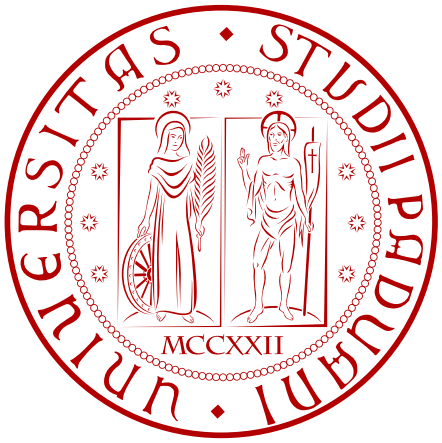
UNIVERSITÀ DEGLI STUDI DI PADOVA



Facoltà di Ingegneria
Corso di Laurea Triennale in Ingegneria dell'Informazione

A.A. 2010 - 2011

Tesina

Architettura della macchina virtuale Dalvik

Relatore

Prof. C. Fantozzi

Presentata da

Ledri Matteo 581881 - INF

Dedico questo lavoro
alla mia famiglia,
senza il cui aiuto
non sarei qui.
Grazie!

Ringraziamenti

Ringrazio in particolar modo il prof. Carlo Fantozzi per la disponibilità e la pazienza dimostrata nella fase di stesura di questo elaborato nonché per aver seguito il lavoro in maniera molto interessata.

Indice

1 Introduzione

Scopo di questa tesina é quello di descrivere le caratteristiche principali della macchina virtuale Dalvik (DVM), elemento fondamentale del sistema operativo Android, mettendola a confronto con la macchina virtuale Java (JVM).

Il **primo capitolo** introduce il concetto di macchina virtuale, ne descrive le differenti possibili realizzazioni e le caratteristiche fondamentali, spiegando il motivo del loro utilizzo.

Il **secondo capitolo** affronta il tema principale, fornendo una breve panoramica dell'architettura del sistema operativo Android, piattaforma per la quale è stata sviluppata la macchina virtuale Dalvik. Elenca poi gli obiettivi per i quali la DVM è stata progettata, fornendo una dettagliata descrizione delle soluzioni trovate.

Il **terzo capitolo** presenta il tema dal punto di vista pratico della creazione di una semplice applicazione: introduce l'ambiente software di sviluppo, descrive le classi e i metodi principali e alcune differenze con il codice Java.

L' **appendice** contiene il codice completo sia di Android che di Java dell'esempio utilizzato per l'analisi del terzo capitolo.

2 Macchina Virtuale

Una macchina virtuale è una realizzazione software, tramite emulazione software o virtualizzazione hardware, di una macchina che esegue programmi come una macchina reale.

L'utilizzo di una VM è legata al fatto di offrire un ambiente di esecuzione indipendente dalla piattaforma utilizzata (come Windows o UNIX per esempio).

La maggior parte dei linguaggi di programmazione di alto livello infatti compila codice sorgente in un codice macchina specifico per la macchina sul quale è stato compilato, con la conseguenza che il programma così ottenuto risulta non portabile: può essere eseguito unicamente da quel particolare processore o sistema operativo sul quale è stato sviluppato. In molte circostanze questo fatto è irrilevante, ma se si vuole del codice che possa funzionare con dispositivi di varia natura, con diverse risorse a disposizione, sia per tipologia che per quantità, è necessario utilizzare una macchina virtuale proprio per questa sua peculiare caratteristica. Questa filosofia corrisponde allo slogan *"Compile once, run anywhere"*. La portabilità del codice è ottenuta in due fasi:

1. **compilazione**: il file originario viene compilato ottenendo un altro file chiamato *bytecode*;

2. **esecuzione**: il *bytecode* viene interpretato dalla VM.

Una descrizione più dettagliata di quello che avviene durante queste due operazioni viene fornita nelle sezioni 3.3.2 e 3.3.3.

Esistono due famiglie di macchine virtuali, diverse per utilizzo e corrispondenza con una macchina reale: System VMs e Process VMs.

2.1 System VMs

Utilizzate prevalentemente in ambito server, simulano un sistema hardware completo e supportano l'esecuzione di più sistemi operativi contemporaneamente.

2.2 Process VMs

Supportano l'esecuzione di un singolo processo alla volta. É possibile creare più esemplari di process VM per permettere l'esecuzione contemporanea di più applicazioni associate a più processi.

Un esemplare di questo tipo di VM è creato quando il corrispondente processo inizia e distrutta quando il corrispondente processo termina.

Il suo principale obiettivo, e il motivo per cui si utilizza, è quello di fornire la già menzionata indipendenza dalla piattaforma.

Le macchina virtuali Java e Dalvik appartengono entrambe a questa categoria.

3 Dalvik Virtual Machine

3.1 Android O.S.

Figura 3.1.1: Logo ufficiale

Sistema operativo open source specificatamente ideato per dispositivi mobili, fu un progetto sviluppato da Android Inc., acquisita poi da Google nel 2005, e ufficialmente presentato al pubblico il 5 Novembre 2007 dall'Open Handset Alliance.

Basato sul kernel Linux, oggi alla versione 2.6, offre un ambiente completo all'esecuzione di applicazioni che devono funzionare in ambienti *embedded*, e quindi su dispositivi dotati di:

- **processore** dalle prestazioni limitate;
- **memoria** RAM non molto estesa;
- **batteria** dall'autonomia ridotta.

Con l'uscita sul mercato degli ultimi smartphone questi limiti prestazionali sono diventati via via meno stringenti. Sin dagli anni '50 è valida infatti la *legge di Moore*, secondo cui la velocità di miglioramento dei componenti, così come quella di abbattimento dei costi, è esponenziale. I dispositivi mobili ora sono dotati di processori sempre più veloci, in alcuni casi anche multicore, RAM oltre il gigabyte di memoria e batterie sempre più efficienti.

Agli inizi del progetto gli uomini di Google si sono scontrati col problema di avere risorse limitate a disposizione e hanno progettato un sistema *ad hoc*. Ora lavorano costantemente per aggiornare il sistema con nuove release per stare al passo di questa forte evoluzione che interessa tutti i settori legati all'elettronica.

La versione di Android è così giunta alla 2.3.4 per gli smartphone e alla 3.2 per i tablet pc.

Nella figura 3.1.2 si osserva la struttura tipica del sistema operativo Android [1]: lo strato basilare è il kernel Linux, utilizzato come *hardware instruction layer* e scelto in quanto fornisce un insieme di driver affidabili. Su questo primo strato poggiano le librerie, scritte in C, in cui si possono individuare componenti indispensabili per l'interazione dell'utente con il sistema tramite il touchscreen del dispositivo, per la grafica,

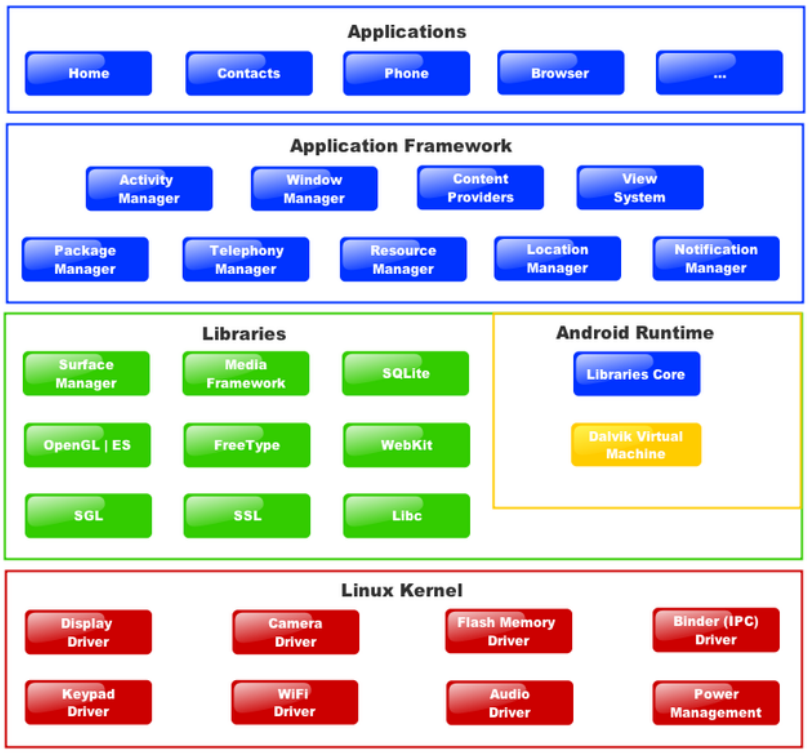


Figura 3.1.2: Diagramma dell'architettura di Android

sia 2d che 3d, e di tutto quello che riguarda la multimedialità e l'immagazzinamento dei dati in memoria. L'Application Framework, scritto interamente in Java, contiene l'ossatura di qualsiasi applicazione, come l'*activity manager*, il *package manager* e il *content manager* che ne permettono la supervisione e lo scambio di informazioni da una all'altra. Lo strato più elevato è quello in cui vengono eseguite le applicazioni lanciate dall'utente. Il cuore del sistema è però costituito dall'Android Runtime, contenente la Dalvik Virtual Machine, così chiamata in onore della città natale del suo ideatore, Dan Bornstein.

3.2 Obiettivi di progetto

Gli obiettivi prefissi nella realizzazione della DVM sono:

- il tempo di esecuzione del codice deve essere il minore possibile a parità di risorse hardware;
- il bytecode deve avere una struttura che ottimizzi l'utilizzo di memoria;
- l'overhead nel lancio di una nuova applicazione deve essere minimo.

3.3 Caratteristiche

La decisione di adottare per il sistema operativo Android una macchina virtuale diversa dalla JVM nasce dal dover sopperire alle limitazioni imposte dalla tipologia di dispositivi su cui deve essere eseguita, come menzionato nel paragrafo 3.1. La sua effettiva necessità viene in questo capitolo valutata analizzando le principali scelte progettuali.

Dagli obiettivi stabiliti per la progettazione della DVM citati nel paragrafo 3.2 discendono le caratteristiche che distinguono maggiormente la DVM dalla JVM:

- è register - based;

- utilizza un diverso bytecode.

Altre caratteristiche sono il compilatore *Just In Time* (JIT), introdotto però solo nelle versioni più recenti di Android, e lo Zygote.

3.3.1 Register - based

Come accennato in precedenza, la principale differenza tra la DVM e la JVM è che, al contrario di quest'ultima che è stack - based, la macchina virtuale Dalvik è register - based.

Negli anni passati è stato molto acceso il dibattito su quali delle due architetture fosse la migliore, con esponenti autorevoli in entrambi i fronti che cercavano di distinguersi dalla concorrenza a forza di dati e statistiche. Ancora oggi però la questione non è ben definita, motivo per cui entrambe le architetture sono tuttora in uso.

alto livello	*register − based*	*stack − based*
c = a + b	sum a, b, c	push(a)
		push(b)
		sum()

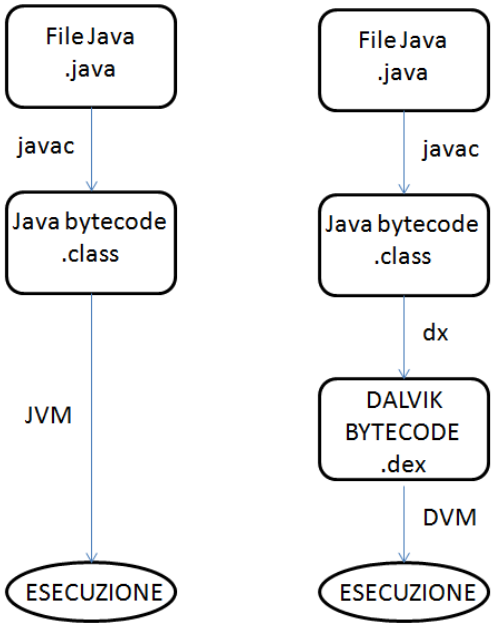
Tabella 3.3.1.1: Semplice esempio che mostra la sostanziale differenza nelle istruzioni tra macchine register - based e stack - based.

Uno studio condotto sul comportamento di diverse VM mostra che l'architettura register - based richiede in media il 47% in meno di istruzioni eseguite dalla VM rispetto all'architettura stack - based e che le istruzioni eseguite da quest'ultima sono mediamente il 25% più corte perché un operando è sempre implicito, come evidenziato dal semplice esempio in tabella 3.3.1.1. Considerando però che la lunghezza media superiore delle istruzioni di una VM register - based comporta per una macchina reale solo l'1.07% di lavoro in più per ogni istruzione della VM, in totale una VM register - based impiega il 32.3% di tempo in meno rispetto ad una VM stack - based per l'esecuzione di benchmark standard [4].

A fronte di questo risultato si capisce la scelta, nella progettazione della macchina virtuale Dalvik, dell'architettura basata sui registri e non su uno stack.

3.3.2 Dex bytecode

Una seconda importante differenza rispetto alla JVM è la seguente: come descritto nel paragrafo 2.2, la popolarità del linguaggio Java sta nella portabilità del codice, proprietà ottenuta mediante la traduzione del codice originario Java nel bytecode. La DVM, discendendo dalla JVM, garantisce la stessa proprietà anche se in modo diverso: il bytecode di Java viene convertito, attraverso uno strumento chiamato *dx*, in un nuovo bytecode, ottenendo un file .dex, come mostrato in figura 3.3.2.3.



(a) Java (b) Android

Figura 3.3.2.1: Confronto tra JVM e DVM

La decisione di lavorare con un bytecode differente da quello interpretato dalla JVM deriva dalla necessità di risparmiare memoria dati i limiti fisici dei primi dispositivi sui quali il sistema operativo Android veniva installato.

Come si osserva nella figura 3.3.2.2, questo risultato è ottenuto mediante una riorganizzazione del file .jar secondo il principio della condivisione. Il file .jar contiene infatti un file .class per ogni classe in cui è suddiviso il codice, con la conseguenza che non vi è una struttura unitaria comportando una frequente ripetizione di oggetti che invece potrebbero essere condivisi più efficientemente evitando uno spreco di risorse di memoria. Partendo da questa situazione, il compilatore che si invoca attraverso il comando *dx* crea un file .dex dove la divisione in classi non è più presente, sostituita da una struttura organizzata unicamente per tipologia di oggetti di codice, più flessibile e snella, come risulta dalla figura 3.3.2.2.

Questa riorganizzazione ha effettivamente un impatto positivo sulle dimensioni di un singolo file: prendendo come riferimento la dimensione di un file .jar non compresso,

un file .dex non compresso ha infatti dimensione di qualche punto percentuale inferiore a quella di un file .jar compresso [2].

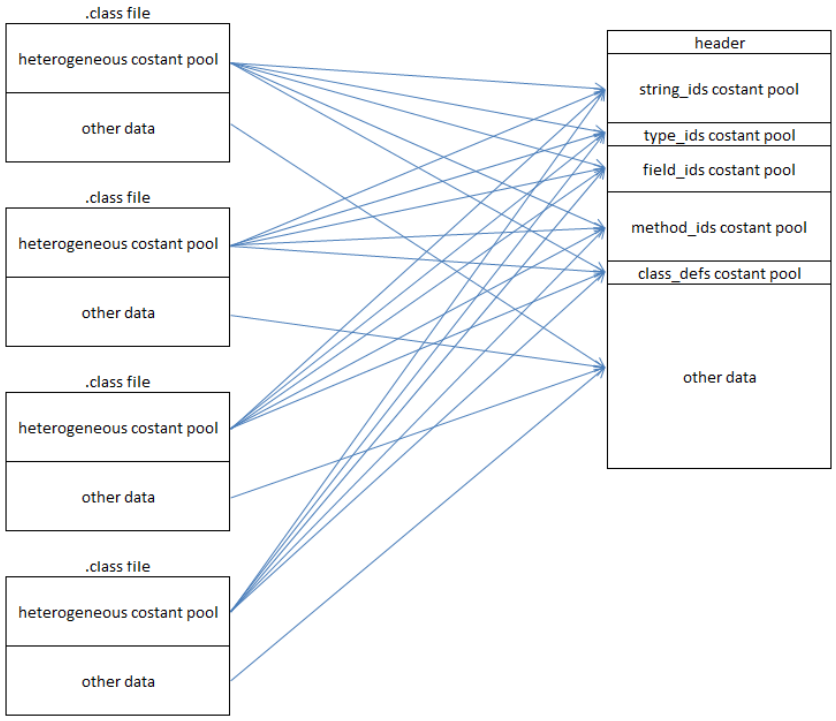


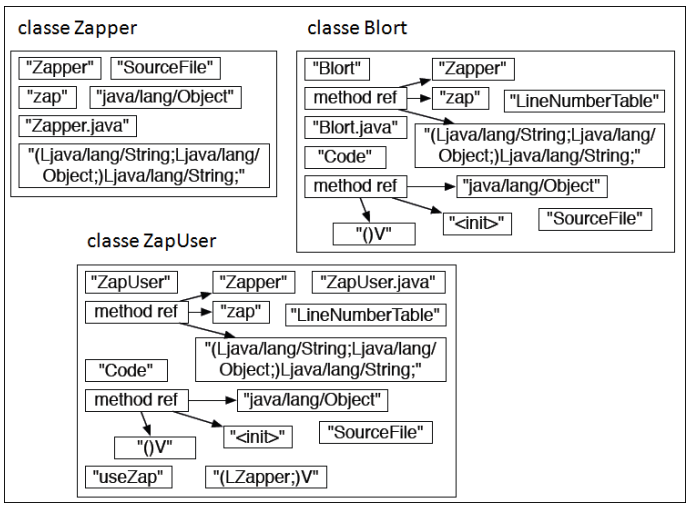
Figura 3.3.2.2: Confronto tra l'anatomia di un file .dex, a destra, e un file .jar, a sinistra.

La diversità nella struttura dei due file .jar e .dex è evidente nel seguente esempio, preso dalla presentazione di Dan Bornstein al Google I/O del 2008.

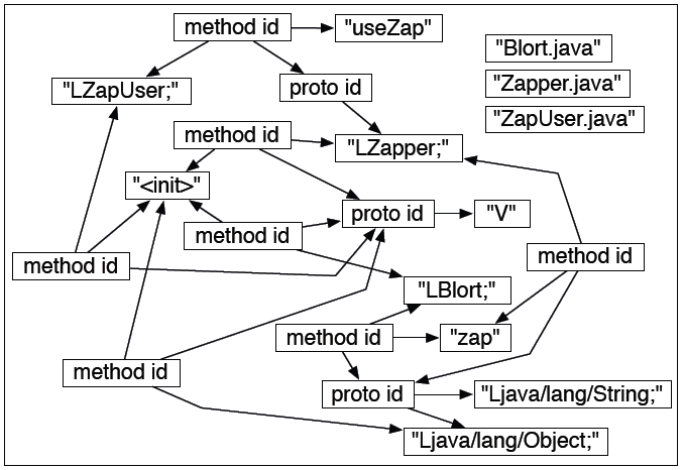
```
public interface Zapper{
        public String zap (String s, Object o);
}
public class Blort implements Zapper{
        public String zap (String s, Object o);
        .....;
}
public class ZapUser{
        public void useZap (Zapper z){
                z.zap (.....)
        }
}
```

Codice 3.3.2.1: Esempio di codice Java preso dalla presentazione di Dan Bornstein al Google I/O del 2008

A seguito della compilazione di questo codice dal compilatore *javac* da una parte e *dx* dall'altra si ottengono infatti risultati molto diversi:



(a) File .jar



(b) File .dex

Figura 3.3.2.3: Confronto tra la struttura di un file prima e dopo la compilazione con *dx*

3.3.3 JIT

Durante l'esecuzione di un'applicazione la DVM interpreta un'istruzione del bytecode alla volta, ma non tutto il codice deve essere compilato: come descritto nel paragrafo 3.1 infatti le librerie, per esempio, sono già scritte in codice macchina e quindi in pratica solo una porzione di tutto il codice deve essere interpretato durante l'esecuzione. Alcune ricerche dimostrano infatti che meno di un terzo del tempo utilizzato per l'esecuzione è speso dall'interprete, il resto è impiegato per l'esecuzione di codice già in linguaggio macchina e ottimizzato. Questo unito alla rappresentazione di per se molto compatta e ottimizzata del bytecode sono i motivi per cui fino alla versione 2.2 di Android, denominata Froyo, si è deciso di non utilizzare un compilatore JIT. Questa scelta non era però perfetta: l'assenza di un compilatore JIT nell'esecuzione di appli-

cazioni ad elevato tasso computazionale comportava un rallentamento dell'esecuzione nell'ordine del 5%, 10% [3].

Un compilatore JIT affianca il lavoro svolto dall'interprete identificando le porzioni di codice maggiormente impegnative dal punto di vista computazionale, cioè quelle parti in cui l'interprete da solo sarebbe abbastanza inefficiente, le compila in codice macchina e le ottimizza. In questo modo ad ogni successiva esecuzione di queste porzioni di codice, esse si troveranno già tradotte in linguaggio macchina aumentando notevolmente le prestazioni.

Cosa compilare e quando sono i due fattori principali nella progettazione del compilatore JIT: si può compilare per esempio quando si installa l'applicazione, quando si invoca per la prima volta un metodo e si può compilare l'intero programma, una pagina fisica, un metodo. Gli obiettivi definiti per il compilatore JIT di Android e su cui è stata basata la scelta sono i seguenti:

- utilizzare la minor quantità di memoria;
- l'utente deve avere un miglioramento delle prestazioni immediato senza attendere troppo a lungo l'avvio di un'applicazione;
- le transizioni tra l'esecuzione del codice dall'interprete e l'esecuzione diretta di codice già compilato deve essere semplice.

Sul mercato sono presenti due valide soluzioni di compilatori JIT: il *method based JIT* e il *trace based JIT*.

Il method based JIT è la soluzione adottata da Java. Il suo funzionamento è il seguente: identifica i metodi maggiormente computazionali, li compila in codice nativo e lo ottimizza. Questa soluzione ha lo svantaggio che viene compilato tutto il metodo, comprese quelle porzioni di codice raramente eseguite, comportando uno spreco di tempo per la traduzione di codice che salvo particolari eccezioni non verrà mai eseguito.

Il trace based JIT veniva utilizzato largamente negli anni '90 nella virtualizzazione software. A differenza del method based, individua solo un insieme di istruzioni altamente computazionali, raggruppandole in tracce, le compila e le ottimizza. Queste tracce compilate vengono collegate tra loro nella *translation cache*: l'esecuzione avviene di traccia in traccia. Questo metodo sopperisce allo svantaggio del metodo precedente perché l'analisi delle istruzioni ignora i casi eccezionali. Un altro vantaggio è che il salto tra interpretazione ed esecuzione di codice compilato è molto frequente: questo comporta che se l'assunzione non era corretta e bisogna eseguire un'altra porzione di codice è molto semplice tornare dall'interprete e lasciare a questo l'esecuzione. I principali svantaggi di questo metodo sono due: il guadagno prestazionale massimo in termini assoluti è minore rispetto al method based ed è difficile condividere la translation cache tra processi.

Per chiarire la differenza tra i due metodi si riporta l'esempio tratto da [3], in cui si analizza il processo *system_server* del sistema operativo Android: l' 8% del codice del programma corrisponde a hot methods e il 2% a hot traces, ovvero al 26% degli hot methods.

A partire da Froyo quindi è stato scelto un compilatore trace JIT perché, come si evince dal precedente esempio, raggiunge maggiormente gli obiettivi garantendo un rapporto tra prestazioni e costo di compilazione maggiore.

Non è comunque esclusa in futuro l'integrazione dei due metodi in quanto possono coesistere:

- trace JIT durante l'alimentazione batteria;
- method JIT in background durante la ricarica.

3.3.4 Zygote

Nel paragrafo 3.1 si è accennato all'importanza di avere un overhead ridotto in fase di lancio di una nuova applicazione per garantire una pronta risposta del dispositivo. Per ottenere questo risultato nelle ultime versioni di Android si è debitamente progettato un compilatore JIT, come visto nel paragrafo 3.3.3.

Essendo la DVM una macchina virtuale di tipo process VM, un'esemplare di questa VM può eseguire un unico processo e affinché possano essere contemporaneamente in esecuzione più applicazioni è necessario creare più esemplari di VM. La soluzione presente sin dalle prime versioni del sistema operativo quindi consiste nel velocizzare l'inizializzazione di nuovi esemplari di VM e lo fa attraverso lo Zygote, un processo del sistema operativo inizializzato in fase di avvio di Android dal processo *init*.

Come si osserva dalla figura 3.3.4.4 il processo Zygote rimane in esecuzione in background. Quando deve essere lanciata una nuova applicazione, il processo esegue il fork: uno dei due processi creati rimane in esecuzione in background mentre l'altro inizializza un nuovo esemplare di DVM. Se durante l'esecuzione di questa applicazione deve essere lanciata una ulteriore applicazione, il processo in background esegue un secondo fork. In ogni momento è quindi disponibile un processo Zygote per l'inizializzazione di una nuova DVM.

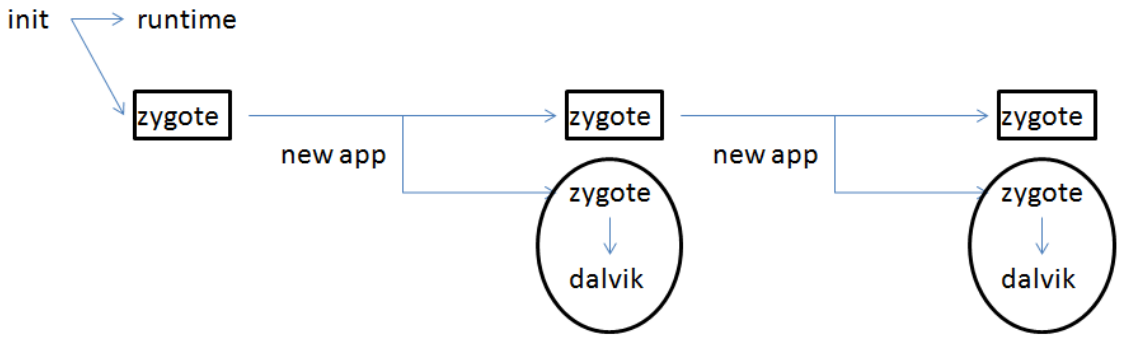


Figura 3.3.4.4: Lancio di una nuova applicazione : il fork di Zygote e la creazione di un nuovo esemplare di DVM

4 Progetto

Viene presentato ora un mini-progetto da utilizzare come semplice esempio per la creazione di applicazioni destinate ad essere eseguite su sistemi dotati del sistema operativo Android.

4.1 Ambiente di sviluppo

Una guida dettagliata per l'installazione di un ambiente di sviluppo per la programmazione e l'esecuzione del codice si trova in [5].

4.2 Pacchetti e metodi

Una guida all'API di Android si trova in [6].
I pacchetti principali utilizzati sono:

1. *android.view.View*;
2. *android.app.Activity*.

4.2.1 View

Questa classe rappresenta il mattone fondamentale nella costruzione di una UI. Un esemplare di questa classe occupa una porzione rettangolare dello schermo ed è responsabile nel disegnare le componenti della UI e nella gestione di eventi legati all'interazione dell'applicazione con l'utente. Queste interazioni sono rese possibili tramite la presenza di apposite interfacce nidificate, chiamate *event listeners*, definite proprio sugli oggetti di questa classe.

4.2.2 Activity

La classe Activity si prende cura di creare una finestra in cui il programmatore può inserire la UI attraverso la chiamata del metodo *setContentView(View)* di un esemplare della classe View.

Un'applicazione può consistere in una sola attività, o può contenerne più di una. Cosa sono le varie attività e come esse sono collegate tra loro dipende, ovviamente, dall'applicazione e dalla sua struttura. Generalmente una delle attività è definita come la principale, e viene presentata all'utente quando l'applicazione viene lanciata. Lo spostamento tra un'attività e l'altra è realizzato attraverso una chiamata da parte della prima alla seconda. Questo è reso possibile grazie all'utilizzo di uno stack: quando una nuova attività inizia viene posizionata in cima allo stack e diventa quella in esecuzione, mentre l'attività precedente non torna in primo piano fino al termine di quella nuova. Un'attività può quindi trovarsi in ogni momento in uno dei tre seguenti stati, come mostrato in figura 4.2.2.1:

- **running**, se è in primo piano;

- **paused**, se è visibile ma ha perso il focus;

- **stopped**, se non è più visibile.

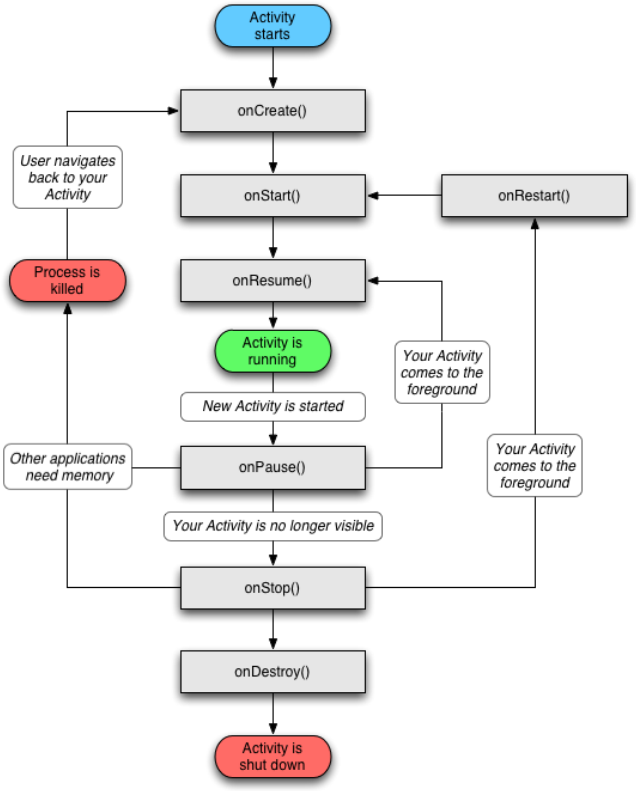


Figura 4.2.2.1: Ciclo di vita di un esemplare della classe Activity

Se un'attività si trova in uno degli ultimo due stati può essere rimossa dalla memoria dal sistema operativo in caso di necessità di risorse per l'applicazione in esecuzione. Questa rimozione viene gestita in due modi: o il sistema operativo costringe l'attività da rimuovere a terminare in breve tempo o semplicemente ne sopprime il processo. In ogni caso quando sarà visualizzata nuovamente dall'utente dovrà essere completamente reinizializzata e riportata al suo stato precedente.

Per quanto riguarda i metodi, quello principale è *onCreate ()*. Viene chiamato quando viene inizializzato un nuovo esemplare della classe Activity ed è dove vengono eseguite tutte le inizializzazioni per la gestione della UI.

4.3 Confronto con Java

La semplice applicazione in esempio consiste in un simulatore di una schedina del TotoCalcio, il cui codice è riportato in appendice.

Tutti gli oggetti della classe View sono stati definiti in un file .xml in modo tale da ottenere una separazione logica tra l'interfaccia grafica e le poche righe di codice

necessarie alla gestione della simulazione legata alla pressione di un bottone da parte dell'utente.

In appendice si può osservare anche il codice riscritto per essere compilato con una normale JVM, in cui non si è utilizzato un file .xml.

La differenza principale e più evidente tra il codice scritto per la DVM rispetto a quello scritto per la JVM è che in quest'ultimo, come in tutte le applicazioni scritte in Java puro, è necessaria la presenza di un file contenente il metodo *main*, attraverso cui avviene l'inizializzazione dell'applicazione stessa. Come sottolineato precedentemente nella sezione 4.2.2, in Android questo lavoro è svolto dal metodo *onCreate*, che quindi formalmente sostituisce il metodo *main* di Java.

Per ulteriori esempi si veda [7]

A Codice Java specifico per Android

A.1 File .xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:orientation="vertical" >
<TextView android:id="@+id/title"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="center"
        android:textColor="#FF0000"
        android:textStyle="bold"
        android:textSize="20px"
        android:text="Risultati ultima giornata"
        android:layout_marginBottom="60px"/>
<LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="left" >
        <TextView android:id="@+id/primo"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginLeft="65px"
                android:text="First match....." />
        <LinearLayout
                android:orientation="horizontal"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:gravity="center" >
                <TextView android:id="@+id/primoris"
                        android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
                        android:layout_marginRight="10px"
                        android:textColor="#FF0000"
                        android:text="-" />
        </LinearLayout>
</LinearLayout>
<LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:gravity="left" >
```

```
44          <TextView android:id="@+id/secondo"
                    android:layout_width="wrap_content"
46                  android:layout_height="wrap_content"
                    android:layout_marginLeft="65px"
48                  android:text="Second_match....." />
            <LinearLayout
50                  android:orientation="horizontal"
                    android:layout_width="fill_parent"
52                  android:layout_height="wrap_content"
                    android:gravity="center" >
54                  <TextView android:id="@+id/secondoris"
                            android:layout_width="wrap_content"
56                          android:layout_height="wrap_content"
                            android:layout_marginRight="10px"
58                          android:textColor="#FF0000"
                            android:text="-" />
60          </LinearLayout>
    </LinearLayout>
62  <LinearLayout
            android:orientation="horizontal"
64          android:layout_width="fill_parent"
            android:layout_height="wrap_content"
66          android:gravity="left" >
            <TextView android:id="@+id/terzo"
68                  android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
70                  android:layout_marginLeft="65px"
                    android:text="Third_match....." />
72          <LinearLayout
                    android:orientation="horizontal"
74                  android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
76                  android:gravity="center" >
                    <TextView android:id="@+id/terzoris"
78                          android:layout_width="wrap_content"
                            android:layout_height="wrap_content"
80                          android:layout_marginRight="10px"
                            android:textColor="#FF0000"
82                          android:text="-" />
            </LinearLayout>
84  </LinearLayout>
    <LinearLayout
86          android:orientation="horizontal"
            android:layout_width="fill_parent"
88          android:layout_height="wrap_content"
            android:gravity="left" >
90          <TextView android:id="@+id/quarto"
```

```
                android:layout_width="wrap_content"
92              android:layout_height="wrap_content"
                android:layout_marginLeft="65px"
94              android:text="Fourth match....." />
        <LinearLayout
96              android:orientation="horizontal"
                android:layout_width="fill_parent"
98              android:layout_height="wrap_content"
                android:gravity="center" >
100             <TextView android:id="@+id/quartoris"
                        android:layout_width="wrap_content"
102                     android:layout_height="wrap_content"
                        android:layout_marginRight="10px"
104                     android:textColor="#FF0000"
                        android:text="-" />
106     </LinearLayout>
    </LinearLayout>
108 <LinearLayout
        android:orientation="horizontal"
110     android:layout_width="fill_parent"
        android:layout_height="wrap_content"
112     android:gravity="left" >
        <TextView android:id="@+id/quinto"
114             android:layout_width="wrap_content"
                android:layout_height="wrap_content"
116             android:layout_marginLeft="65px"
                android:text="Fifth match....." />
118     <LinearLayout
                android:orientation="horizontal"
120             android:layout_width="fill_parent"
                android:layout_height="wrap_content"
122             android:gravity="center" >
                <TextView android:id="@+id/quintoris"
124                     android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
126                     android:layout_marginRight="10px"
                        android:textColor="#FF0000"
128                     android:text="-" />
        </LinearLayout>
130 </LinearLayout>
    <LinearLayout
132     android:orientation="horizontal"
        android:layout_width="fill_parent"
134     android:layout_height="wrap_content"
        android:gravity="left" >
136     <TextView android:id="@+id/sesto"
                android:layout_width="wrap_content"
```

```
138                 android:layout_height="wrap_content"
                    android:layout_marginLeft="65px"
140                 android:text="Sixth match....." />
            <LinearLayout
142                 android:orientation="horizontal"
                    android:layout_width="fill_parent"
144                 android:layout_height="wrap_content"
                    android:gravity="center" >
146                 <TextView android:id="@+id/sestoris"
                            android:layout_width="wrap_content"
148                         android:layout_height="wrap_content"
                            android:layout_marginRight="10px"
150                         android:textColor="#FF0000"
                            android:text="-" />
152         </LinearLayout>
    </LinearLayout>
154 <LinearLayout
            android:orientation="horizontal"
156         android:layout_width="fill_parent"
            android:layout_height="wrap_content"
158         android:gravity="left" >
            <TextView android:id="@+id/settimo"
160                 android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
162                 android:layout_marginLeft="65px"
                    android:text="Seventh match....." />
164         <LinearLayout
                    android:orientation="horizontal"
166                 android:layout_width="fill_parent"
                    android:layout_height="wrap_content"
168                 android:gravity="center" >
                    <TextView android:id="@+id/settimoris"
170                         android:layout_width="wrap_content"
                            android:layout_height="wrap_content"
172                         android:layout_marginRight="10px"
                            android:textColor="#FF0000"
174                         android:text="-" />
            </LinearLayout>
176 </LinearLayout>
    <LinearLayout
178         android:orientation="horizontal"
            android:layout_width="fill_parent"
180         android:layout_height="wrap_content"
            android:gravity="left" >
182         <TextView android:id="@+id/ottavo"
                    android:layout_width="wrap_content"
184                 android:layout_height="wrap_content"
```

```
                android:layout_marginLeft="65px"
186             android:text="Eigth match....." />
        <LinearLayout
188             android:orientation="horizontal"
                android:layout_width="fill_parent"
190             android:layout_height="wrap_content"
                android:gravity="center" >
192             <TextView android:id="@+id/ottavoris"
                        android:layout_width="wrap_content"
194                     android:layout_height="wrap_content"
                        android:layout_marginRight="10px"
196                     android:textColor="#FF0000"
                        android:text="-" />
198     </LinearLayout>
    </LinearLayout>
200 <LinearLayout
        android:orientation="horizontal"
202     android:layout_width="fill_parent"
        android:layout_height="wrap_content"
204     android:gravity="left" >
        <TextView android:id="@+id/nono"
206             android:layout_width="wrap_content"
                android:layout_height="wrap_content"
208             android:layout_marginLeft="65px"
                android:text="Ninenth match....." />
210     <LinearLayout
                android:orientation="horizontal"
212             android:layout_width="fill_parent"
                android:layout_height="wrap_content"
214             android:gravity="center" >
                <TextView android:id="@+id/nonoris"
216                     android:layout_width="wrap_content"
                        android:layout_height="wrap_content"
218                     android:layout_marginRight="10px"
                        android:textColor="#FF0000"
220                     android:text="-" />
        </LinearLayout>
222 </LinearLayout>
    <LinearLayout
224     android:orientation="horizontal"
        android:layout_width="fill_parent"
226     android:layout_height="wrap_content"
        android:gravity="left" >
228     <TextView android:id="@+id/decimo"
                android:layout_width="wrap_content"
230             android:layout_height="wrap_content"
                android:layout_marginLeft="65px"
```

```
232                 android:text="Tenth match....." />
        <LinearLayout
234                 android:orientation="horizontal"
                    android:layout_width="fill_parent"
236                 android:layout_height="wrap_content"
                    android:gravity="center" >
238                 <TextView android:id="@+id/decimoris"
                            android:layout_width="wrap_content"
240                         android:layout_height="wrap_content"
                            android:layout_marginRight="10px"
242                         android:textColor="#FF0000"
                            android:text="-" />
244     </LinearLayout>
</LinearLayout>
246 <Button android:id="@+id/button"
        android:layout_gravity="center"
248     android:layout_width="wrap_content"
        android:layout_height="wrap_content"
250     android:layout_marginTop="80px"
        android:text="Simulate" />
252 </LinearLayout>
```

A.2 File .java

```
1
package prova.prima;
3
import android.app.Activity;
5 import android.os.Bundle;
import android.view.View;
7 import android.view.View.OnClickListener;
import android.widget.Button;
9 import android.widget.TextView;

11 public class Prova<EditView> extends Activity {
    // Called when the activity is first created.
13  @Override
    public void onCreate(Bundle savedInstanceState) {
15      super.onCreate(savedInstanceState);
        //richiama il file .xml dove sono definiti gli oggetti grafici
17      setContentView(R.layout.main);
        //inizializzazione degli oggetti tramite riferimento id dal file .xml
19      final TextView primo = (TextView) findViewById(R.id.primoris);
        final TextView secondo = (TextView) findViewById(R.id.secondoris);
21      final TextView terzo = (TextView) findViewById(R.id.terzoris);
        final TextView quarto = (TextView) findViewById(R.id.quartoris);
23      final TextView quinto = (TextView) findViewById(R.id.quintoris);
```

```
        final TextView sesto = (TextView) findViewById(R.id.sestoris);
25      final TextView settimo = (TextView) findViewById(R.id.settimoris);
        final TextView ottavo = (TextView) findViewById(R.id.ottavoris);
27      final TextView nono = (TextView) findViewById(R.id.nonoris);
        final TextView decimo = (TextView) findViewById(R.id.decimoris);
29      final Button button = (Button) findViewById(R.id.button);
        //NUM e' il numero di partite
31      final int NUM = 10;
        //LB e UB sono i limiti di decisione per i risultati delle partite
33      final double LB = 0.3;
        final double UB = 0.7;
35      //OnClickListener perche' si interagisce con il pulsante tramite pressione dello stesso
        button.setOnClickListener(new OnClickListener()
37      {
                //metodo onClick chiamato in caso di pressione del pulsante
39              public void onClick(View v)
                {
41                      //generazione di risultati casuali delle partite
                        double[] tmp = new double[NUM];
43                      String[] tmp2 = new String[NUM];
                        for (int i=0 ; i < NUM ; i++ )
45                              tmp[i] = Math.random();
                        for (int i=0 ; i < NUM ; i++ )
47                      {
                                if (tmp[i] < LB) tmp2[i] = "1";
49                              else if (tmp[i] < UB) tmp2[i] = "X";
                                else tmp2[i] = "2";
51                      }
                        //assegnazione dei risultati alle partite
53                      primo.setText(tmp2[0]);
                        secondo.setText(tmp2[1]);
55                      terzo.setText(tmp2[2]);
                        quarto.setText(tmp2[3]);
57                      quinto.setText(tmp2[4]);
                        sesto.setText(tmp2[5]);
59                      settimo.setText(tmp2[6]);
                        ottavo.setText(tmp2[7]);
61                      nono.setText(tmp2[8]);
                        decimo.setText(tmp2[9]);
63              }
        });
65  }
}
```

B Codice Java

```
1
package prova.seconda;
3
import javax.swing.JFrame;
5 import javax.swing.JPanel;
import javax.swing.JLabel;
7 import javax.swing.JButton;
import java.awt.event.ActionListener;
9 import java.awt.event.ActionEvent;
import java.awt.Container;
11 import java.awt.BorderLayout;
import java.awt.GridLayout;
13
public class ProvaJava extends JFrame implements ActionListener
15 {

17         protected JLabel titolo, primo, primoris, secondo, secondoris, terzo, terzoris ;
           protected JLabel quarto, quartoris, quinto, quintoris, sesto, sestoris, settimo, settimoris ;
19         protected JLabel ottavo, ottavoris, nono, nonoris, decimo, decimoris;

21         public ProvaJava()
     {
23         setTitle("Totocalcio");
           setDefaultCloseOperation(EXIT_ON_CLOSE);
25         setExtendedState(MAXIMIZED_BOTH);

27         //crea le componenti del pannello
           titolo = new JLabel("Risultati␣ultima␣giornata");
29         primo = new JLabel("Primo␣match");
           primoris = new JLabel("-");
31         secondo = new JLabel("Secondo␣match");
           secondoris = new JLabel("-");
33         terzo = new JLabel("Terzo␣match");
           terzoris = new JLabel("-");
35         quarto = new JLabel("Quarto␣match");
           quartoris = new JLabel("-");
37         quinto = new JLabel("Quinto␣match");
           quintoris = new JLabel("-");
39         sesto = new JLabel("Sesto␣match");
           sestoris = new JLabel("-");
41         settimo = new JLabel("Settimo␣match");
           settimoris = new JLabel("-");
43         ottavo = new JLabel("Ottavo␣match");
           ottavoris = new JLabel("-");
45         nono = new JLabel("Nono␣match");
```

```
        nonoris = new JLabel("-");
47      decimo = new JLabel("Decimo match");
        decimoris = new JLabel("-");
49      JButton simulazione = new JButton("Simula");
        //aggiunge simulazione agli oggetti che richiedono la gestione di eventi
51      //derivanti da azioni dell'utente.
        //Quando l'evento accade, viene invocato il metodo actionPerformed
53      simulazione.addActionListener(this);

55      //aggiunge le componenti ai pannelli p1, p2, p3
        JPanel p1 = new JPanel();
57      p1.add(titolo);

59      JPanel p2 = new JPanel();
        p2.setLayout(new GridLayout(10, 2));
61      p2.add(primo);
        p2.add(primoris);
63      p2.add(secondo);
        p2.add(secondoris);
65      p2.add(terzo);
        p2.add(terzoris);
67      p2.add(quarto);
        p2.add(quartoris);
69      p2.add(quinto);
        p2.add(quintoris);
71      p2.add(sesto);
        p2.add(sestoris);
73      p2.add(settimo);
        p2.add(settimoris);
75      p2.add(ottavo);
        p2.add(ottavoris);
77      p2.add(nono);
        p2.add(nonoris);
79      p2.add(decimo);
        p2.add(decimoris);
81
        JPanel p3 = new JPanel();
83      p3.add(simulazione);

85      //aggiunge i pannelli al frame
        Container c = getContentPane();
87      c.setLayout(new BorderLayout());
        c.add(p1, BorderLayout.NORTH);
89      c.add(p2, BorderLayout.CENTER);
        c.add(p3, BorderLayout.SOUTH);
91
    }
```

```
93
        //metodo chiamato quando accade l'evento e che lo gestisce
95      public void actionPerformed(ActionEvent e)
        {
97              //NUM e' il numero di partite
        final int NUM = 10;
99      //LB e UB sono i limiti di decisione per i risultati
        final double LB = 0.3;
101     final double UB = 0.7;
                double[] tmp = new double[NUM];
103             String[] tmp2 = new String[NUM];
                for (int i=0 ; i < NUM ; i++ )
105                     tmp[i] = Math.random();
                for (int i=0 ; i < NUM ; i++ )
107             {
                        if (tmp[i] < LB) tmp2[i] = "1";
109                     else if (tmp[i] < UB) tmp2[i] = "X";
                        else tmp2[i] = "2";
111             }
                //assegnazione dei risultati alle partite
113             primoris.setText(tmp2[0]);
                secondoris.setText(tmp2[1]);
115             terzoris.setText(tmp2[2]);
                quartoris.setText(tmp2[3]);
117             quintoris.setText(tmp2[4]);
                sestoris.setText(tmp2[5]);
119             settimoris.setText(tmp2[6]);
                ottavoris.setText(tmp2[7]);
121             nonoris.setText(tmp2[8]);
                decimoris.setText(tmp2[9]);
123     }

125 public static void main(String[] args)
    {
127     ProvaJava totocalcio = new ProvaJava();
        totocalcio.setVisible(true);
129 }

131 }
```

Riferimenti bibliografici

[1] Introduzione all'architettura di Android O.S.: http://developer.android.com/videos/index.html#v=QBGfUs9mQYY

[2] Breve presentazione della DVM a cura di Dan Bornstein: http://sites.google.com/site/io/dalvik-vm-internals

[3] Breve presentazione del compilatore JIT: http://www.youtube.com/watch?v=Ls0tM-c4Vfo&feature=player_embedded

[4] Approfondimento su alcune tematiche: http://imsciences.edu.pk/serg/wp-content/uploads/2009/07/Analysis-of-Dalvik-VM.pdf

[5] Guida all'installazione dell'ambiente de lavoro: http://developer.android.com/sdk/index.html

[6] Android API: http://developer.android.com/reference/packages.html

[7] Altri esempi pratici: http://developer.android.com/resources/samples/index.html