ROM richiede la possibilità di sovrascrivere i file di sistema, per cui necessita dei permessi di root.

8. Guida ai permessi di root

Per comprendere realmente cosa sia il root nel mondo Android, dobbiamo analizzarlo nel contesto da cui discende il sistema operativo, Linux. In questa piattaforma, con il termine root si identifica l’utente che dispone dei massimi privilegi all’interno della macchina, in un concetto del tutto simile all’utente amministratore in Windows. Il termine root, in italiano radice, identifica la cartella principale di un sistema Unix. Ottenere i permessi sulla cartella radice, significa poter accedere sia in lettura che in scrittura all’intero contenuto del file system.
Applicato in un contesto Android, equivale ad autentificarsi all’interno del sistema con un grado di privilegi massimo. In questo modo, è possibile oltrepassare ogni forma di restrizione imposta dal sistema operativo. Se da una parte questo comporta la possibilità di accedere a contenuti e risorse fino ad ora inaccessibili, dall’altra parte il rischio di danneggiare il telefono è decisamente elevato, per questo è consigliato seguire una guida attendibile. La procedura di root del telefono è del tutto legale, ma comporta la perdita di eventuali garanzie.
Di seguito si riporta la procedura per ottenere i permessi di root per lo specifico Samsung Galaxy S5 SM-G900F KLTE (si tenga conto che la responsabilità per eventuali danni è sempre e soltanto di chi decide di modificare il proprio telefono):

1. Premesso che l’operazione di root di un dispositivo non comporti l’eliminazione dei dati personali, è consigliato eseguire un backup del sistema tramite l’utilizzo di tool come AirDroid. In aggiunta, bisogna ricordarsi di effettuare tramite Google Sync una sincronizzazione dei contatti Gmail e della rubrica telefonica.
2. Una volta installati i driver Samsung USB, bisogna scaricare la versione di CF Auto Root corrispondente al modello sopracitato dal sito

"autoroot.chainfire.eu". Al suo interno sono contenuti un eseguibile di Odin, un programma per l'installazione di firmware e custom-ROM per dispositivi Samsung, e un file di estensione ".tar.md5", una recovery personalizzata.

3. Una volta aperto Odin, premere sul bottone PDA. Si aprirà un'esplora risorse, per cui selezionare il file "CF-Auto-Root-klte-kltexx-smg900f.tar.md5". Controllate accuratamente che non sia selezionata la casella re-partition.
4. Aprire il menu di impostazioni dal telefono e selezionare "generale". Una volta raggiunta, premere la voce "Numero Build" per sette volte: una notifica vi avviserà dell'attivazione della modalità sviluppatore. A questo punto è necessario spegnere il telefono. Una volta spento, avviarlo in modalità download, premendo contemporaneamente il tasto di accensione, il tasto home e il tasto per abbassare il volume fino ad avvertire una vibrazione. Si presenterà sullo schermo un robot verde, quindi selezionate il tasto per alzare il volume. Vi ritroverete in modalità download.
5. Tramite un cavo USB, collegare il dispositivo al computer, assicurandosi che Odin riconosca il telefono. Cliccare sul tasto start e attendete il messaggio "Pass" su Odin. A questo punto il telefono si ravvierà automaticamente e si dovrebbe vedere l'applicazione SuperSu installata.
6. Per assicurarsi ulteriormente che il tutto sia andato a buon fine, scaricare dal Play Store l'applicazione Root Checker, la quale vi notificherà l'acquisizione o meno dei permessi di root.

9. Guida alla build di una ROM Cyanogenmod 13.0

La guida qui presente è relativa alla creazione di una ROM CM 13.0 per un Samsung Galaxy S5 KLTE.
Requisiti:

- Una buona connessione a internet per il download del codice sorgente.
- Una macchina con prestazioni elevate, in particolare con almeno 8gb di memoria RAM e 100gb di spazio libero.
- Un sistema operativo Linux.

Procedura:

1. Se non si è ancora installato adb e fastboot, installare l'SDK Android. L'acronimo "SDK", Software Developer Kit, indica un insieme di tool che

possono essere utilizzati per “flashare” software o controllare in diretta il file di log dal computer.

2. Una volta ottenuta l’SDK, installare un package manager. In Linux, è un sistema utilizzato per installare o rimuovere software sul computer. In alternativa si può utilizzare il comando “*apt-get install*” direttamente dal terminale.

   Indipendentemente dal tipo di sistema utilizzato, copiare la seguente stringa: *bison build-essential curl flex git gnupg gperf libesd0-dev liblz4-tool libncurses5-dev libsdl1.2-dev libwxgtk2.8-dev libxml2 libxml2-utils lzop maven openjdk-7-jdk pngcrush schedtool squashfs-tools xsltproc zip zlib1g-dev*

   In aggiunta, per i sistemi 64-bit:

   g++-multilib gcc-multilib lib32ncurses5-dev lib32readline-gplv2-dev lib32z1-dev

   Per Ubuntu 15.10 e successivi, effettuare la seguente modifica:

   lib32readline-gplv2-dev → *lib32readline6-dev*

   Ora è necessario scaricare la JDK: per CyanogenMod 13.0, quella più corretta risulta essere la versione OpenJDK 1.7. Tuttavia, dalla versione di Ubuntu 16.04 non è più disponibile all’interno dei repository ufficiali. È comunque possibile reperirla al seguente link:
   https://launchpad.net/~openjdk-r/+archive/ubuntu/ppa

3. Eseguire i seguenti comandi per scaricare il folder “*repo*” e conferirgli il permesso di esecuzione:

- $ curl https://storage.googleapis.com/git-repo-downloads/repo > ~/bin/repo
- $ chmod a+x ~/bin/repo

4. Nelle più recenti versioni di Ubuntu, la directory *~/bin* dovrebbe essere già inclusa all’interno del path. Per verificare la sua presenza, aprire con un editore *~/.profile* e verificate che siano presenti i seguenti comandi (nel caso in cui fossero assenti, aggiungeteli):

   if [-d "$HOME/bin"]

   PATH="$HOME/bin:$PATH"

   fi

5. A questo punto si inizializza il repository Cyano:

   $ cd ~/android/system/

   $ repo init -u https://github.com/CyanogenMod/android.git -b cm-13.0

   Per avviare il download del codice sorgente, eseguire:

   $ repo sync

   Il manifest CM include parametri di configurazione prestabiliti molto sensibili. I valori di default utilizzati sono:

   - *-j 4*: indica che contemporaneamente verranno stabiliti quattro diversi thread di connessione. Nel caso in cui si verificassero dei problemi relativi alla rete, si può provare ad abbassare il numero.
   - *-c*: forzerà repo ad "entrare" solo nel branch corrente e non in tutta la storia Cyano.

6. Dopo aver completato il download, è necessario scaricare la configurazione specifica per il nostro dispositivo e il codice del kernel:

   cd ~/android/system

   $ source build/envsetup.sh

   $ breakfast klte

7. Ora assicurarsi che il telefono sia collegato al computer tramite un cavo USB. Il prossimo passo prevede la copia dei file proprietari presenti sul telefono all'interno della ROM. Questo risulta fondamentale in quanto, se dovessero succedere errori non gestiti durante la fase di copiatura, il dispositivo non potrebbe utilizzare diverse funzionalità, tra le quali le librerie grafiche. Se il vostro telefono dispone già di una ROM CyanogenMod installata, eseguite:

   cd ~/android/system/device/samsung/klte

   $ ./extract-files.sh

   altimenti è necessario aggiungere un repository nel local manifest. Spostatevi in *~/CyanoGenMod/android/system/.repo/local_manifests$* e all'interno del file *roomservice.xml* aggiungete il seguente nodo:

   <project name="TheMuppets/proprietary_vendor_samsung" path="vendor/samsung" remote="github" />

   Salvate il file e chiudetelo.

8. Per velocizzare la procedura di building è consigliato utilizzare CCache. Per far questo aggiungre "*export USE_CCACHE=1*" al file *~/.bashrc* ed eseguite:

   prebuilts/misc/linux-x86/ccache/ccache -M 50G

   dove 50G identifica lo spazio dedicato su disco all'utilizzo come area di cache. Ricordarsi che la quantità che è specificata verrà permanentemente occupata, per cui è consigliato non utilizzare cifre superiori alle cento unità.

9. A questo punto è possibile effettuare il building della ROM vero e proprio:

   $ croot

   $ brunch klte

   Una volta terminata la procedura digitare "*$ cd $OUT*".

Come si può notare, il comando brunch avrà generato come output diversi file. I due più importanti ai nostri fini sono una recovery CyanogenMod e un file cm-13.0-20160617-UNOFFICIAL-klte.zip, il quale contiene il package di installazione della ROM.

10. Guida all'installazione di una ROM

1. Una volta generato il file ".zip" contenente il sorgente del sistema operativo, copiarlo all'interno della cartella root della memoria SD del telefono. Per questioni di licenze, le diverse ROM reperibili su internet o home made non dispongono delle licenze per le Google Apps. Per questo, riavviando il telefono al termine della procedura, non si avrebbero installate applicazioni come Play Store, Google Chrome o Gmail. Per ovviare a questo inconveniente, bisogna scaricare dal sito "opengapps.org" il pacchetto di app necessarie e copiarlo nel terminale.
2. A questo punto è necessario riavviare il telefono in modalità recovery. Una volta spento, premere contemporaneamente il tasto per alzare il volume, il tasto home e il tasto di accensione del telefono fino ad avvertire una vibrazione.
3. Spostarsi utilizzando i tasti del volume alla voce "Wipe Data/Factory Reset" e confermare. Questo comando provvederà a cancellare tutti i dati presenti nella memoria interna del telefono. Ora eseguire una "Wipe Cache Partition". Quest'ultima risulta un'operazione opzionale, ma riduce

notevolmente le possibilità per le quali il telefono si blocchi durante la fase di boot.

4. Tornare sulla schermata principale e selezionare prima “Install Zip” e poi “Choose zip from /sdcard”. A questo punto scorrere il file system fino al raggiungimento del file “.zip” contenente la ROM prodotta. Selezionarlo e confermare l’installazione.
5. Ora ripetere dal punto due le stesse procedure applicandole alle Google Apps. al termine di quest’ultima operazione si esegue un “Reboot System Now”. Attendete per una decina di minuti, al termine dei quali il terminale sarà pronto a utilizzare la ROM autoprodotta.

11. Firma digitale in Android

In riferimento alla documentazione ufficiale, Android utilizza la firma digitale in contesti diversi. Il sistema operativo richiede che ciascun file “.apk” sia firmato digitalmente per poter essere correttamente installato e successivamente eseguito. Questo avviene ogni qualvolta un’applicazione viene installa, aggiornata (le due versioni devono condividere la stessa chiave) o richiede di accedere tramite uno User ID ai dati di un’altra applicazione (le due applicazioni devono condividere lo stesso ID e la stessa chiave).
Ciascuna chiave è composta da due diversi file:

- il certificato, il quale compare con un’estensione “.x509.pem”. Esso contiene solo la metà pubblica della chiave, per questo può essere distribuita pubblicamente. Viene utilizzato per verificare che un determinato file “.apk” sia firmato dalla corrispondente chiave privata del certificato;
- una chiave privata, con estensione “.pk8”. Viene utilizzata per firmare le diverse applicazioni e, per questioni di sicurezza, dovrebbe essere mantenuta segreta. La conoscenza da parte di un attaccante della chiave privata corrispondente a un determinato certificato, potrebbe comportare un furto di dati o la modifica delle applicazioni.

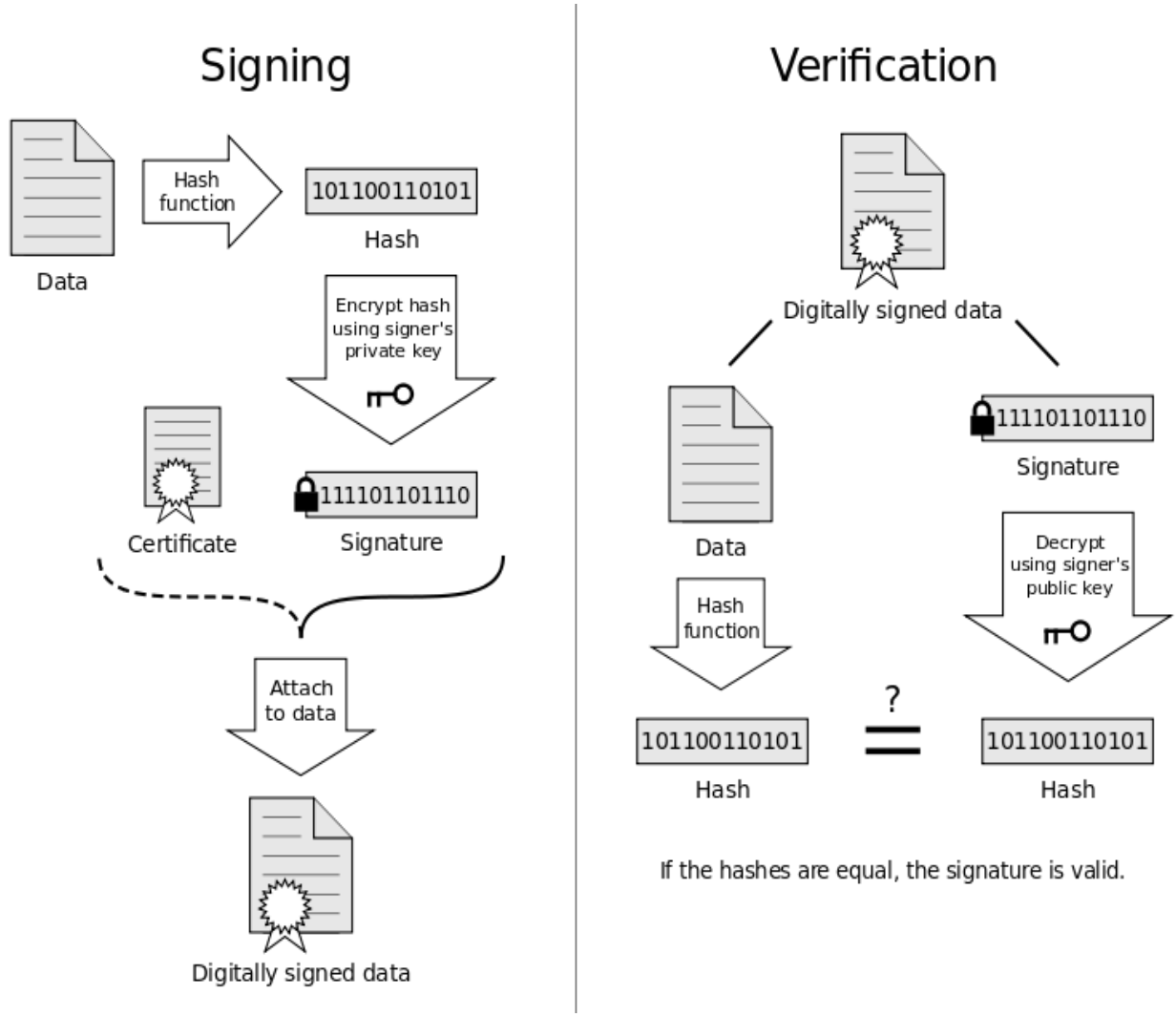
Di default Android dispone di quattro diverse chiavi per la firma digitale: platform key, shared key, media key e testkey. Tutti i package considerati parte della versione natia dell’immagine (interfaccia grafica, impostazioni, bluetooth, WiFi)

sono firmati utilizzando la platform key; i package per la ricerca e dei contatti, equivalenti al processo "home/contacts", tramite la shared key; le applicazioni per la gestione dei contenuti multimediali (galleria, musica) con la media key; i restanti, compresi tutti coloro che non specificano all'interno del file "Android.mk" la chiave di firma, attraverso la testkey (o releasekey per le build di release).

Il processo di firma digitale ha inizio con l'applicazione di una funzione hash al package di riferimento, una funzione non invertibile capace di mappare una stringa di lunghezza arbitraria in una stringa di lunghezza predefinita. Questa viene poi criptata tramite una chiave privata e salvata con il rispettivo certificato. Al contrario, durante la fase di verifica, viene di nuovo calcolata la funzione hash sul package e confrontata con la stringa decriptata calcolata precedentemente. Nel caso in cui siano identiche, la verifica si considera valida. A livello applicativo questa fase di validazione viene effettuata tramite una chiamata alla funzione di sistema RecoverySystem.verifyPackage().

Vista la particolare esigenza da parte di Android di ricevere firmato ciascun file ".apk" per consentirne l'esecuzione, ciascun ambiente di sviluppo integra un processo di firma digitale. Quando un'applicazione viene eseguita direttamente dall'IDE, viene firmata utilizzando un certificato di debug generato dai tools dell'Android SDK. Questo è reperibile all'interno del keystore "*$HOME/.android/debug.keystore*". Un keystore è un file all'interno del quale sono contenute una o più chiavi private, di cui esistono due versioni: una di debug e una di release. La prima dovrebbe essere utilizzata durante la fase di sviluppo dell'applicazione tramite l'ambiente di sviluppo; la seconda quando si intende rilasciare una versione definitiva dell'applicazione, per esempio tramite il Play Store ufficiale di Google. In questo modo Android sarà sempre capace di riconoscervi come legittimi proprietari dell'applicazione, scongiurando eventuali aggiornamenti provenienti da terze parti. Ciascun keystore, indipendentemente dal fatto che sia di release o debug, viene identificato tramite due caratteristiche: il filename che contiene e gli alias. Visto che potenzialmente ciascun keystore può supportare a sua volta più keystore, ciascuno deve essere identificato tramite un alias. Nella maggioranza dei casi conterrà un'unica coppia certificato-chiave privata, ma comunque dovrà disporre del secondo attributo. Ciascun keystore è inoltre

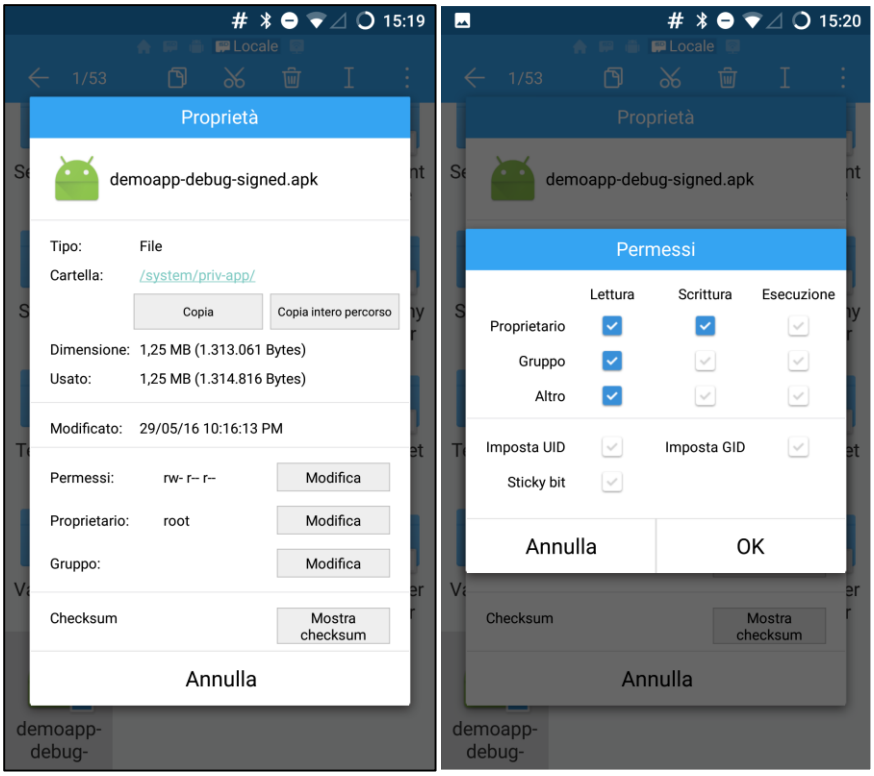
protetto da una coppia di password: una per controllare l'accesso allo stesso e un'altra per ogni coppia keystore-alias presente all'interno del file.
Questa procedura di firma digitale automatica tuttavia non consente l'installazione di applicazioni di sistema se non modificando i keystore utilizzati. Si propone a seguire una guida per la firma di applicazioni con permessi di sistema utilizzando il tool SignApk.



12. Firma digitale di un'applicazione e installazione

Dopo aver seguito tutte le guide precedenti, si potrà a questo punto installare l'applicazione di sistema.

1. Dal sito "http://androidxref.com/4.4.3_r1.1/xref/build/tools/signapk/", scaricare il file Signapk.jar. Questo risulta un comodo tool per firmare applicazioni Android. Ora copiare nella stessa directory i file platform.x509.pem e platform.pk8, reperibili in *"<root-of-android-source-tree>/build/target/product/security*", o in alternativa dall'apposito repository GitHub nel caso in cui si utilizzano le chiavi di default.
2. Aggiungere nel manifest dell'applicazione sotto il nodo <manifest> il seguente attributo: *android:sharedUserId="android.uid.system*".
3. Utilizzando Android Studio, aprire il menù di Build e selezionare Build APK.
4. Copiare il file "app-debug.apk" all'interno della directory contenente il file Signapk.jar. Ora aprire il terminal e spostarsi all'interno della cartella. Eseguire il seguente comando:
   java -jar signapk.jar -w platform.x509.pem platform.pk8 app-debug.apk demoapp-debug-signed.apk
   sostituendo al posto dei caratteri in rosso il nome che si vuole attribuire all'apk firmato.
5. Ora collegare il telefono al computer tramite cavo USB. Installate dal Play Store l'applicazione ES File Explorer, in alternativa si può utilizzare un qualsiasi gestore di file capace di ottenere i permessi di root. Copiare all'interno del telefono il file "*demoapp-debug-signed.apk*".
6. Aprite il file explorer del telefono e concedere i permessi di root all'apk. Per quanto riguarda ES File Explorer, selezionare il menù in alto a sinistra e attivare la voce "Root Explorer".
7. A questo punto navigare il file system fino al raggiungimento del file copiato in precedenza. Selezionarlo e copiarlo. Ora spostarsi nella cartella "*<root-of-android-source-tree>/system/priv-app*" e incollare l'apk. Una volta fatto ciò, selezionate di nuovo il file, aprire la voce Proprietà e modificate i permessi come nella figura qui presente:



A questo punto confermare e riavviare il telefono. L'applicazione dovrebbe essere riconosciuta a livello di sistema.

13. Iniezione di eventi touch

L'iniezione di eventi touch ad altre componenti applicative sono gestite tramite la classe Instrumentation, utilizzata in ambito professionale durante la fase di testing, viene utilizzata in particolare per comprendere dinamicamente il comportamento dell'applicazione con determinati input. Questo è possibile perché all'esecuzione la classe Instrumentation viene instanziata prima di ogni riga di codice, permettendo al programmatore di visualizzare ogni interazione con il sistema operativo. Per poter funzionare, questa richiede di essere eseguita in un thread differente rispetto a quello principale.

Si prendono in considerazione le diverse interazioni con il telefono, con riferimento particolare alla casistica del click per comprendere il funzionamento della classe Instrumentation.

- Click e long click

```
//Thread per l'esecuzione del click
private final Thread thread = new Thread() {
   @Override
   public void run() {
      //Esegue un click alle coordinate X e Y
      Instrumentation m_Instrumentation = new Instrumentation();
      try {
         m_Instrumentation.sendPointerSync(MotionEvent.obtain(
               SystemClock.uptimeMillis(),
               SystemClock.uptimeMillis() + 10,
               MotionEvent.ACTION_DOWN, X, Y, 0));
         m_Instrumentation.sendPointerSync(MotionEvent.obtain(
               SystemClock.uptimeMillis(),
               SystemClock.uptimeMillis() + 10,
               MotionEvent.ACTION_UP, X, Y, 0));
      } catch (Exception e) {}
   }
};
```

Quando un utente intende interagire con il proprio telefono mediante un click, in realtà compie due eventi diversi: un primo evento quando preme sullo schermo, seguito da un secondo evento equivalente al distacco tra esso e il dito. La durata del primo evento espressa in millisecondi indica se il click è normale o lungo. Per poter simulare l'iniezione via software di un click è quindi necessario ripetere gli stessi identici eventi. Per potere iniettare un singolo evento è necessario utilizzare il seguente metodo:

`void sendPointerSync (MotionEvent event)`

Parametri	
event	MotionEvent: Un oggetto "motion event" che descrive l'azione che si vuole compiere.

E' possibile creare un oggetto MotionEvent con il seguente metodo:

`MotionEvent obtain (long downTime, long eventTime, int action, float x, float y, int metaState)`

Parametri	
downTime	long: Il tempo in millisecondi di quando l'utente inizia a premere sullo schermo. Questo deve essere ottenuto mediante il metodo uptimeMillis().
eventTime	long: Il tempo in millisecondi di quando questo specifico evento è stato generato. Deve essere ottenuto mediante la funzione uptimeMillis().
action	int: Il tipo di azione che deve essere svolta.
x	float: La cordinata x dell'evento.
y	float: La coordinata y dell'evento.
metaState	int: Lo stato di ogni metadato che era presente quando l'evento è stato generato (Default 0)
Dato restituito	
MotionEvent	

Assumendo X e Y come coordinate del puntatore in un preciso momento, per simulare un click non resta che invocare per due volte i metodi soprariportati. Inizialmente l'azione dell'evento dovrà essere del tipo ACTION_DOWN e successivamente ACTION_UP.
Nel caso in cui si voglia implementare il long click, non resta che aumentare la distanza temporale dei due eventi aggiungendo le qui riportare istruzioni:

```
try{
   Thread.sleep(750);
}catch (Exception e){}
```

- Home e indietro

```
//Esegue il click indietro
case "Back":{
   try{
```

```
      Runtime.getRuntime().exec("input keyevent " +
KeyEvent.KEYCODE_BACK);
   }catch (IOException e){}
   break;
}
//Esegue il tasto home
case "Home":{
   try{
      Runtime.getRuntime().exec("input keyevent " +
KeyEvent.KEYCODE_HOME);
   }catch (IOException e){}
   break;
}
```

Non tutte le funzionalità richieste dall'applicazione possono essere implementate tramite la classe Instrumentation. Ciascuna applicazione Java dispone di un'istanza della classe Runtime che le permette di interfacciarsi con il sistema. Per poterla richiamare esiste il metodo "*getRuntime()*". Con questa istanza, disponibile solo per dispositivi "*rootati*", è possibile eseguire dei comandi specifici, in un sistema del tutto simile alla Shell di Linux tramite il seguente metodo:

```
Process exec (String[] cmdarray)
```

Parametri	
cmdarray	String: array contenente il comando e i suoi argomenti
Valore di ritorno	
Process	Un nuovo oggetto Process per gestire il sottoprocesso
Genera	
SecurityException	Se un manager di sicurezza non consente la creazione di un sottoprocesso
IOException	Se accade un errore di input o output

NullPointerException	Se cmdarray o uno dei suoi elementi è nullo
IndexOutOfBoundsException	Se cmdarray è un array vuoto (lunghezza uguale a 0)

Il comando per notificare al sistema operativo di voler iniettare un determinato input prende il nome di "*input keyevent*" seguito da una costante numerica. Questo parametro di tipo intero identifica ciascun evento. Su internet è possibilie reperire l'intera lista, comprendente per esempio i codici per regolare il volume dell'audio o la luminosità dello schermo. In particolare possiamo notare come il tasto indietro sia equivalente al codice 0x00000004, mentre il tasto home a 0x00000003.

Per riassumere, quando il servizio attivo riceve un messaggio di broadcast con l'intenzione di voler emulare il tasto back, viene eseguita la seguente riga di codice: Runtime.*getRuntime*().exec(**"input keyevent "** + KeyEvent.***KEYCODE_BACK***); o in alternativa volendo utilizzare il valore in chiaro:

Runtime.*getRuntime*().exec(**"input keyevent "** + 4);

14. I messaggi di broadcast

Supponendo di avere una connessione Internet disponibile, all'accensione del telefono automaticamente i diversi Social Network si collegano ai propri server in ricerca di eventuali notifiche, senza che l'utente lanci le rispettive applicazioni. All'avvio infatti il sistema operativo genera un messaggio di completamento dell'accensione ("*android.intent.action.BOOT_COMPLETED*"), opportunatamente gestito dalle sopracitate applicazioni. Questo prende il nome di messaggio di broadcast. Esistono due tipologie di questi messaggi:

- Normali (trasmessi tramite il metodo *Context.sendBroadcast()*) o asincroni: i riceventi di un determinato messaggio saranno "risvegliati" in un ordine del tutto casuale.
- Ordinati (trasmessi tramite il metodo *Context.sendOrderedBroadcast()*): sono consegnati a ciascun ricevente in un ordine stabilito tramite l'attributo "*android:priority*" del corrispettivo intent-filter all'interno del file

manifest.xml. Ciascun destinatario a turno può decidere se generare un nuovo messaggio, contenente nuovi valori, oppure se continuare o arrestare la propagazione di quello ricevuto.

Escludendo quelli resi accessibili al solo sistema operativo, ciascuna applicazione può trasmettere dei messaggi di broadcast personalizzati, in particolare qualora si dovesse far comunicare due servizi. Non avendo bisogno di stabilire determinate priorità, all'interno dell'applicazione è stato utilizzato il seguente metodo:

`Context.sendbroadcast(Intent intent)`

Parametri	
intent	Intent: L'intent da trasmettere come broadcast

Si propone il messaggio di broadcast generato ogni qualvolta l'utente pronunci la parola chiave "click":

```
Intent intent=new Intent();
intent.setAction("demo.project.hmi.demoapp.msg");
intent.putExtra("Action", "Click");
sendBroadcast(intent);
```

Per ricevere correttamente il messaggio, è necessario registrare un receiver. Questo è possibile tramite due soluzioni:

- Utilizzando direttamente il codice java:

```
@Override
public void onCreate() {
   super.onCreate();
//Registro il receiver dei messaggi di broadcast
registerReceiver(new Receiver(), new
IntentFilter("demo.project.hmi.demoapp.msg"));
}
```

- Aggiungendo un nodo all'interno del file manifest.xml:

```
<receiver android:name="Receiver" >
   <intent-filter>
      <action android:name=" demo.project.hmi.demoapp.msg " />
```

```
    </intent-filter>
</receiver>
```

A questo punto è possibile creare la classe "*Receiver*":

```
private class Receiver extends BroadcastReceiver {
    @Override
    //Funzione eseguita alla ricezione di un messaggio
    public void onReceive(Context context, Intent intent) {
        //Prende l'azione richiesta (click, swipe..)
        String action = intent.getStringExtra("Action");
        if(action == null)
            return;
        switch (action){
            //Esegue il click
            case "Click":{
                OnClick();
                break;
            }
            …
        }
    }
}
```

In questo caso è riportata una semplificazione del Receiver implementato all'interno del servizio per l'iniezione di eventi touch. Alla ricezione di un messaggio, la stringa contenuta all'interno della variabile action determina l'eventuale azione da compiere.

15. Widget

Se il riconoscimento vocale ha il vantaggio di permettere l'interazione dell'utente con l'applicazione, d'altra parte introduce dei compromessi sull'utilizzo delle risorse, in particolare del microfono. Per esempio, qualora si stesse ascoltando la musica senza l'utilizzo degli auricolari, il servizio di riconoscimento vocale potrebbe riconoscere all'interno della canzone delle parole chiave e interpretarle di conseguenza. Lo stesso problema si presenta durante le chiamate telefoniche. Per questo motivo è stato introdotto all'interno dell'applicazione un widget dotato di tre diverse funzionalità: un pulsante indietro, un tasto home e un bottone per disattivare momentaneamente il servizio vocale.

Il widget è stato realizzato tramite il seguente overlay:

```
public class OverlayService extends Service {
   private boolean isMoved = false, isOn = true;
   private WindowManager windowManager;
   private ImageView image;
   private WindowManager.LayoutParams params;
   @Override public IBinder onBind(Intent intent) {
      // Not used
      return null;
   }
   @Override public void onCreate() {
      super.onCreate();
      windowManager = (WindowManager)
getSystemService(WINDOW_SERVICE);
      image = new ImageView(this);
      image.setImageResource(R.drawable.bar);
      params = new WindowManager.LayoutParams(
            WindowManager.LayoutParams.WRAP_CONTENT,
            WindowManager.LayoutParams.WRAP_CONTENT,
            WindowManager.LayoutParams.TYPE_PHONE,
```

```
            WindowManager.LayoutParams.FLAG_NOT_FOCUSABLE,
            PixelFormat.TRANSLUCENT);
        params.gravity = Gravity.TOP | Gravity.LEFT;
        params.x = 0;
        params.y = 100;
        windowManager.addView(image, params);
```

Con il codice qui riportato, si costituisce uno strato trasparente sul quale è possibile posizionare l'immagine mostrata precedentemente (R.drawable.***bar***) sovrapposta a ogni applicazione.

Si pone l'attenzione sull'implementazione dei bottoni. Ciascun overlay supporta la visualizzazione di un'unica immagine, non consentendo l'introduzione di oggetti "Button". Tuttavia è possibile registrare un listener sul widget che calcoli le coordinate x e y in pixel alle quali è stato generato ciascun evento touch. L'immagine precedente è il risultato dell'unione di tre diverse icone, distanziate da un intervallo trasparente espresso in pixel. Per cui si noti come il tasto indietro occupi le coordinate comprese tra 130 e 200; il tasto home tra 500 e 600 mentre quello relativo al riconoscimento vocale tra 900 e 1000. Ogni qualvolta l'utente preme sul widget, conoscendo le coordinate dell'evento e la precisa posizione di ciascun'icona, è possibile determinare quale "bottone" sia stato premuto.

```
image.setOnTouchListener(new View.OnTouchListener() {
   private int initialX;
   private int initialY;
   private float initialTouchX;
   private float initialTouchY;
   @Override
//Ogni qualvolta l'utente clicca sul widget
 public boolean onTouch(View v, MotionEvent event) {
      switch (event.getAction()) {
//Contatto con lo schermo, salvo le coordinate
         case MotionEvent.ACTION_DOWN:
                ...
//Quando rilascia il dito, controllo le coordinate dell'evento
         case MotionEvent.ACTION_UP:
         //Controllo che il widget non sia stato trascinato
            if(isMoved){
               isMoved = false;
```

```
                return true;
            }
        //Se le coordinate sono comprese tra 130 e 200
            if((initialTouchX > 130)&&(initialTouchX < 200)){
                //Invio un messaggio per eseguire il tasto indietro
                Intent intent=new Intent();
                intent.setAction("demo.project.hmi.demoapp.msg");
                intent.putExtra("Action", "Back");
                sendBroadcast(intent);
        //Se invece sono comprese tra 500 e 600
            }else if((initialTouchX > 500)&&(initialTouchX < 600)){
                //invio un messaggio per eseguire il tasto home
                Intent intent=new Intent();
                intent.setAction("demo.project.hmi.demoapp.msg");
                intent.putExtra("Action", "Home");
                sendBroadcast(intent);
            }...
    }
});
```

16. Sitografia

- http://forum.xda-developers.com/showthread.php?t=2696537
- https://www.androidpit.it
- https://developer.android.com/index.html
- https://wiki.cyanogenmod.org/w/Build_for_klte
- https://coronalabs.com/blog/2014/08/26/tutorial-understanding-android-app-signing/
- https://boundarydevices.com/android-security-part-1-application-signatures-permissions/
- http://www.piwai.info/chatheads-basics/
- http://www.html.it/pag/19502/lavoriamo-in-backgroud-con-i-service/