UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Facoltà di Ingegneria
Corso di Studi in Ingegneria Informatica

Tesi di laurea triennale

Strumento di supporto alla configurazione automatica del test di applicazioni Android

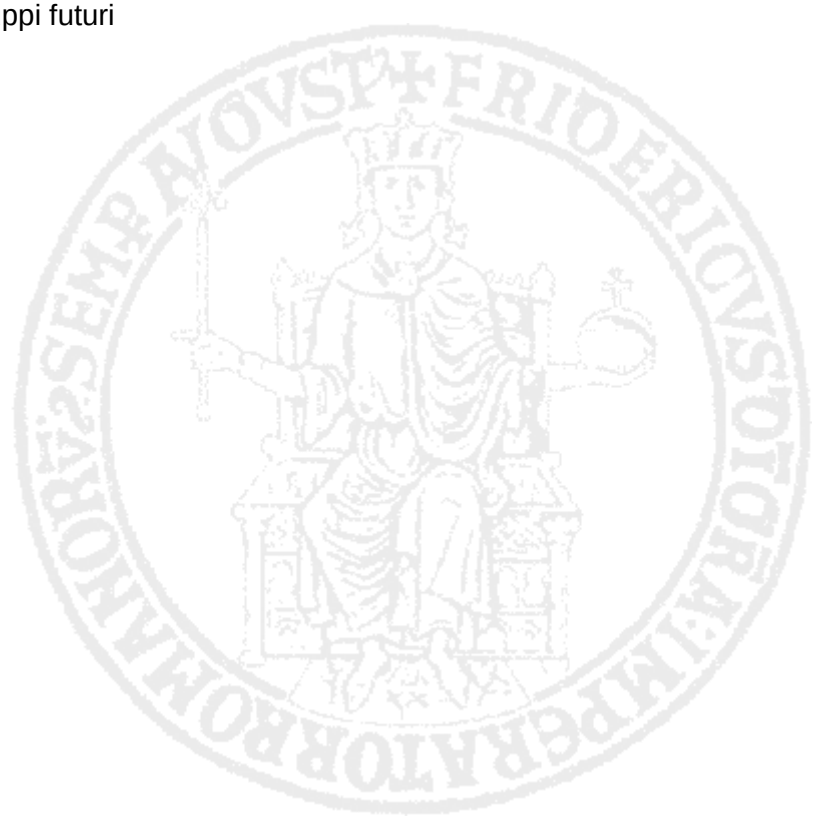
Anno Accademico 2012 - 2013

relatore
Ch.mo prof. Porfirio Tramontana

correlatore
Ing. Domenico Amalfitano

candidato
Riccardo Vaccaro
matr. 534/3247

Indice



Introduzione

L'evoluzione della tecnologia informatica-elettronica ha coinvolto, in tutto il suo percorso, principalmente i calcolatori elettronici (principalmente i computer). Da qualche anno però essa sta coinvolgendo in maniera preponderante i dispositivi "mobili", facendoli diventare sempre più dei veri e propri computer tascabili.

Negli ultimi anni i dispositivi mobili (quali palmari, smartphone e tablet) hanno raggiunto un enorme livello di diffusione (in ambito “personal” e aziendale) e in tempi brevi.

Di conseguenza, si è registrato un forte aumento della popolarità di applicazioni mobili per i dispositivi mobili. Ciò si traduce però nella richiesta, da parte di utenti e sviluppatori, di una maggiore qualità nel software, cioè affidabilità, usabilità, prestazioni, e sicurezza [2].

Per soddisfare la crescente richiesta di applicazioni di alta qualità, gli sviluppatori devono dedicare maggiore impegno e attenzione ai processi di sviluppo del software: l’uso di tecniche di ingegneria del software ben definite diventa indispensabile, e in particolare il test di applicazioni mobili svolge un ruolo strategico.

Il test del software in generale rimane una delle attività più critiche e costose nel ciclo di vita del software, ma per le applicazioni mobili può essere un’attività ancora più complessa a causa delle specifiche caratteristiche e problematiche che caratterizzano queste applicazioni [1].

C'è la necessità dunque di tecniche specifiche per testare efficientemente ed efficacemente le applicazioni mobili (in termini di rilevazione di crash e bug, di costi e tempi). A questo

scopo, le soluzioni di test automatici sono altamente desiderabili [2].

Il test di applicazioni mobili nell'industria, però, è ancora lontano da essere completamente automatico: ad esempio gli approcci più pratici per la generazione automatica di casi di test hanno gravi limitazioni.Gli autori di [2] hanno recentemente presentato una nuova tecnica di test di sistema basato su eventi (Event-Driven-System EDS) per generare "automaticamente" i casi di test nel contesto delle applicazioni Android. Tale tecnica di test è chiamata **GUI Ripping**, si focalizza sull'interfaccia grafica (GUI) dell'applicazione Android e fa uso dello strumento **GUI Ripper** [5].

Il GUI Ripper effettua un'analisi dinamica "automatica" della GUI dell'applicazione al fine di trovare e scatenare eventi nella GUI. In questo modo vengono dedotti automaticamente le possibili sequenze di eventi, che infine possono essere tradotte automaticamente in casi di test.

La tecnica del GUI Ripping si basa su una tecnica "configurabile" di analisi della GUI: infatti il comportamento del GUI Ripper può essere regolato in più modi (attraverso i suoi parametri), a seconda della specifica applicazione sotto test e degli obiettivi del test.

I risultati di vari esperimenti condotti [2] hanno dimostrato che la tecnica del GUI Ripping è più efficiente ed efficace (nella rilevazione di guasti, nei costi e tempi) rispetto ad altre tecniche di test automatico della GUI. Tali risultati, però, hanno evidenziato anche che tale successo è legato strettamente alla specifica configurazione del GUI Ripper: scegliere una configurazione piuttosto che un altra può incidere anche molto sui risultati del test, nonché sul corretto funzionamento del GUI Ripper stesso.

Il mio lavoro di tesi, quindi, mira a fornire uno strumento di supporto alla tecnica del GUI Ripping, che permetta la scelta guidata della corretta configurazione del GUI Ripper.

Capitolo 1

Android e il test

Applicazione mobile: software autonomo progettato per un dispositivo mobile e per l'esecuzione di compiti specifici per gli utenti mobili [1].

Negli ultimi anni si è assistito alla rapida evoluzione e all'impetuosa diffusione dei dispositivi mobili quali palmari, smartphone e tablet. Tale diffusione è legata a due principali caratteristiche dei dispositivi mobili moderni:

costi contenuti, che permettono l'accessibilità a tutti i tipi di utenti.

presenza di hardware di tutto rispetto e la possibilità di far girare applicazioni, che favoriscono il supporto di funzionalità paragonabili a quelle dei comuni pc.

Come conseguenza di ciò si è assistito anche all'aumento di popolarità delle applicazioni mobili.

Le prime applicazioni mobili apparvero circa dieci anni fa, ma non ebbero molto successo poiché limitate dalle caratteristiche dei primi dispositivi mobili, quali hardware limitato e quasi nessun meccanismo di astrazione offerto dalle piattaforme software dei dispositivi.

Negli ultimi anni però, in parallelo ai progressi delle piattaforme hardware dei dispositivi mobili (hardware migliore, sensori, etc.), vi sono stati anche progressi nelle piattaforme software (sistemi operativi mobili, framework di sviluppo, etc.): tutto ciò ha spianato la strada alle applicazioni mobili.

La loro diffusione, in particolare, è aumentata in modo esponenziale con l'apertura della piattaforma Apple App Store il 30 Luglio 2008 [6]: il numero di applicazioni mobili

disponibili in App Store è cresciuto da poche centinaia iniziali a circa mezzo milione di applicazioni il 30 luglio 2011 [7]. Dopo il lancio dell'App Store, diverse altre piattaforme di distribuzione digitale sono apparse, che forniscono anche il software mobile per i dispositivi mobili (come Android, BlackBerry, Symbian, Windows Phone etc.)

Tutte le piattaforme hanno un proprio market dove gli utenti possono navigare e scaricare le applicazioni mobili disponibili (gratuitamente o a pagamento). Ciò ha favorito una sempre più crescente domanda di applicazioni di questo tipo, sia per uso personale sia per i contesti inter/intra aziendali e industriali (come applicazioni mobili per accedere in remoto a servizi o postazioni).

Dopo il successo della piattaforma App Store, nel 2011 si è registrato un nuovo (e più rivoluzionario) successo della piattaforma Android, sviluppata da Google. Dalla sua comparsa sul mercato, il sistema operativo Android è continuato a crescere in popolarità, superando BlackBerry e iOS nel 2011 [8] e rappresentando il 52,5% delle vendite di smartphone nel terzo trimestre del 2011 [9]. Inoltre nel dicembre 2011, Android Market ha superato i 10 miliardi di download di applicazioni con un tasso di crescita di un miliardo di download di applicazioni al mese [10].

1.1 Android

L'ingegnere Andy Rubin, nel 2004 lasciò la carica di *CEO* (Chief Executive Officer - Amministratore Delegato) della Danger Inc e riunì attorno a sé uno staff di ingegneri per sviluppare una nuova e innovativa piattaforma per dispositivi mobili.

Fondò la Startup Android Inc., la quale venne acquisita dalla Google Inc. nel 2005. Quest'ultima nel 2007 costituì assieme a colossi come ASUS, HTC, Samsung, Intel, Motorola, Qualcomm, T-Mobile e NVIDIA, un'alleanza il cui obiettivo è stabilire nuovi standard per l'open source per i dispositivi mobili: **l'Open Handset Alliance** (**OHA**).

Sempre nel 2007 venne presentata ufficialmente la prima versione di **Android [**11**]**. Il primo dispositivo equipaggiato con Android che venne lanciato sul mercato fu l'HTC Dream, il 22 ottobre del 2008 [12].

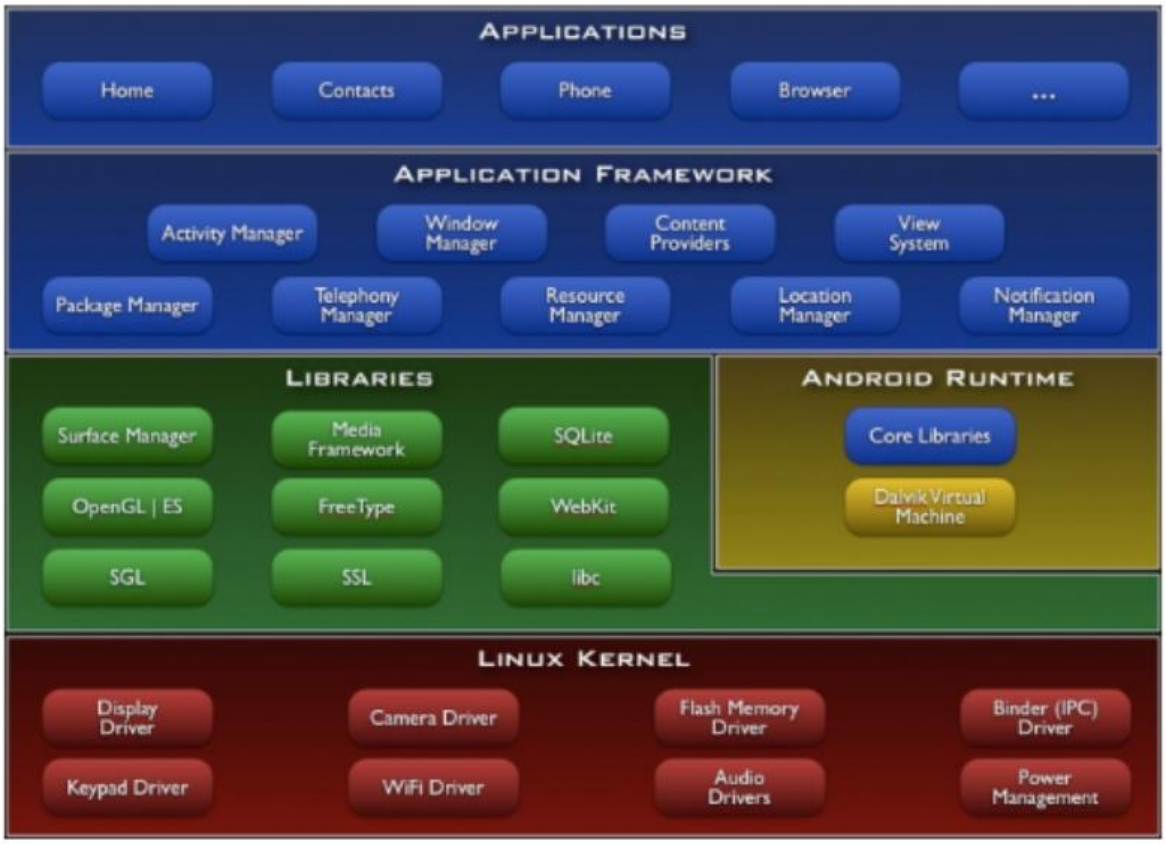
Da allora, il successo della piattaforma Android è andato sempre crescendo, facendole

raggiungere il primato nel 2011 come sistema operativo mobile più diffuso al mondo.

Android, a differenza delle altre piattaforme software mobili, è "open": è liberamente disponibile ed offre la possibilità di utilizzare e di modificare la piattaforma a utenti, sviluppatori e produttori di dispositivi (da cui deriva la sua larghissima diffusione).

1.1.1 Struttura di Android

Android è una "stack-platform" (piattaforma a pila) per i dispositivi mobili: è composta di quattro livelli [1].



1.1 – Struttura a livelli di Android

Il **livello Applicazioni** occupa lo strato superiore dello stack, includendo sia applicazioni di base che di terze parti (client di posta elettronica, di messaggistica, mappe, navigatore, etc).

Il **livello Application Framework** fornisce un insieme di componenti riutilizzabili per la creazione di nuove applicazioni Android ed include i servizi per le applicazioni (gestori di risorse, di notifiche, etc).

Il **livello Librerie Statiche** contiene un insieme standard di librerie, di cui l'Application Framework ne espone le funzionalità (tecnologia del browser, connettività con database, grafica avanzata, supporto audio/video, etc.) , e l'ambiente run-time di Android. Il **Runtime Android** include le librerie principali di run-time per il sistema operativo e la *Dalvik Virtual Machine* (DVM), una macchina virtuale Java ottimizzata per eseguire le applicazioni Android.

Il **Kernel Linux** fornisce un livello di astrazione hardware e i servizi di base (processore, memoria e gestione del file-system) mediante i driver hardware specifici per il dispositivo.

1.1.2 Android Application Framework

L'**Android SDK** (Software Development Kit) fornisce gli strumenti e le API necessarie per iniziare a sviluppare applicazioni sulla piattaforma Android, utilizzando il linguaggio di programmazione Java.

Un'applicazione Android può includere quattro tipi fondamentali di componenti Java, messi a disposizione dall'**Android Application Framework**: Activity, Service, Broadcast Receiver e Content Provider.

Activity

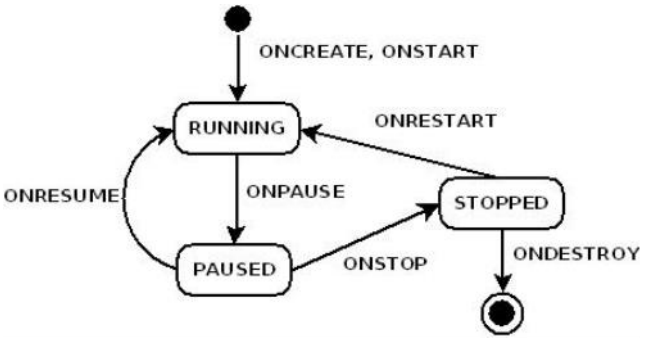
I componenti *Activity* forniscono funzioni fondamentali per l'interfaccia grafica utente (GUI) dell'applicazione **[**13**]** e presentano un'interfaccia visiva per ciascun compito specifico che l'utente può intraprendere. L'interfaccia grafica utente di un'applicazione Android è composta da una serie di Activity.

La navigazione da un'Activity all'altra avviene a seguito dello scatenarsi di eventi, che possono essere generati dall'utente, dall'ambiente software o dal sistema a seguito di eventi hardware.

Nel suo ciclo di vita, un Activity passa attraverso tre stati principali:

- *in esecuzione*: l'Activity ha il controllo completo ed esclusivo dello schermo del dispositivo (solo un'istanza di Activity alla volta può essere in esecuzione)
- *in pausa*: l'Activity perde il controllo esclusivo dello schermo, ma rimane visibile

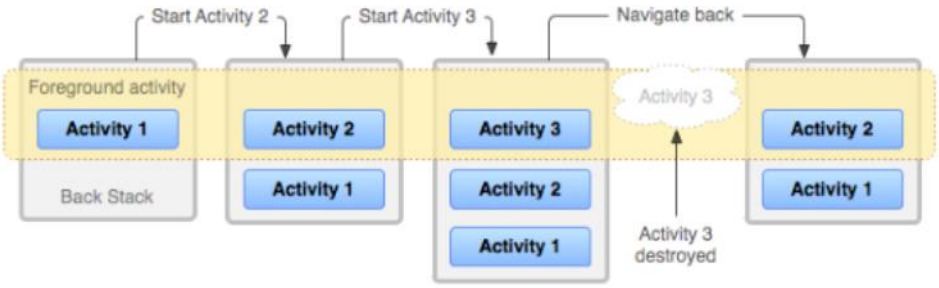
- *fermato*: quando l'Activity diventa completamente oscurata da un'altra Activity



1.1 - Ciclo di vita dell'Activity

Un'applicazione è un insieme di Activity collegate tra loro: solitamente l'Activity visualizzata, quando si avvia per la prima volta un'applicazione Android, è detta **main Activity**.

Ogni Activity può avviarne altre per eseguire diverse azioni ed il sistema conserva l'Activity precedente in una struttura *Back Stack* (a pila) [14]. La nuova Activity viene inserita in cima al Back Stack e diventa l'Activity in esecuzione, mentre quella precedente resta memorizzata nello stack nello stato fermato o in pausa. In particolare, l'utente può tornare all'Activity precedente tramite la pressione del pulsante BACK sul dispositivo (da cui il nome "back" della struttura).



1.2 – Back Stack structure

Service

I componenti *Service* possono eseguire operazioni in background (transazioni di rete, riproduzione di musica, trasferimento di file di I / O, etc.) senza offrire alcuna interfaccia utente: solitamente svolgono operazioni che richiedono molto tempo.

L'esecuzione di un servizio può arrestarsi da se (quando viene lasciato da tutti i componenti a cui era legato) o essere fermato da un altro componente.

Broadcast Receiver

I componenti *Broadcast Receiver* ascoltano e reagiscono agli annunci di trasmissione a livello di sistema, provenienti sia dal sistema (trasmissioni che annunciano che lo schermo è spento, che la batteria è scarica, che una foto è stata catturata, etc.) sia da altre applicazioni.

In genere essi non hanno un'interfaccia grafica, anche se possono creare notifiche nella barra di stato.

Il Broadcast Receiver è diverso dal componente Activity: l'Activity può ascoltare gli eventi solo durante l'esecuzione, mentre un Broadcast Receiver attende gli eventi in modo continuo.

Content Provider

I componenti Content Provider gestiscono i dati per le applicazioni: li incapsulano e ne gestiscono la condivisione fra più applicazioni. Opzioni per l'archiviazione dei dati includono il file-system, un database SQLite, il web, o in qualsiasi altro luogo di memorizzazione persistente.

I Content Provider controllano anche l'accesso dei dati di un'applicazione da parte di altre applicazioni.

1.1.3 Gestione dei processi

Un aspetto unico del sistema Android è il supporto per *l'attivazione a run-time dei componenti* [1]: qualsiasi applicazione può attivare e riutilizzare dei componenti di un'altra applicazione. Tuttavia, poiché il sistema esegue ogni applicazione in un processo separato e

applica i permessi dei file (limitandone l'accesso), l'applicazione non può attivare direttamente un componente di un'altra applicazione, ma deve consegnare un messaggio **Intent** al sistema che specifica appunto l'intenzione di avviare un particolare componente. In particolare, i messaggi di Intent asincroni attivano tre dei quattro tipi di componenti (Activitiy, Service, e Broadcast Receivers).

Come conseguenza di questo meccanismo di attivazione dei componenti, le applicazioni Android, a differenza della maggior parte delle applicazioni tipiche, non hanno un unico punto di ingresso (assenza di funzioni "main()" o simili).

In parte a causa di ciò, la piattaforma Android impone un interessante approccio di gestione dei processi.

- per ottimizzare l'utilizzo delle limitate risorse disponibili sul dispositivo mobile, le applicazioni Android implementano un modello di elaborazione multi-thread nel quale soltanto un particolare thread, detto *thread principale* (o *main thread*), è in grado di accedere all'interfaccia utente, mentre tutti gli altri lavorano in background
- quando un componente di un'applicazione viene avviato e l'applicazione non ha altri componenti in esecuzione, il sistema Android avvia un nuovo processo Linux per l'applicazione con un singolo thread di esecuzione
- di default, tutti i componenti della stessa applicazione sono eseguiti nello stesso processo e thread (thread principale)
- se un componente viene avviato ed esiste già un processo per la sua applicazione, allora il sistema inizializza il nuovo componente all'interno del processo esistente e nello stesso thread di esecuzione: tuttavia, diversi componenti di un'applicazione possono essere eseguiti in processi separati, se desiderato, e i thread aggiuntivi per ogni processo possono essere creati da componenti esistenti

1.1.4 Le risorse di un'applicazione Android

Un'applicazione Android necessita di *risorse aggiuntive* (come audio, video e tutto ciò che concerne l'aspetto visivo dell'applicazione). Utilizzando appositi file XML, lo sviluppatore può definire tutte le risorse grafiche (animazioni, menù, stili grafici, colori, layout grafico,

etc.).

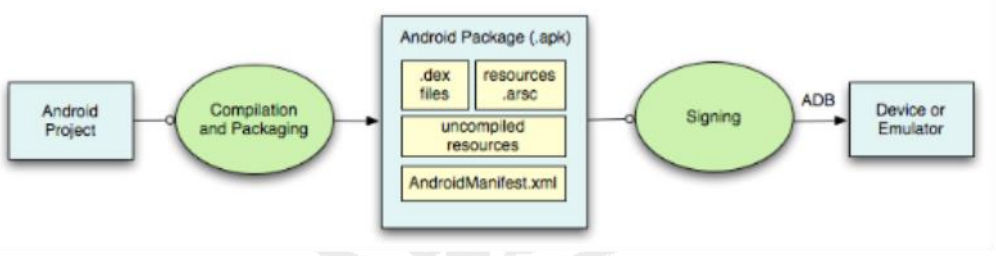
Per ogni risorsa presente nel file XML il sistema genera un ID univoco (numero intero che identifica la risorsa) che può essere utilizzato per accedervi da codice o per effettuare riferimenti incrociati con altre risorse.

1.1.5 AndroidManifest e il file APK

Sul sistema Android ogni applicazione deve avere, nella sua directory principale, un file **AndroidManifest.xml** che contiene le informazioni essenziali circa l'applicazione. Esso dichiara:

- il nome del package, che identifica in modo univoco l'applicazione
- i componenti (Activity, Service, Broadcast Receiver e Content Provider) che compongono l'applicazione, gli Intent che esse possono gestire, le condizioni necessarie per la loro esecuzione (come la versione minima del sistema Android richiesta)
- le autorizzazioni necessarie all'applicazione per poter accedere a parti protette del sistema ed interagire con altre applicazioni
- i permessi necessari alle altre applicazioni per interagire con le componenti dell'applicazione

Gli sviluppatori compilano e pacchettizzano le applicazioni Android in un singolo file di applicazione Android con l'estensione ".apk" (**Android PacKage**). In analogia al formato JAR usato per le applicazioni Java, anche APK è una semplice estensione del formato ZIP. Questo file include tutto il codice dell'applicazione (file ".Des"), le risorse, le Activity, e il file Manifest.



1.4 – Android Package

Compilato il file APK, bisogna assegnargli una **release key** prima che possa essere installato su un dispositivo.

1.1.6 Sviluppo delle applicazioni Android

L'**Android SDK** (Software Development Kit), messo a disposizione liberamente da Google, offre tutti gli strumenti necessari per lo sviluppo e l'esecuzione delle applicazioni Android: l'SDK è disponibile per le più diffuse piattaforme (Windows, Linux e MacOS).

Esso fornisce le classi Java con cui programmare le applicazioni, gli *SDK Tools* (strumenti per la compilazione, il debug ed il deploying) e un driver USB che permette di collegare un dispositivo mobile alla macchina su cui è installato l'SDK (per usarlo nei test dell'applicazione).

L'Android SDK include anche l'**Android Emulator** [20], un emulatore di dispositivo mobile che permette di eseguire le applicazioni su un comune computer, anziché utilizzare dispositivi reali.

Durante la sua esecuzione, l'emulatore si comporta come un comune dispositivo su cui gira il sistema Android e permette lo stesso tipo di interazioni utente. In particolare il "touch" viene emulato attraverso il click del mouse, mentre la tastiera del dispositivo viene emulata con la tastiera del computer oppure con quella virtuale presente nell'emulatore.

L'Android Emulator consente la creazione di uno o più *Android Virtual Device* (AVD), nonché la configurazione della loro dotazione hardware/software (ram, memoria, display, versione di Android, etc.): infine consente l'esecuzione di uno o più AVD (anche contemporaneamente).

1.2 Testing di Android

A differenza delle applicazioni software a cui siamo stati sinora abituati, incluse le applicazioni Web, l'utilizzo di un'applicazione mobile è immediato: si scarica e si usa.

Se funziona bene si continua a usarla, altrimenti se ne cerca un'altra.

In tale contesto, il test delle applicazioni mobili ricopre un ruolo fondamentale.

Il test in ambito mobile non può trascurare gli aspetti di base che caratterizzano il test di

qualsiasi tipo di applicazione [15]:

- **Modelli di test**, rappresentano le relazioni tra gli elementi di un componente del software
- **Livelli di test**, specificano i diversi ambiti in cui eseguire i test
- **Strategie di test**, algoritmi per la creazione di casi di test a partire dai modelli
- **Processi di test**, definiscono il flusso delle attività di testing e qualsiasi decisione riguardante tali attività

Il test del software è un procedimento utilizzato per individuare le carenze di *correttezza*, *completezza* e *affidabilità* di un prodotto software nelle fasi di sviluppo [16]. Con tale attività si vuole quindi assicurare la qualità del software tramite la ricerca di "bug", che si manifestano sotto forma di "guasti" dell'applicazione (in particolari condizioni di esecuzione e con particolari input).

- *bug*: difetto nei requisiti, nella progettazione o nell'implementazione dell'applicazione
- *guasto*: comportamento del software dissimile dalle specifiche e dai requisiti definiti per l'applicazione [17]

Le principali tecniche del test del software sono:

- **test statico**: valutazione di un sistema (o di un suo componente) basato sulla sua forma, sulla sua struttura, sul suo contenuto o sulla documentazione. La valutazione avviene a prescindere dall'esecuzione dell'oggetto che si sta testando
- **test dinamico**: valutazione di un sistema (o di un suo componente) basato sull'osservazione del suo comportamento in esecuzione. La valutazione necessita la conoscenza dei comportamenti attesi per poterli confrontare con quelli osservati

Solitamente col termine "test" si fa riferimento al test dinamico.

Strategie di test definiscono i metodi per la progettazione dei casi di test, come:

- **test black-box** (funzionale): si progettano casi di test basati sulle funzionalità specifiche dell'elemento da testare
- **test white-box** (strutturale): si considera il codice sorgente sottostante per sviluppare casi di test

- **test gray-box**: si progettano casi di test utilizzando entrambi gli approcci **[**18**]**

Livelli di test definiscono i diversi ambiti e punti di inizio per i test.

- **test di unità**: verifica il codice sorgente di ogni singolo componente dell'applicazione
- **test di integrazione**: considera parti messe assieme dell'applicazione per testare le loro funzionalità combinate
- **test di sistema*:*** mira a scoprire difetti testando il sistema nel suo complesso

1.2.1 Test di applicazioni Android

Una serie di questioni rende la progettazione e l'esecuzione di *attività di test* delle applicazioni mobili (in particolare quelle Android) un problema non banale.

Il sistema operativo Android è in prevalenza installato su dispositivi mobili dotati di risorse hardware limitate rispetto ai tradizionali calcolatori, ma con una quantità molto maggiore di sensori e dispositivi di comunicazione. Ne segue che, dopo aver sviluppato e testato un'applicazione su un computer (tramite l'AVD), si dovrà poi passare a testarla sul dispositivo reale (l'emulatore non può assicurare la totale compatibilità col dispositivo reale) [19].

Utilizzare tecniche di test sviluppate per altri sistemi nell'ambito delle applicazioni Android richiederebbe un considerevole lavoro di adattamento dei modelli e delle strategie adottate per poter tenere in conto le peculiarità della piattaforma Android.

Infine un'analisi svolta da Hu et al. (2011) [21] mostra che, sebbene la presenza di bug specifici dell'implementazione dell'applicazione, non sono rari i malfunzionamenti specifici della piattaforma Android. Quindi fonti di guasti comprendono bug della realizzazione dell'applicazione, dell'ambiente d'esecuzione e dell'interfaccia tra l'applicazione e il suo ambiente.

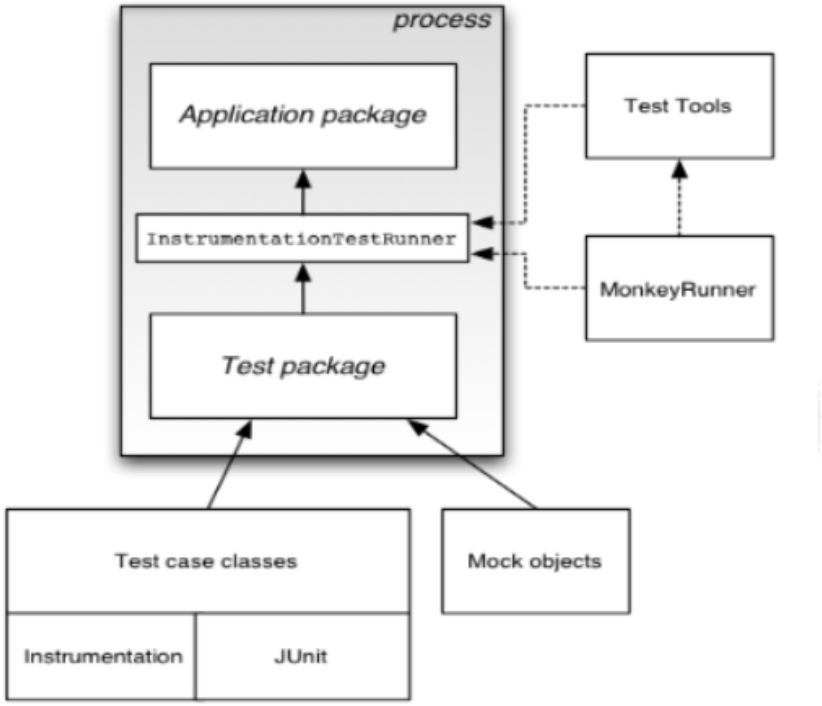
1.2.2 Android Testing Framework

La piattaforma di sviluppo di Android include l'**Android Testing Framework**, un ricco set di API basato su JUnit e ampliato con un framework di strumentazione e con le classi di test

specifiche per Android [22] . Il framework di test include:

- classi per la strumentazione automatica del codice dell'applicazione e per il monitoraggio delle interazione tra il sistema e l'applicazione
- classi per emulare gli eventi del ciclo di vita di Android
- classi per il test diretto dei componenti Activity, Content Provider, e Service (supporta l'automazione del test di unità)
- classi per l'inizializzazione del contesto dell'applicazione (risorse specifiche per l'applicazione, sensori, servizi di sistema, etc.) e per l'isolamento di un componente dal reale contesto di esecuzione (supporta l'automazione di test di integrazione e di sistema)

Mentre in JUnit si utilizza direttamente un test runner per l'esecuzione dei casi di test, in Android è necessario impiegare appositi tool per caricare i package del progetto di test e l'applicazione da testare; quindi, un test runner specifico per Android (**InstrumentationTestRunner**) viene controllato dal tool per l'esecuzione dei test.



1.5 – Android Testing Framework

1.2.3 Monkeyrunner e Monkey

Android SDK fornisce alcuni semplici applicazioni di testing.

Monkeyrunner permette di installare un'applicazione test su un dispositivo, inviare comandi alla GUI, catturare un'immagine che ne rappresenta lo stato e confrontare due immagini catturate in momenti successivi.

Monkeyrunner non va confuso con **UI/Application Exerciser Monkey** (noto semplicemente come **Monkey**). Questi, a differenza di Monkeyrunner, invia comandi generati pseudo-random al dispositivo (reale o emulato) sotto vincoli definibili dall'utente. Monkey si arresta qualora l'applicazione dovesse andare in crash, generando un report. Per questi motivi, Monkey risulta utile per eseguire lo stress test di un'applicazione.

1.2.4 Instrumentation

L'**Instrumentation** di Android è un insieme di system-call grazie alle quali è possibile controllare i componenti dell'Application Framework di Android indipendentemente dal loro normale lifecycle.

In generale, lo sviluppatore Android può implementare i metodi che permettono il passaggio da uno stato all'altro, specificando il codice che dovrà essere eseguito alla loro chiamata, ma non può invocare tali metodi a piacimento: ciò viene gestito dal framework.

In fase di test, invece, si ha la necessità di invocare tali metodi per forzare il componente nello stato desiderato (stato il cui deve essere testato). Ciò è possibile grazie all'Instrumentation.

1.2.5 Robotium

Robotium [23] è una libreria di metodi creata da Jayway [24] che rende più facile scrivere potenti e robusti casi di test per le applicazioni Android. L'utilizzo di questi metodi consente al tester di concentrarsi sulla logica del caso di test, lasciando a Robotium il compito di eseguire le particolari interazioni richieste di volta in volta attraverso l'instrumentation.

Si é visto che Robotium eccelle dal punto di vista del test black-box. È l'unico framework

che permette il test sul singolo file APK, è in continuo sviluppo ed è corredato da una ricca documentazione.

Tuttavia, testando attraverso l'interfaccia utente, Robotium "vede" unicamente la struttura dei dati rappresentata dalla View, ovvero dal display del dispositivo.

Poiché Android mette a disposizione quattro risoluzioni, è possibile che lo stesso test effettuato su due dispositivi potrebbe dare risultati diversi a causa della diversa dimensione dello schermo e potrebbe accadere che le strutture dati visualizzate (le sole che Robotium testa) potrebbero variare in relazione alla risoluzione del display.

1.3 Testing dell'interfaccia utente

Oggigiorno le *interfacce utente grafiche* (GUI) hanno assunto un'importanza fondamentale nel rapporto tra il software e l'utente: forniscono un ottimo mezzo di interazione con il software e spesso ne determinano la sua *usabilità*/*accettazione*/*diffusione*. Ne consegue che un'adeguata fase di convalida e verifica della GUI è indispensabile per assicurare qualità all'intero prodotto software[25]

Le tecniche di test della GUI si suddividono fondamentalmente in:

- **Model-Based testing**: esiste un modello (descrizione formale dell'applicazione sotto test, in particolare della GUI) che avrà un livello di dettaglio sufficiente alla generazione automatica di casi di test per la verifica del software
- **Random testing**: in assenza di un modello, l'applicazione viene esercitata in maniera casuale, o pseudocasuale, alla ricerca di eventuali guasti (causati da eccezioni non gestite)
- **Ripping-Based testing**: in assenza di un modello, l'applicazione viene esercitata in maniera metodica, seguendo una precisa strategia di esplorazione: vengono individuati eventuali guasti (causati da eccezioni non gestite) e viene ricostruito un modello (impiegabile nella generazione di casi di test)

Nel contesto del test della GUI, il modello assume un ruolo fondamentale: infatti, un adeguato modello dell'interfaccia utente permette la generazione automatica dei casi di test e ciò rende fattibile l'automazione dell'esecuzione dei test. Di seguito vengono trattati

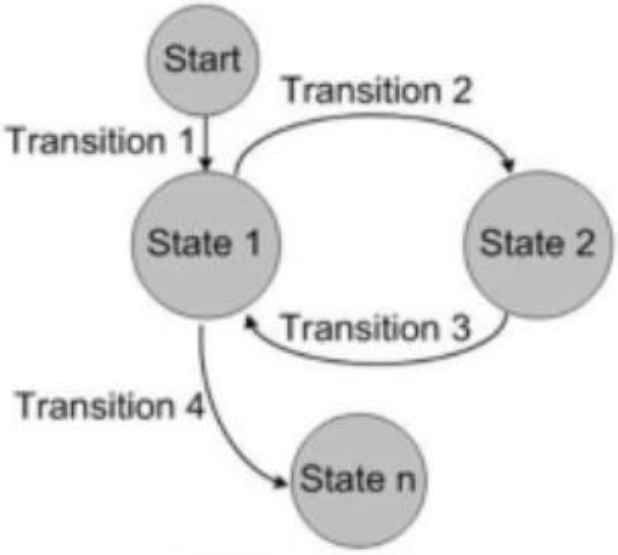
brevemente i modelli di interfaccia utente più utilizzati nei test delle GUI.

1.3.1 FSM model

Si descrive il funzionamento dell'interfaccia utente tramite l'utilizzo di **Macchine a Stati Finiti** (FSM - Finite State Machine) [26]. Ogni nodo del grafo rappresenta un particolare stato dell'interfaccia utente, mentre ogni arco indica il passaggio da uno stato all'altro ed è innescato dalle interazioni dell'utente (*eventi*).

Per snellire il modello, si può pensare di ridurre il numero di stati nella macchina (ad esempio si accorpandoli in classi di equivalenza [25]).

Dal modello FSM è possibile in seguito generare automaticamente casi di test per la GUI. In particolare, ogni *sequenza di archi* che parte da un nodo iniziale viene opportunamente codificata in un linguaggio di programmazione e va a comporre un caso di test.



1.6 – Diagramma di una FSM

1.3.2 Event-Flow model

Si associa il funzionamento dell'interfaccia utente ad un generico software guidato da eventi (EDS – Event Driven Software). Il modello astratto più utilizzato per gli EDS è tipicamente l'**Event-Flow Model** (modello del Flusso di Eventi). Si prenderà come esempio

il modello proposto da Memon (2007) [25], il quale prevede due fasi.

Una prima fase descrive le possibili sequenze di eventi che possono essere eseguite sull'interfaccia.I nodi del grafo rappresentano gli eventi, quindi le transizioni non sono associate ad un particolare oggetto dell'interfaccia grafica ma indicano semplicemente la sequenza degli eventi consecutivi.

Una seconda fase descrive l'insieme delle proprietà dei nodi del grafo, ossia degli eventi: essi si dividono in:

- *Pre-condizioni*: l'insieme degli stati della GUI in cui l'evento può avere luogo
- *Effetti*: i cambiamenti che l'esecuzione dell'evento apporta allo stato della GUI

Una visita ordinata del grafo, secondo una certa strategia, porta alla generazione automatica di casi di test per la GUI.

1.3.3 Problematiche sulla generazione del modello

Produrre un modello formale dell'interfaccia utente può risultare un procedimento costoso. La GUI infatti deve consentire all'utente di utilizzare tutte le funzionalità del software e deve rispondere in maniera appropriata a tutte le possibili interazioni dell'utente. Da tale contesto emerge che:

- la GUI ha una complessità proporzionale (spesso paragonabile) a quella del software che controlla (il modello sarà altrettanto complesso)
- la GUI deve essere aggiornata non solo quando si scoprono bug, ma anche quando vengono aggiunte nuove funzionalità (il modello deve essere aggiornato continuamente)
- testare la GUI come componente a sé, isolato dal resto del software, può fornire informazioni non attendibili sul suo reale funzionamento nell'ambiente finale (il modello può risultare non valido rispetto al reale comportamento dell'intero software)
- il modello della GUI deve essere sufficientemente accurato da distinguere fra i diversi tipi di messaggi provenienti dall'utente (*eventi*) e fra i diversi tipi di destinatari (*widget*)

Ne consegue che uno strumento in grado di generare automaticamente un modello della GUI è di fondamentale importanza per un'efficace automazione dei test.

1.3.4 Tecnica del GUI Ripping

GUI Ripping è una tecnica di "reverse engineering" che mira a ricostruire un modello dell'interfaccia grafica di un'applicazione software esistente interagendo dinamicamente con la sua interfaccia utente [27].

La tecnica di GUI Ripping utilizza un tool **GUI Ripper** che esplora automaticamente l'interfaccia grafica dell'applicazione, simulando eventi reali degli utenti nell'interfaccia utente dell'applicazione per:

- individuare guasti dovuti ad eccezioni non gestite nell'applicazione
- ricavare automaticamente un modello dell'interfaccia grafica

Tale modello supporta la derivazione di casi di test che potranno essere eseguiti automaticamente per scopi diversi (es. per crash test e test di regressione).

L'esplorazione dell'applicazione è eseguita in maniera metodica, implementando una strategia che pone fra i suoi obiettivi quello di massimizzare la copertura del codice. Le informazioni ottenute vengono utilizzate internamente dal tool per portare più in profondità l'esplorazione. Lo scatenarsi di un evento porta infatti il GUI Ripper a scoprire nuovi stati dell'applicazione nei quali sarà possibile scatenare nuovi eventi che potranno portare a scoprire nuovi stati, e così via.

1.3.4.1 GUI Tree model

Il GUI Ripper produce un modello di interfaccia grafica che è una struttura ad albero chiamata **GUI Tree**: esso è la rappresentazione esplicita dei cammini eseguibili sulla GUI dell'applicazione sotto test, a partire da un certo stato iniziale, scatenando eventi sui widget dell'interfaccia utente.

I nodi dell'albero rappresentano le interfacce utente individuali nell'applicazione Android, mentre gli archi descrivono le transizioni basate su eventi tra le interfacce.

Attraverso l'analisi dinamica (sparando eventi sull'interfaccia utente dell'applicazione) il

GUI Ripper costruisce il modello della GUI secondo le seguenti regole:

- a partire da uno stato A e scatenando un evento, la GUI viene a trovarsi in uno stato B: ciò da luogo nel GUI Tree ad un nuovo nodo B collegato al nodo A attraverso un arco orientato da A a B
- a partire da uno stato A e non scatenando nessun evento, il GUI Tree rimane invariato

Il criterio di esplorazione del GUI Ripping, senza opportuni criteri di limitazione, può portare ad un altissimo numero di stati (anche ad un esplosione del numero) : per ovviare a ciò, vengono utilizzati nell'esplorazione due criteri.

Il *criterio di terminazione*, che serve per terminare l'esplorazione: può essere "naturale"(non vi sono più eventi da scatenare) oppure imposto dal tester (come il raggiungimento di una profondità massima).

Il *criterio di astrazione e confronto*, che serve per ridurre considerevolmente il numero degli stati. Ogni istanza di interfaccia (stato) trovata dal GUI Ripper viene astratta (rappresentata con un sottoinsieme delle sue caratteristiche) e poi confrontata con altri stati già astratti in precedenza: se risulta equivalente ad uno di questi, allora l'istanza di interfaccia corrente non viene esplorata. Da ciò segue infine che due stati, anche se equivalenti, posso certamente avere proprietà diverse: non ci sono soluzioni uniche per la scelta delle caratteristiche per l'astrazione e sta al tester sceglierle.

1.3.4.2 Algoritmo del GUI Ripping

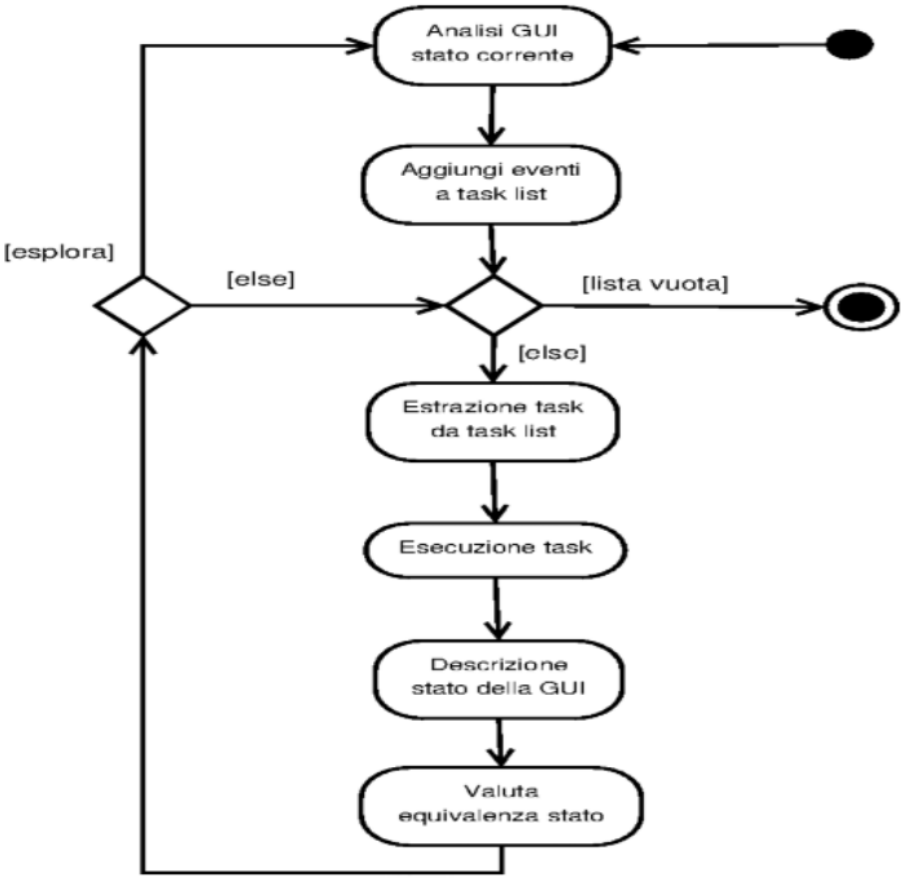
La tecnica di GUI Ripping è iterativa e si basa sui seguenti *concetti chiave* [2]:

- *evento*: è un'azione utente eseguita su un Widget della GUI
- *azione*: consiste di una sequenza di zero o più eventi
- *attività*: è una coppia (azione, stato della GUI) , cioè un'azione eseguita in uno stato della GUI. Un attività viene eseguita raggiungendo prima lo stato della GUI e poi eseguendo l'azione
- *elenco di attività*: comprende una serie di attività
- *criterio di esplorazione della GUI*: stabilisce se l'esplorazione di una data GUI deve

continuare o essere interrotta (criterio di terminazione e criterio di astrazione/confronto)

L'algoritmo del GUI Ripper è il seguente:

1. Il GUI Ripper analizza la GUI dell'applicazione alla ricerca di possibili eventi da esercitare e le inserisce nella lista delle attività da eseguire.
2. Se la lista delle attività risulta vuota il GUI Ripper termina l'esecuzione, altrimenti viene estratta ed eseguita la prima della lista.
3. Gli eventi contenuti nella lista dovrebbero aver cambiato lo stato della GUI. Tale stato viene quindi descritto e confrontato con gli stati precedentemente visitati. Se lo stato corrente deve essere esplorato si torna al punto 1, altrimenti si torna al punto 2.



1.7 – Algoritmo del GUI Ripper

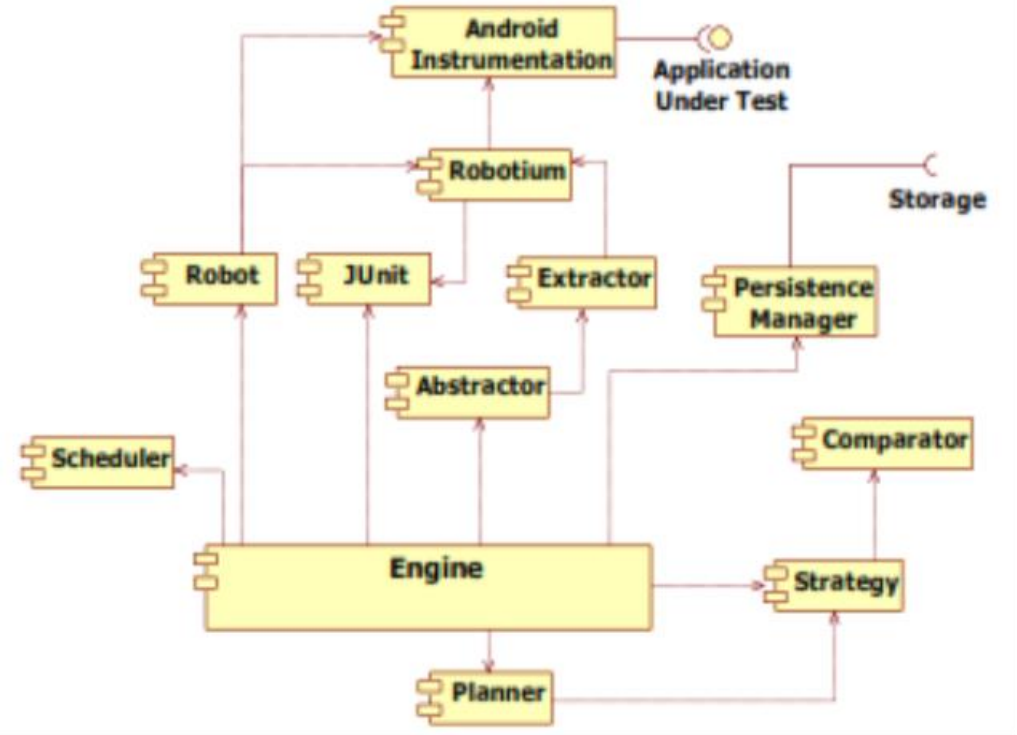
1.4 Strumento di GUI Ripping

Lo strumento **GUI Ripper** proposto dagli autori di [1,2,4] è stato sviluppato in Java, cercando di soddisfare i seguenti requisiti:

- strategia di esplorazione altamente personalizzabile
- alta evolvibilità per l'attuazione di nuove strategie di esplorazione con un ridotto impatto sull'architettura complessiva del Ripper
- capacità di affrontare i problemi di efficienza relativi all'esplosione degli stati e degli eventi dell'interfaccia grafica
- soluzioni idonee per terminare l'esplorazione dell'interfaccia grafica

1.4.1 GUI Ripper software architecture

Per soddisfare queste esigenze, il GUI Ripper è stato progettato in maniera modulare e la cui architettura software è composta da nove principali componenti: Scheduler, Robot, Abstractor, Extractor, Engine, Strategy, Planner, Comparator e Persistence Manager [4].



1.8 – Software architecture of the GUI Ripper

Engine implementa la logica di business principale del Ripper. Esso avvia un processo di GUI Ripping iterativo che parte da una GUI iniziale dell'AUT (stato iniziale). Ad ogni iterazione, Engine utilizza il **Robot** per emulare gli eventi utente sull'interfaccia grafica e l'Extractor per ottenere lo stato corrente della GUI.

Extractor e **Abstractor** cooperano per definire una "stato astratto" da associare alla GUI attuale.

La descrizione dell'attuale stato della GUI viene quindi inviata al **Planner** per selezionare un insieme di eventi scatenabili nella GUI: il Planner trasforma questi eventi in attività eseguibili che verranno memorizzate in un elenco di attività gestito dallo Scheduler.

Scheduler stabilisce l'ordine di esecuzione delle attività, in accordo con la strategia di esplorazione della GUI scelta e fornisce ad Engine la prossima attività.

Dopo l'esecuzione dell'attività, l'Engine delega il componente **Strategy** per stabilire se l'esplorazione della GUI può fermarsi, in accordo con il criterio di terminazione del Ripping, oppure no. La valutazione del criterio di terminazione può richiedere un componente aggiuntivo, il **Comparator**, che valuta l'equivalenza tra lo stato corrente della GUI e gli altri stati definiti in precedenza.

Il processo iterativo di GUI Ripping va avanti fino a quando la lista delle attività è vuota. Durante il processo, il componente **Persistence Manager** è invocato per memorizzare i risultati parziali e quelli complessivi dell'esplorazione della GUI sul disco.

Gli output finali del GUI Ripper includono:

- una rappresentazione della GUI esplorata, detta GUI Tree
- un file di report dei guasti, che fornisce i dettagli circa ogni guasto dell'applicazione Android osservato durante il processo di GUI Ripping

1.4.2 Parametri di configurazione del GUI Ripper

La caratteristica chiave della tecnica di GUI Ripping proposta dagli autori [1, 2, 4] è l'alta "configurabilità" dell'analisi della GUI: il comportamento del GUI Ripper infatti può essere regolato in base all'applicazione da testare e agli obiettivi del test.

Il tester imposta i parametri del GUI Ripper in fase di inzializzazione: essi comprendono

- l'identificazione dell'applicazione da testare
- il tipo di eventi e di input da scatenare sui widget, nonché il numero massimo per widget
- i tipi di widget con cui interagire
- i tempi di attesa
- il criterio di terminazione e il criterio di equivalenza
- il salvataggio di screenshot (schermate dell'emulatore)
- la possibilità di ripristino (dopo eventuali interruzioni "esterne" al processo di test)

1.4.3 Sensor Ripper

La maggior parte dei dispositivi Android possiede dei sensori, utili sia nell'interazione con l'utente che per la determinazione di particolari condizioni esterne al dispositivo.

La piattaforma Android supporta tre categorie di sensori:

- *sensori di movimento* (accelerometro, giroscopio, sensore di gravità e del vettore di rotazione)
- *sensori ambientali* (termometro, barometro, sensore di illuminazione e di umidità)
- s*ensori di posizione* (sensori di orientazioni, magnetometro)

L'**Android Sensor Framework** metta a disposizione classi ed interfacce per accedere si sensori del dispositivo mobile: è possibile determinare quali sensori sono disponibili sul dispositivo, le loro capacità e proprietà, nonché acquisire dati o registrare cambiamenti dai sensori.

Poichè le interfacce grafiche delle applicazioni Android sono guidate dagli eventi, risulta desiderabile e necessario considerare casi di test in cui la GUI sia sollecitata non solo da eventi dell'utente, ma anche da eventi scatenati dai sensori [29].

A tale scopo, gli autori di [1, 2, 4] stanno sviluppando un'estensione del GUI Ripper che può considerare anche eventi dei sensori: il ***Sensor Ripper*** [28].

Dato che è un'estensione, le sue caratteristiche sono uguali a quelle del GUI Ripper (comportamento, tipi di configurazione, modalità d'utilizzo, output del processo di test, etc.).

A causa dello stato sperimentale del progetto, non è stata ancora prodotta documentazione sufficiente alla sua trattazione.

Capitolo 2

PreferencesGui tool

Alcuni esperimenti [2] hanno dimostrato la bontà, l'efficienza e l'efficacia della tecnica di test GUI Ripping (per rilevazione di bug, per costi e tempi) rispetto ad altre basate ad esempio su Monkey: tutto ciò è legato strettamente al tipo di configurazione del GUI Ripper.

Infatti, la scelta di configurazioni non "ottimali" o non corrette può compromettere non solo i risultati del test, ma anche il corretto funzionamento del GUI Ripper stesso.

Il mio lavoro di tesi si è focalizzato sulla progettazione e sviluppo di un utile strumento di configurazione del GUI Ripper: il **PreferencesGui**.

2.1 Specifica dei requisiti del PreferencesGui tool

I *requisiti* di un sistema software descrivono i servizi, le funzionalità e le caratteristiche del sistema, nonchè i vincoli del sistema e del suo processo di sviluppo. L'*analisi dei requisiti* è un'attività che mira a definire esplicitamente i requisiti del sistema e termina con la produzione della *specifica dei requisiti,* un documento che descrive i requisiti in modo completo, non ambiguo e comprensibile.

Il **PreferencesGui** è un'applicazione per computer che supporta la configurazione guidata del GUI Ripper: esso offre un interfaccia grafica per il settaggio dei suoi parametri. Come tale, il PreferencesGui deve soddisfare una serie di requisiti.

2.1.1 Supporto alla generazione automatica di test cases

"*Il programma deve supportare l'automazione della generazione dei casi di test del processo di GUI Ripping*".

La tecnica di test GUI Ripping supporta la generazione automatica di casi di test per l'AUT: il GUI Ripper, al termine dell'esplorazione, genera automaticamente come output il modello GUI Tree e il report dei guasti rilevati, che possono entrambi essere usati per la generazione automatica dei casi di test.

La configurazione del GUI Ripper consiste nel settare un insieme di parametri con adeguati valori: le coppie parametro-valore vengono scritte in un apposito file *preferences* in formato XML (con una specifica struttura), che sarà letto dal GUI Ripper prima di inziare la sessione di Ripping.

Da ciò si evince che un'operazione "manuale" di settaggio dei parametri (direttamente sul file preferences) non è accettabile nello scenario del "test automatizzato".

Il tool PreferencesGui deve quindi consentire una facile, efficace/efficiente ed automatizzata configurazione del GUI Ripper (settare i parametri e scriverli nell'apposito file preferences).

2.1.2 Supporto alla configurazione guidata

"*Il programma deve consentire la configurazione guidata del GUI Ripper e la validazione delle configurazioni*".

Il successo di una tecnica di test non è dato solo dalla sua efficienza ed efficacia (per rilevazione di bug, per costi e tempi), ma anche dalla sua accessibilità. A tale scopo, progetti correlati al GUI Ripper (wizard, documentazione, articoli, etc.) stanno rendendo tale strumento (e quindi la relativa tecnica di test) accessibile e usabile a tutti. Un'operazione "manuale" di settaggio dei parametri comporta un duplice problema:

- preclude la configurazione e l'utilizzo del GUI Ripper ai "non addetti ai lavori"
- non garantisce la corretta ed ottimale configurazione del GUI Ripper (i controlli tramite ispezioni "visive" richiedono tempo e non sono sempre precisi)

Il tool PreferencesGui deve quindi guidare il tester nella scelta della configurazione del GUI

Ripper e validarla.

2.1.3 Parametri del GUI Ripper

I parametri che costituiscono la configurazione del GUI Ripper sono classificati in *sezioni* (Automation, Scheduler, Planner, etc.) [30], ognuna delle quali ha il compito di configurare un corrispondente componente dell'architettura software del GUI Ripper.

Di seguito viene elencata la maggior parte dei parametri esistenti (non tutti) : ciò è dovuto al fatto che alcuni parametri sono ancora "sperimentali" e quindi non ancora ben implementati e documentati (il GUI Ripper è un progetto in continua evoluzione e revisione).

- General:

Nome	*Descrizione*	*Tipo*
Package Name	Nome del package contenente le classi dell'applicazione	Stringa (specifico dell'applicazione)
Class Name	Nome completo (con package) della classe dell'Activity principale	Stringa (specifico dell'applicazione)
Random Seed	Valore usato per impostare il meccanismo di generazione random di numeri. Se = 0, il GUI Ripper provvederà ad impostarne uno	Intero positivo
Activity Description in Session	Possibilità di unire la descrizione delle istanze di Activity incontrate durante il Ripping nel file xml del GUI Tree oppure di salvare tale descrizione in un file xml apposito	True o False
Enable Resume	Possibilità di salvare lo stato interno del GUI Ripper dopo ogni traccia (per il ripristino della sessione di Ripping in caso di interruzioni causate dall'esterno)	True o False

- Automation:

Sleep after Event	Quanti millisecondi aspettare dopo l'esecuzione di ogni azione (di un'attività)	Intero positivo diverso da zero
Sleep after Restart	Quanti millisecondi aspettare prima dell'esecuzione delle azioni di un'attività	Intero positivo
Sleep after Task	Quanti millisecondi (aggiuntivi) aspettare dopo aver eseguito tutte le azioni di un'attività	Intero positivo
Sleep on Throbber	Quanti millisecondi aspettare quando viene visualizzato un throbber (animazione di caricamento) sullo schermo	Intero positivo
Force Restart	Possibilità di far partire tutte le tracce dallo stesso punto di partenza	True o False
Screenshot for States	Possibilità di salvare un'immagine dello schermo dopo l'esecuzione dell'ultima azione dell'attività	True o False
Screenshot for only new States	Possibilità di salvare un'immagine dello schermo dopo l'esecuzione dell'attività, ma solo se viene scoperto un nuovo stato	True o False

- Scheduler:

Scheduler Algorithm	Definisce la strategia usata per l'esplorazione: Depth First: un nodo viene esplorato a fondo prima di passare ai nodi vicini. Breadth First: strategia opposta	Stringa (definita)
Max Tasks in Scheduler	Massimo numero di attività che possono essere eseguite nella sessione di ripping	Intero positivo

- Interactions:

Events	Definisce il tipo di eventi e i widget su cui il GUI Ripper può scatenarli	Array di stringhe (definite)

Inputs	Definisce il tipo di input e i widget su cui il GUI Ripper può scatenarli	Array di stringhe (definite)
Key Events	Definisce quali bottoni fisici del dispositivo il GUI Ripper può premere	Array di interi (definiti)
Back Button Event	Possibilità di premere il pulsante "back"	True o False
Menu Events	Possibilità di premere il pulsante "menu"	True o False
Orientation Events	Possibilità di cambiare l'orientamento dello schermo (da verticale ad orizzontale)	True o False
Scroll down Event	Possibilità di scorrere il display una pagina in basso	True o False

- Planner:

Max Events per Widget	Numero di volte che un evento è generato su un "group widget" (widget che contiene altri widget). Se = 0 non c'è limite.	Intero positivo
Planner	Definisce la strategia che dovrà seguire il componente Planner del GUI Ripper	Stringa (definita)
Tab Event Start only	Possibilità di pianificare l'evento "cambia scheda" solo sull'Activity iniziale	True o False
Event when no ID	Possibilità di scatenare eventi sui widget che non hanno un ID	True o False
Use Sensor	Possibilità di scatenare eventi proveniente dai sensori (solo per il Sensor Ripper)	True o False
Use GPS	Possibilità di scatenare eventi provenienti dal GPS (solo per il Sensor Ripper)	True o False

- Storage:

Max Traces in Ram	Numero massimo di tracce (che compongono il GUI Tree) che possono stare nella memoria ram, prima di essere trasferite nel file xml del GUI Tree. Se = 0 non c'è limite.	Intero positivo

File Name	Nome del file xml del GUI Tree	Stringa
Task List File Name	Nome del file xml contenente la lista delle attività (file temporaneo)	Stringa
Activity List File Name	Nome del file xml contenente la descrizione di tutte le istanze di Activity incontrate durante il Ripping	Stringa
Parameters File Name	Nome del file obj contenente i parametri salvati dai componenti del GUI Ripper (file temporaneo)	Stringa

- Comparator:

Comparator Type	Definisce il criterio di equivalenza tra gli stati: - NullComparator: due stati non sono mai equivalenti - NameComparator: due stati sono equivalenti se hanno lo stesso nome (istanze della stessa Activity) - CustomWidgetsSimpleComparator: due stati sono equivalenti se hanno lo stesso set di widget - CustomWidgetsIntensiveComparator: due stati sono equivalenti se hanno lo stesso set di widget con stesse proprietà	Stringa (definita)
Widget Types	Definisce il set di widget con cui applicare il criterio di equivalenza	Array di stringe (definite)
Compare Activity Name	Possibilità di usare il confronto dei nomi nel criterio di equivalenza	True o False
Compare State Title	Possibilità di usare il confronto dei titoli nel criterio di equivalenza	True o False
Compare List Count	Possibilità di usare il confronto delle dimensioni delle liste nel criterio di equivalenza	True o False
Compare Menu Count	Possibilità di usare il confronto delle dimensioni dei menu nel criterio di equivalenza	True o False
Compare Values	Possibilità di usare il confronti dei valori dei widget nel criterio di equivalenza	True o False

- Strategy:

Max Num Traces	Numero massimo di tracce, eseguite le quali la sessione di ripping termina. Se = 0 non c'è limite	Intero positivo
Pause After Traces	Numero massimo di tracce, eseguite le quali la sessione di ripping va in pausa. Se = 0 non c'è limite	Intero positivo
Pause After Time	Dopo quanti secondi trascorsi la sessione di ripping va in pausa. Se = 0 non c'è limite	Intero positivo
Trace Max Depth	Valore che limita l'esplorazione della GUI ad una certa profondità del GUI Tree. Se = 0 non c'è limite	Intero positivo
Trace Min Depth	Valore che forza l'esplorazione della GUI a raggiungere almeno una certa profondità del GUI Tree	Intero positivo
Explore only new States	Possibilità di non esplorare uno stato equivalente ad altri già esplorati	True o False

2.1.3.1 Vincoli di coerenza

Il GUI Ripper implementa un esplorazione "metodica" (che segue una o più metodologie) dell'interfaccia grafica, quindi i suoi parametri devono rispettare dei vincoli (valori precisi o intervalli limitati di valori) imposti proprio dalle metodologie scelte: questi sono detti *vincoli di coerenza*. Un esempio: in un esplorazione della GUI, la scelta di nessun criterio di comparazione degli stati sottintende l'assenza di nessun tipo di comparazione. Pertanto, scegliere parametri che non rispettano tali vincoli, può provocare risultati falsati nei test e malfunzionamenti del GUI Ripper stesso.

Da ciò si evince che per scegliere configurazioni "corrette", alcuni parametri (i loro valori) devono imporre dei vincoli di valore su altri parametri.

Di seguito è riportata una tabella che elenca i parametri coinvolti in tali vincoli:

Parametro indipendente	*Parametro dipendente*
Esplorazione random della GUI (parametro non presente nel GUI Ripper)	- Scheduler Algorithm = Depth First

	- Max Tasks in Scheduler = 2 - Comparator Type = NullComparator - Explore only new States = False
Activity Description in Session = False	Enable Resume = True
Enable Resume = True	Max Traces in Ram = 1
Screenshot for States = False	Screenshot only new States = False
Planner = SimplePlanner oppure Planner = SimpleReflectionPlanner	Text Value from Dictionary = False Dictionary Ignore Content Type = False
Planner = DictionarySimplePlanner	Text Values ID Hash = False
Comparator Type = NullComparator	- Widget Types = "" - Compare Activity Name = False - Compare State Title = False - Compare List Count = False - Compare Menu Count = False - Compare Values = False - Max Num Traces e Max Depth non possono essere entrambi = 0
Comparator Type = NameComparator	- Widget Types = "" - Compare State Title = True - Compare List Count = False - Compare Menu Count = False - Compare Values = False
Comparator Type = CustomWidgetsSimpleComparator	Compare Values = False
Comparator Type = CustomWidgetsIntensiveComparator	Compare Values = True

2.1.4 Scelte e vincoli progettuali

Il GUI Ripper può eseguire due tipi di esplorazioni della GUI (random o sistematico), che

richiedono specifici vincoli di coerenza, nonché uno specifico nome del file preferences (*preferences_Ripper.xml* per il sistematico, *preferences_Random.xml* per il random). Pertanto, come scelta progettuale, il PreferencesGui presenta un parametro aggiuntivo (non presente nel GUI Ripper) chiamato **Kind of Exploration** per la scelta del tipo di esplorazione.

Una seconda scelta progettuale consiste nell'uso di valori di default ottimali, piuttosto che avere parametri non settati: se l'utente non sa quali valori inserire o ne inserisce di sbagliati, i parametri coinvolti restano con i loro valori di default.

Un'altra scelta progettuale prevede di avviare il PreferencesGui con una *configurazione base* iniziale del GUI Ripper con esplorazione sistematica (e con parametri di default): l'utente può subito utilizzare tale configurazione base per la generazione del file preferences, oppure modificare tutti i parametri che vuole.

Altra scelta progettuale riguarda i quattro parametri *Sleep*: nella documentazione questi devono avere valori in millisecondi (interi positivi). La scelta progettuale prevede invece che il PreferencesGui accetti, per tali parametri, valori in secondi (reali positivi) per una maggiore praticità di settaggio per l'utente.

Infine, un'ultima scelta progettuale consiste nel generare il file preferences nella stessa directory del file APK se l'utente non specifica nessuna directory di destinazione.

Un primo vincolo progettuale prevede l'implementazione del progetto in Java: in particolare deve essere utilizzata la stessa versione del GUI Ripper (JDK 1.6.0_43) e lo stesso IDE (Eclipse), per garantire una certa compatibilità tra le evoluzioni dei due progetti in futuro.

Un secondo vincolo progettuale riguarda la natura dell'applicazione: essa deve essere basata su un'interfaccia utente, per garantire una facile, rapida ed automatica configurazione dei parametri.

Altro vincolo progettuale è la necessità di poter configurare tutti i parametri del GUI Ripper (e non solo una parte), per supportare l'alta configurabilità del GUI Ripper.

Infine, un ultimo vincolo progettuale riguarda i comportamenti specifici del PreferencesGui, in particolare l'impossibilità di generare il file preferences nei casi in cui:

- non viene selezionato il file APK
- si verificano dei problemi nell'estrarre le informazioni dal file APK (dopo la sua selezione)

2.2 Progetto e sviluppo del PreferencesGui

PreferencesGui è un'applicazione stand-alone implementata in Java seguendo le regole della modularità, dell'incapsulamento e infomation-hiding e dell'ereditarietà: ciò garantisce la *portabilità*, l'*evolvibilità* e la *manutenibilità* del software.

L'applicazione fornisce un'interfaccia grafica per il settaggio dei parametri del GUI Ripper e per la produzione del file *preferences* in formato XML (che sarà letto dal GUI Ripper prima dell'inizio della sessione di Ripping).

L'applicazione fornisce:

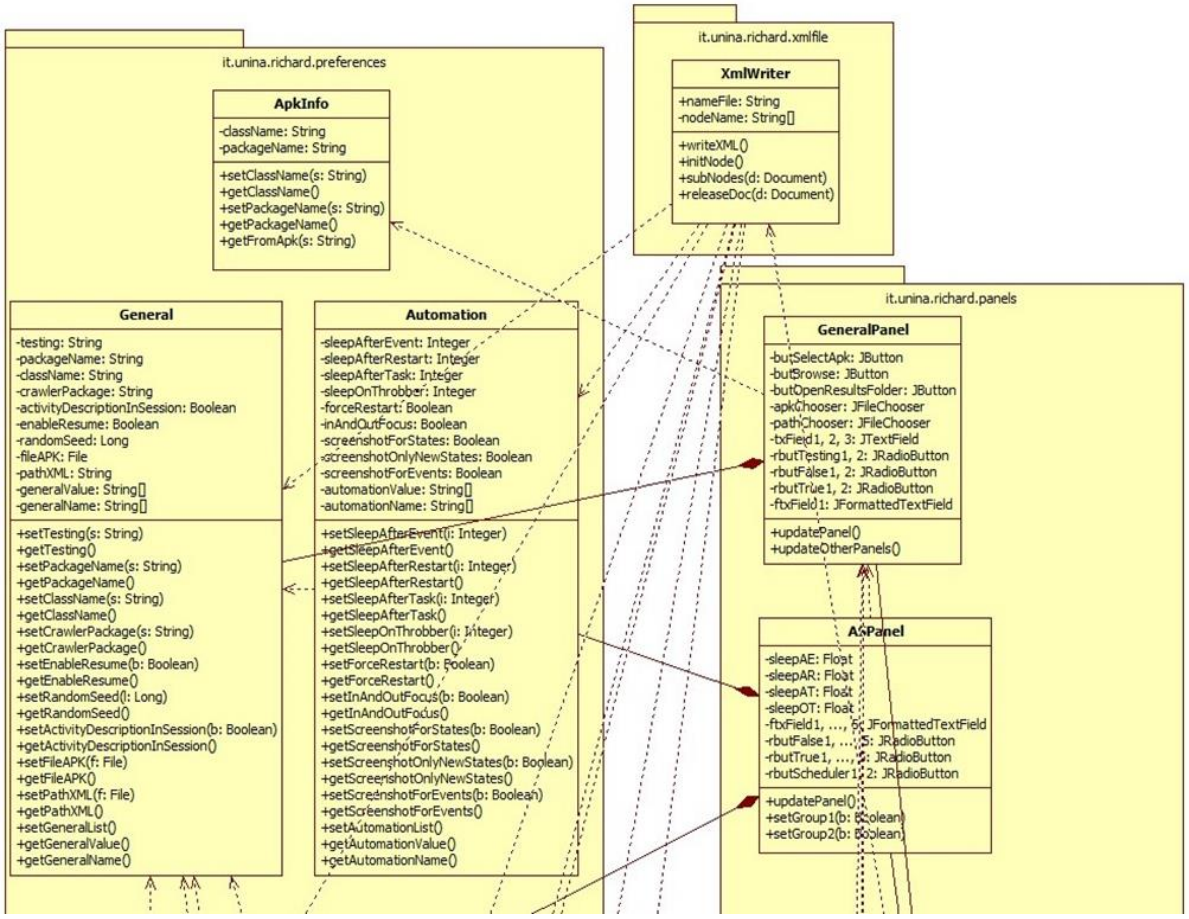
- un'interfaccia grafica intuitiva, semplice, correlata di opportuni dialog e suggerimenti per una configurazione guidata dei parametri del GUI Ripper
- meccanismi per la produzione di un opportuno file *preferences*
- meccanismi che impostano in automatico i parametri con valori di default (ottimali)
- meccanismi di controllo, verifica e validazione dei parametri (tipi e vincoli di coerenza)

2.2.1 Architettura software

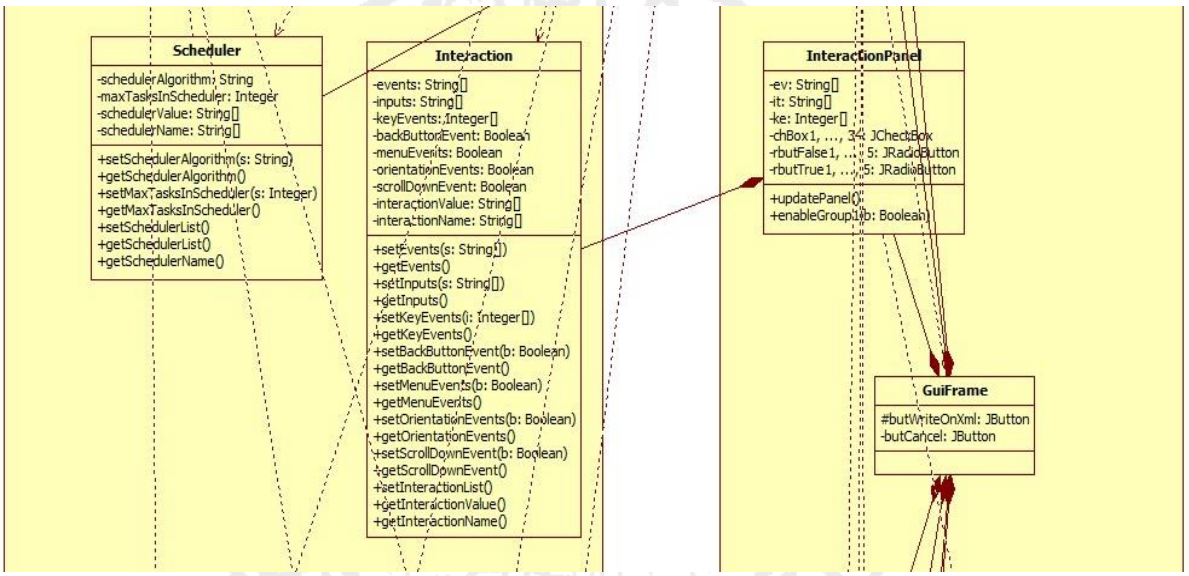
L'applicazione PreferencesGui è stata progettata considerando due livelli:

- livello contenente componenti software che gestiscono tutta la logica di verifica/validazione, impostazione/salvataggio e reperimento dei valori dei parametri
- livello contenente componenti software che gestiscono l'interfaccia grafica e il suo comportamento

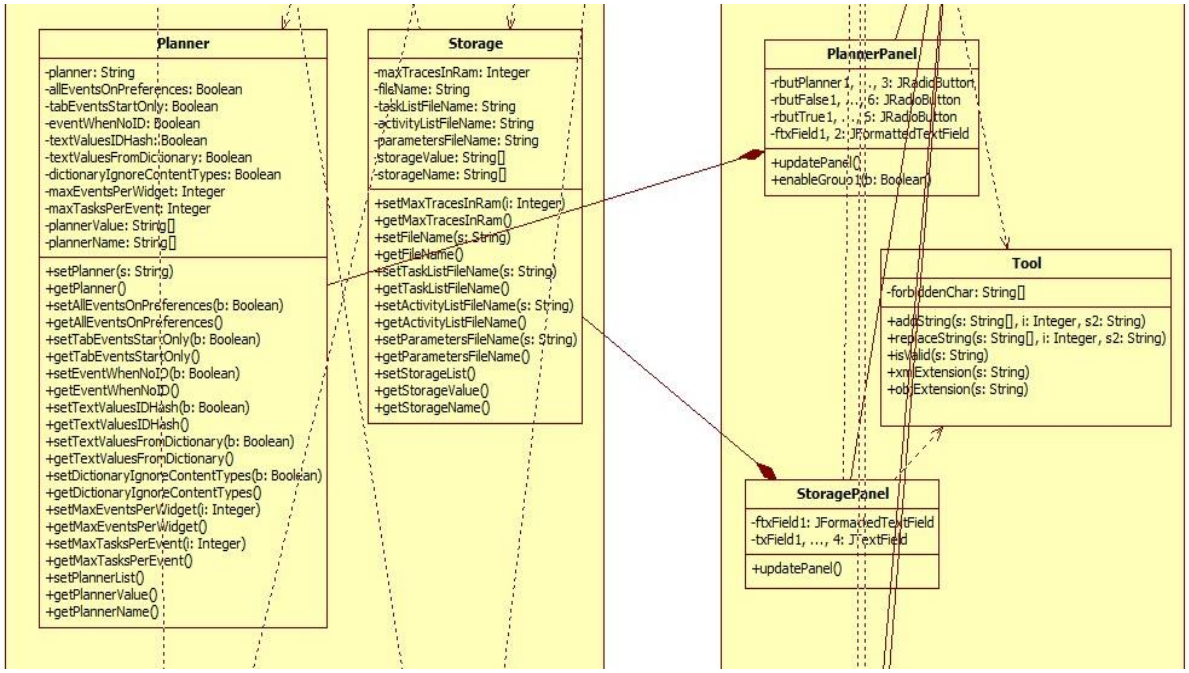
Di seguito è riportato il *class diagram* dell'applicazione:



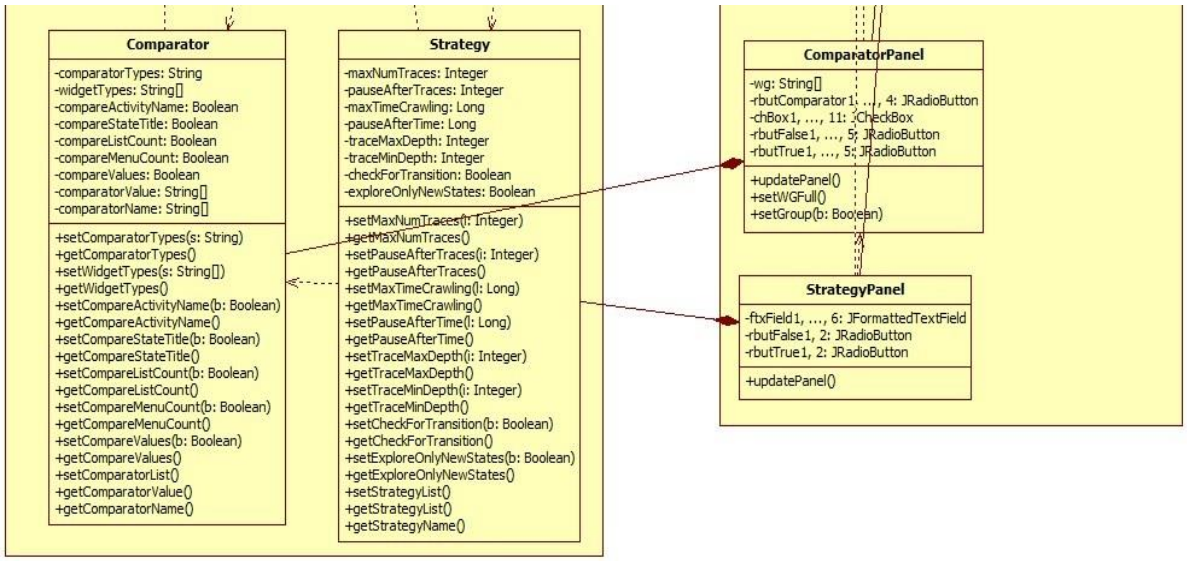
2.1 – Class diagram of the PreferencesGui (part 1)



2.2 – Class diagram of the PreferencesGui (part 2)



2.3 – Class diagram of the PreferencesGui (part 3)



2.4 Class diagram of the PreferencesGui (part 4)

2.2.2 Descrizione delle classi

Il package *preferences* contiene le classi responsabili della gestione dei valori dei parametri del GUI Ripper. Tali classi sono state create seguendo la classificazione in sezioni dei parametri: la classe *General* gestisce i parametri della sezione "general", la classe *Automation* gestisce quelli della sezione "automation", e così via.

Le classi permettono l'impostazione/memorizzazione e il reperimento dei valori dei

parametri attraverso gli appositi metodi *set()* e *get()*: inoltre meccanismi di controlli sulla validazione (tipi e vincoli di coerenza) dei parametri sono stati implementati negli stessi metodi *set()*.

La classe *ApkInfo* è responsabile del reperimento dei valori di parametri legati all'applicazione Android (Package Name e Class Name), presenti nel file Manifest dell'APK: la classe fornisce il metodo *getFromApk()*, richiamato ad ogni selezione del file APK, che ricava informazioni dal file ed imposta automaticamente i parametri Package Name e Class Name.

Il package *panels* contiene le classi responsabili della gestione dell'interfaccia grafica, del suo comportamento e delle interazioni con l'utente.

L'interfaccia grafica del PreferencesGui è suddivisa in schede, seguendo la classificazione in sezioni dei parametri del GUI Ripper, e ognuna delle quali è implementata da una specifica classe "Panel": la classe *GeneralPanel* implementa la scheda "General", la classe *ASPanel* implementa la scheda "A & S", e così via.

Ogni classe Panel contiene un'istanza della corrispondente classe del package preferences, che utilizza sia per la memorizzazione dei valori inseriti dall'utente (facendo uso dei corrispondenti metodi set()) sia per mostrare gli attuali valori dei parametri (utilizzando i metodi get()). Per quest'ultima funzionalità le classi Panel implementano il metodo *updatePanel()*, che consente l'aggiornamento della visualizzazione degli attuali valori della scheda: tale metodo è richiamato ad ogni inserimento di valore nella scheda e utilizza i metodi get().

Le classi Panel supportano una configurazione guidata dei parametri, fornendo messaggi (informazioni, alert, suggerimenti) e abilitando/disabilitando elementi grafici (per garantire la non violazione dei vincoli di coerenza dei parametri).

La classe *Tool* fornisce metodi comuni a tutte le classi Panel del package panels: è stata sviluppata per ottimizzare il codice dell'applicazione.

Infine, la classe *GuiFrame* implementa un frame (finestra) e contiene tutte le classi Panel: inoltre fornisce il pulsante *Write On XML* per la generazione del file preferences.

Il package *xmlfile* contiene la sola classe *XmlWriter*, che fornisce i metodi per l'operazione

di scrittura dei parametri nel file preferences. Tale classe utilizza i metodi *getValue()* e *getName()*, forniti dalle classi del package preferences, per ottenere la lista dei parametri con gli attuali valori settati. I metodi per la scrittura del file preferences vengono richiamati infine dalla classe *GuiFrame* (package panels) quando verrà cliccato il pulsante *Write on XML*.

2.2.3 Funzionamento dell'interfaccia grafica

Vediamo ora in che modo l'interfaccia utente opera, durante l'esecuzione del PreferencesGui.

Innanzitutto, l'applicazione è scritta in Java e quindi per avviarla occorre aprire il suo file eseguibile Java *PreferencesGui.jar* (basta anche cliccarci due volte su). L'apertura del file provoca l'invocazione del metodo *main()* presente nella classe *Main* dell'applicazione (presente in tutte le applicazioni Java).

In particolare il main() creerà un oggetto GuiFrame (contenitore di pannelli), il quale a sua volta creerà gli oggetti *Panel* (uno per ogni classe *Panel*) e questi ultimi creeranno gli oggetti delle corrispondenti classi del package *preferences*: l'interfaccia grafica viene così creata.

La GUI del PreferencesGui è composta da una serie di elementi grafici (bottoni, checkbox, textfield, etc.), ognuno dei quali è associato ad un particolare paramentro del GUI Ripper e permette di impostarlo con un valore inserito dall'utente. Tali elementi grafici sono ereditati/riutilizzati dai set di componenti grafici *Swing* e *Awt*, e sono implementati nelle classi Panel del package panels.

Ogni elemento grafico ha con se un *listener* (opportunamente implementato nella classe Panel contenente l'elemento) che, ad ogni interazione dell'utente con l'elemento (per inserire un valore), esegue un'azione composta dalle seguenti operazioni:

- impostazione del parametro associato con il valore inserito dall'utente, richiamando il relativo metodo set()
- aggiornamento della scheda corrente della GUI, richiamando il metodo updatePanel()

- visualizzazione di un messaggio se il valore inserito non è accettabile o se l'impostazione di quel valore ha fatto cambiare valore ad altri parametri

Non tutti i listener presenti, però, implementano esattamente queste operazioni: in particolare vi sono due listener che effettuano operazioni leggermente diverse dagli altri.

Il listener del bottone *Select .apk,* quando l'utente decide di inserire il file APK, esegue le seguenti operazioni:

- richiama il metodo getFromApk() della classe ApkInfo per ricavare le informazioni utili dal file
- imposta i parametri Package Name e Class Name utilizzando i relativi metodi set() e aggiorna la scheda con il metodo updatePanel()

Il listener del bottone *Write on XML,* quando l'utente decide di generare il file preferences, esegue le seguenti operazioni:

- aggiorna prima tutte le schede con gli attuali valori dei parametri
- richiama il metodo WritreXml() della classe XmlWriter per generare il file preferences

2.2.3.1 Il metodo set()

I metodi set() sono presenti ed implementati in ogni classe del package preferences associata ad una sezione dei parametri del GUI Ripper: ogni metodo è logicamente associato ad uno di questi parametri.

Il metodo riveste un ruolo chiave nel settaggio del parametro associato, poiché non solo salva il valore inserito, ma controlla anche la validità del valore e l'eventuale vincolo di coerenza a cui potrebbe sottostate il parametro.

Buona parte della logica di controllo e validazione è stata implementata nei metodi set(). Prima di salvare un parametro inserito nell'interfaccia, infatti, il metodo set() richiamato effettua dei controlli, valutando la possibilità o meno di compiere l'operazione di salvataggio.

2.2.3.2 Il medoto updatePanel()

I metodi updatePanel() sono presenti ed implementati in ogni classe Panel del package

panels.

Il metodo implementa buona parte del comportamento visivo dell'interfaccia grafica del PreferencesGui, in quanto:

- aggiorna la scheda dell'interfaccia con i valori attuali dei parametri presenti in essa
- attiva/disattiva determinati elementi grafici in base ai vincoli di coerenza a cui possono sottostare parametri

Il metodo riveste un ruolo chiave nell'operazione di settaggio dei parametri, in quanto supporta il corretto inserimento dei valori: non permette infatti l'inserimento dei valori per tutti quei parametri sottostanti a vincoli di coerenza.

Buona parte della logica di verifica è stata implementata nei metodi updatePanel().

Quando viene settato un parametro con un certo valore, il metodo updatePanel() non solo aggiorna la scheda corrente, ma verifica quali parametri della scheda si trovano a sottostare ai vincoli di coerenza e disattiva, se il caso, i componenti grafici associati a quei parametri.

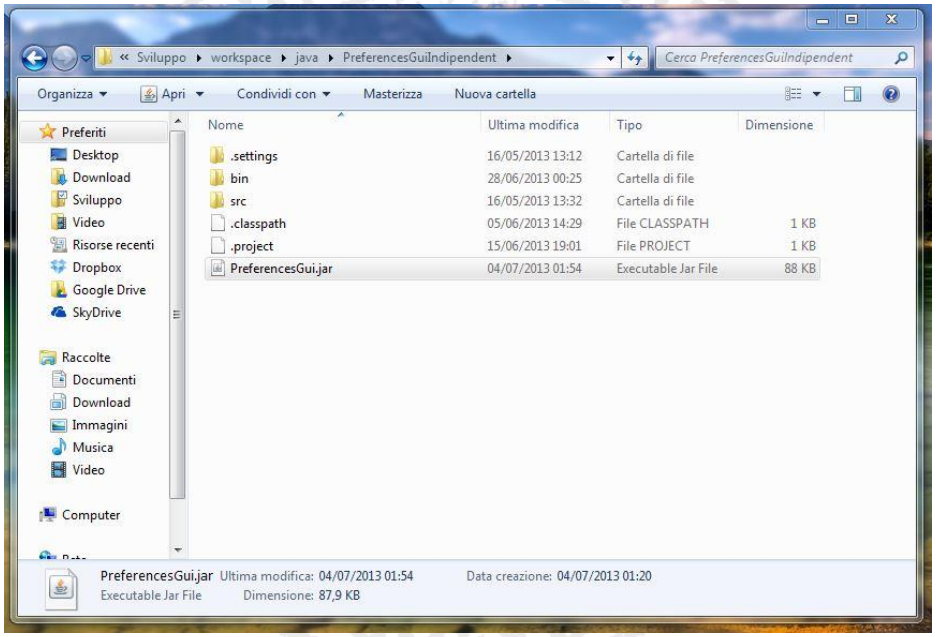
Capitolo 3

Esempi di utilizzo

In questo capitolo verrà illustrato un tipico utilizzo del PreferencesGui. In particolare, dato che lo strumento è un'applicazione stand-alone, ma lavora in funzione del GUI Ripper, si dovranno avviare entrambi i programmi per dimostrare la *correttezza* del PreferencesGui.

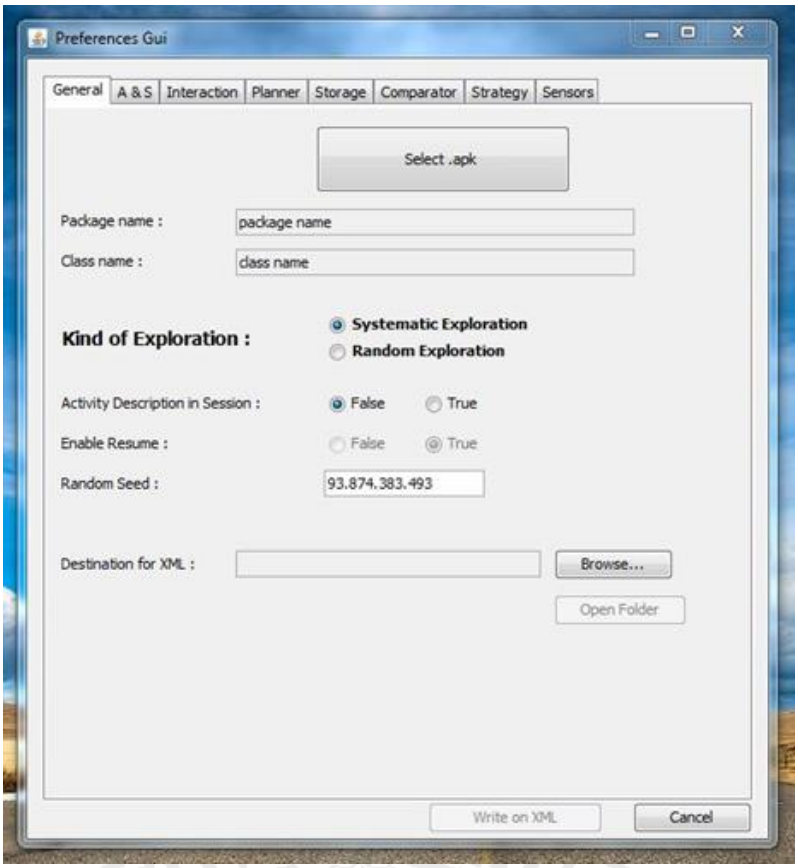
3.1 Un tipico utilizzo del PreferencesGui

Il PreferencesGui è scritto in Java, dunque per avviare l'applicazione occorre aprire il corrispondente file ".jar" (*PreferencesGui.jar*), anche semplicemente col doppio click.



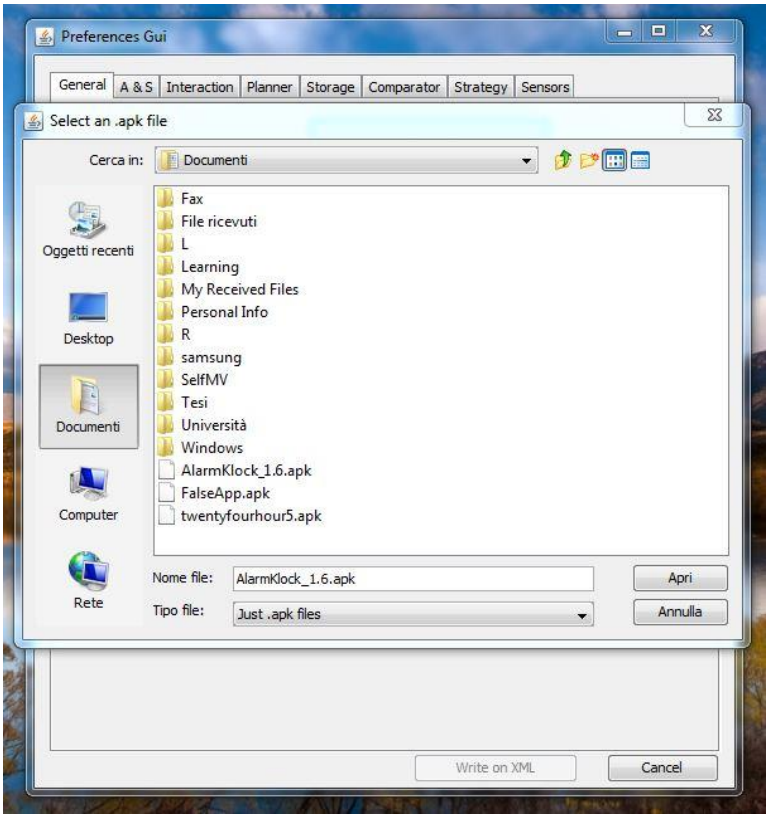
4.1 – File Preferences.jar

Il PreferencesGui viene così avviato. L'applicazione presenta un'interfaccia grafica intuitiva a schede, di cui la scheda "General" rappresenta la schermata principale del programma.

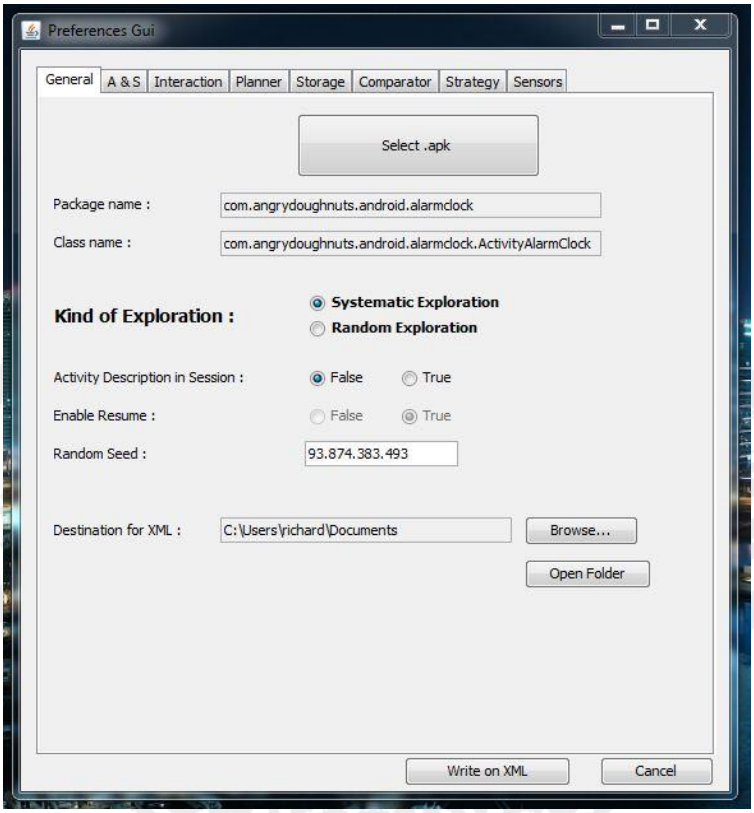


4.2 – Schermata principale del PreferencesGui

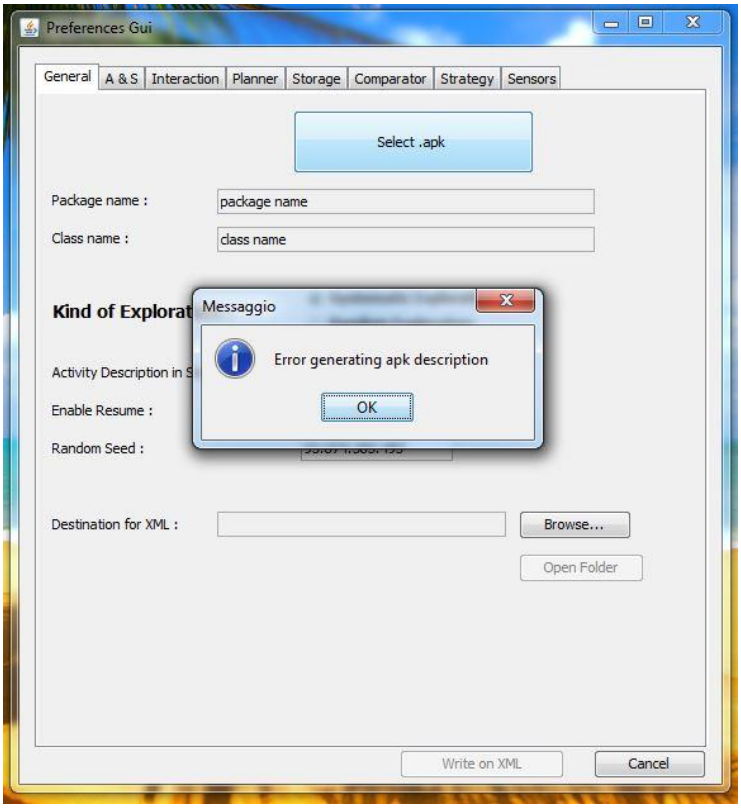
Da ora è possibile svolgere qualunque operazione legale. Logicamente, la prima operazione consigliata è quella della selezione del file APK: ciò può essere fatto attraverso il click sul bottone grande *Select .apk*. La selezione di un file corretto provocherà la generazione di un avviso circa la cartella di destinazione del file preferences, il settaggio automatico dei parametri *Package Name* e *Class Name* e l'abilitazione del pulsante *Write on XML*; la selezione di un file non corretto provocherà invece la generazione di un alert di errore.



4.3 Selezione del file APK



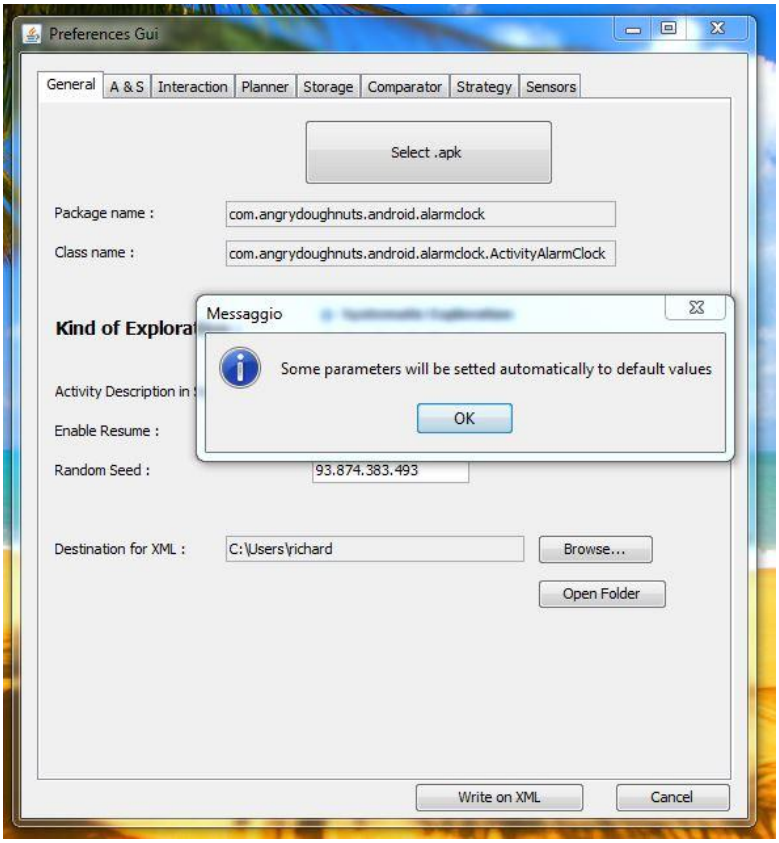
4.4 – File APK corretto



4.5 – File APK non corretto

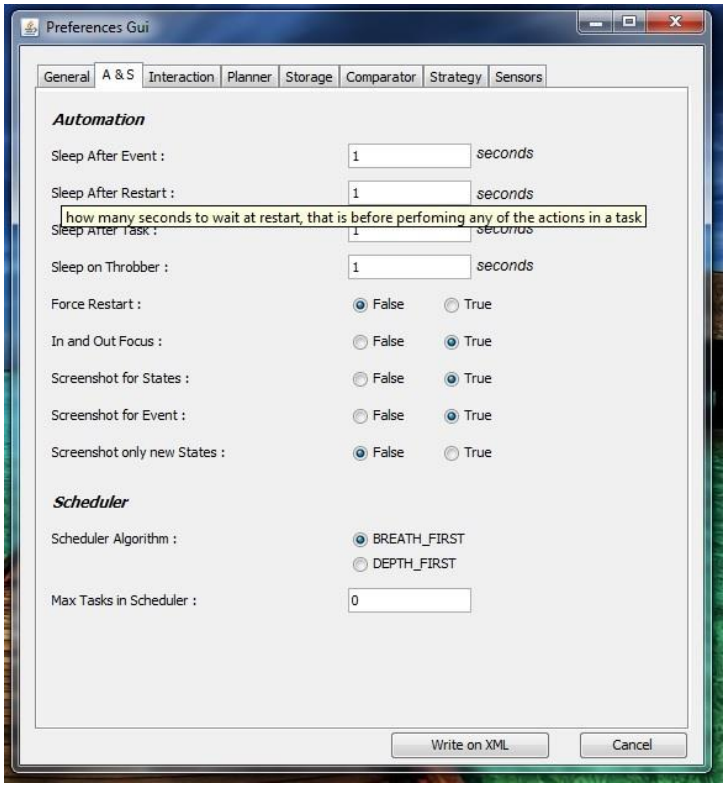
Si può ora iniziare a configurare i parametri. L'interfaccia grafica del PreferencesGui è composta da schede, che seguono la classificazione in sezione dei parametri del GUI Ripper (i parametri di una sezione si troveranno nella scheda corrispondente). Inoltre i parametri sono già settati ai loro valori di default.

Il settaggio di ogni parametro avviene interagendo con gli elementi grafici (campi di testo o bottoni) presenti a fianco del nome del parametro. Si consiglia di impostare innanzitutto il parametro *Kind of Exploration*: a seconda del suo valore , infatti, il PreferencesGui imposta automaticamente il valore di altri parametri (anche se erano stati già impostati, con conseguente perdita dei settaggi precedenti).

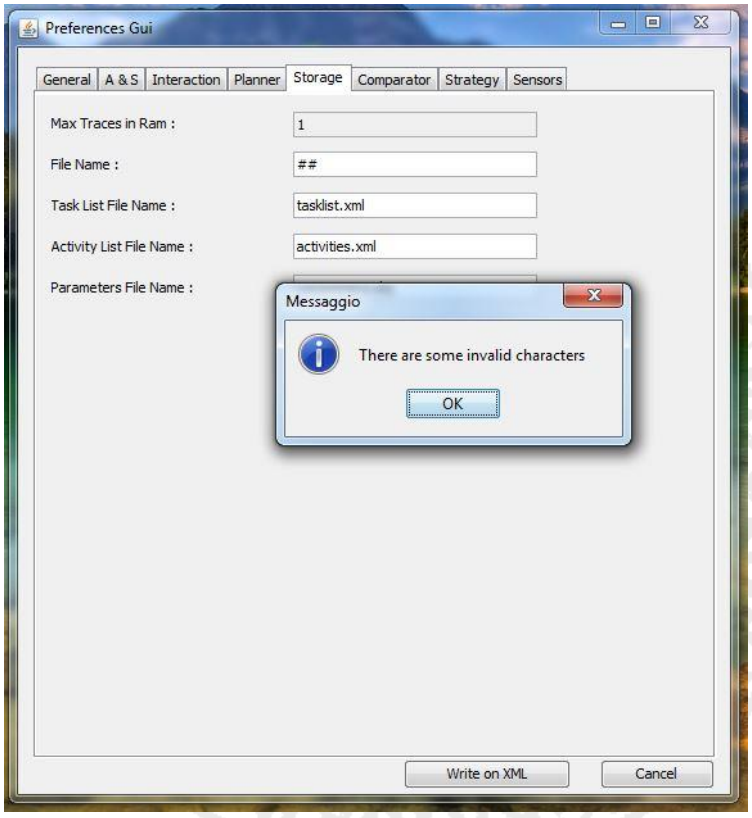


4.6 – Settaggio di Kind of Exploration

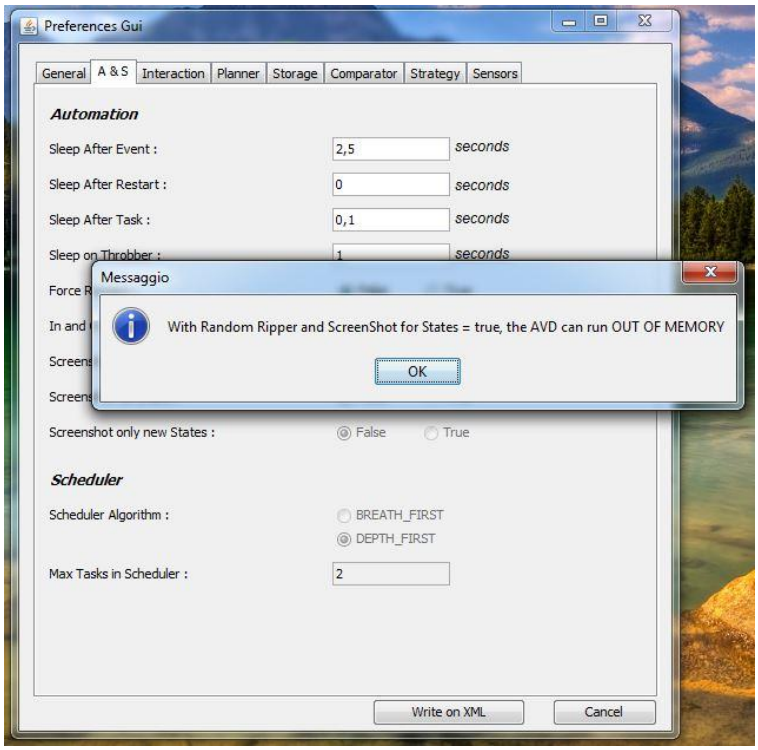
La configurazione dei parametri si presenta come un'operazione guidata per l'utente, data la presenza di tooltips (suggerimenti) a fianco ai nomi dei parametri, di alert di errore di inserimento di valore (e automatica re-impostazione al valore di default), di alert di informazione, di componenti grafici disabilitati (che rispecchiano i vincoli di valore dei parametri)



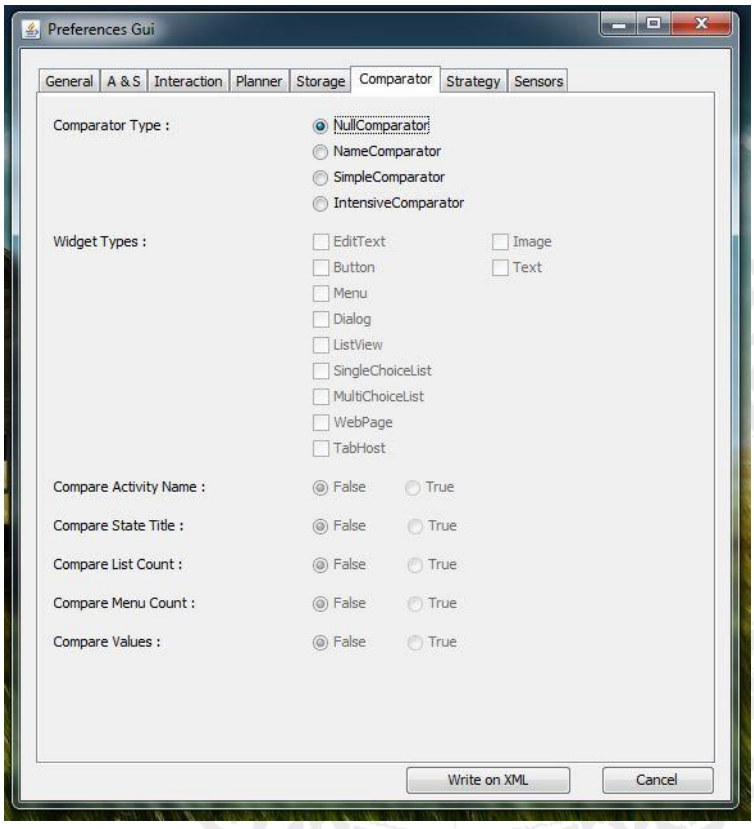
4.7 – Tooltip



4.8 – Alert di errore di inserimento valore

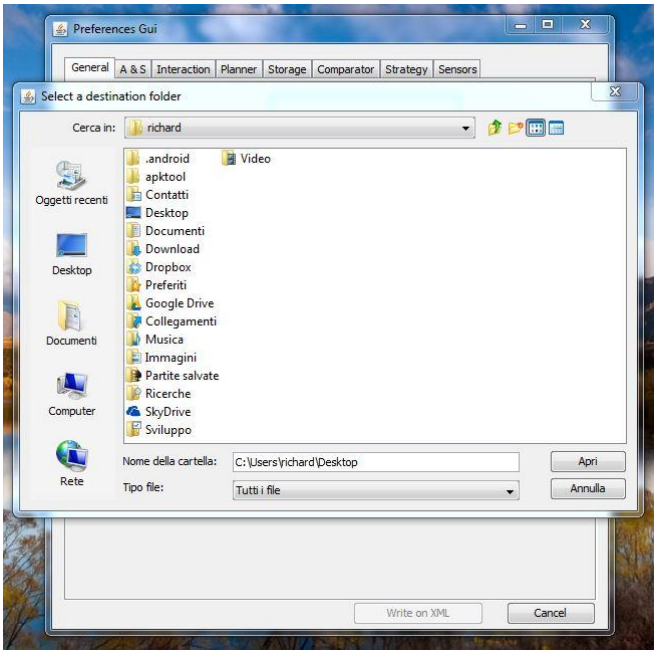


4.9 – Alert di informazione

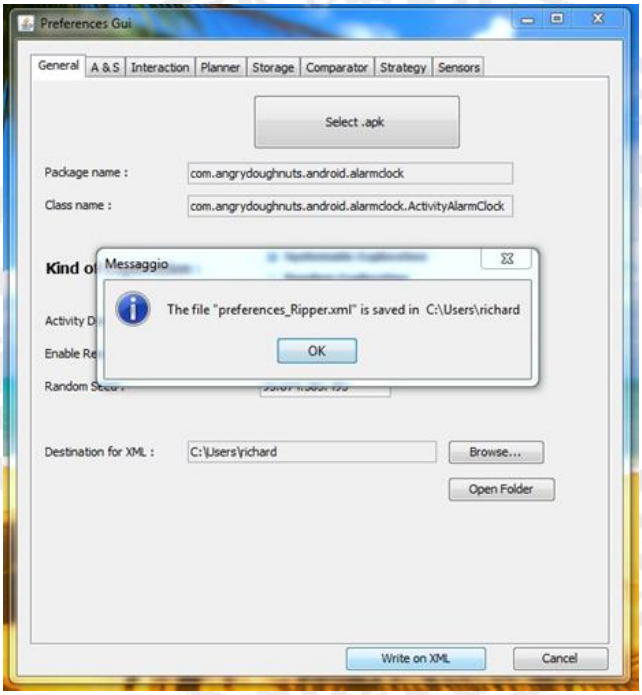


4.10 – Elementi grafici disabilitati

Una volta completato il settaggio dei parametri, l'ultima operazione consigliata, prima della generazione del file preferences, è la selezione della cartella di destinazione di quest'ultimo (se non si vuole usare quella di default), cliccando sul bottone *Browse* (scheda *General*). Infine la configurazione può essere trascritta nel file preferences cliccando sul bottone *Write on XML*.



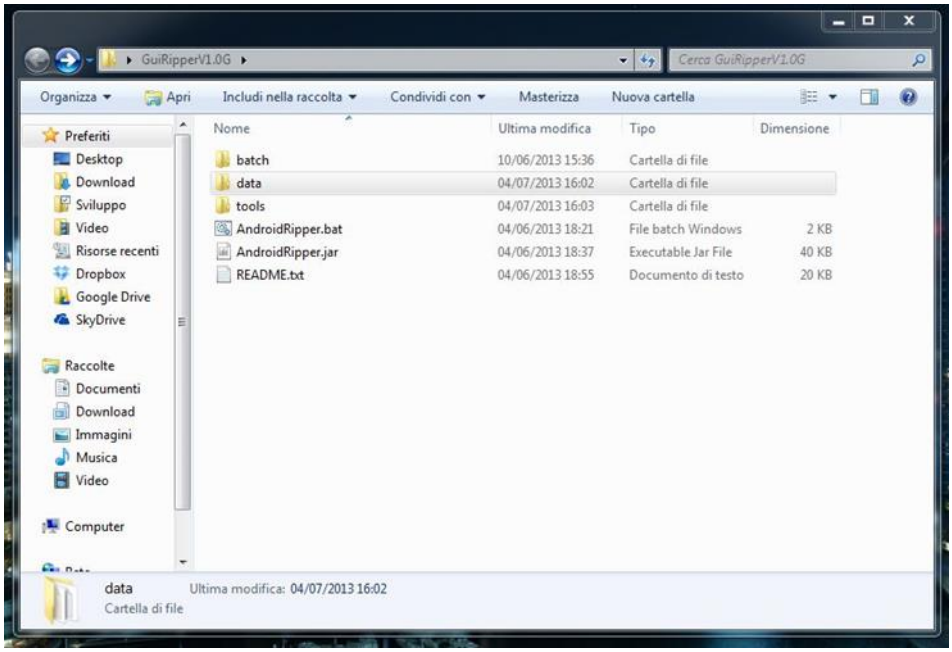
4.11 – Selezione cartella di destinazione



4.12 – Generazione del file preferences

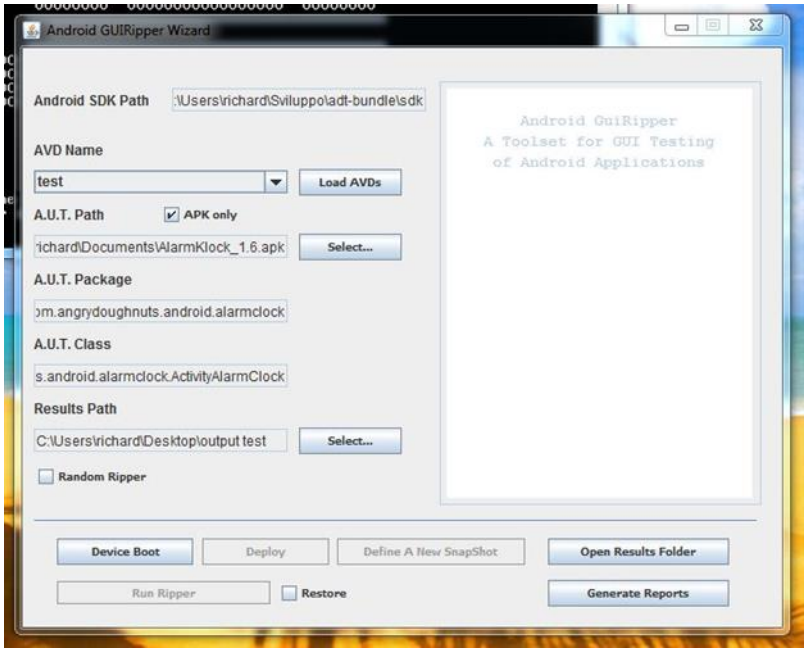
3.2 Utilizzo del file preferences generato

Una volta generato il file preferences, occorre spostarlo nella cartella *data* del GUI Ripper: dato che quest'ultimo ha già un file preferences di default, occorrerà sostituirlo con quello precedentemente creato con il PreferencesGui.



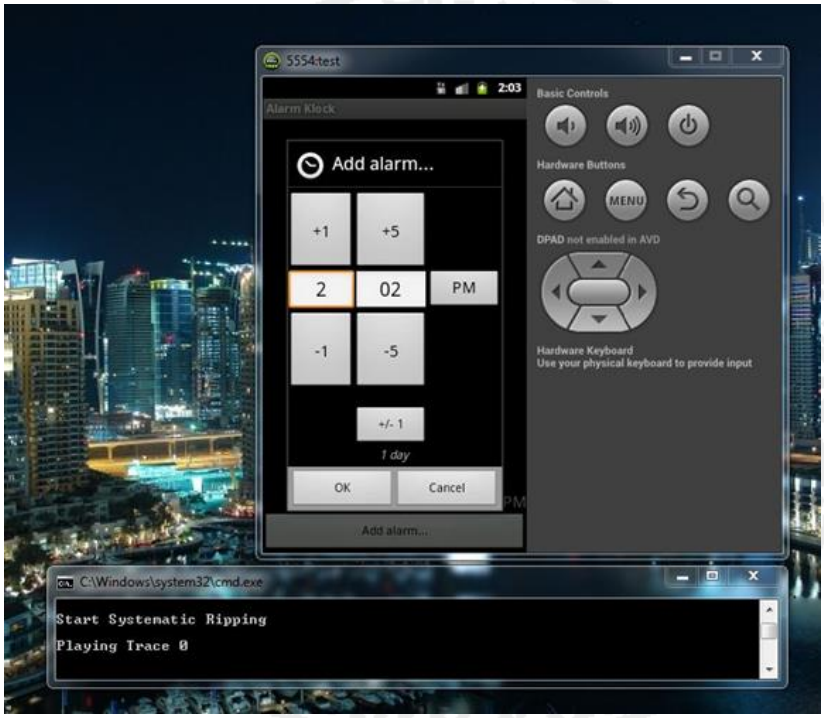
4.13 – Cartella "data" del GUI Ripper

In seguito, occorrerà avviare il GUI Ripper, attraverso l'apertura del file *AndroidRipper.bat*. Ora, poiché PreferencesGui è un applicazione stand-alone, una volta comparso il wizard del GUI Ripper si dovrà configurare anch'esso (in particolare si dovrà ri-selezionare il file APK utilizzato nel PreferencesGui).



4.14 – Wizard del GUI Ripper

Infine, si potrà avviare il GUI Ripper cliccando sul pulsante *Device Boot* (e seguendo le istruzioni che compariranno nel wizard). Il GUI Ripper eseguirà il test dell'applicazione Android utilizzando tutte le impostazioni scelte nel PreferencesGui (cioè leggendo il file preferences creato precedentemente).



4.15 – GUI Ripper in azione

Capitolo 4

Test del PreferencesGui

Il PreferencesGui, come qualunque software, necessita di attività di test per la rilevazione di guasti e per la verifica della qualità.

Di seguito verrà documentato un piano di test del PreferencesGui in cui i casi di test saranno "focalizzati sugli input" dell'interfaccia. In particolare, dati gli input da inserire, e i conseguenti comportamenti e output "previsti" del PreferencesGui, si inseriranno tali input e poi si valuteranno i risultati (comportamenti e output effettivi) rispetto alle previsioni. Tale piano di test prevede di testare gli input in base:

- al tipo (test di input numerici, di input booleani, etc.)
- alla validità (test di input che rispettano/non rispettano i tipi e i vincoli di coerenza dei parametri)

Gli input coinvolgeranno solo una parte dei parametri esistenti e si cercherà di considerarli sempre diversi rispetto ai valori di default. Invece, i parametri non coinvolti, saranno lasciati ai loro valori di default (non avranno input). Alla fine dell'inserimento degli input, si tenterà di generare il file preferences, per valutare se i valori scritti nel file coicidono con quelli visualizzati nell'interfaccia.

Nella trattazione seguente verrà considerato, di volta in volta, una "coppia" di casi di test: la modalità con cui saranno scelti gli input verrà così applicata ad entrambi i tipi di esplorazione (sistematica e random).

Tutto ciò dovrebbe garantire una buona copertura di tutte le combinazioni possibili degli

input.

4.1 Test con input validi

Per input *valido* si intende quel valore che rispetta il tipo e i vincoli di coerenza del parametro corrispondente.

4.1.1 Input numerici

- Input: inserimento di valori numerici validi per tutti i parametri di tipo numerico, in particolare:

Parametro	*Systematic*	*Random*
Random Seed	9000000000	10000000000
Sleep after Event	1,5	0,5
Sleep after Restart	1,5	0,5
Sleep after Task	1,5	0,5
Sleep on Throbber	1,5	0,5
Max Tasks in Scheduler	5	nessuno (non modificabile)
Max Events Per Widget	1	2
Max Tasks Per Event	nessuno (non modificabile)	nessuno (non modificabile)
Max Traces in Ram*	2	3
Max Num Traces	5	4
Pause after Traces	2	3
Max Time Crawling	6	8
Pause after Time	9	10
Trace Max Depth	10	8
Trace Min Depth	6	4
restanti parametri	lasciati a default	lasciati a default

*per inserire un valore, si è considerato Enable Resume = false

I casi di test terminano con la generazione del file preferences.

- Previsioni: l'applicazione dovrebbe riconoscere gli input inseriti come validi/accettabili: dovrebbe accettare l'inserimento degli input e visualizzare correttamente nella GUI tali input inseriti; l'applicazione dovrebbe generare correttamente il file preferences: i valori contenuti nel file dovrebbero essere uguali agli input visualizzati nella GUI

- Risultati: l'applicazione ha riconosciuto gli input come accettabili/validi: accetta l'inserimento di tutti gli input e visualizza correttamente nella GUI tutti gli input inseriti;

l'applicazione genera il file preferences correttamente (contenente gli stessi input visualizzati nella GUI)

4.1.2 Input booleani

- Input: inserimento di valori booleani per tutti i parametri di tipo booleano, in particolare:

Parametro	*Systematic*	*Random*
Activity Description in Session	true	true
Enable Resume	false	false
Force Restart	true	true
In and Out Focus	false	false
Screenshot for States	false	true
Screenshot for Event*	nessuno (non modificabile)	true
Back Button Event	false	false
Menu Button Events	false	false
Change Orientation Events	false	false
Scrolldown Event	true	true
All Events on Preferences	false	false
Tab Events Start only	true	true
Event when no ID	false	false
Text Values ID Hash**	false	nessuno (non modificabile)
Text Values from Dictionary**	nessuno (non modificabile)	true
Dictionary Ignore Content Type**	nessuno (non modificabile)	true
Compare Activity Name***	false	nessuno (non modificabile)
Compare State Title***	false	nessuno (non modificabile)
Compare List Count***	true	nessuno (non modificabile)
Compare Menu Count***	false	nessuno (non modificabile)
Compare Values***	nessuno (non modificabile)	nessuno (non modificabile)
Check for Transition	true	true
Explore only new States***	false	nessuno (non modificabile)
restanti parametri	lasciati a default	lasciati a default

* ScreenShot for States=false impone che Screenshot for Event=false

**in Systematic si è considerato Planner=SimplePlanner, in Random invece Planner=DictionarySimplePlanner

*** Random impone che Comparator Type=NullComparator ed Explore only new States=false

I casi di test terminano con la generazione del file preferences.

- Previsioni: l'applicazione dovrebbe riconoscere gli input inseriti come validi: dovrebbe accettare l'inserimento degli input e visualizzare correttamente nella GUI tali input inseriti; l'applicazione dovrebbe generare correttamente il file preferences: i valori contenuti nel file dovrebbero essere uguali agli input visualizzati nella GUI

- Risultati: l'applicazione ha riconosciuto gli input come accettabili/validi: accetta

l'inserimento di tutti gli input e visualizza correttamente nella GUI tutti gli input inseriti; l'applicazione genera il file preferences correttamente (contenente gli stessi input visualizzati nella GUI)

4.1.3 Input stringhe

- Input: inserimento di valori per tutti i parametri di tipo stringa, in particolare:

Parametro	*Systematic*	*Random*
Scheduler Algorithm*	depth_first	nessuno (non modificabile)
Events	click(button), setbar(seekbar, ratingbar), selectlistitem, selectspinneritem, swaptab, writetext in searchbar	longclick(image), longclicklistitem, selectradioitem, focus,
Key Events	home button, volume -	volume +
Inputs	click(checkbox), writetext, selectspinneritem	setbar(seekbar), edittext
Planner	simplereflectionplanner	dictionarysimpleplanner
File Name	file1.xml	file11.xml
Task List File Name	file2.xml	file22.xml
Activity List File Name	file3.xml	file33.xml
Parameters File Name	file4.obj	file44.obj
Comparator Type*	intensivecomparator	nessuno (non modificabile)
Widget Types*	edittext, menu, singlechoicelist, text	"" (stringa vuota)
restanti parametri	lasciati a default	lasciati a default

*Random impone Scheduler Algorithm=depth_first, Comparator Type=nullcomparator e Widget Types=""

I casi di test terminano con la generazione del file preferences.

- Previsioni: l'applicazione dovrebbe riconoscere gli input inseriti come validi: dovrebbe accettare l'inserimento degli input e visualizzare correttamente nella GUI tali input inseriti; l'applicazione dovrebbe generare correttamente il file preferences: i valori contenuti nel file dovrebbero essere uguali agli input visualizzati nella GUI

- Risultati: l'applicazione ha riconosciuto gli input come accettabili/validi: accetta l'inserimento di tutti gli input e visualizza correttamente nella GUI tutti gli input inseriti; l'applicazione genera il file preferences correttamente (contenente gli stessi input visualizzati nella GUI)

4.1.4 Input file e cartelle

Osservazione: per file APK valido si intente quello ottenuto dal Market Android o dal sito

dello sviluppatore

- Input: selezione del file APK e della cartella di destinazione del file preferences, in particolare:

Parametro	*Systematic*	*Random*
File .apk	AlarmKlock_1.6.apk	twentyfourhour5.apk
Cartella di destinazione	C:\Users\richard\Desktop	C:\Users\richard\Documents
restanti parametri	lasciati a default	lasciati a default

I casi di test terminano con la generazione del file preferences.

- Previsioni: l'applicazione dovrebbe riconoscere gli input inseriti come validi: dovrebbe accettare la selezione del file APK e della cartella di destinazione del file preferences, dovrebbe visualizzare correttamente nella GUI i valori di Package Name e Class Name, nonché l'input inserito circa la cartella di destinazione; l'applicazione dovrebbe generare correttamente il file preferences nella cartella di destinazione: il file dovrebbe contenere i valori di Package Name e Class Name che sono visualizzati nella GUI

- Risultati: l'applicazione riconosce gli input inseriti come validi: accetta la selezione del file APK e della cartella di destinazione, visualizza correttamente nella GUI i valori di Package Name e Class Name e l'input della cartella di destinazione; l'applicazione genera il file preferences nella cartella di destinazione e il contenuto del file è corretto

4.1.5 Input numerici "limite"

- Input: inserimento di valori numerici per tutti i parametri di tipo numerico, tali che si avvicinano ad un valore limite (superiore o inferiore) imposto dal programma o dall'elaboratore stesso (cioè numeri molto grandi o molto piccoli). Per gli input reali si è scelto un numero piccolo (i valori devono essere convertiti in millisecondi), mentre per gli input interi un numero molto grande, in particolare:

Parametro	*Systematic*	*Random*
Random Seed	900000000000000000	800000000000000000
Sleep after Event	0,001	0,002
Sleep after Restart	0,001	0,002
Sleep after Task	0,001	0,002
Sleep on Throbber	0,001	0,002
Max Tasks in Scheduler	900000000	nessuno (non modificabile)
Max Events Per Widget	900000000	800000000
Max Tasks Per Event	nessuno (non modificabile)	nessuno (non modificabile)

Max Traces in Ram*	900000000	800000000
Max Num Traces	900000000	800000000
Pause after Traces	900000000	800000000
Max Time Crawling	900000000	800000000
Pause after Time	900000000	800000000
Trace Max Depth	900000000	800000000
Trace Min Depth	900000000	800000000
restanti parametri	lasciati a default	lasciati a default

*per inserire un valore, bisogna considerare Enable Resume = false

I casi di test terminano con la generazione del file preferences.

- Previsioni: l'applicazione dovrebbe riconoscere gli input inseriti come validi: dovrebbe accettare l'inserimento degli input e visualizzare correttamente nella GUI tali input inseriti; l'applicazione dovrebbe generare correttamente il file preferences: i valori contenuti nel file dovrebbero essere uguali agli input visualizzati nella GUI

- Risultati: l'applicazione ha riconosciuto gli input come accettabili/validi: accetta l'inserimento di tutti gli input e visualizza correttamente nella GUI tutti gli input inseriti; l'applicazione genera il file preferences correttamente (contenente gli stessi input visualizzati nella GUI)

4.1.6 Input che impongono vincoli di coerenza

- Input: inserimento di tutti quei particolari valori che impongono vincoli di coerenza su altri parametri, in particolare:

Parametro	Systematic	Random
Activity Description in Session	false	false
Enable Resume	nessuno (non modificabile)	nessuno (non modificabile, già vincolato)
Screenshot for States	false	false
Events	click, longclick, setbar	click, longclick, setbar
Inputs	click, setbar	click, setbar
Planner	dictionarysimpleplanner	simplereflectionplanner
Comparator Type	intensivecomparator	nessuno (non modificabile)
restanti parametri	lasciati a default	lasciati a default

I casi di test terminano con la generazione del file preferences

- Previsioni: l'applicazione dovrebbe riconoscere gli input inseriti come validi: dovrebbe accettare l'inserimento degli input e visualizzare correttamente nella GUI tali input inseriti; l'applicazione dovrebbe generare correttamente il file preferences: i valori contenuti nel file

dovrebbero essere uguali agli input visualizzati nella GUI

- Risultati: l'applicazione ha riconosciuto gli input come accettabili/validi: accetta l'inserimento di tutti gli input e visualizza correttamente nella GUI tutti gli input inseriti; l'applicazione genera il file preferences correttamente (contenente gli stessi input visualizzati nella GUI)

4.2 Test con input non validi

Per input *non valido* si intende quel valore che non rispetta il tipo o i vincoli di valore del parametro corrispondente.

Il PreferencesGui offre un'interfaccia grafica avente componenti grafici per settare i parametri e comportamenti particolari (come la disattivazione di tali elementi), tali da evitare l'inserimento di valori non corretti: tutto ciò si traduce in un minor numero di casi di test rispetto alla sezione precedente.

In realtà una migliore strategia di test, per questo tipo di input, sarebbe stata quella che considera per ogni caso di test un solo input non valido: ciò avrebbe consentito una maggiore copertura di tutte le possibili combinazioni di input, ma d'altro canto avrebbe aumentato di molto il numero dei casi di test da considerare. Per brevità di trattazione, si sono considerati solo casi di test aventi tutti input non validi.

4.2.1 Input numerici

- Input: inserimento di valori numerici non validi per tutti i parametri di tipo numerico, in particolare

Parametro	*Systematic*	*Random*
Random Seed	-9000000000	-10000000000
Sleep after Event	-1,5	-0,5
Sleep after Restart	-1,5	-0,5
Sleep after Task	-1,5	-0,5
Sleep on Throbber	-1,5	-0,5
Max Tasks in Scheduler	-5	nessuno (non modificabile)
Max Events Per Widget	-1	-2
Max Tasks Per Event	nessuno (non modificabile)	nessuno (non modificabile)
Max Traces in Ram*	-2	-3
Max Num Traces	-5	-4
Pause after Traces	-2	-3
Max Time Crawling	-6	-8

Pause after Time	-9	-10
Trace Max Depth	-10	-8
Trace Min Depth	-6	-4
restanti parametri	lasciati a default	lasciati a default

*per inserire un valore, bisogna considerare Enable Resume = false

I casi di test terminano con la generazione del file preferences.

- Previsioni: l'applicazione dovrebbe riconoscere come non validi gli input inseriti: dovrebbe permettere il typing, ma non il salvataggio ne la visualizzazione degli input nella GUI, poichè dovrebbe re-impostare i parametri coinvolti ai loro valori di default; l'applicazione dovrebbe generare il file preferences non contenente gli input, ma i valori di default visualizzati nella GUI.

- Risultati: l'applicazione ha riconosciuto gli input come non validi: permette il typing, ma non il salvataggio ne la visualizzazione degli input nella GUI; l'applicazione re-imposta i parametri coinvolti ai loro valori di default; l'applicazione genera il file preferences non contenente gli input, ma i valori di default visualizzati nella GUI.

4.2.2 Input stringhe

- Input: inserimento di valori per quei parametri di tipo stringa che possono essere editati, in particolare:

Parametro	*Systematic*	*Random*
File Name	*.xml	*.xml
Task List File Name	#.xml	*.xml
Activity List File Name	.xml	.xml
Parameters File Name	'.obj	'.obj
restanti parametri	lasciati a default	lasciati a default

I casi di test terminano con la generazione del file preferences.

- Previsioni: le stesse del caso degli input numerici non validi

- Risultati: le stesse del caso degli input numerici non validi

4.2.3 Input file

Osservazione: non si è potuto considerare la scelta di una cartella non valida, in quanto per la selezione della cartella di destinazione il PreferencesGui fa uso del file-manager dell'OS (che già gestisce l'esistenza e la raggiungibilità delle directory).

- Input: selezione di un file APK non valido (non preso dal Market Android o dal sito dello

sviluppatore) : a tale scopo viene utilizzato un file con un'altra estensione, che viene cambiata in ".apk" prima di essere selezionato (in particolare il file *fakeapp.txt* è cambiato in *fakeapp.apk*)

I casi di test terminano con la generazione del file preferences.

- Previsioni: l'applicazione dovrebbe riconoscere l'input inserito come non valido: dovrebbe permettere la selezione del file APK, ma non dovrebbe visualizzare nessun valore per i parametri Package Name e Class Name; l'applicazione non dovrebbe generare il file preferences.

- Risultati: l'applicazione riconosce come non valido l'input inserito: permette la selezione del file APK, ma non visualizza nessun valore per i parametri Package Name e Class Name; l'applicazione non permette la generazione del file preferences.

4.2.4 Input numerici che superano il "limite"

- Input: valori numerici per tutti i parametri di tipo numerico, tali che superano un valore limite (superiore o inferiore) imposto dal programma o dall'elaboratore stesso (numeri molto grandi o molto piccoli). Per gli input reali si è scelto un numero piccolo (i valori devono essere convertiti in millisecondi), mentre per gli input interi un numero molto grande, in particolare:

Parametro	*Systematic*	*Random*
Random Seed	9000000000000000000	8000000000000000000
Sleep after Event	0,0001	0,0002
Sleep after Restart	0,0001	0,0002
Sleep after Task	0,0001	0,0002
Sleep on Throbber	0,0001	0,0002
Max Tasks in Scheduler	9000000000	2 (non modificabile)
Max Events Per Widget	9000000000	8000000000
Max Tasks Per Event	1 (non modificabile)	1 (non modificabile)
Max Traces in Ram*	9000000000	8000000000
Max Num Traces	9000000000	8000000000
Pause after Traces	9000000000	8000000000
Max Time Crawling	9000000000	8000000000
Pause after Time	9000000000	8000000000
Trace Max Depth	9000000000	8000000000
Trace Min Depth	9000000000	8000000000
restanti parametri	lasciati a default	lasciati a default

Osservazione: si sono considerati gli input dei casi di test "input che si avvicinano al limite"

e si è aggiunto, per ognuno di essi, una sola cifra in più.

I casi di test terminano con la generazione del file preferences.

- Previsioni: le stesse dei casi di test con input numerici non validi

- Risultati: gli stessi dei casi di test con input numerici non validi

Conclusioni e sviluppi futuri

Il test automatico risulta essere una soluzione, nel processo di verifica e validazione del software, di gran lunga preferibile al test manuale, con comprovati vantaggi rispetto a quest'ultimo. Rendere il test "automatico", però, non vuol dire solo automatizzare l'esecuzione dei test pratici, ma anche la loro generazione e valutazione, nonché automatizzare l'uso e il funzionamento degli strumenti che eseguono i test.

Nell'ambito del test delle applicazioni Android, la tecnica di test GUI Ripping si è mossa nella direzione giusta del test automatico: lo strumento GUI Ripper analizza automaticamente le GUI delle applicazioni Android e fornisce risultati che permettono la generazione automatica di casi di test (utilizzabili per altri tipi di test). Inoltre, peculiarità non di poco conto, il GUI Ripper implementa l'esplorazione in un modo altamente configurabile.

Il PreferencesGui risulta essere un tassello importante nell'automazione della tecnica di test GUI Ripping. Il GUI Ripping ha già riscontrato risultati molto positivi nel test delle applicazioni Android, ma l'assenza di uno strumento di configurazione automatica e guidata del GUI Ripper potrebbe impedire il raggiungimento di certi livelli di efficiente automazione del test. Il PreferencesGui supporta la tecnica di GUI Ripping, fornendo l'automazione nella configurazione e una migliore accessibilità/usabilità del GUI Ripper. In particolare le caratteristiche e le peculiarità dell'interfaccia utente del PreferencesGui permettono a chiunque (ricercatori, tester, etc.) di poter utilizzare e comprovare la tecnica di GUI Ripping.

Come qualunque software, anche il PreferencesGui necessita di opportuni test: in

particolare, dato il suo ruolo nella tecnica di GUI Ripping, esso dovrà essere sottoposto a ben più specifici e accurati piani di test, che ne studino le qualità e i limiti per un suo eventuale (e necessario) miglioramento.

Lo sviluppo modulare del PreferencesGui permette una facile manutenibilità ed evolvibilità dell'applicazione. Il PreferencesGui infatti lavora in funzione del GUI Ripper, che è tuttora un progetto in continua evoluzione e revisione: la futura introduzione in quest'ultimo di nuove caratteristiche e funzionalità dovranno essere adeguatamente supportate dal PreferenceGui.

Per concludere, gli scopi finali del progetto del PreferencesGui porteranno, a breve, ad includere l'applicazione in un progetto (tutt'ora in sviluppo) per la creazione di un nuovo wizard del GUI Ripper: il PreferencesGui dunque sarà presto integrato direttamente nel GUI Ripper.

Appendice

PreferencesGui integrato al GUI Ripper

Data la natura stand-alone del PreferencesGui, la configurazione del GUI Ripper non risulta ancora abbastanza automatizzata. In particolare:

- Il file preferences generato deve essere salvato nella cartella *data* del GUI Ripper
- Il PreferencesGui non supporta l'avvio automatico del GUI Ripper (e viceversa)
- Bisogna selezionare il file APK sia nel PreferencesGui che nel GUI Ripper

Per automatizzare e unire meglio le operazioni di configurazione ed esecuzione, si è pensato pertanto di integrare il PreferencesGui nel GUI Ripper: in particolare si è optato per la progettazione e lo sviluppo di un "nuovo wizard" del GUI Ripper tale che, oltre ad offrire nuove caratteristiche, possa anche contenere ed interagire col PreferencesGui. Questo richiede però alcuni "adattamenti" al progetto originario del PreferencesGui: nasce così il nuovo progetto del **PreferencesGui integrato**.

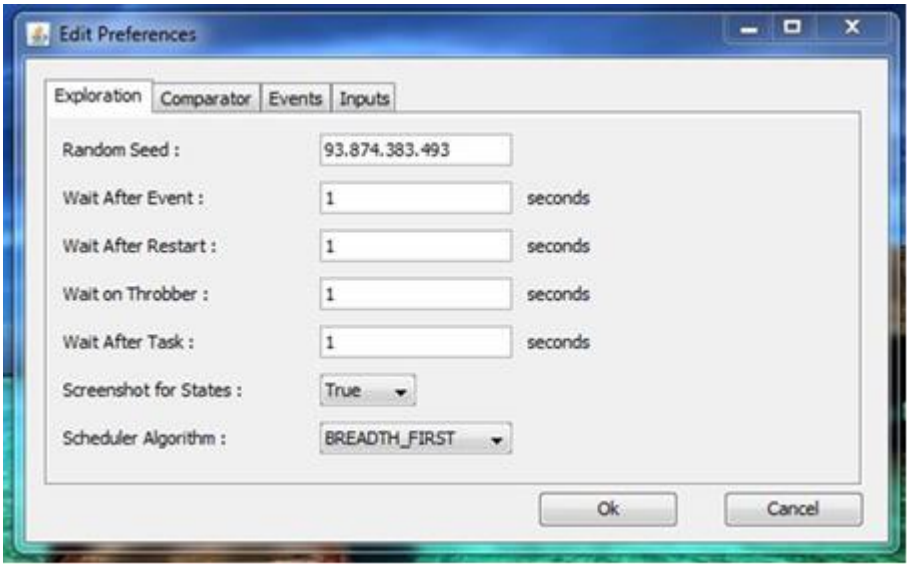
Gli adattamenti richiesti alla nuova applicazione possono riaussumersi nei seguenti:

- Cambiamento della natura dell'applicazione, da stand-alone a "integrato"
- Gestione di un numero minore di parametri (quelli più utilizzati)
- Capacità di gestire le informazioni passate dal wizard

Il risultato dei due progetti dovrà essere quello di presentare un wizar tale che possa permettere, durante la configurazione base del GUI Ripper, di richiamare ed attivare il PreferencesGui per il settaggio dei parametri: la combinazione wizard-PreferencesGui

permetterà così di avere una fase di configurazione del GUI Ripper che sia finalmente automatizzata completamente.

Dato lo stato sperimentale del progetto, esso non permette ancora una trattazione completa del PreferencesGui.



PreferencesGui integrato

Bibliografia

[1] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana
"Testing Android Mobile Applications: Challenges, Strategies and Approaches"

[2] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine
"Using GUI Ripping for Automated Testing of Android Applications**"**

[3] Domenico Amalfitano, Anna Rita Fasolino, Gennaro Imparato
"Sperimentazione di tecniche di testing automatico per applicazioni Android"

[4] Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, Gennaro Imparato
"A Toolset for GUI Testing of Android Applications"

[5] Atif Memon, Ishan Banerjee, and Adithya Nagarajan. 2003.
"GUI Ripping: Reverse Engineering of Graphical User Interfaces for Testing". In Proceedings of the 10th Working Conference on Reverse Engineering (WCRE '03). IEEE Computer Society, Washington, DC, USA, 260-269.
http://dx.doi.org/10.1109/WCRE.2003.1287256

[6] Apple App Store, http://www.apple.com/iphone/apps-for-iphone/ , last accessed on July 30th, 2011

[7] Guardian.co.uk. Apple's iOS App Store reaches 15bn downloads milestone http://www.guardian.co.uk/technology/appsblog/2011/jul/07/apple-iphone-app-store-downloads, last accessed on July 30th, 2011

[8] Gartner Newsroom. Gartner Says Android to Become No. 2 Worldwide Mobile Operating System in 2010 and Challenge Symbian for No. 1 Position by 2014. Available from http://www.gartner.com/it/page.jsp?id=1434613 Last accessed July 30th, 2011

[9] Gartner. 2011 Gartner Says Sales of Mobile Devices Grew5.6 Percent in Third Quarter of 2011; Smartphone Sales Increased 42 Percenthttp://www.gartner.com/it/page.jsp?id=1848514 last accessed on February 29th, 2012

[10] Eric Chu. 2011. 10 Billion Android Market Downloads and Counting, http://androiddevelopers.blogspot.com/2011/12/10-billionandroidmarketdownloadsand.html last accessed on February 29th, 2012

[11] Android. http://www.android.com/.

[12] HTC http://www.htc.com/www/about/newsroom/?id=66338&lang=1033

[13] "Android Developers. The Developer's Guide. Activities." http://developer.android.com/guide/topics/fundamentals/activities.html, last accessed on July 30th, 2011

[14] O'Rell, Android Community Experts,Ian Darwin and Contributors 2011 "Android Cookbook" Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472

[15] Robert V. Binder. “Testing object-oriented systems: models, patterns, and tools.” Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1999.

[16] Hass and Anne Mette Janassen. “Guide to Advanced Software Testing.” Artech House Publishers, 2003.

[17] IEEE Std. 610.12- 1990 , “Glossary of Software Engineering Terminology, in Software Engineering Standard Collection” 1990, IEEE CS Press, Los Alamitos, California.

[18] R.V. Binder, “Testing Object-Oriented Systems- Models, Patterns, and Tools, Addison” - Wesley, Boston, MA, USA, 1999

[19] I. Satoh. “Software testing for wireless mobile application.” IEEE Wireless Communications, pages 58–64, October 2004.

[20] Android Developers, The Developer’s Guide. Android SDK. http://developer.android.com/sdk/index.html, last accessed on July 30th, 2011

[21] Cuixiong Hu and Iulian Neamtiu. Automating gui testing for android applications. In: “Proc. of AST 2011, 6th international workshop on Automation of software test”, ACM Press, pages 77–83, 2011.

[22] Junit, “Resources for Test Driven Development”, http://www.junit.org, last accessed July 30th, 2011

[23] Robotium - It’s like Selenium, but for Android http://code.google.com/p/robotium/.

[24] Jayway. http://www.jayway.com/.

[25] Atif M. Memon.” An event-flow model of gui-based applications for testing.” Wiley InterScience, 2007.

[26] T. S. Chow.” Testing software design modeled by finite-state machines.” IEEE Transactions on Software Engineering, SE-4:178–187, May 1978.

[27] A. Memon, L. Banerjee, A. Nagarajan, “GUI ripping: reverse engineering of graphical user interfaces for testing”, Proceedings of the 10th Working Conference on Reverse Engineering (WCRE 2003), 2003, IEEE CS Press, pp.260 – 269

[28] Configurable Sensor & GUI Ripper: User Guide http://wpage.unina.it/ptramont/SensorGUIRipperConfigurable.htm

[29] Domenico Amalfitano, Porfirio Tramontana, Nicola Amatucci “Gui testing automatico di applicazioni Android tramite emulazione di input ed eventi provenienti da sensori”

[30] “Configurable GUI Ripper: a User Guide” http://wpage.unina.it/ptramont/GUIRipperConfigurable.htm