

Realizzazione di un framework per Android per l'esecuzione automatica di test con input offuscati

Relatore: *Prof.ssa Micucci Daniela*

Co-relatori: *Dott.ssa Briola Daniela*
Dott. Ginelli Davide

Relazione della prova finale di:

Stefano Androni

Matricola 845811

Indice

Introduzione	1
Introduzione allo stage	1
Obiettivo dello stage	4
Struttura della relazione	6
1 Background	7
1.1 Soluzioni candidate al riutilizzo	8
1.2 Libreria di offuscamento	8
1.2.1 Dettagli progettuali e implementativi	8
1.2.2 Tecniche di offuscamento	9
1.2.3 Valutazione della soluzione	11
1.3 LoadDataPool	12
1.3.1 Architettura della soluzione	12
1.3.2 Valutazione della soluzione	13
1.4 AndroidApp tool	13
1.4.1 Architettura della soluzione	14
1.4.2 Valutazione della soluzione	16
2 Panoramica del Framework sviluppato	17
2.1 Risoluzione delle limitazioni e criticità	17
2.2 Implementazione di nuove funzionalità	18
2.3 Miglioramento complessivo del sistema	19
3 Architettura del Framework sviluppato	20
3.1 Introduzione	20
3.1.1 Requisiti funzionali informali	20
3.1.2 Architettura della soluzione	21
3.2 Funzionalità	22

3.2.1	Offuscamento dei dati	23
3.2.2	Automatizzazione dei test	28
3.2.3	Offuscamento dei dati e automatizzazione dei test	35
4	Dettagli implementativi del Framework sviluppato	39
4.1	Architettura software del tool	39
4.2	Configurazione	41
4.2.1	Struttura del package di lancio	41
4.2.2	File di configurazione e file di input	43
4.3	Lancio del tool da linea di comando	49
4.3.1	Prerequisiti	49
4.3.2	Lancio del tool	49
4.4	Gestione dell'emulatore per l'automazione dei test	51
4.5	Passaggio dei parametri alla classe di test	52
4.6	Gestione del salvataggio degli screen record	52
4.7	Gestione del salvataggio degli screenshot	53
5	Validazione del Framework	56
5.1	Introduzione	56
5.2	Applicazioni	57
5.2.1	ActivityDiary	57
5.2.2	PrivacyFriendlyShoppingList	58
5.2.3	OpenScale	59
6	Conclusione e sviluppi futuri	61
	Appendice A	62
	Operazioni utili	62
	Variabili d'ambiente	62
	Migrazione del progetto ad AndroidX	62
	Registrazione del caso di test con Espresso Test Recorder	62
	Generazione degli APK in Android Studio	63
	Configurazione degli AVD	63
	Lettura degli alberi xml	64
	Generazione degli APK	65
	Strumenti utilizzati	70
	Intellij IDEA	70

Android Studio	70
adb	70
Emulator	70
AAPT2	70
AVD Manager	71
apksigner	71
Espresso	71
Gradle	71
AndroidTestWithOutSource Project	71
UI Automator Viewer	72
Sitografia	73

Introduzione

Lo stage è stato svolto presso il Dipartimento di Informatica, Sistemistica e Comunicazione (DISCO) dell'Università degli Studi di Milano-Bicocca nel periodo compreso tra il 30/06/2021 e il 28/09/2021 in modalità smart working (causa emergenza Covid-19).

Introduzione allo stage

Contesto

Protezione dei dati e della privacy online

Nel processo di trasformazione digitale, fortemente accelerato dalla pandemia da Covid-19, il tema del trattamento dei dati personali ha assunto una rilevanza sempre maggiore finendo al centro di molti dibattiti.

L'Unione Europea, in tale ambito, ha emanato nel 2016 il GDPR¹ con l'obiettivo di uniformare le leggi europee sul trattamento dati e garantire ai cittadini il diritto a essere in pieno controllo delle informazioni che li riguardano[1]. Il regolamento considera il formato dei dati irrilevante ed obbliga le aziende (e le organizzazioni) a chiedere il consenso al soggetto prima di potere raccogliere i suoi dati personali[2]. Le conseguenze pecuniarie per chi viola la norma possono arrivare fino a 20 milioni di euro o, per le imprese, fino al 4 % del fatturato mondiale totale annuo[3].

Il resto del mondo sta adottando legislazioni che vanno nella stessa direzione obbligando le aziende a dover rivoluzionare il modo con cui gestiscono i dati sensibili e rendendo le persone più consapevoli sull'argomento.

¹**General Data Protection Regulation:** il regolamento di riferimento dell'Unione europea in materia di trattamento dei dati personali e di privacy

Problema

Bug report e privacy

Le risorse temporali ed economiche destinate alla fase di testing di un applicativo prima del suo rilascio, in molti casi, non sono adeguate all'esecuzione di un collaudo che possa considerarsi esaustivo, con effetti limitanti nell'identificazione dei difetti del software.

Generalmente, disporre delle risorse necessarie all'esecuzione di un testing completo non potrebbe comunque garantire il rilascio di applicativi privi di bug, con particolare riferimento ai difetti che dipendono dall'ambiente di esecuzione. Il limite, in quest'ultimo caso, è dovuto alla difficoltà/impossibilità di prevedere a priori (e quindi simulare) tutti gli ambienti in cui il software sarà eseguito dagli utenti finali, in particolar modo quando l'applicativo è destinato a diversi sistemi.

Queste tipo di problematiche portano spesso al rilascio di software non perfetti, che presentano ancora molti difetti. Come conseguenza, molti dei bug vengono identificati e risolti quando il software è già stato rilasciato.

Il contributo degli utenti finali con la compilazione e l'invio di bug report velocizza ed economizza l'identificazione e la risoluzione dei difetti. Infatti la reportizzazione permette allo sviluppatore di riprodurre il bug in-house ricreando l'ambiente in cui il difetto si è verificato e riproducendo le operazioni eseguite dall'utente.

Tuttavia **le informazioni di esecuzione contenute nel report potrebbero includere dei dati sensibili**, che l'utente potrebbe non voler divulgare a terzi e decidere quindi di non inviare o inviare in maniera incompleta/inconsistente. Questa situazione è molto limitante per il processo di individuazione dei bug con conseguenze negative sia per gli sviluppatori, che si vedrebbero obbligati ad investire molte più risorse, che per gli utenti, che potenzialmente si ritroverebbero ad utilizzare applicativi più difettati.

Possibile soluzione

Offuscamento dei dati

Per i motivi normativi sopra citati, l'interesse in tale ambito è ampio e molti degli studi identificano nell'offuscamento dei dati una possibile soluzione.

L'offuscamento dei dati è una delle metodologie di data masking che, attraverso l'applicazione di diverse tecniche, persegue l'obiettivo di mascheramento dei dati con la qualità di preservarne la consistenza e la coerenza[4].

Nella risoluzione del problema considerato, l'applicazione delle tecniche di offuscamento sui dati sensibili potrebbe apportare un duplice contributo:

- gli utenti dovrebbero essere più disponibili all'invio di report che non contengono i loro dati sensibili (sono stati offuscati).
- lo sviluppatore riceverebbe più dati completi ed utilizzabili, anche se in parte offuscati.

In sostanza, l'offuscamento dei dati non fa cadere l'obbligo di richiesta del consenso per il trattamento dei dati imposto dal GDPR, ma cerca di convincere gli utenti a compilare ed inviare dei bug report completi garantendo che i loro dati sensibili non vengano divulgati.

I dati offuscati possono sembrare una soluzione definitiva al problema, ma hanno in realtà un limite: non possono garantire con sicurezza che l'applicativo si comporti nello stesso modo in cui si sarebbe comportato ricevendo in input i dati originali e quindi che sia indifferente fornire allo sviluppatore i dati non offuscati o i dati offuscati in ambito di riproducibilità dei bug.

Obiettivo dello stage

Lo stage si inserisce in un progetto più ampio, a cui hanno contribuito negli anni passati altri colleghi, con obiettivo complessivo la valutazione dell'efficienza delle tecniche di offuscamento in ambito di riproducibilità di bug. Con "efficienza" si intende la capacità delle varie tecniche di generare dati offuscati che riproducano il bug originale, cioè il difetto sollevato dall'inserimento dei dati non offuscati. La scelta degli applicativi su cui effettuare lo studio è ricaduta sulle applicazioni destinate al sistema operativo per dispositivi mobile Android, in quanto ampiamente diffuse e statisticamente più soggette alla presenza di bug[16]. I bug interessanti per lo studio sono evidentemente quelli che implicano l'inserimento di dati da parte dell'utente.

L'obiettivo dello stage, in termini generali, può essere identificato nella **realizzazione di un framework per l'esecuzione automatica di test con input offuscati** e della conseguente implementazione di un **tool eseguibile da linea di comando** che permetta di sfruttarne le capacità. Il perseguimento dell'obiettivo include la prosecuzione degli studi e del lavoro svolto in precedenza da altri colleghi e del conseguente refactoring delle soluzioni già adottate. Prima di ogni altra cosa quindi, è essenziale comprendere le scelte progettuali/implementative effettuate in precedenza ed individuare limitazioni e criticità la cui risoluzione rappresenterà uno dei fini fondamentali dello stage. In termini generali, lo stage implica il raggiungimento di **sotto-obiettivi** che possono essere identificati nelle seguenti macrocategorie:

Ob.1 Offuscamento dei dati

Realizzazione di un componente in grado di applicare le tecniche di offuscamento e restituire le tuple offuscate.

Nota - Tool già realizzato da un collega e in gran parte riutilizzato nel progetto

Ob.2 Creazione del caso di test

Ricerca della migliore soluzione per la registrazione e parametrizzazione del caso di test che genera il bug.

Nota - Soluzione già trovata da un collega, ma non riutilizzata nel progetto per motivi che saranno esposti nei capitoli successivi

Ob.3 Automazione dell'esecuzione di test parametrici in ambiente Android

Realizzazione di un componente in grado di eseguire automaticamente instrumented tests parametrici e capace di catturarne il risultato. (nel progetto utile a lanciare automaticamente un test per ogni tupla offuscata)

Nota - Tool già realizzato da un collega, ma non riutilizzato nel progetto per motivi che saranno esposti nei capitoli successivi

Ob.4 Verifica della riproduzione dei bug

Realizzazione di uno strumento in grado di comprendere se il test ha prodotto il bug atteso.

Ob.5 Validazione del tool

Breve validazione del tool utilizzando alcune delle applicazioni difettate individuate negli studi precedenti.

Un obiettivo secondario è identificato nella produzione di una buona documentazione e reportizzazione dei progressi, oltre che alla segnalazione di possibili problematiche e punti aperti, in modo da permettere un più fluido avanzamento del progetto in caso di futuro proseguimento degli studi.

Struttura della relazione

La relazione si struttura nei seguenti capitoli:

Capitolo 1 - Background

Nel Capitolo 1 verranno esposti ed analizzati gli ultimi studi, effettuati all'interno del gruppo di ricerca in cui ho lavorato, in ambito di offuscamento dei dati e automazione dei test.

Capitolo 2 - Panoramica del Framework sviluppato

Nel Capitolo 2 verrà presentato complessivamente il lavoro svolto durante lo stage.

Capitolo 3 - Architettura del Framework sviluppato

Nel Capitolo 3 verrà analizzato il framework sviluppato, esponendo ed esplorando le funzionalità offerte dal tool eseguibile da linea di comando che ne sfrutta le capacità.

Capitolo 4 - Dettagli implementativi del Framework sviluppato

Nel Capitolo 4 verranno analizzati i dettagli implementativi del framework ritenuti più interessanti.

Capitolo 5 - Validazione del Framework

Nel Capitolo 5 verrà discusso il breve processo di validazione del Framework.

Capitolo 6 - Conclusione e sviluppi futuri

Nel Capitolo 6 verranno ricapitolati i risultati ottenuti e verranno identificati possibili sviluppi futuri.

Capitolo 1

Background

In questo primo capitolo verranno esposti ed analizzati, nelle parti che interessano lo stage, gli ultimi studi effettuati all'interno del gruppo di ricerca con cui ho lavorato in ambito di offuscamento dei dati e automazione dei test in ambiente Android. In particolare modo verranno presentati ad alto livello ed esaminati brevemente gli strumenti di potenziale interesse già realizzati in questi studi con il fine di valutarne un possibile riutilizzo. L'analisi includerà inoltre l'identificazione di possibili limitazioni e criticità delle soluzioni adottate.

Studi di riferimento

- **Belisario A.** (2016/2017) Tesi magistrale
Analisi empirica dell'influenza delle tecniche di offuscamento dei dati sulla riproduzione dei fallimenti del software.
- **Triolo D.** (2016/2017) Relazione triennale
Test app Android con dati offuscati: setup ambiente di test e valutazione empirica dei fallimenti del software.
- **Mendieta A.** (2016/2017) Relazione triennale
Realizzazione di uno strumento per l'esecuzione automatica di test parametrici in Android.
- **Sormani D.** (2019/2020) Relazione triennale
Analisi empirica dell'impatto di tecniche di offuscamento di dati sulla riproducibilità di bug.

1.1 Soluzioni candidate al riutilizzo

Per il raggiungimento di alcuni sotto-obiettivi dello stage, è stato valutato un possibile riutilizzo delle seguenti soluzioni realizzate negli studi precedenti:

Ob.1 Offuscamento dei dati

Obiettivo Realizzazione di un componente in grado di applicare le tecniche di offuscamento e restituire le tuple offuscate.

Soluzione Libreria di offuscamento + LoadDataPool

Ob.3 Automazione dell'esecuzione di test parametrici in ambiente Android

Obiettivo Realizzazione di un componente in grado di eseguire automaticamente instrumented tests parametrici e capace di catturarne il risultato.

Soluzione AndroidApp tool

Tutte le soluzioni elencate (Libreria di offuscamento, LoadDataPool, AndroidAppTool) vengono analizzate e valutate singolarmente nei capitoli successivi (Sezione 1.2, Sezione 1.3, Sezione 1.4) .

1.2 Libreria di offuscamento

La libreria di di offuscamento considerata è stata sviluppata nella tesi magistrale “Analisi empirica dell’influenza delle tecniche di offuscamento dei dati sulla riproduzione dei fallimenti del software” del collega Andrea Belisario. La libreria **offre la possibilità di ottenere, dato un input appartenente ad un certo dominio, un output modificato secondo una particolare tecnica di offuscamento.**

1.2.1 Dettagli progettuali e implementativi

Nel contesto in cui è stata sviluppata la libreria, è essenziale che il dominio dei dati offuscati (output) sia coerente con il dominio dei dati non offuscati (input). Si può facilmente intuire, per esempio, l'impossibilità di eseguire test che ammette solo numeri avendo a disposizione solo stringhe. Per questo motivo la libreria è stata progettata in modo tale che ogni tecnica di offuscamento offra due metodi:

■ **offuscaDato(dato) = datoOffuscato**

Il metodo va ad offuscare il dato in input "dato" e restituisce un valore che potrebbe non essere coerente con il suo dominio, ma che rispetta la definizione della tecnica di offuscamento di turno.

■ **rendiDominioCoerente(datoOffuscato)**

Il metodo prende in input il risultato del metodo precedente e restituisce un valore coerente con il dominio dell'attributo offuscato, che sia quindi utilizzabile per i test.

La coerenza dei domini richiesta dalla libreria, ottenuta tramite l'applicazione dei metodi sopra illustrati, implica l'utilizzo di una logica che gestisca le varie **tipologie di dato**. La logica è stata implementata con la realizzazione di una classe astratta dalla quale ereditano tutte le tipologie di dato previste dalla libreria:

■ **Categorici** (Categorical)

I valori categorici possono assumere solamente un numero finito di valori. In ambito informatico l'equivalente dei valori categorici è rappresentato dai tipi enumerativi.

■ **Continui** (Continuos)

I valori continui possono assumere un numero non finito di valori all'interno di un range.

■ **Stringhe** (StringType)

I valori che non possono essere definiti come 'Categorici' o 'Continui', nel dominio informatico, possono essere rappresentati mediante stringhe.

1.2.2 Tecniche di offuscamento

La libreria realizzata racchiude le tecniche di offuscamento più utilizzate e rivisitate dal collega per essere adattate al meglio alle esigenze dello studio. Le tecniche incluse possono essere *perturbative*, quando il dato viene sostituito e alcuni dettagli vengono rimossi, oppure *non perturbative*, quando il dato originale non viene alterato, ma vengono sopprese alcune informazioni.

Tecniche non perturbative

Local Suppression

Definizione

Data una tupla T, la nuova tupla T' contiene tutti gli attributi contenuti in T con i valori di alcuni attributi soppressi.

Rivisitazione

Ai fini della libreria, non ha utilità inserire valore NULL (soppressione) in quanto si otterrebbero sempre le stesse risposte in qualunque esecuzione del test. La soluzione adottata prevede la sostituzione dei valori da offuscare con valori generati randomicamente nel loro dominio.

Global Recoding

Definizione

Può essere utilizzata solo sui tipi di valori continui. Il dominio di un attributo viene suddiviso in N intervalli disgiunti non necessariamente della stessa dimensione, ad ognuno dei quali viene assegnata un'etichetta. Data una tupla T , i valori degli attributi contenuti in T saranno sostituiti dalle etichette degli intervalli a cui appartengono.

Rivisitazione

Ai fini della libreria, non ha senso sostituire il valore di un dato con l'etichetta dell'intervallo a cui appartiene (inoltre la coerenza del dominio non sarebbe rispettata). In questo caso viene generato un valore random all'interno dell'intervallo in cui si trova il dato originale.

Top/Bottom Coding

Definizione

Può essere utilizzata solo sui tipi di valori continui. A partire dai valori presenti in una tabella T viene definito un limite top o bottom, che indicano rispettivamente il limite superiore e il limite inferiore nel dominio di un valore continuo. I campioni per i quali l'attributo da offuscare supera, o è inferiore, al valore di soglia, vengono sostituiti con l'etichetta '> TL', se si tratta di un limite superiore o '< BL', se si tratta di un limite inferiore.

Rivisitazione

Ai fini della libreria, non ha senso sostituire il valore di un dato con un'etichetta (inoltre la coerenza del dominio non sarebbe rispettata). In questo caso viene generato un valore random inferiore o maggiore alla soglia.

Generalizzazione

Definizione

Può essere applicata sia a valori continui che categorici. La tecnica prevede di sostituire il valore di un attributo con un suo rappresentante gerarchico superiore. Le gerarchie possono essere rappresentate mediante gli alberi: nella radice dell'albero si trova il valore più generico mentre le foglie rappresentano i valori più specifici. Il valore offuscato sarà un antenato del valore stesso.

Rivisitazione

Nessuna rivisitazione.

Tecniche perturbative

PRAM

Definizione

La tecnica PRAM prevede di sostituire il valore categorico della tupla con un altro valore categorico adottando la tecnica della probabilità. Il risultato è l'introduzione di volontari errori nella classificazione dei dati raccolti, in modo che non sia possibile risalire all'identità del proprietario. Per l'utilizzo di questa tecnica viene creata la matrice Markoviana, una matrice di transizione contenente le probabilità di passare da uno stato ad un altro, nel nostro caso di osservare un valore categorico in corrispondenza del valore da offuscare.

Rivisitazione

Nessuna rivisitazione.

Rounding

Definizione

La tecnica rounding opera in maniera analoga rispetto alla tecnica Global Recoding separando il dominio del valore in N intervalli e calcolando per ognuno di questi il rounding point, che solitamente è il valore medio di ogni intervallo. Il valore viene quindi offuscato sostituendolo con il rounding point dell'intervallo a cui il valore appartiene.

Rivisitazione

Ai fini della libreria, ha senso utilizzare sempre come rounding point il valore medio dell'intervallo.

1.2.3 Valutazione della soluzione

Per le finalità dello stage la libreria non ha bisogno di modifiche e risulta utile riutilizzarla nella sua interezza.

1.3 LoadDataPool

LoadDataPoll è il componente principale dell'architettura dello strumento realizzato da A. Belisario nella tesi magistrale "Analisi empirica dell'influenza delle tecniche di offuscamento dei dati sulla riproduzione dei fallimenti del software". Il tool è in grado di riprodurre automaticamente test per applicazioni desktop utilizzando come parametri dati precedentemente offuscati con le tecniche della libreria trattata nella sezione precedente (Sezione 1.2).

1.3.1 Architettura della soluzione

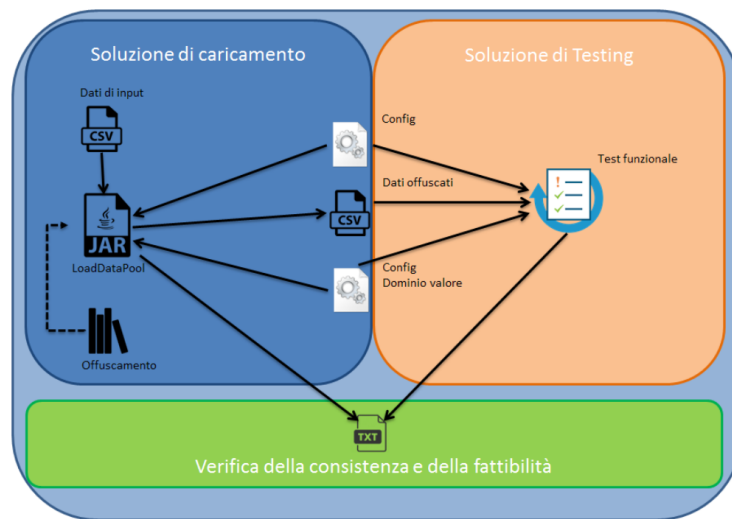


Figura 1.1: Architettura della soluzione ¹

Come illustrato nella Figura 1.1, l'architettura della soluzione si compone di 3 parti:

- **Modulo di caricamento**

Il modulo di caricamento si occupa di:

- leggere le tuple in input da un file csv;
- leggere le informazioni necessarie all'offuscamento da dei file xml di configurazione ;
- offuscare i dati in input in base alle informazioni lette (come tecnica di offuscamento e dominio dei dati).

Il modulo produce in output un file csv contenente le tuple offuscate. Il file prodotto sarà uno dei file in input per il modulo di testing.

¹Immagine realizzata da A. Belisario

- **Modulo di testing**

Il modulo di testing si preoccupa di lanciare in maniera automatica un test per ogni tupla del file ricevuto dal modulo di caricamento e di catturarne il risultato.

- **Modulo di verifica**

Il modulo di verifica valuta l'efficacia dell'offuscamento (se è o non è in grado di riprodurre il bug) e si occupa della creazione di un file di log in output.

1.3.2 Valutazione della soluzione

Per le finalità del sotto-obiettivo considerato risulta utile riutilizzare solo il modulo di caricamento (che si occupa di orchestrare l'offuscamento), in quanto comunque la parte di automazione dei test è stata realizzata ad hoc per applicativi desktop e non potrebbe essere in nessun modo riutilizzata in ambiente Android. Il riutilizzo di parte di questo componente implica il dover adottare la stessa architettura prevista da A. Belisario, che prevede in input due file di configurazione distinti: uno per il dominio e uno per le tecniche di offuscamento.

1.4 AndroidApp tool

Un altro dei sotto-obiettivi che si vogliono raggiungere include la realizzazione di un componente in grado di eseguire automaticamente instrumented tests parametrici su applicazioni Android e capace di catturarne il risultato. A questo sotto-obiettivo è in realtà fortemente legata anche la ricerca di una soluzione per la registrazione e parametrizzazione del caso di test che genera il bug (*Ob.2*). Nella relazione “Realizzazione di uno strumento per l'esecuzione automatica di test parametrici in Android” del collega Anthony Mendieta, viene esposto e realizzato uno strumento, chiamato AndroidApp tool, che risponde quasi in toto ai requisiti del sotto-obiettivo.

1.4.1 Architettura della soluzione

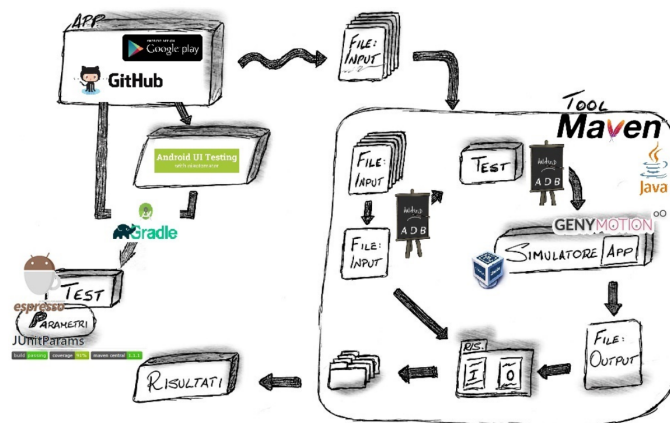


Figura 1.2: Architettura della soluzione ²

L'**architettura della soluzione** realizzata dal collega, illustrata in Figura 1.2, prevede in input:

- **file di input**

Il file di input è un file txt che contiene i parametri con cui eseguire i test, separati da una virgola. Il documento può essere formato da più righe, ognuna delle quali corrisponde ai parametri di esecuzione di un singolo test.

- **APK dell'applicazione**

L' APK dell'applicazione che si vuole testare.

- **APK di test**

L'APK di test dell'applicazione che si vuole testare contenente la classe di test parametrica.

- **file di configurazione**

File di configurazione utile al funzionamento del tool.

Il **flusso di normale funzionamento** del tool può essere sintetizzato come segue:

1. Lettura del file di configurazione
2. Installazione dell'APK dell'applicazione e dell'APK di test sull'emulatore (se prevista dal file di configurazione)
3. Lettura del file di input
4. Per ogni riga letta dal file di input
 - 1 Creazione di un file contenente la singola riga letta

²Immagine realizzata da A. Mendieta

- 2 Push del file creato nel punto precedente sull'emulatore (nel percorso interno specificato nel file di configurazione)
- 3 Lancio della classe di test parametrica (utilizza come parametri i valori contenuti nel file caricato sull'emulatore nel punto precedente)
- 4 Cattura del risultato del test in un file di testo (in caso di errore salva il log in un file di testo a parte)

File di configurazione

La soluzione proposta prevede la presenza di un file di configurazione in cui devono essere specificati i seguenti parametri:

PathAPK

Indirizzo interno del pacchetto in cui si trova l'APK dell'applicazione

PathAPKTest

Indirizzo interno del pacchetto in cui si trova l'APK di test dell'applicazione

PathOutput

Indirizzo interno del pacchetto in cui creare i file prodotti dal tool

PathInput

Indirizzo interno del pacchetto in cui si trovano i file di input

PathIntoAndroid

Indirizzo interno del device in cui si trova la cartella riservata all'applicazione da testare, nella quale verranno copiati i file di input da testare

dirClassTest

Nome del pacchetto all'interno del quale si trova la classe di test

dirRunWith

Indirizzo completo della strumentazione che si occuperà del lancio della classe di test

installAPK

Valore booleano che indica se si deve procedere con l'installazione o la reinstallazione dell'APK sul device

installAPKTest

Valore booleano che indica se si deve procedere con l'installazione o la reinstallazione dell'APK di test sul device

eseguireTest

Valore booleano che indica se si deve procedere con il lancio dei test parametrici

Creazione della classe di test

Nella soluzione vengono proposte due modalità per la creazione della classe di test:

- **Scrittura a mano del caso di test**

Modalità obbligatoria quando non si ha a disposizione il codice sorgente. In questo caso è fondamentale l'utilizzo di UI Automator Viewer (Appendice A - UI Automator Viewer).

- **Registrazione del caso di test**

Modalità utilizzabile solo quando si è in possesso del codice sorgente. Per la registrazione del caso di test viene utilizzato Espresso Test Recorder (Appendice A - Registrazione del caso di test con Espresso Test Recorder).

Parametrizzazione del caso di test

Per la parametrizzazione del caso di test viene utilizzata la libreria JUnitParams, che è in grado di leggere automaticamente i parametri da un file nel device. La libreria in realtà potrebbe lanciare automaticamente un test per ogni riga del file letto, ma il sollevamento di un'eccezione interromperebbe il ciclo di esecuzione. Il problema è stato risolto dal collega Mendieta dividendo il file di input in più file, ognuno dei quali contenente una sola riga del file originale, in modo da poterli caricare sul device singolarmente, come analizzato nel 'flusso di normale funzionamento'.

1.4.2 Valutazione della soluzione

La relazione è stata presa di ispirazione nella progettazione del nuovo componente e alcune soluzioni trovate, come la registrazione della classe di test, sono state riutilizzate. Nonostante questo è stato deciso di realizzare il componente da zero sia per l'impossibilità di reperirne il codice sorgente, sia per la presenza di alcune limitazioni e criticità tra le quali:

- L1** - utilizzo di librerie ormai obsolete, tra le quali JUnitParams, utilizzata per la parametrizzazione dei test e non aggiornata da più di 3 anni.
- L2** - processo di lancio dei test parametrici macchinoso (implica il push di un file sull'emulatore contenente i parametri del test prima di ogni lancio)
- L3** - previsto l'inserimento di informazioni ridondanti nel file di configurazione (tra le quali, per esempio, dirClassTest e dirRunWith, entrambe ricavabili dall'APK di test con apposito strumento)
- L4** - in caso di fallimento del test non è in grado di catturarne automaticamente la causa (tipo di eccezione o crash), ma salva semplicemente il log

Capitolo 2

Panoramica del Framework sviluppato

In questo capitolo verrà presentato complessivamente il lavoro svolto durante lo stage, che può essere complessivamente sintetizzato in:

- Risoluzione delle limitazioni e criticità
- Implementazione di nuove funzionalità
- Miglioramento complessivo del sistema

2.1 Risoluzione delle limitazioni e criticità

Nella progettazione del tool sono state risolte le limitazioni e criticità identificate nel Capitolo 1. Per ognuna di queste, segue una breve esposizione della soluzione adottata:

L1 Limitazione Utilizzo di librerie ormai obsolete, tra le quali JUnitParams, utilizzata per la parametrizzazione dei test e non aggiornata da più di 3 anni.

Soluzione La nuova soluzione prevede la parametrizzazione delle classi di test senza l'utilizzo di librerie esterne: i parametri vengono ottenuti come argomenti del bundle della strumentazione di test (Appendice A - Generazione degli APK - 2.Parametrizzazione della classe di test).

L2 Limitazione Processo di lancio dei test parametrici macchinoso (implica il push di un file sull'emulatore contenente i parametri del test prima di ogni lancio).

Soluzione La nuova soluzione prevede il passaggio dei parametri alla strumentazione di test, specificandoli come argomento custom del comando adb (Appendice A - adb)

che lancia la classe di test: in questo modo può essere evitato il meccanismo di push dei file sul device. (Sezione 4.5 - Passaggio dei parametri alla classe di test)

L3 Limitazione Previsto l'inserimento di informazioni ridondanti nel file di configurazione (tra le quali, per esempio, `dirClassTest` e `dirRunWith`).

Soluzione La nuova soluzione prevede l'utilizzo di strumenti appositi in grado di ottenere le informazioni richieste direttamente dal file APK. (Appendice A - AAPT2)

L4 Limitazione In caso di fallimento del test non è in grado di catturarne automaticamente la causa (tipo di eccezione o crash), ma salva semplicemente il log .

Soluzione La nuova soluzione è in grado di identificare automaticamente il tipo di fallimento del test, crash o eccezione, e, in quest'ultimo caso, è in grado di catturarne la tipologia. (Sezione 3.2.2 - Automatizzazione dei test)

2.2 Implementazione di nuove funzionalità

Verifica della riproduzione del bug

La progettazione del tool è stata orientata all'aggiunta di una nuova funzionalità che permettesse la verifica della riproduzione del bug atteso. L'obiettivo funzionale è stato raggiunto attraverso diversi step :

- Impostazione di una logica che definisce se il bug è stato riprodotto in base a:
 - risultato del test
 - bug atteso
- Reimpostazione del file di configurazione con l'aggiunta di nuove informazioni relative al bug atteso
- Realizzazione di un componente in grado di gestire il processo di verifica

Gestione degli emulatori

Lo strumento realizzato permette di poter associare ad ogni bug uno specifico emulatore su cui lanciare i test. L'implementazione di questa funzionalità è stata di fondamentale importanza in quanto:

- Alcune applicazioni potrebbero funzionare solo con determinate versioni dell'Sdk
- Il verificarsi di alcuni bug potrebbe dipendere dalla versione dell'Sdk

Funzionalità secondarie

Sono state implementate inoltre alcune funzionalità secondarie, tra le quali:

- Possibilità ottenere uno screen record per ogni esecuzione di un caso di test
- Possibilità di ottenere uno screenshot per ogni esecuzione di un caso di test

2.3 Miglioramento complessivo del sistema

Command Line Tool

Il tool è stato progettato per essere lanciato da linea di comando, con la possibilità di utilizzare diverse funzioni con argomenti differenti grazie all'implementazione di un parser ad hoc. Inoltre è stato previsto un helper in grado di aiutare l'utente nell'utilizzo dello strumento.

Nuova struttura del package di lancio

La soluzione adottata implica l'utilizzo di una struttura fissa del package di lancio, che permette di:

- Mantenere in memoria, in modo organizzato, i file di esecuzione in modo da poterli rieseguire facilmente in qualsiasi momento
- Mantenere in memoria, in modo organizzato, i file di output di ogni esecuzione del tool
- Evitare di dover specificare ad ogni esecuzione il percorso in cui sono posizionati i vari file di input

Gestione dell'intero processo

La soluzione adottata permette la gestione dell'intero processo di offuscamento dei dati, automazione dei test e verifica della riproduzione del bug atteso. Le funzionalità dello strumento, una volta configurato correttamente, possono essere utilizzate in maniera fluida e rapida senza dover preoccuparsi ad ogni esecuzione della configurazione dell'ambiente in cui viene lanciato. Infatti gestisce automaticamente anche il processo di avvio e chiusura dell'emulatore, effettuando tutti i controlli del caso.

Capitolo 3

Architettura del Framework sviluppato

In questo terzo capitolo verrà analizzato il framework esponendo ed esplorando le funzionalità offerte dal tool eseguibile da linea di comando che ne sfrutta le capacità.

3.1 Introduzione

In questa sezione vengono analizzate brevemente ed in maniera informale le funzionalità offerte e l'architettura della soluzione.

3.1.1 Requisiti funzionali informali

Lo strumento offre le seguenti funzionalità:

Fu.1 **Offuscamento dei dati**

Il tool offre la possibilità di applicare le tecniche di offuscamento e restituire le tuple offuscate.

Fu.2 **Automazione dei test**

Il tool offre la possibilità di eseguire automaticamente test parametrici su applicazioni Android. Lo strumento è inoltre in grado di catturare il risultato dei test e, in base a questo, verificare se il bug atteso è stato riprodotto. Sono inoltre offerte le seguenti funzionalità secondarie:

- Salvataggio di uno screenshot per ogni test lanciato
- Salvataggio di uno screen record per ogni test lanciato

Fu.3 Offuscamento dei dati + Automazione dei test

Il tool offre la possibilità di utilizzare in sequenza le funzionalità precedenti permettendo il raggiungimento dell'obiettivo dello stage. E' possibile infatti eseguire automaticamente test parametrici utilizzando in input le tuple offuscate ottenute applicando le tecniche di offuscamento sui dati originali che provocano il bug. In questo modo lo strumento è in grado di verificare, per ogni tupla offuscata, se il bug atteso (provocato dalla tupla non offuscata) è stato riprodotto.

Risulta opportuno evidenziare che il tool è orientato a quest'ultima funzionalità in quanto è quella che permette il raggiungimento dell'obiettivo per cui è stato creato.

3.1.2 Architettura della soluzione

La soluzione adottata, in grado di offrire le funzionalità identificate nella sezione precedente (Sezione 3.1.1), segue il modello esposto nella Figura 3.1.

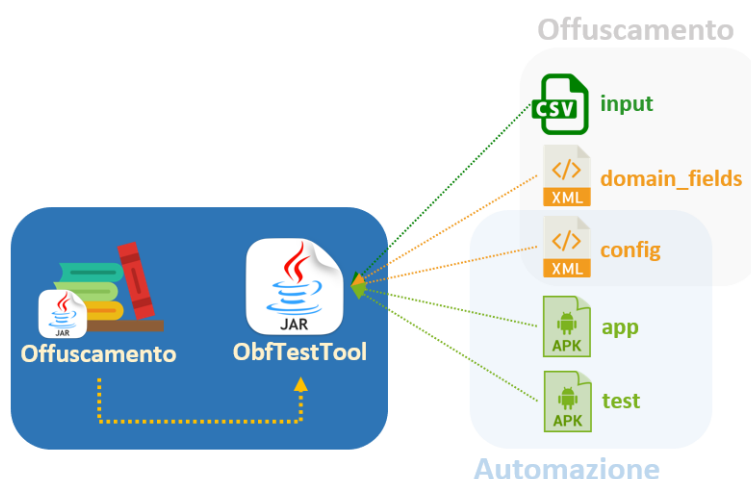


Figura 3.1: Architettura della soluzione

Il tool è stato impacchettato in un file JAR (*ObfTestTool*) lanciabile da linea di comando, che utilizza la libreria di offuscamento (*Offuscamento*) citata nella Sezione 1.2. Come evidente nella Figura 3.1, lo strumento necessita di alcuni file di configurazione (*domain_fields.xml*, *config.xml*) e di alcuni file di input (*input.csv*, *app.apk*, *test.apk*) in maniera differente in base alla funzionalità che si vuole sfruttare.

3.2 Funzionalità

Vengono ora presentate e spiegate nei minimi dettagli le funzionalità offerte dal Framework:

- **Offuscamento dei dati**
- **Automatizzazione dei test**
- **Offuscamento dei dati e automatizzazione dei test**

Per ogni funzionalità verrà presentata l'architettura della soluzione ponendo particolare attenzione a:

- ciò che permette di fare
- parametri di esecuzione (obbligatori e facoltativi)
- informazioni necessarie al funzionamento (ottenute dai file di configurazione e dai file di input)
- file prodotti in output

Nel capitolo non verranno quindi analizzati i dettagli implementativi, che saranno oggetto invece dell'analisi nel capitolo successivo (Capitolo 4).

3.2.1 Offuscamento dei dati

La funzionalità 'Offuscamento dei dati' si occupa dell'offuscamento dei dati rispondendo al requisito funzionale *Fu.1*. Specificatamente la funzionalità permette di offuscare una o più colonne del file in input (ognuna delle quali appartenente ad un certo dominio) applicando una determinata tecnica di offuscamento per un numero specifico di iterazioni.

Architettura della soluzione

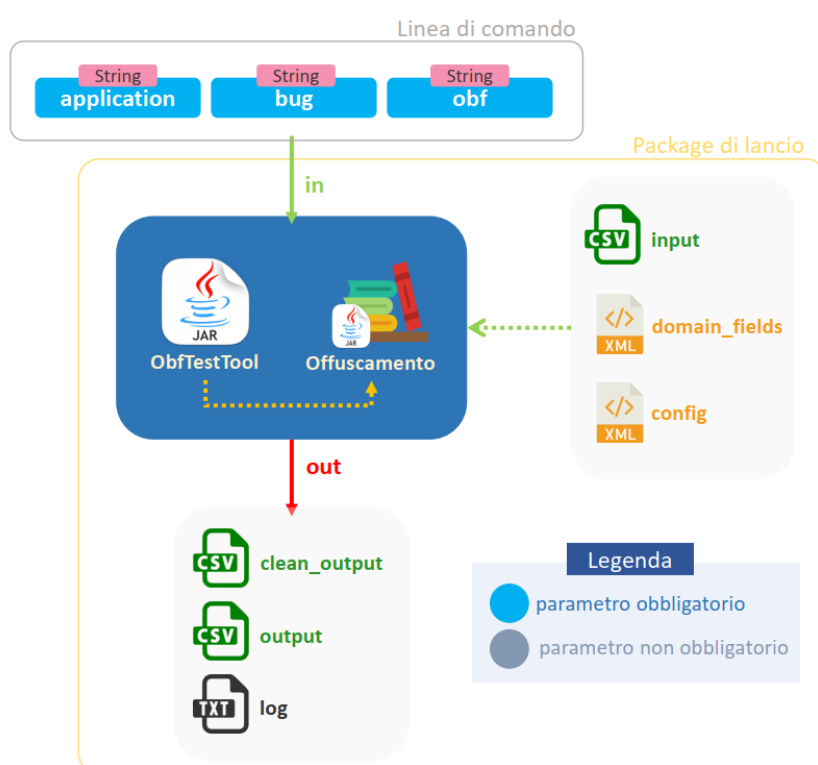


Figura 3.2: Architettura della soluzione

Come illustrato in Figura 3.2, il tool, nell'eseguire la funzionalità considerata, richiede obbligatoriamente alcuni parametri:

- **@application** id dell'applicazione [*dominio: String*]
- **@bug** id del bug [*dominio: String*]
- **@obf** id della funzione di offuscamento [*dominio: String*]

Grazie ai parametri passati, lo strumento è in grado di leggere dal package di lancio (Sezione 4.2.1) alcune informazioni essenziali all'esecuzione della funzionalità:

■ tuple da offuscare

Ottiene le tuple da offuscare leggendo il file di input posizionato nel package di lancio nel percorso `src/@application/@bug/input.csv`

■ tecnica di offuscamento, colonne da offuscare, numero di ripetizioni

Ottiene queste informazioni leggendo il file di configurazione posizionato nel percorso `src/config.xml`. In particolar modo, effettuando il parsing del file xml, è in grado di ottenere le informazioni della tecnica di offuscamento con id `@obf` relativa al bug con id `@bug` dell'applicazione con id `@application`.

■ id del dominio delle colonne

Ottiene l'id del dominio di ogni colonna leggendo il file di configurazione posizionato nel percorso `src/config.xml`. In particolar modo, effettuando il parsing del file xml, è in grado di ottenere gli identificativi dall'attributo delle colonne relative al bug con id `@bug` dell'applicazione con id `@application`.

■ dominio delle colonne

Ottiene le informazioni relative al dominio delle colonne leggendo il file di configurazione dei domini posizionato nel percorso `src/@application/domain_fields.xml`. In particolar modo, effettuando il parsing del file xml è in grado di ottenere le informazioni del dominio con un certo id, che ricava come spiegato nel punto precedente.

In Figura 3.3 viene riportato un esempio della lettura delle informazioni 'tecnica di offuscamento', 'colonne da offuscare' e 'numero di ripetizioni'.

```
<Applications>
  <Application id = "ActivityDiary">
    <Bug id = "bug1-ActivityExistingName" type = "crash" description = "In case an activity is
      created but the name already exists, the app crashed instead of rejecting the name.">
      <Columns>
        <Column id = "1" name = "activityName1" domain = "1"/>
        <Column id = "2" name = "activityName2" domain = "1"/>
      </Columns>
      <ObfuscationFunction id = "obf1" technique = "localSuppression" columns = "1;2" times = "5"/>
      <ObfuscationFunction id = "obf2" technique = "globalRecoding" columns = "1;2" times = "10"/>
      <Emulator id = "e30"/>
    </Bug>
  </Application>
</Applications>
```

Figura 3.3: Esempio: lettura dal file di configurazione

Lettura di tecnica di offuscamento (technique), colonne da offuscare (columns) e numero di ripetizioni (times) per l'esecuzione della funzionalità con parametri in input @application="ActivityDiary", @bug="bug1-ActivityExistingName", @obf="obf1".

A questo punto il tool è in possesso di tutte le informazioni necessarie ad eseguire la

funzionalità e può offuscare le tuple utilizzando i metodi offerti dalla libreria di offuscamento.

L'esecuzione della funzionalità, nel caso in cui vada a buon fine, porta alla produzione dei seguenti file:



Il file *clean_output.csv* contiene semplicemente le tuple prodotte dall'offuscamento, precedute da un'intestazione composta dai nomi delle colonne. Il file 'pulito' può essere utilizzato, senza nessuna modifica, come file di input per l'automazione dei test (funzionalità 'Automatizzazione dei test'). Il suo mantenimento in memoria è di fondamentale importanza in quanto non basterebbe rieseguire la funzionalità con gli stessi parametri per ottenere i medesimi valori offuscati, poichè alcune delle tecniche di offuscamento dipendono da un fattore randomico. In Figura 3.4 viene mostrato un esempio del file prodotto.



Il file *output.csv* può essere considerato il file riepilogativo dell'esecuzione della funzionalità in quanto contiene le tuple originali, le tuple offuscate e la tecnica di offuscamento applicata. Il file ha una struttura del tipo [Val1; Val2; ...; Valn; NULL; Val1_Off; Val2_Off; ...; Valn_Off; Technique;] in cui, nell'intestazione, il nome delle colonne che sono state offuscate è seguito da un '*'. In Figura 3.5 viene mostrato un esempio del file prodotto.



Il file *log.txt* contiene semplicemente l'output generato dall'esecuzione della funzionalità sul Command Prompt. Inoltre nel documento vengono salvati come copie chiave-valore i parametri con cui il tool è stato lanciato. In Figura 3.6 viene mostrato un esempio del file prodotto.

	A	B
1	activityName1	activityName2
2	Q	Nfs
3	Q	J
4	E	Xm
5	Vm	Y
6	Jx	Tnkkv
7	A	Swi
8	P	Tn
9	Blrwv	Ytl
10	Uakvyvgh	O
11	Rw	F
12	Rg	H
13	H	Ot
14	Z	K
15	K	F
16	B	Gr

	A	B	C	D	E	F
1	activityName1	activityName2		activityName1_Off*	activityName2_Off*	Technique
2	corsA	corsa		Q	Nfs	localSuppression
3	corsA	corsa		Q	J	localSuppression
4	corsA	corsa		E	Xm	localSuppression
5	corsA	corsa		Vm	Y	localSuppression
6	corsA	corsa		Jx	Tnkkv	localSuppression
7	dentista	medico		A	Swi	localSuppression
8	dentista	medico		P	Tn	localSuppression
9	dentista	medico		Blrwv	Ytl	localSuppression
10	dentista	medico		Uakvyvgh	O	localSuppression
11	dentista	medico		Rw	F	localSuppression
12	camminata	camminata		Rg	H	localSuppression
13	camminata	camminata		H	Ot	localSuppression
14	camminata	camminata		Z	K	localSuppression
15	camminata	camminata		K	F	localSuppression
16	camminata	camminata		B	Gr	localSuppression

Figura 3.5: Esempio: *output.csv*Figura 3.4: Esempio: *clean_output.csv*

```
JAR launched with the following parameters: [command:"Obf" application:"ActivityDiary" bug:"bug1-ActivityExistingName" obf:"obf1"]

*****
* OBFUSCATION TOOL *
*****

> Execution parameters
  application: ActivityDiary
  bug: bug1-ActivityExistingName
  function: obf1

> Obfuscation info
  technique: localSuppression
  times: 5
  affected columns: activityName1, activityName2

RESULT
-----
> output.csv created in \C:\Users\Stefano\Desktop\Tool\src\ActivityDiary\bug1-ActivityExistingName\obf1\date17-10-2021time15-6-47\output.csv
  *Columns that have been obfuscated are marked with an asterisk in the output file
> clean_output.csv (file containing only the generated tuples) created in
  \C:\Users\Stefano\Desktop\Tool\src\ActivityDiary\bug1-ActivityExistingName\obf1\date17-10-2021time15-6-47\clean_output.csv
-----
```

Figura 3.6: Esempio: *log.txt*

Per ogni esecuzione della funzionalità viene creata una cartella contenente i file prodotti appena discussi. Il nome della cartella dipende dal momento in cui il tool è stato eseguito e segue un pattern di questo tipo: 'dateDD-MM-YYYYtimeHH-MM-SS'. La cartella viene posizionata all'interno del package di lancio nel percorso `src/@application/@bug/@obf`. In Figura 3.7 vengono mostrate le cartelle create per ogni esecuzione della funzionalità con specifici parametri .

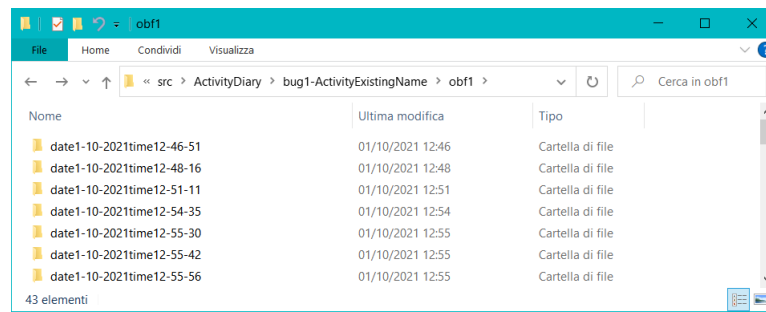


Figura 3.7: Esempio: cartelle di esecuzione

Cartelle create per le esecuzioni della funzionalità con parametri in input
@application="ActivityDiary", @bug="bug1-ActivityExistingName", @obf="obf1"

3.2.2 Automatizzazione dei test

La funzionalità 'Automatizzazione dei test' si occupa dell'automazione dei test rispondendo al requisito funzionale *Fu.2*. Specificatamente la funzionalità permette di eseguire automaticamente test parametrici su applicazioni Android e, per ogni test, di catturare il risultato e verificare se il bug atteso è stato riprodotto. Inoltre viene offerta la possibilità di salvare uno screenshot o uno screen record per ogni esecuzione del test parametrizzato.

Architettura della soluzione

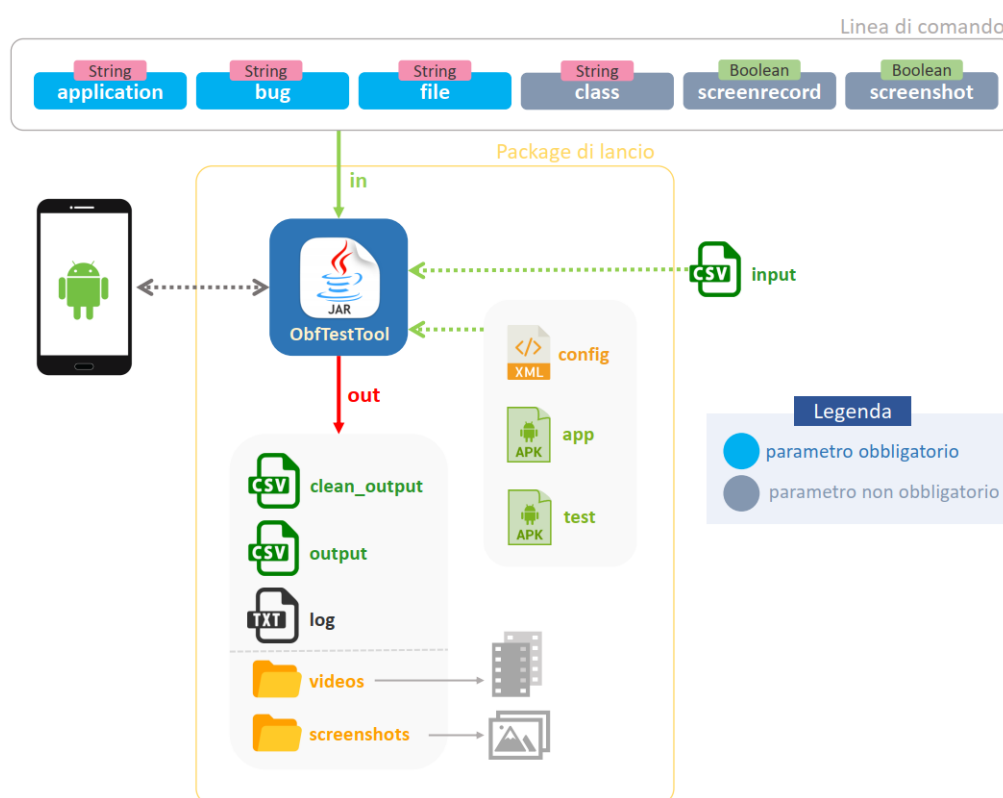


Figura 3.8: Architettura della soluzione

Come illustrato in Figura 3.8, il tool, nell'eseguire la funzionalità considerata, richiede obbligatoriamente alcuni parametri:

- **@application** id dell'applicazione [*dominio: String*]
- **@bug** id del bug [*dominio: String*]
- **@file** path del file csv contenente i parametri con cui lanciare i test (deve essere strutturato nello stesso modo del file di input dei dati trattato nella Sezione 4.2.2) [*dominio: String*]

L'esecuzione della funzionalità 'Automatizzazione dei test' può essere personalizzata specificando altri parametri non obbligatori:

- **@class** pacchetto interno in cui si trova la classe di test con aggiunta del nome della classe di test (vedere Appendice A - Generazione degli APK - 1.Creazione della classe di test - Posizione della classe di test) [*dominio: String*]
valore di default nome della classe lanciabile del file app.apk + "Test"
- **@screenrecord** valore booleano che indica se si vuole utilizzare la funzionalità di salvataggio di uno screen record per ogni esecuzione del test parametrizzato [*dominio: Boolean (true, false)*]
valore di default false
- **@screenshot** valore booleano che indica se si vuole utilizzare la funzionalità di salvataggio di uno screenshot per ogni esecuzione del test parametrizzato [*dominio: Boolean (true, false)*]
valore di default false

Grazie ai parametri passati, lo strumento è in grado di leggere dal package di lancio (Sezione 4.2.1) alcune informazioni essenziali all'esecuzione della funzionalità:

■ **app.apk**

Ottiene il file APK dell'applicazione da testare cercando un file con estensione 'apk' all'interno del package di lancio nel percorso `src/@application`.

■ **test.apk**

Ottiene il file APK di test dell'applicazione da testare cercando un file con estensione 'apk' all'interno del package di lancio nel percorso `src/@application/@bug`.

■ **tipo di bug atteso**

Ottiene il tipo di bug atteso leggendo il file di configurazione posizionato nel percorso `src/config.xml`. In particolar modo, effettuando il parsing del file xml, è in grado di ottenere il tipo di bug del bug con id `@bug` dell'applicazione con id `@application`.

■ **id dell'emulatore**

Ottiene l'id dell'emulatore su cui lanciare i test leggendo il file di configurazione posizionato nel percorso `src/config.xml`. In particolar modo, effettuando il parsing del file xml, è in grado di ottenere l'id dell'emulatore associato al bug con id `@bug` dell'applicazione con id `@application`.

■ **nome dell'AVD**

Ottiene il nome dell'AVD su cui lanciare i test leggendo il file di configurazione posizionato nel percorso `src/config.xml`. In particolar modo, effettuando il parsing del file xml, è in grado di ottenere il nome dell'AVD avente come id l'identificativo

trovato al punto precedente.

A questo punto il tool è in possesso di tutte le informazioni necessarie ad eseguire la funzionalità e può lanciare i test parametrici sull'emulatore.

L'esecuzione della funzionalità, nel caso in cui vada a buon fine, porta alla produzione dei seguenti file:



Il file *clean_output.csv* è semplicemente una copia del file in input (percorso *@file*) contenente i parametri con cui lanciare i test. Il suo mantenimento in memoria permette in qualsiasi momento di poter rieseguire la funzionalità con gli stessi parametri di test.



Il file *output.csv* può essere considerato il file riepilogativo dell'esecuzione della funzionalità in quanto contiene i parametri del test, il risultato di ogni test (trattato nel Paragrafo 'Risultato del test') e l'esito della riproduzione del bug atteso (trattato nel Paragrafo 'Logica per la verifica della riproduzione del bug'). Il file ha una struttura del tipo [Val1; Val2; ...; Valn; NULL; TestResult;Info;BugReproduction;]. In Figura 3.9 viene mostrato un esempio del file prodotto.



Il file *log.txt* contiene semplicemente l'output generato dall'esecuzione della funzionalità sul Command Prompt. Inoltre nel documento vengono salvati come coppie chiave-valore i parametri con cui il tool è stato lanciato. In Figura 3.10 viene mostrato un esempio del file prodotto.

	A	B	C	D	E	F
1	activityName1	activityName2		TestResult	Info	BugReproduction
2	corsA	corsa		test passed		not reproduced
3	dentista	medico		test passed		not reproduced
4	camminata	camminata		crash		reproduced
5		palestra		exception	java.lang.IllegalFormatException	new bug
6	lezione	palestra		test passed		not reproduced

Figura 3.9: Esempio: *output.csv*

```

JAR launched with the following parameters: [application:"ActivityDiary"  bug:"bug1-ActivityExistingName"  class:"MainActivityTest"
file:"C:\Users\Stefano\Desktop\Tool\src\ActivityDiary\bug1-ActivityExistingName\input.csv"  command:"Auto"]

*****
*  AUTOMATION TOOL  *
*****
> Execution parameters
  application: ActivityDiary
  bug: bug1-ActivityExistingName
  test class: MainActivityTest
  input file path: C:\Users\Stefano\Desktop\Tool\src\ActivityDiary\bug1-ActivityExistingName\input.csv
> Bug info
  type: crash
  description: In case an activity is created but the name already exists, the app crashed instead of rejecting the name.
> Running emulator...
  (Emulator 'emulator-5554' killed)
> Emulator started... [android-version: 11] [api-level: 30]
> App APK installed...
  (Test Apk already exists, it will be uninstalled before installation)
> Test APK installed...
> Test run...
  Index  Test input          TestResult          BugReproduction
  -----
  (1)    'corsa;corsa'         succes [test passed] [not reproduced]
  (2)    'dentista;medico'     succes [test passed] [not reproduced]
  (3)    'camminata;camminata' failed [crash]        [reproduced]
  (4)    ';palestra'          failed [exception]    [new bug]
                                     {java.lang.IllegalFormatException}
  (5)    'lezione;palestra'    succes [test passed] [not reproduced]

RESULT -----
> output.csv created in \C:\Users\Stefano\Desktop\Tool\src\ActivityDiary\bug1-ActivityExistingName\no_obfuscation\date17-10-2021time18-18-31\output.csv

```

Figura 3.10: Esempio: *log.txt*

Inoltre, in base ai parametri con cui la funzionalità viene lanciato, potrebbero essere prodotti anche:



La cartella videos contiene la registrazione dello schermo (in formato mp4) per ogni test lanciato. La funzionalità viene trattata nel dettaglio nella Sezione 4.6.



La cartella screenshots contiene un'istantanea dello schermo (in formato png) per ogni test lanciato. La funzionalità viene trattata nel dettaglio nella Sezione 4.7.

Per ogni esecuzione della funzionalità viene creata una cartella contenente i file prodotti appena discussi. Il nome della cartella dipende dal momento in cui il tool è stato eseguito e segue un pattern di questo tipo: 'dateDD-MM-YYYYtimeHH-MM-SS'. La cartella viene posizionata all'interno del package di lancio nel percorso `src/@application/no_obfuscation`.

Risultato del test

L'esecuzione di ogni test parametrico può terminare nei seguenti modi:

- **crash**

L'applicazione termina in crash senza sollevare nessuna eccezione.

- **exception**

L'esecuzione del test termina con il sollevamento di un'eccezione non gestita. Il tool è in grado di catturare il tipo di eccezione.

- **test passed**

L'esecuzione del test termina con successo: non è stata sollevata nessuna eccezione, l'applicazione non è terminata in crash e, nel caso in cui il caso di test la preveda, l'asserzione è stata verificata.

- **test not passed**

L'esecuzione del test fallisce perchè l'asserzione non è stata verificata (non perchè l'applicazione è terminata con un crash o è stata sollevata un'eccezione). Questo tipo risultato quindi può verificarsi solo nel caso in cui il caso di test preveda un'asserzione.

Logica per la verifica della riproduzione del bug

Lo strumento, come analizzato, è in grado di verificare se il bug atteso è stato riprodotto. Nel file di configurazione generale *config.xml*, come illustrato nella Sezione 4.2.2, per ogni bug può essere specificato il tipo di bug atteso, con la possibilità di scegliere tra uno dei seguenti valori:

- **assertion**

Il bug atteso si manifesta quando una o più asserzioni non vengono verificate e quindi se il risultato del test è *test not passed*.

- **crash**

Il bug atteso si manifesta quando l'applicazione termina in crash e quindi se il risultato del test è *crash*.

- **exception**

Il bug atteso si manifesta quando l'applicazione solleva un'eccezione, e quindi se il risultato del test è *exception*, e il tipo di eccezione sollevato corrisponde con il tipo di eccezione atteso. Infatti, in questo caso, è obbligatorio aggiungere una colonna chiamata 'expectedException' sia nel file di configurazione che nel file di input, che abbia come valore il tipo di eccezione attesa. La Figura 3.11 potrebbe essere utile ad una migliore comprensione della situazione appena illustrata.

■ NS

Il bug atteso non può essere classificato in nessuno dei casi precedenti. In questa situazione risulta utile utilizzare le funzionalità di salvataggio degli screenshots e degli screen record ed effettuare la verifica manualmente.

	A	B	C
1	val1	val2	expectedException
2	28	2	java.lang.ArithmeticException
3	8	4	java.lang.ArithmeticException
4	0	4	java.lang.ArithmeticException
5	0	0	java.lang.ArithmeticException
6	216	4	java.lang.ArithmeticException

```

<Columns>
  <Column id="1" name="val1" domain="18"/>
  <Column id="2" name="val2" domain="1"/>
  <Column id="3" name="expectedException"/>
</Columns>

```

Figura 3.11: Esempio: configurazione bug di tipo *exception*

La verifica della riproduzione del bug atteso (campo 'BugReproduction' nel file di output *output.csv*) può terminare con uno dei seguenti valori:

■ reproduced

Il bug atteso è stato riprodotto, in base alla logica appena esposta.

■ not reproduced

Il bug atteso non è stato riprodotto, in base alla logica appena esposta, e non si sono presentati altri bug.

■ new bug

Il bug atteso non è stato riprodotto, in base alla logica appena esposta, ma si è verificato un nuovo bug. Questo caso può verificarsi nelle seguenti situazioni:

- Il bug atteso è di tipo 'assertion', ma è stata sollevata un'eccezione
- Il bug atteso è di tipo 'assertion', ma l'applicazione è terminata con un crash
- Il bug atteso è di tipo 'exception', ma l'applicazione è terminata con un crash
- Il bug atteso è di tipo 'crash', ma è stata sollevata un'eccezione

La logica per la verifica della riproduzione del bug può essere sintetizzata nella tabella illustrata in Figura 3.12, che considera tutte le possibili combinazioni del caso.

		Test Result			
		passed	not passed	exception	crash
Bug Type	assertion	not reproduced	reproduced	new bug	new bug
	exception	not reproduced	not reproduced*	reproduced**	new bug
	crash	not reproduced	not reproduced*	new bug	reproduced
	NS	na	na	na	na

* Casistica che non dovrebbe mai verificarsi

** Se l'eccezione sollevata corrisponde con l'eccezione attesa, altrimenti 'new bug'

Figura 3.12: Logica per la verifica della riproduzione del bug

Nella tabella sono state inserite anche delle casistiche che non dovrebbero idealmente mai verificarsi, infatti:

■ caso **BugType = 'exception', TestResult = "not passed"**

Nel caso di test che dovrebbe sollevare un'eccezione, non ha senso inserire un'asserzione (il risultato del test 'not passed' si verifica solo quando è presente un'asserzione e questa non è verificata)

■ caso **BugType = 'crash', TestResult = "not passed"**

Nel caso di test che dovrebbe provocare un bug dell'applicazione, non ha senso inserire un'asserzione (il risultato del test 'not passed' si verifica solo quando è presente un'asserzione e questa non è verificata)

3.2.3 Offuscamento dei dati e automatizzazione dei test

La funzionalità 'Offuscamento dei dati e automatizzazione dei test' permette il raggiungimento dell'obiettivo principale dello stage rispondendo al requisito funzionale *Fu.3*. La funzionalità permette di eseguire automaticamente test parametrici utilizzando in input le tuple offuscate ottenute applicando le tecniche di offuscamento sui dati originali che provocano il bug. In questo modo lo strumento è in grado di verificare, per ogni tupla offuscata, se il bug atteso (provocato dalla tupla non offuscata) è stato riprodotto.

Architettura della soluzione

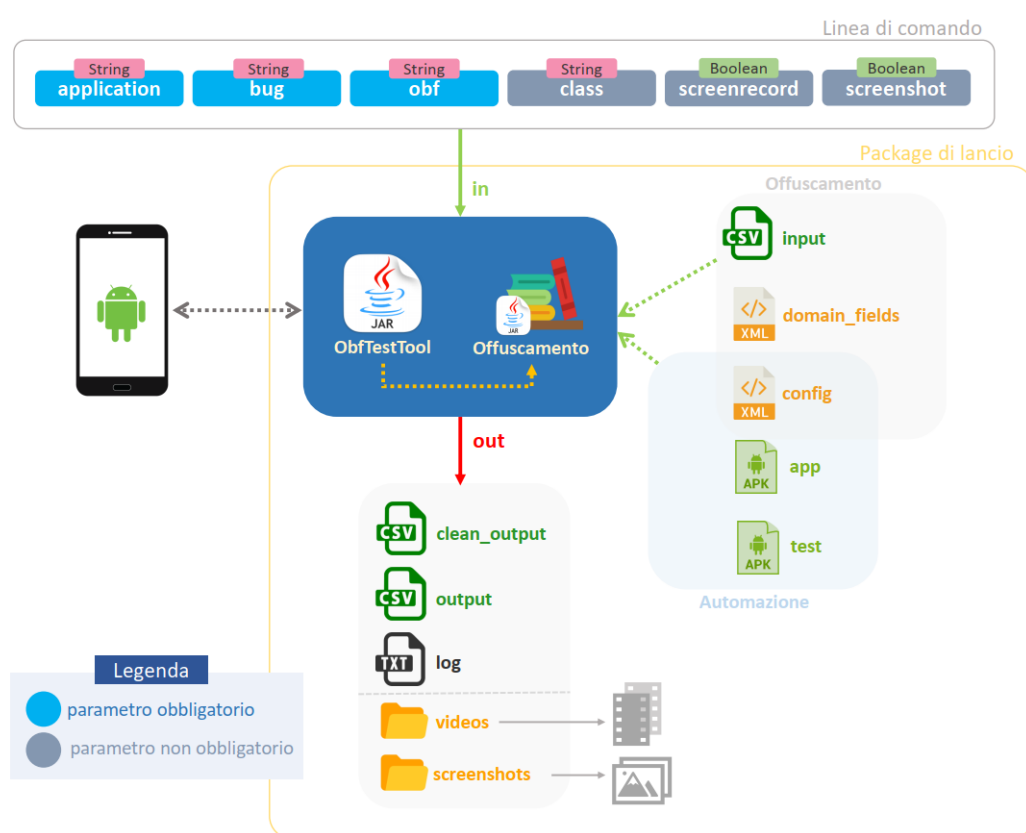


Figura 3.13: Architettura della soluzione

Come illustrato in Figura 3.13, il tool, nell'eseguire la funzionalità considerata, richiede obbligatoriamente alcuni parametri:

- **@application** id dell'applicazione [dominio: String]
- **@bug** id del bug [dominio: String]
- **@obf** id della funzione di offuscamento [dominio: String]

L'esecuzione della funzionalità 'Offuscamento dei dati e automatizzazione dei test' può essere personalizzata specificando altri parametri non obbligatori:

- **@class** pacchetto interno in cui si trova la classe di test con aggiunta del nome della classe di test (vedere Appendice A - Generazione degli APK - 1.Creazione della classe di test - Posizione della classe di test) [*dominio: String*]
valore di default nome della classe lanciabile del file app.apk + "Test"
- **@screenrecord** valore booleano che indica se si vuole utilizzare la funzionalità di salvataggio di uno screen record per ogni esecuzione del test parametrizzato [*dominio: Boolean (true, false)*]
valore di default false
- **@screenshot** valore booleano che indica se si vuole utilizzare la funzionalità di salvataggio di uno screenshot per ogni esecuzione del test parametrizzato [*dominio: Boolean (true, false)*]
valore di default false

Grazie ai parametri passati, lo strumento è in grado di leggere dal package di lancio (Sezione 4.2.1) le informazioni essenziali all'esecuzione della funzionalità, che sono complessivamente quelle elencate per le singole funzionalità 'Automatizzazione dei test' e 'Offuscamento dei dati' nelle sezioni precedenti.

A questo punto il tool è in possesso di tutte le informazioni necessarie ad eseguire in sequenza prima l'offuscamento delle tuple e poi l'automazione dei test. L'automazione dei test viene effettuata utilizzando come file di input il file *clean_output.csv* prodotto dall'offuscamento dei dati.

L'esecuzione della funzionalità, nel caso in cui vada a buon fine, porta alla produzione dei seguenti file:



Corrisponde completamente al file *clean_output.csv* prodotto dalla funzionalità 'Offuscamento dei dati'.



Il file *output.csv* può essere considerato il file riepilogativo dell'esecuzione della funzionalità in quanto contiene le tuple originali, le tuple offuscate, la tecnica di offuscamento, il risultato di ogni test e l'esito della riproduzione del bug. Il file ha una struttura del tipo [Val1; Val2; ...; Valn; NULL; Val1_Off; Val2_Off; ...; Valn_Off; Technique; NULL; TestResult; Info; BugReproduction] in cui, nell'intestazione, il nome delle colonne che sono state offuscate è seguito da un '*'. In Figura 3.14 viene mostrato un esempio del file prodotto.



Il file *log.txt* contiene semplicemente l'output generato dall'esecuzione della funzionalità sul Command Prompt. Inoltre nel documento vengono salvati come coppie chiave-valore i parametri con cui il tool è stato lanciato. In Figura 3.15 viene mostrato un esempio del file prodotto.

	A	B	C	D	E	F	G	H	I	J
1	activityName1	activityName2		activityName1_Off*	activityName2_Off*	Technique		TestResult	Info	BugReproduction
2	corsA	corsa		K	C	localSuppression		test passed		not reproduced
3	corsA	corsa		Jauhh	Q	localSuppression		test passed		not reproduced
4	corsA	corsa		Jz	Jz	localSuppression		crash		reproduced
5	corsA	corsa		Chqj	E	localSuppression		test passed		not reproduced
6	corsA	corsa		O	G	localSuppression		test passed		not reproduced
7	dentista	medico		C	I	localSuppression		test passed		not reproduced
8	dentista	medico		Mqc	Od	localSuppression		test passed		not reproduced
9	dentista	medico		Plxqqn	S	localSuppression		test passed		not reproduced
10	dentista	medico		L	T	localSuppression		test passed		not reproduced
11	dentista	medico		C	C	localSuppression		crash		reproduced
12	camminata	camminata		Sx	Sx	localSuppression		crash		reproduced
13	camminata	camminata		L	Fb	localSuppression		test passed		not reproduced
14	camminata	camminata		Wmo	Vr	localSuppression		test passed		not reproduced
15	camminata	camminata		Lxq	A	localSuppression		test passed		not reproduced
16	camminata	camminata		C	U	localSuppression		test passed		not reproduced

Figura 3.14: Esempio: *output.csv*

```
JAR launched with the following parameters: [application:"ActivityDiary" bug:"bug1-ActivityExistingName" class:"MainActivityTest" command:"ObfAut" obf:"Obf1"]

*****
* OBFUSCATION TOOL *
*****
> Execution parameters
  application: ActivityDiary
  bug: bug1-ActivityExistingName
  function: obf1
> Obfuscation info
  technique: localSuppression
  times: 5
  affected columns: activityName1, activityName2

RESULT
> output.csv created in 'C:\Users\Stefano\Desktop\Tool\src\ActivityDiary\bug1-ActivityExistingName\obf1\date17-10-2021time22-49-11\output.csv'
  *Columns that have been obfuscated are marked with an asterisk in the output file*
> clean_output.csv (file containing only the generated tuples) created in 'C:\Users\Stefano\Desktop\Tool\src\ActivityDiary\bug1-ActivityExistingName\obf1\date17-10-2021time22-49-11\clean_output.csv'

*****
* AUTOMATION TOOL *
*****
> Execution parameters
  application: ActivityDiary
  bug: bug1-ActivityExistingName
  test class: MainActivityTest
> Bug info
  type: crash
  description: In case an activity is created but the name already exists, the app crashed instead of rejecting the name.
> Running emulator...
  (emulator "emulator-5554" killed)
> emulator started... [android-version: 11] [api-level: 30]
  (App Apk already exists, it will be uninstalled before installation)
> App APK installed...
  (Test Apk already exists, it will be uninstalled before installation)
> Test APK installed...
> Test run...
  Index   Test input   TestResult   BugReproduction
  (1)     'KjC'        succes [test passed]   [not reproduced]
  (2)     'Jaahh;Q'    succes [test passed]   [not reproduced]
  (3)     'Jz;Jz'      failed [crash]         [reproduced]
  (4)     'Chqj;E'     succes [test passed]   [not reproduced]
  (5)     'OjG'        succes [test passed]   [not reproduced]
  (6)     'G;I'        succes [test passed]   [not reproduced]
  (7)     'Mq;Od'      succes [test passed]   [not reproduced]
  (8)     'Plxqgn;S'   succes [test passed]   [not reproduced]
  (9)     'L;T'        succes [test passed]   [not reproduced]
  (10)    'CjC'        failed [crash]         [reproduced]
  (11)    'Sw;Sx'     failed [crash]         [reproduced]
  (12)    'L;PB'      succes [test passed]   [not reproduced]
  (13)    'Wmo;Vn'    succes [test passed]   [not reproduced]
  (14)    'Lxq;A'     succes [test passed]   [not reproduced]
  (15)    'C;U'       succes [test passed]   [not reproduced]

RESULT
> output.csv updated in 'C:\Users\Stefano\Desktop\Tool\src\ActivityDiary\bug1-ActivityExistingName\obf1\date17-10-2021time22-49-11\output.csv'
```

Figura 3.15: Esempio: *log.txt*

Inoltre, in base ai parametri con cui la funzionalità viene lanciata, potrebbero essere prodotti anche:



videos

La cartella videos contiene la registrazione dello schermo (in formato mp4) per ogni test lanciato. La funzionalità viene trattata nel dettaglio nella Sezione 4.6.



screenshots

La cartella screenshots contiene un'istantanea dello schermo (in formato png) per ogni test lanciato. La funzionalità viene trattata nel dettaglio nella Sezione 4.7.

Per ogni esecuzione della funzionalità viene creata una cartella contenente i file prodotti appena discussi. Il nome della cartella dipende dal momento in cui il tool è stato eseguito e segue un pattern di questo tipo: 'dateDD-MM-YYYYtimeHH-MM-SS'. La cartella viene posizionata all'interno del package di lancio nel percorso *src/@application/@bug/@obf*.

Capitolo 4

Dettagli implementativi del Framework sviluppato

In questo capitolo verranno analizzati alcuni dei dettagli implementativi più interessanti.

4.1 Architettura software del tool

Nella Figura 4.1 viene illustrata una semplificazione dell'architettura software del tool con aggiunta delle interazioni con l'ambiente più interessanti.

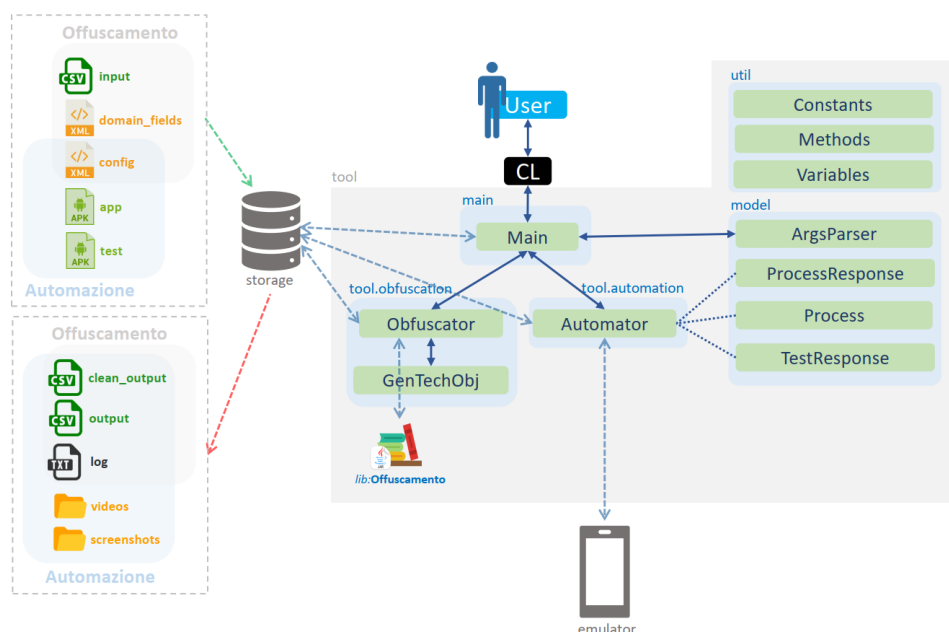


Figura 4.1: Architettura software

Come ripetuto più volte, lo strumento deve essere lanciato dall'utente da linea di comando. Il tool interagisce con l'emulatore (su cui esegue i casi di test) e con la memoria (da cui legge e in cui produce file).

Brevemente si analizza l'utilità della classi in base al package di appartenenza:

■ **package: main**

Il package 'main' contiene solamente la classe 'Main'. La classe si interfaccia direttamente con l'utente e svolge la funzione di lettura dei parametri (e lancio del parsing) e chiamata ai metodi in base al comando da eseguire.

■ **package: tool.obfuscation**

Il package 'tool.obfuscation' contiene le classi che si occupano di gestire l'intero processo di offuscamento dei dati. Il package si interfaccia con la libreria di offuscamento.

■ **package: tool.automation**

Il package 'tool.automation' contiene solamente la classe 'Automator' che si occupa di gestire l'intero processo di automazione dei test. La classe si interfaccia con l'emulatore gestendo inoltre il suo ciclo di vita.

■ **package: util**

Il package 'util' contiene le classi con parametri/metodi utili all'intero tool.

■ **package: model**

Il package 'model' contiene le classi che modellano concetti utili ad altre classi.

4.2 Configurazione

4.2.1 Struttura del package di lancio

Per il corretto funzionamento del tool risulta fondamentale strutturare in modo corretto il package di lancio (Figura 4.2) in modo da permettere una corretta lettura dei file di configurazione e dei file di input.

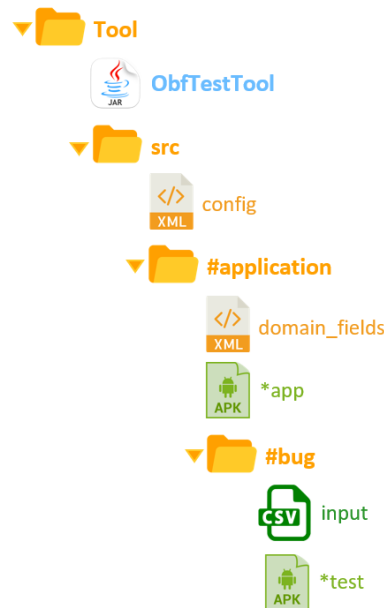


Figura 4.2: Struttura package di lancio

La struttura del package di lancio prevede la possibilità di:

- inserire nella cartella 'src' più cartelle '#application'
(il tool permette di tenere in memoria più applicazioni)
- inserire nella cartella '#application' più cartelle '#bug'
(il tool permette di tenere in memoria più bug per applicazione)

Nella struttura del package il nome dei file e delle cartelle è vincolante, con alcune eccezioni:

- alle cartelle il cui nome nella Figura 4.2 inizia con il carattere '#' dovrà essere attribuito un nome significativo
- il nome dei file il cui nome nella Figura 4.2 inizia con il carattere '*' è indifferente al funzionamento del tool
- il nome della cartella radice, chiamata nella Figura 4.2 'Tool', è indifferente al funzionamento del tool

L'architettura della soluzione e le scelte strutturali effettuate per il package di lancio possono essere comprese maggiormente andando ad analizzare brevemente l'utilità dei singoli file:



ObfTestTool

File JAR lanciabile da linea di comando in cui è stato impacchettato il tool che offre diverse funzionalità. Comprende al suo interno anche la libreria di offuscamento.



config

File di configurazione dell'intero tool contenente parametri utili all'offuscamento dei dati e all'automazione dei test. I parametri nel file sono definiti in modo differenziato per ogni bug di ogni applicazione.



domain_fields

File xml contenente i domini dei campi dell'applicazione utili alla libreria di offuscamento nell'applicazione delle tecniche.



input

File csv contenente le tuple da offuscare. Ai fini dello stage saranno inserite le tuple che generano il bug.



app

File APK dell'applicazione da testare.



test

File APK di test dell'applicazione da testare che permette il lancio del caso di test parametrico che dovrebbe riprodurre il bug.

4.2.2 File di configurazione e file di input

I file di configurazione e i file di input, come spiegato nella sezione precedente (Sezione 4.2.1), sono essenziali al funzionamento del tool e richiedono di essere prodotti rispettando alcuni restringenti requisiti.

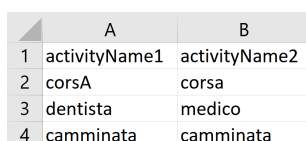
File di input dei dati `input.csv`

Il file *input.csv* contiene il dataset dei valori inseriti nell'applicazione dall'utente, idealmente ricavati dal log di esecuzione. Le tuple contenute sono destinate ad essere totalmente o in parte offuscate, in base ai parametri del file di configurazione.

Il file deve essere in formato csv in modo da essere facilmente leggibile (se aperto con un CSV viewer i dati vengono incolonnati come in Figura 4.3) e facile da modificare sia dall'utente che dal tool. Il file deve rispettare un pattern semplice:

- prima riga** nomi/etichette delle colonne/campi divise da un ','
- righe successive** tupla di valori ordinati delle colonne/campi divisi da un ','

La figura 4.4 mostra l'esempio di un file di input dei dati in cui il pattern è rispettato.



	A	B
1	activityName1	activityName2
2	corsA	corsa
3	dentista	medico
4	camminata	camminata

Figura 4.3: File di input (viewer)

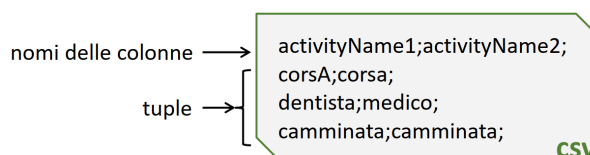


Figura 4.4: File di input (pattern)

File di configurazione generale `config.xml`

Il file di configurazione generale permette di parametrizzare in modo semplice l'applicativo, con informazioni che riguardano sia l'offuscamento dei dati che l'automazione dei test (e conseguente verifica della riproduzione del bug atteso). I parametri nel file sono definiti in modo differenziato per ogni bug di ogni applicazione. Il file di configurazione deve esser in formato xml e deve rispettare una struttura ben precisa.

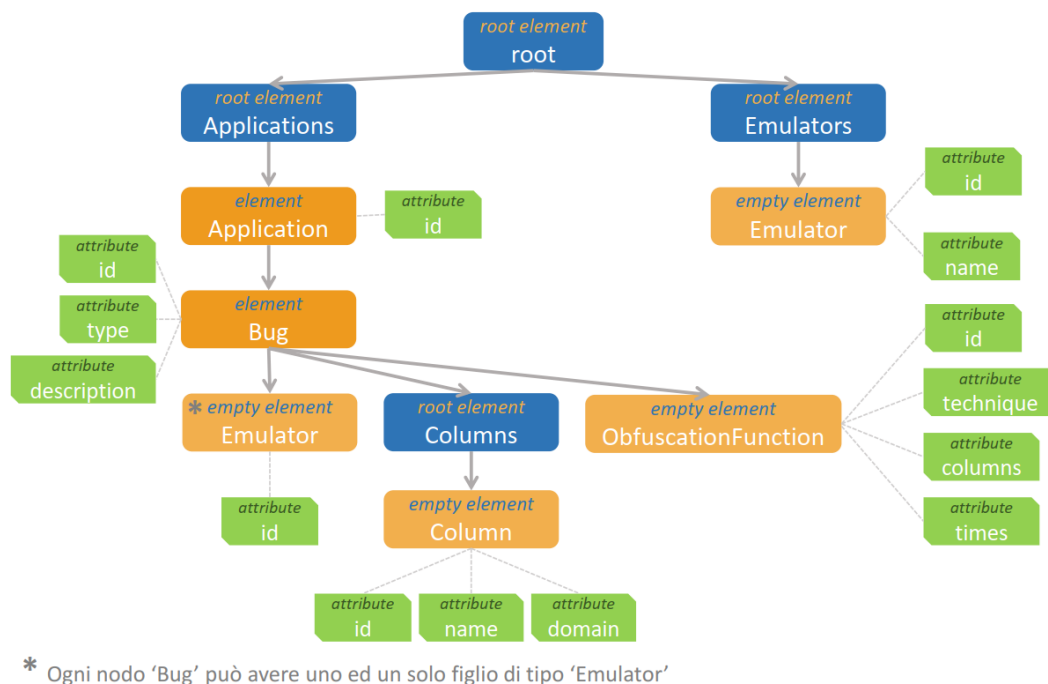


Figura 4.5: Struttura ad albero del file xml di configurazione

Il pattern richiesto dal documento è schematizzato in Figura 4.5: la struttura dell'albero xml può essere compresa nei dettagli solo conoscendo le varie tipologie di nodo/attributo (Appendice A - Lettura degli alberi xml).

Per sfruttare a pieno le potenzialità del tool (utilizzare sia le funzionalità di offuscamento che di automazione) è necessario compilare il file di configurazione in ogni sua parte e non tralasciare quindi nessun nodo e nessun attributo dell'albero.

La correlazione tra la struttura del package di lancio (Figura 4.2) e la struttura del file xml (Figura 4.5) è molto forte per ovvi motivi. Le scelte strutturali effettuate per il file di configurazione potrebbero essere comprese maggiormente andando ad analizzare l'utilità dei singoli nodi e dei relativi attributi:

■ root

Il nodo 'root' è il nodo radice dell'albero. Non svolge una funzionalità particolare, se non quella di permettere il rispetto delle regole previste dallo standard xml, che impone di avere un singolo nodo radice nel documento.

■ Applications

Il nodo 'Applications' è il nodo genitore di tutte le applicazioni.

■ Application

I nodi 'Application' rappresentano le applicazioni Android.

- **id** L'attributo rappresenta l'id con cui si vuole memorizzare l'applicazione. Il valore di questo campo deve corrispondere al nome della relativa cartella nel

package di lancio, indicata nella Figura 4.2 con '#application' (join).

■ Bug

I nodi 'Bug' rappresentano i bug delle applicazioni sui quali si vogliono sfruttare le funzionalità del tool. Per come è strutturato il file di configurazione, per ogni applicazione possono essere inseriti più bug.

- **id** L'attributo rappresenta l'id con cui si vuole memorizzare il bug. Il valore di questo campo deve corrispondere al nome della relativa cartella nel package di lancio, indicata nella Figura 4.2 con '#bug' (join).
- **type** L'attributo è opzionale e rappresenta il tipo di bug atteso. Può assumere uno tra i valori 'assertion', 'exception', 'crash' o 'NS'. L'attributo influisce sulla valutazione della riproduzione del bug atteso, che viene trattata nella Sezione 3.2.2.
- **description** L'attributo è opzionale e rappresenta la descrizione testuale del bug atteso.

■ Emulator (figlio del nodo 'Bug')

Il nodo 'Emulator' rappresenta l'emulatore su cui lanciare i test.

- **id** L'attributo rappresenta l'id dell'emulatore su cui lanciare i test. Il valore di questo campo deve corrispondere con l'id di uno dei nodi Emulator figli del nodo Emulators (join).

■ Columns

Il nodo 'Columns' è il nodo genitore delle colonne del file di input.

■ Column

I nodi 'Column' rappresentano le colonne/i campi del file di input. Per ogni colonna del file *input.csv*, deve essere presente un nodo con tag 'Column'.

- **id** L'attributo rappresenta l'id numerico della colonna.
- **name** L'attributo rappresenta il nome della colonna. Il valore di questo campo deve corrispondere con il relativo nome della colonna nel file *input.csv*.
- **domain** L'attributo rappresenta l'id numerico del dominio del campo. Il valore dell'attributo deve corrispondere con l'id di uno dei domini configurati nel file *domain_fields.xml*.

■ ObfuscationFunction

I nodi 'ObfuscationFunction' rappresentano le funzioni di offuscamento applicabili.

- **id** L'attributo rappresenta l'id della funzione di offuscamento.
- **technique** L'attributo rappresenta il nome letterale della tecnica di offuscamento. Serve per invocare la corretta funzione nella libreria delle tecniche di offuscamento ('localSuppression', 'generalization'...).

- **columns** L'attributo rappresenta gli id delle colonne da offuscare, separati dal ','.
- **times** L'attributo rappresenta il numero di volte di applicazione della tecnica di offuscamento sui dati.

■ Emulators

Il nodo 'Emulators' è il nodo genitore di tutti gli emulatori.

■ Emulator (figlio del nodo 'Emulators')

I nodi 'Emulator' rappresentano gli emulatori assegnabili ai vari bug.

- **id** L'attributo rappresenta l'id dell'emulatore.
- **name** L'attributo rappresenta il nome dell'emulatore. Il valore di questo campo deve corrispondere con il nome di uno degli AVD¹ configurati (Appendice A - Configurazione degli AVD).

```
<root>
  <Applications>
    <Application id = "ActivityDiary">
      <Bug id = "bug1-ActivityExistingName" type = "crash" description = "In case an activity is
        created but the name already exists, the app crashed instead of rejecting the name.">
        <Columns>
          <Column id = "1" name="activityName1" domain="1"/>
          <Column id="2" name="activityName2" domain="1"/>
        </Columns>
        <ObfuscationFunction id = "obf1" technique = "localSuppression" columns="1;2" times="5"/>
        <ObfuscationFunction id = "obf2" technique = "globalRecoding" columns="1;2" times="10"/>
        <Emulator id="e30"/>
      </Bug>
    </Application>
  </Applications>

  <Emulators>
    <Emulator id="e30" name="mobile1" /> <!--API 30-->
    <Emulator id="e22" name="mobile2"/> <!--API 22-->
    <Emulator id="e25" name="mobile3"/> <!--API 25-->
  </Emulators>
</root>
```

Figura 4.6: Esempio: *config.xml*

Nella Figura 4.6 viene mostrato lo screenshot di un file di configurazione correttamente impostato.

File di configurazione dei domini *domain_fields.xml*

Il file di configurazione dei domini ha il fine di contenere i domini dei campi dell'applicazione, utili alla libreria nell'applicazione delle tecniche di offuscamento. Ogni nodo 'Column' del file di configurazione generale infatti, come analizzato nella sezione precedente (Sezione 4.2.2), ha un attributo dominio il cui valore deve corrispondere all'identificativo di un

¹**AVD**: Android Virtual Device, configurazione che definisce le caratteristiche di un dispositivo che si vuole simulare

dominio nel relativo file *domain_fields.xml*. Il documento è stato strutturato in maniera completamente identica a quella prevista dal collega A. Belisario nella sua tesi magistrale.

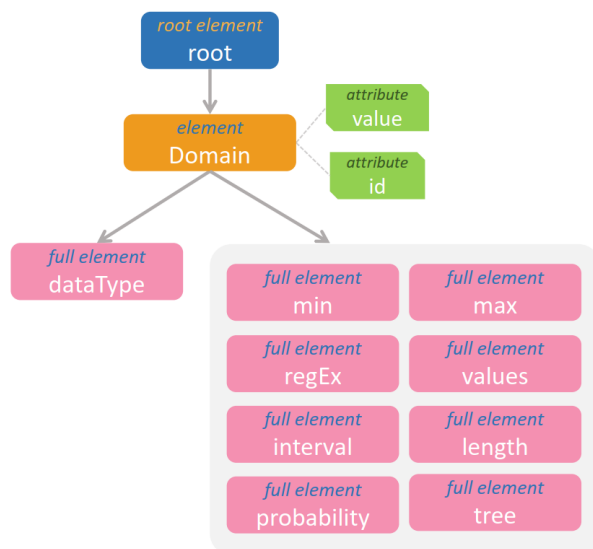


Figura 4.7: Struttura ad albero del file xml di configurazione

La struttura dell'albero xml (Figura 4.7) può essere compresa nei dettagli solo conoscendo le varie tipologie di nodo/attributo (Appendice A - Lettura degli alberi xml). La struttura del file è dinamica in base:

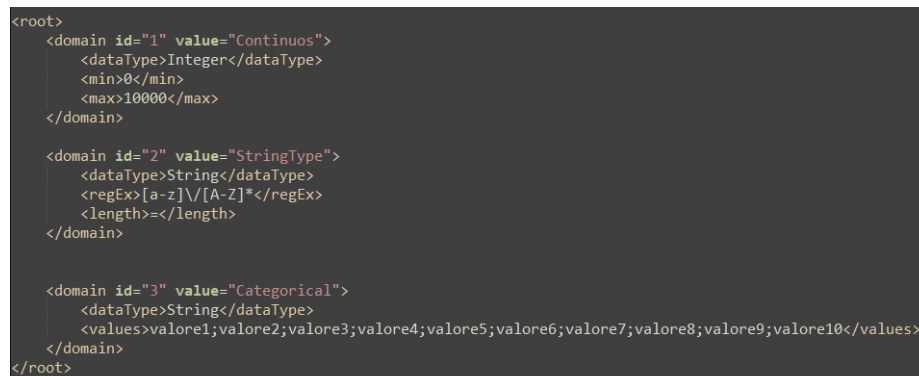
- alle informazioni necessarie per definire il dominio del dato
- alle informazioni necessarie ad ogni tecnica per poter offuscare nel modo corretto

Per quanto riguarda il dominio del dato, solo il nodo 'dataType' è sempre obbligatorio, mentre l'obbligatorietà degli altri nodi dipende dal suo valore:

- dataType=**Continuous**
 - **min** valore limite inferiore dell'insieme
 - **max** valore limite superiore dell'insieme
- dataType=**Categorical**
 - **values** lista dei valori che il campo può assumere, separati da un ','
- dataType=**StringType**
 - **regEx** espressione regolare che definisce il dominio del campo
 - **length** numero che definisce la lunghezza della stringa in output, se viene indicato il carattere '=' questa sarà generata con una lunghezza uguale a quella passata in input

Per quanto riguarda le informazioni necessarie alla tecnica di offuscamento, possono essere definiti i nodi:

- **tree** albero di gerarchia (indicando con root il valore più generico e ricorsivamente attraverso il tag node tutti i figli)
- **probability** matrice $N \times N$ (N sono i possibili valori che assume la variabile)



```

<root>
  <domain id="1" value="Continuos">
    <dataType>Integer</dataType>
    <min>0</min>
    <max>10000</max>
  </domain>

  <domain id="2" value="StringType">
    <dataType>String</dataType>
    <regEx>[a-z]\|[A-Z]*</regEx>
    <length>=</length>
  </domain>

  <domain id="3" value="Categorical">
    <dataType>String</dataType>
    <values>valore1;valore2;valore3;valore4;valore5;valore6;valore7;valore8;valore9;valore10</values>
  </domain>
</root>
    
```

 Figura 4.8: Esempio: *domain.xml*

Nella Figura 4.8 viene mostrato lo screenshot di un file di configurazione dei domini correttamente impostato.

File APK *app.apk*, *test.apk*

La generazione dei file *app.apk* e *test.apk* implica l'esecuzione di diversi passaggi che possono variare in base a differenti fattori (uno tra tutti la disponibilità del codice sorgente dell'applicazione). Per questo motivo il tema verrà trattato in un capitolo a parte (Capitolo 6).

4.3 Lancio del tool da linea di comando

4.3.1 Prerequisiti

I prerequisiti per il lancio del tool possono essere sintetizzati in:

- **configurazione del package di lancio**

Le cartelle e i file devono essere creati e posizionati all'interno del package di lancio come stabilito nella Sezione 4.2.1 e nella Sezione 4.2.2.

- **configurazione dell'ambiente di lancio**

Le variabili d'ambiente devono essere impostate correttamente (Appendice A - Variabili d'ambiente) e gli AVD devono essere creati adeguatamente (Appendice A - Configurazione degli AVD).

4.3.2 Lancio del tool

Il tool è stato interamente impacchettato in un file jar lanciabile da linea di comando. L'esecuzione dello strumento prevede l'apertura del Command Prompt dalla cartella root del package di lancio² e il lancio di un comando che deve rispettare il seguente pattern:

java -jar ObfTestTool.jar [parametri di lancio del tool]

Parametri di lancio del tool

I parametri di lancio del tool devono rispettare il seguente pattern:

command [options]

Le opzioni possono esser specificate in 2 modi:

1. [option] [value]
ex: '-a ApplicationName'
2. [option]:[value]
ex: '-a:ApplicationName'

Command

Command specifica la funzionalità del tool che si vuole utilizzare tra quelle disponibili (Sezione 3.2). Per ogni funzionalità è previsto un comando in versione breve e un comando in versione lunga:

- Funzionalità: **Offuscamento dei dati**

short command: Obf

²**cartella root del package di lancio** : la cartella contenente il file jar 'ObfTestTool' e la cartella 'src' (negli esempi chiamata 'Tool')

long command: Obfuscate

■ Funzionalità: **Automatizzazione dei test**

short command: Auto

long command: Automate

■ Funzionalità: **Offuscamento dei dati e automatizzazione dei test**

short command: ObfAut

long command: ObfuscateAutomate

Options

Options si riferisce ai parametri che possono/devono essere specificati per utilizzare la funzionalità. Per ogni parametro è previsto un flag (per specificare la option) in versione breve e un flag in versione lunga:

■ Parametro: **application**

short flag: -a

long flag: --application

■ Parametro: **bug**

short flag: -b

long flag: --bug

■ Parametro: **obf**

short flag: -o

long flag: --obf

■ Parametro: **file**

short flag: -f

long flag: --file

■ Parametro: **class**

short flag: -c

long flag: --class

■ Parametro: **screenshot**

short flag: -s

long flag: --screenshot

■ Parametro: **screenrecord**

short flag: -r

long flag: --screenrecord

Helper

Lo strumento prevede anche un helper, che può essere lanciato in uno dei seguenti modi:

- java -jar ObfTestTool.jar **-h**

- java -jar ObfTestTool.jar --help

Esempi

Sono elencati alcuni esempi di comandi correttamente impostati:

- e1 java -jar ObfTestTool.jar Obf --application:ActivityDiary -b bug1-ActivityExistingName
-o:obf1 -class MainActivityTest
- e2 java -jar ObfTestTool.jar Auto -a:ActivityDiary -b bug1-ActivityExistingName
-c:MainActivityTest -f:C:\Users\...\input.csv
- e3 java -jar ObfTestTool.jar ObfAut -a ActivityDiary -b:bug1-ActivityExistingName
-obf:obf1 -c:MainActivityTest

In Figura 4.9 viene illustrato uno schema riassuntivo di come deve essere strutturata la stringa di lancio.

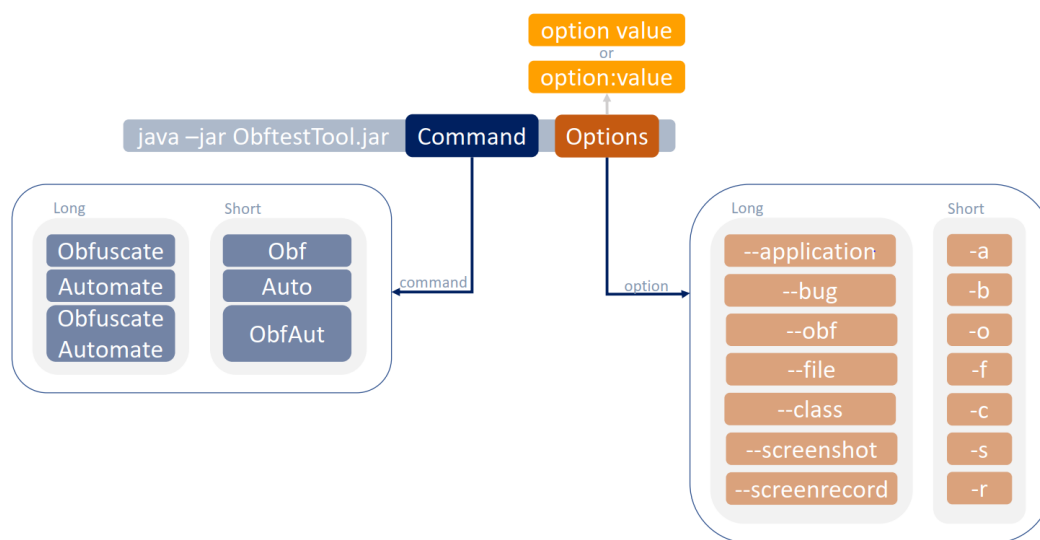


Figura 4.9: Struttura stringa di lancio del tool

4.4 Gestione dell'emulatore per l'automazione dei test

L'automazione dei test è resa possibile dall'utilizzo di un emulatore e di un gestore dei device che ne permetta un controllo completo. Per ogni esecuzione della funzionalità di automazione dei test, il tool gestisce il ciclo di vita dell'emulatore seguendo, in termini generali, i passi sottostanti:

1. nel caso ci siano emulatori in esecuzione, li chiude;
2. avvia l'emulatore specificato nel file di configurazione;
3. attende che l'emulatore sia stato completamente avviato;

4. porta l'emulatore alla home screen;
5. installa l'APK dell'applicazione (se già installata sull'emulatore, prima la disinstalla);
6. installa l'APK di test dell'applicazione (se già installata sull'emulatore, prima la disinstalla);
7. ottiene la strumentazione dell'APK di test ;
8. per ogni tupla (di parametri) del file di input:
 1. elimina i dati dell'applicazione;
 2. lancia il test passando i parametri di esecuzione (Sezione 4.5);

4.5 Passaggio dei parametri alla classe di test

Nell'automazione dei test, il tool deve essere in grado di lanciare, per ogni tupla del file di input, un test parametrico. La soluzione adottata prevede di sfruttare la possibilità di passare dei parametri alla strumentazione di test, specificandoli come argomento custom del comando adb che lancia i test sull'emulatore. La classe di test può leggere questi parametri accedendo al bundle degli argomenti della strumentazione (Appendice A - Generazione degli APK - 2.Parametrizzazione della classe di test). I parametri vengono passati nel formato JSON in modo da essere ben organizzati e facilmente leggibili dalla classe di test.

Il comando adb utilizzato per lanciare il test passando i parametri come argomento custom è il seguente:

```
adb shell am instrument -w -r -e clearPackageData true -e args [parametri] -e
debug false -e class testClassName testPackageName/testRunner
```

Nell'esecuzione del comando appena illustrato, la stringa *[parametri]* viene sostituita con i parametri in formato JSON compresi tra i singoli apici. Lo strumento, in particolare modo, crea un oggetto JSON con i parametri da passare e inserisce nel comando la stringa ottenibile dall'oggetto. Un esempio di una stringa valida è:

```
'activityName1="activity1";activityName2="activity2"'
```

4.6 Gestione del salvataggio degli screen record

Per ogni esecuzione della funzionalità di automazione dei test con parametro *screenrecord* impostato a true, il tool :

1. crea la cartella 'videos' nel path interno del device /sdcard/DCIM (se esiste una cartella con lo stesso nome, prima la elimina);
2. per ogni caso di test:
 1. inizia la registrazione dello schermo prima di lanciare il caso di test;
 2. termina la registrazione dello schermo al termine dell'esecuzione del caso di test;
 3. salva la registrazione dello schermo nella cartella creata al punto 1 (viene utilizzato un identificativo incrementale nel nome - testX.mp4);
3. effettua un pull della cartella 'videos' nella cartella di output dell'esecuzione nel package di lancio;
4. elimina la cartella 'videos' dal device.

Quindi, la registrazione dello schermo inizia prima che l'applicazione venga aperta e si conclude alla sua chiusura. Nel file di output dell'esecuzione *output.csv* viene aggiunta la colonna 'VideoPath' che contiene, per ogni test, il path della relativa registrazione dello schermo.

	A	B	C	D	E	F	G
1	activityName1	activityName2		TestResult	Info	BugReproduction	VideoPath
2	corsA	corsa		test passed		not reproduced	videos/test1.mp4
3	dentista	medico		test passed		not reproduced	videos/test2.mp4
4	camminata	camminata		crash		reproduced	videos/test3.mp4

Figura 4.10: Esempio: *output.csv*

In Figura 4.10 viene illustrato in esempio un file di output per l'esecuzione della funzionalità 'Automatizzazione dei test' con parametro *screenrecord* settato a true.

4.7 Gestione del salvataggio degli screenshot

Il salvataggio di uno screenshot per ogni esecuzione del caso di test, rispetto al salvataggio degli screen record (trattato nella Sezione 4.6), risulta molto più complicato e richiede un maggiore livello di configurazione. Il tool che gestisce il lancio dei test parametrici non è in grado di conoscere l'esatto momento in cui il bug si presenta (o non si presenta) e non è in grado quindi di effettuare uno screenshot nel momento interessante ai fini dello studio. La soluzione adottata affida la cattura della schermata alla classe di test, in quanto in grado di intervenire in un momento interno all'esecuzione del caso di test.

Comando

Lo screenshot può essere effettuato attraverso il seguente comando:

```
UiDevice device = UiDevice.getInstance(InstrumentationRegistry.getInstrumentation());
boolean taked = device.takeScreenshot(new File("/sdcard/DCIM/screenshots/screen.png"));
```

La cattura dello schermo è resa possibile dall'utilizzo della classe UiDevice offerta da UiAutomator, che è in grado di accedere alle funzionalità e alle informazioni del device su cui viene eseguita la classe di test. Risulta opportuno notare, che l'utilizzo di UiAutomator implica l'utilizzo di AndroidX nel progetto: nel caso così non fosse, è necessario effettuare la migrazione del progetto ad AndroidX (Appendice A - Migrazione del progetto ad AndroidX).

Struttura della classe di test che effettua lo screenshot

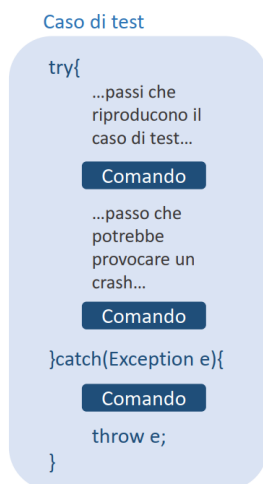


Figura 4.11: Struttura della classe che cattura la schermata

Il comando appena presentato deve essere posizionato in più punti del metodo che riproduce il caso di test, come esposto in Figura 4.11:

- **nel ramo catch di un blocco try-catch che racchiude tutta il caso di test**

In questo modo, se il caso di test sollevasse un'eccezione verrebbe effettuato immediatamente uno screenshot. Dopo il comando l'eccezione viene risolta in modo che sia catturabile dal tool.

- **prima del passo che potrebbe provocare un crash dell'applicazione**

In questo modo, se il caso di test provocasse un crash dell'applicazione senza sollevamento di un'eccezione, sarebbe comunque salvata una schermata del momento antecedente al bug.

■ al termine del caso di test

In questo modo, anche nel caso in cui il caso di test termini con successo, sarà comunque salvata una schermata del momento antecedente alla chiusura dell'applicazione.

Per ogni esecuzione della funzionalità di automazione dei test con parametro *screenshot* settato a true, il tool :

1. crea la cartella 'screenshots' nel path interno del device /sdcard/DCIM (se esiste una cartella con lo stesso nome, prima la elimina);
2. al termine dell'esecuzione di ogni caso di test, effettua un pull del file /sdcard/DCIM/screenshots/screen.png (ultimo screenshot effettuato) nella cartella di output 'screenshots' nel package di lancio (viene utilizzato un identificativo incrementale nel nome - testX.png);
3. elimina la cartella 'screenshots' dal device.

Nel file di output dell'esecuzione *output.csv* viene aggiunta la colonna 'ScreenshotPath' che contiene, per ogni test, il path della relativa immagine di cattura dello schermo.

	A	B	C	D	E	F	G	H	I	J	K
1	activityName1	activityName2		activityName1_Off*	activityName2_Off*	Technique		TestResult	Info	BugReproduction	ScreenshotPath
2	corsA	corsa		E	Nx	localSuppression		test passed		na	screenshots/test1.png
3	corsA	corsa		I	R	localSuppression		test passed		na	screenshots/test2.png
4	corsA	corsa		Op	lh	localSuppression		test passed		na	screenshots/test3.png
5	corsA	corsa		L	T	localSuppression		test passed		na	screenshots/test4.png
6	corsA	corsa		Ew	Jhpgxj	localSuppression		test passed		na	screenshots/test5.png
7	dentista	medico		D	U	localSuppression		test passed		na	screenshots/test6.png
8	dentista	medico		Cd	J	localSuppression		test passed		na	screenshots/test7.png
9	dentista	medico		C	Bs	localSuppression		test passed		na	screenshots/test8.png
10	dentista	medico		D	S	localSuppression		test passed		na	screenshots/test9.png
11	dentista	medico		Q	F	localSuppression		test passed		na	screenshots/test10.png
12	camminata	camminata		V	Y	localSuppression		test passed		na	screenshots/test11.png
13	camminata	camminata		Dr	J	localSuppression		test passed		na	screenshots/test12.png
14	camminata	camminata		D	Z	localSuppression		test passed		na	screenshots/test13.png
15	camminata	camminata		U	Bn	localSuppression		test passed		na	screenshots/test14.png
16	camminata	camminata		P	I	localSuppression		test passed		na	screenshots/test15.png

Figura 4.12: Esempio: *output.csv*

In Figura 4.12 viene illustrato in esempio un file di output per l'esecuzione della funzionalità 'Offuscamento dei dati e automatizzazione dei test' con parametro *screenshot* settato a true.

Capitolo 5

Validazione del Framework

In questo capitolo verrà brevemente discusso il processo di validazione del Framework.

5.1 Introduzione

Durante la fase di sviluppo il Framework è stato testato con applicazioni create ad hoc che permettessero di poter agilmente riprodurre specifiche casistiche, senza dover ogni volta cercare bug di applicazioni che riproducessero la situazione desiderata. Al termine dello sviluppo, per garantire che lo strumento funzionasse anche con applicazioni reali, è stata effettuata una breve validazione. I bug utilizzati per il processo sono stati selezionati tra quelli individuati dal collega D. Triolo nella sua relazione di laurea triennale. Focus della validazione sono le funzioni di: automazione dei test, cattura del risultato e verifica della riproduzione del bug. Per questo motivo non è stata posta attenzione alla funzione di offuscamento dei dati, già validata da altri colleghi.

5.2 Applicazioni

5.2.1 ActivityDiary



Introduzione

Activity Diary è un'applicazione che consente di registrare le proprie attività personalizzandole con note e foto.

(repository: <https://github.com/ramack/ActivityDiary>)

Bug

Nella release 0.4, l'inserimento di un'attività con nome già esistente nell'elenco, provoca un crash dell'applicazione.

Caso di test

Il caso di test prevede l'inserimento di due attività, i cui nomi sono stati resi parametri (*activityName1*, *activityName2*).

Riproduzione del bug

tipologa del bug atteso: crash

Se l'app termina con un crash allora il bug è riprodotto.

Validazione

activityName1	activityName2		TestResult	Info	BugReproduction
corsaA	corsa		test passed		not reproduced
dentista	medico		test passed		not reproduced
camminata	camminata		crash		reproduced
palestra	palestra		crash		reproduced
lezione	palestra		test passed		not reproduced

Figura 5.1: *output.csv*

Come visibile in Figura 5.1, il Framework considera correttamente il bug riprodotto (BugReproduction='reproduced') quando l'esecuzione del caso di test parametrico fa termi-

nare l'applicazione con un crash (`TestResult="crash"`). Nell'esempio considerato infatti, quando il caso di test prevede l'inserimento di 2 attività con lo stesso nome (caso [`activityName1="camminata"`, `activityName2="camminata"`], caso [`activityName1="palestra"`, `activityName2="palestra"`]), l'applicazione termina con un crash e il bug viene riprodotto. Quindi l'attività di validazione per questo specifico caso è andata a buon fine.

5.2.2 PrivacyFriendlyShoppingList



Introduzione

PrivacyFriendlyShoppingList è un'applicazione che consente di creare liste della spesa e gestirle aggiungendo, modificando e eliminando oggetti e liste stesse.

(repository: <https://github.com/SecUSo/privacy-friendly-shopping-list>)

Bug

Nella release 1.0.5, il click del tasto “-“ al momento della creazione dell'item, senza aver inserito un valore nel campo del numero di occorrenze del prodotto, provoca un crash dell'applicazione.

Caso di test

Il caso di test prevede l'inserimento di una nuova lista con specifica del nome (parametro *listName*) in cui viene aggiunto un nuovo item con specifica del nome (parametro *productName*). Prima di terminare l'aggiunta dell'item, senza aver inserito nessun valore nel campo del numero di occorrenze, viene effettuato un click sul tasto ‘-’.

Riproduzione del bug

tipologa del bug atteso: exception (expectedException: 'java.lang.NumberFormatException')

Se l'applicazione solleva l'eccezione 'java.lang.NumberFormatException', allora il bug è stato riprodotto.

Validazione

listName	productName	expectedException		TestResult	Info	BugReproduction
listaMD	Sale	java.lang.NumberFormatException		exception	[java.lang.NumberFormatException]	reproduced
listaConad	Olio	java.lang.NumberFormatException		exception	[java.lang.NumberFormatException]	reproduced
lista1	Pasta	java.lang.NumberFormatException		exception	[java.lang.NumberFormatException]	reproduced
lista2	Pasta	java.lang.NumberFormatException		exception	[java.lang.NumberFormatException]	reproduced

Figura 5.2: *output.csv*

Come visibile in Figura 5.2, il Framework considera correttamente il bug riprodotto (BugReproduction='reproduced') quando l'esecuzione del caso di test parametrico fa sollevare l'eccezione specificata in input nella colonna 'expectedException' (TestResult="exception", Info="java.lang.NumberFormatException"). Nell'esempio considerato, l'esecuzione del caso di test solleva sempre l'eccezione attesa indipendentemente dai parametri inseriti dall'utente (il bug non dipende dai valori in input). Quindi l'attività di validazione per questo specifico caso è andata a buon fine.

5.2.3 OpenScale



Introduzione

Open Scale è un'applicazione utile a tenere un registro delle metriche del proprio corpo. Supporta funzioni Bluetooth, e può tenere traccia di tutti i dati inseriti in un grafico. (repository: <https://github.com/oliexdev/openScale>)

Bug

Nella release 1.5, nel momento in cui si vogliono inserire un peso maggiore di 300 o un indice di massa grassa superiore a 40, oppure entrambi, l'applicazione non lo consente (pur essendo valori corretti secondo le specifiche del programma).

Caso di test

Il caso di test prevede l'inserimento di un nuovo valore per il campo Weight (parametro *weight*) e un nuovo valore per il campo Body fat percentage (parametro *bodyFatPercentage*).

Riproduzione del bug

tipologia del bug atteso: exception (expectedException: 'NoMatchingViewException')

Se l'applicazione solleva l'eccezione 'NoMatchingViewException', allora il bug è stato riprodotto.

Validazione

weight	bodyFatPercentage	expectedException		TestResult	Info	BugReproduction
320	30	NoMatchingViewException		exception	[NoMatchingViewException]	reproduced
200	56	NoMatchingViewException		exception	[NoMatchingViewException]	reproduced
312	48	NoMatchingViewException		exception	[NoMatchingViewException]	reproduced
120	30	NoMatchingViewException		test passed		not reproduced
250	35	NoMatchingViewException		test passed		not reproduced
350	20	NoMatchingViewException		exception	[NoMatchingViewException]	reproduced

Figura 5.3: *output.csv*

Come visibile in Figura 5.3, il Framework considera correttamente il bug riprodotto (BugReproduction='reproduced') quando l'esecuzione del caso di test parametrico fa sollevare l'eccezione specificata in input nella colonna 'expectedException' (TestResult="exception", Info="NoMatchingViewException"). Nell'esempio considerato infatti, quando il caso di test prevede un parametro *weight* maggiore di 300 o un parametro *bodyFatPercentage* maggiore di 40 (o entrambi), viene sollevata l'eccezione attesa e il bug viene riprodotto. Quindi l'attività di validazione per questo specifico caso è andata a buon fine.

Capitolo 6

Conclusione e sviluppi futuri

L'obiettivo dello stage può considerarsi raggiunto in quanto il framework realizzato risponde ai requisiti che sono stati individuati all'origine. In particolar modo, il framework ha permesso di sviluppare un tool da linea di comando in grado di eseguire automaticamente test parametrici su applicazioni Android utilizzando in input le tuple offuscate. La potenzialità maggiore può essere individuata però nella sua capacità di verificare se il bug atteso è stato riprodotto e quindi se l'applicativo si è comportato nello stesso modo in cui si sarebbe comportato ricevendo in input i dati originali.

Lo strumento potrebbe quindi contribuire al raggiungimento dell'obiettivo finale del progetto in cui lo stage si inserisce, cioè la valutazione dell'efficacia delle tecniche di offuscamento in ambito di riproducibilità di bug, dove con efficacia si intende la capacità delle varie tecniche di generare dati offuscati che reproducano il bug originale, cioè il difetto sollevato dall'inserimento dei dati non offuscati.

Lo strumento è sicuramente ancora lontano dall'essere perfetto e ottimale, inoltre non è stata effettuata una validazione che possa considerarsi esaustiva. A tal fine, come attività futura, sarà necessario testare il framework sviluppato durante lo stage con ulteriori applicazioni e tipologie di bug.

Il lavoro effettuato dai colleghi in precedenza è stato fondamentale alla riuscita dello stage e si spera che anche questa relazione possa aiutare nel proseguimento degli studi.

Appendice A

Operazioni utili

Variabili d'ambiente

Per il corretto funzionamento del tool devono essere configurate correttamente le variabili d'ambiente di sistema. In particolar modo, bisogna aggiungere alla variabile 'PATH' i seguenti valori:

- **Emulator** C:\Users\User\AppData\Local\Android\Sdk\Emulator
- **platform-tools** C:\Users\User\AppData\Local\Android\Sdk\platform-tools
- **build-tools** C:\Users\User\AppData\Local\Android\Sdk\build-tools\31.0.0

I percorsi potrebbe variare in base alle scelte effettuate in fase di installazione di Android Studio.

Migrazione del progetto ad AndroidX

AndroidX è un'importante miglioramento della libreria di supporto Android originale, che non viene più mantenuta. I pacchetti AndroidX sostituiscono completamente la libreria di supporto fornendo parità di funzionalità e nuove librerie.

La migrazione del progetto ad AndroidX può essere facilmente effettuata seguendo la guida ufficiale (<https://developer.android.com/jetpack/androidx/migrate>). In caso si incontrino dei problemi/errori, può essere di aiuto la risorsa 'Migrating to AndroidX: Tip, Tricks, and Guidance' (<https://medium.com/androiddevelopers/migrating-to-androidx-tip-tricks-and-guidance-88d5de238876>).

Registrazione del caso di test con Espresso Test Recorder

Espresso Test Recorder è uno strumento già integrato all'interno di Android Studio che permette la registrazione automatica dei casi di test[5]. Dopo aver caricato il progetto in AS, basterà avviare il tool cliccando sulla voce 'Record Espresso Test' nel menù 'Run'.

Lo strumento registra automaticamente (utilizzando la libreria Espresso) le operazioni effettuate sull'emulatore che sarà avviato. Espresso Test Recorder permette inoltre di aggiungere delle asserzioni. La Figura 6.1 mostra un'istantanea effettuata durante la registrazione del caso di test utilizzando lo strumento.

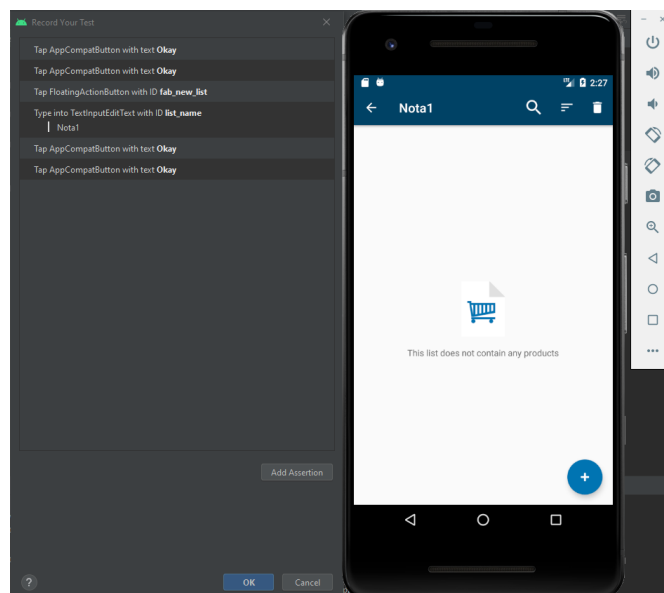


Figura 6.1: Espresso Test Recorder

Generazione degli APK in Android Studio

Android Studio offre un comando che genera automaticamente l'APK di test e l'APK dell'applicazione. Per utilizzarlo basterà scegliere nella barra del menu:

Build -> Build Bundle(s) / APK(s) -> Build APK(s)

Prima di utilizzare il comando, è necessario controllare la configurazione delle Build variant del progetto (File -> Project structure -> BuildVariants). In particolare modo, nella build variant che ci interessa (solitamente 'debug'), bisogna controllare che i campi Application ID Suffix e Version Name Suffix siano vuoti e in caso contrario cancellare il contenuto.

Configurazione degli AVD

Gli AVD possono essere configurati lanciando lo strumento 'AVD Manager' (Appendice A - AVD Manager) direttamente da Android Studio. Nell'interfaccia di avvio di Android Studio (Figura 6.2), basta cliccare sui 3 puntini in alto a destra e, nel menu a tendina, selezionare 'AVD Manager'. Dalla schermata che si apre (Figura 6.3) possono essere facilmente creati e gestiti gli emulatori. Il nome di ogni emulatore nel file di configurazione

generale deve corrispondere con il nome di uno degli AVD in questa schermata. Il tool non è in grado di gestire gli AVD creati con spazi nel nome.

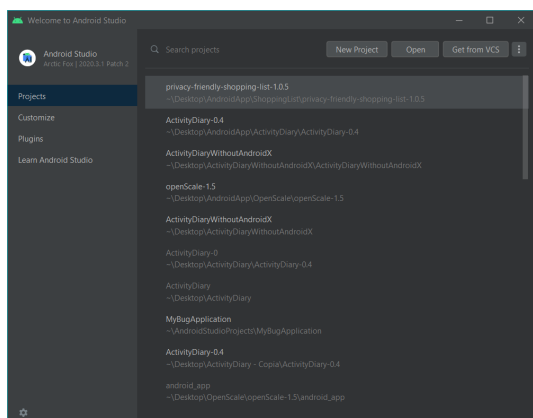


Figura 6.2: Schermata iniziale AS

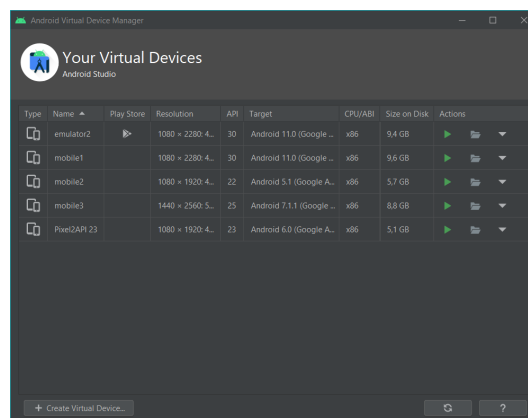


Figura 6.3: Schermata iniziale AVD Manager

Lettura degli alberi xml

In questa sezione vengono illustrate le tipologie di nodo/attributo utilizzate per la rappresentazione degli alberi xml nella relazione, in modo da garantirne una migliore comprensione. L'esposizione comprende una serie di esempi, che fanno riferimento all'albero in Figura 4.5.

- **root element**

Un nodo di tipo 'root element' può avere figli. Non possono coesistere nello stesso livello dell'albero più nodi di tipo 'root element' con lo stesso tag.

Esempio - Il nodo con tag 'Applications', in quanto di tipo 'root element', può avere figli, ma deve essere l'unico nodo con il tag 'Applications' nel livello dell'albero in cui si trova. Quindi il di configurazione dovrà avere, come figli di 'root' solo un nodo con tag 'Applications' e solo un nodo con tag 'Emulators'. Il nodo 'root' non può quindi avere, per esempio, più di un nodo figlio con tag 'Applications'.

- **element**

Un nodo di tipo 'element' può avere figli. Possono coesistere nello stesso livello dell'albero più nodi di tipo 'element' con lo stesso tag.

Esempio - Il nodo 'Applications' può avere più figli con il tag 'Application' in quanto, essendo quest'ultimo di tipo 'element', è consentita la coesistenza di nodi con lo stesso tag nel medesimo livello dell'albero. Inoltre il tag 'Application', sempre per la sua tipologia, può avere dei nodi figli, in questo caso con tag 'Bug'.

- **empty element**

Un nodo di tipo 'empty element' non può avere figli (nodo foglia). Possono coesistere

nello stesso livello dell'albero più nodi di tipo 'empty element' con lo stesso tag.

Esempio - Il nodo 'Columns' può avere più figli con il tag 'Column' in quanto, essendo quest'ultimo di tipo 'empty element', è consentita la coesistenza di nodi con lo stesso tag nel medesimo livello dell'albero. Inoltre il tag 'Column', sempre per la sua tipologia, non può avere dei nodi figli'.

- **full element**

Un nodo di tipo full element' non può avere figli (nodo foglia). Non possono coesistere nello stesso livello dell'albero più nodi di tipo 'full element' con lo stesso tag.

- **attribute**

Ogni nodo può e deve avere come attributi solo gli attributi 'figli' rappresentati nell'albero.

Esempio - Il nodo 'Bug' può e deve avere come attributi 'id', 'type', 'description'.

Generazione degli APK

La generazione dei file APK dell'applicazione (*app.apk* nella relazione) e APK di test (*test.apk* nella relazione), entrambi file di input per la funzionalità di automazione dei test, verrà affrontata nei seguenti passi:

1. Creazione della classe di test
2. Parametrizzazione della classe di test
3. Generazione dei file APK

Al termine verrà dedicato un paragrafo (Generazione degli APK senza disporre del codice sorgente) alla generazione dei file APK nel caso in cui non si disponga del codice sorgente.

1.Creazione della classe di test

Il primo step da affrontare nella generazione degli APK riguarda la creazione della classe che contiene, come unico caso di test, il caso di test in grado di riprodurre il bug. Una volta aperto il progetto dell'applicazione in Android Studio (Appendice A - Android Studio), la classe può essere:

- **scritta manualmente**

Utilizzando la libreria Espresso (Appendice A - Espresso)

- **registrata automaticamente**

Utilizzando lo strumento, già integrato nell'editor, Espresso Test Recorder (Appendice A - Espresso Test Recorder) (Appendice A - Registrazione del caso di test con Espresso Test Recorder)

In ogni caso, la classe di test creata dovrà essere posizionata nella cartella "androidTest" (cartella generato automaticamente da Android Studio per contenere gli instrumented tests) visibile nella vista 'Android' di Android Studio (Figura 6.4).

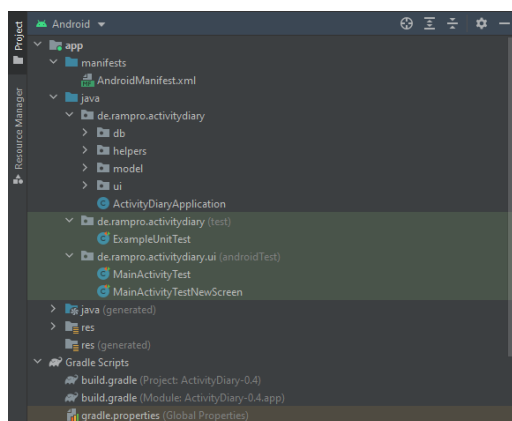


Figura 6.4: Vista 'Android' in Android Studio - progetto 1

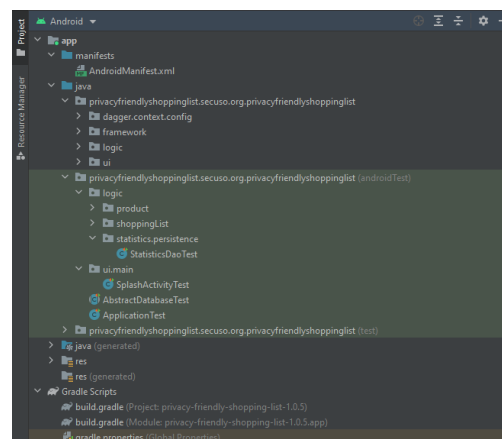


Figura 6.5: Vista 'Android' in Android Studio - progetto 2

Posizione della classe di test

Il posizionamento della classe di test all'interno della cartella 'androidTest' influisce sul valore del parametro `@class`, essenziale all'esecuzione della funzione di automazione dei test. In particolare:

■ Caso 1 (consigliato)

Nel caso in cui la classe sia posizionata nella radice (direttamente nella cartella 'androidTest'), il valore del parametro `@class` dovrà semplicemente coincidere con il nome della classe di test.

Esempio - In Figura 6.4, nel caso in cui la classe contenente il caso di test che riproduce il bug sia 'MainActivityTest', il parametro `@class` dovrà semplicemente assumere il valore 'MainActivityTest'

■ Caso 2

Nel caso in cui la classe sia posizionata in un package interno alla cartella 'androidTest', il valore del parametro `@class` dovrà rispettare il pattern `[pacchetto].[className]`.

Esempio - In Figura 6.5, nel caso in cui la classe contenente il caso di test che produca il bug sia 'StatisticsDaoTest', il parametro `@class` dovrà assumere il valore 'logic.statistics.persistence.StatisticsDaoTest'

2.Parametrizzazione della classe di test

Il secondo step da affrontare nella generazione degli APK riguarda la parametrizzazione della classe di test di cui la creazione è stata affrontata nel punto precedente. Come analizzato nella Sezione 4.5, nell'automazione dei test, per ogni lancio del test parametrico, i parametri vengono passati alla strumentazione di test in formato JSON. Dalla classe di test si può avere accesso agli argomenti del bundle della strumentazione e quindi ai parametri di esecuzione.

Per rendere la classe di test parametrica è necessario inserire, all'inizio di questa, il codice in Figura 6.6;

```
HashMap<String, String> values = new HashMap<String, String>();
@Before
public void before() { values = getHashMapByArguments(); }

private HashMap getHashMapByArguments(){
    String args = InstrumentationRegistry.getArguments().getString( key: "args");
    HashMap<String, String> map = new HashMap<String, String>();
    JSONObject jsonObject = null;
    try {
        jsonObject = new JSONObject(args);
    } catch (JSONException e) {
        e.printStackTrace();
    }
    Iterator<?> keys = jsonObject.keys();

    while(keys.hasNext() ){
        String key = (String)keys.next();
        String value = null;
        try {
            value = jsonObject.getString(key);
        } catch (JSONException e) {
            e.printStackTrace();
        }
        map.put(key, value);
    }
    return map;
}
```

Figura 6.6: Codice per la parametrizzazione delle classi di test

Il funzionamento del codice, nel dettaglio, può essere sintetizzato come segue :

1. dichiara e inizializza una hashmap (*values*);
2. prima di eseguire qualsiasi caso di test, assegna un nuovo valore alla struttura dati appena creata chiamando il metodo *getHashMapByArguments()* che:
 1. ottiene dalla strumentazione di test la stringa passata contenente i parametri in formato json;
 2. dichiara e inizializza una nuova hasmap;
 3. ricrea l'oggetto JSON dalla stringa ottenuta;
 4. per ogni coppia chiave-valore dell'oggetto, aggiunge un elemento alla hasmap appena creata (con la stessa coppia chiave-valore);
 4. ritorna l'hashmap;

A questo punto, l'hasmap *values* contiene i parametri del test come coppia chiave-valore, in cui la chiave è rappresentata dall'etichetta della colonna del file di input a cui il valore fa riferimento. Si può quindi procedere alla parametrizzazione del caso di test in quanto l'hasmap *values* ha visibilità in tutta la classe.

Quindi, nei punti in cui il caso di test prevede l'inserimento di dati in input da parte dell'utente, il valore da inserire può essere ottenuto dall'hasmap specificando come chiave la colonna del file di input che contiene il dato (*values.get("inputColumnName")*). In Figura 6.7 viene mostrato un esempio di parametrizzazione di un caso di test.

```
ViewInteraction appCompatEditText2 = onView(
    allOf(withId(R.id.edit_activity_name),
        childAtPosition(
            childAtPosition(
                withId(R.id.content_fragment),
                position: 0),
                position: 1),
        isDisplayed()));
appCompatEditText2.perform(replaceText(values.get("activityName2")), closeSoftKeyboard());
```

Figura 6.7: Esempio: parametrizzazione della classe di test

3. Generazione dei file APK

L'ultimo step riguarda la vera e propria generazione dei file *app.apk* e *test.apk*. Android Studio è in grado di generare i file richiesti automaticamente grazie all'integrazione di Gradle (Appendice A - Generazione degli APK in Android Studio) (Appendice A - Gradle). Al termine della generazione dei file, nell'Event Log di AS viene comunicata la directory in cui questi sono stati creati. Solitamente il file che nella relazione è chiamato *app.apk* viene generato come *app-debug.apk*, mentre il file che nella relazione è chiamato *test.apk* viene generato come *app-debug-androidTest.apk*.

Generazione degli APK senza disporre del codice sorgente

Nei paragrafi precedenti è stata trattata la generazione degli APK supponendo di essere sempre in possesso del codice sorgente. In realtà, nella maggior parte dei casi, le applicazioni non sono open source e risulterebbe quindi utile poter utilizzare lo strumento anche nel caso in cui si disponga solamente dell'APK. Il collega A. Mendieta, nella sua relazione di laurea triennale, ha trovato un metodo per la generazione dell'APK di test anche nel caso in cui non si possieda il codice sorgente (ma solo l'APK dell'applicazione), che ancora oggi è risultato utilizzabile.

Brevemente, la soluzione individuata dal collega, utilizza un progetto open source (Ap-

pendice A - AndroidTestWithOutSource Project) che permette di generare solamente l'APK di test contenente la classe parametrizzata. In questo caso, la classe di test non può essere registrata con Espresso Test Recorder, ma deve essere scritta manualmente sfruttando le funzionalità offerte dallo strumento Ui Automator Viever (Appendice A - UI Automator Viewer).

Prima di poter utilizzare l'APK dell'applicazione e l'APK di test generata come file di input per il tool, risulta essenziale firmare gli APK con la stessa firma, operazioni facilmente effettuabile grazie allo strumento apksigner (Appendice A - apksigner).

Strumenti utilizzati

IntelliJ IDEA

IntelliJ IDEA è un ambiente di sviluppo integrato (IDE) per il linguaggio di programmazione Java sviluppato da JetBrains [6]. L'IDE è stato utilizzato per lo sviluppo del tool.

Android Studio

Android Studio è l'ambiente di sviluppo integrato (IDE) ufficiale per il sistema operativo Android di Google, basato sul software IntelliJ IDEA di JetBrains e progettato specificamente per lo sviluppo Android[7]. Lo strumento è stato utilizzato principalmente per generare l'APK dell'applicazione e l'APK di test contenente la classe di test parametrica.

adb

Android Debug Bridge (adb) è uno strumento versatile lanciabile da riga di comando che consente di comunicare con un dispositivo. Il comando adb facilita una serie di azioni del dispositivo, come l'installazione e il debug di app, e fornisce l'accesso a una shell Unix che può essere utilizzata per eseguire una varietà di comandi su un dispositivo. Il tool è incluso nell'sdk di Android Studio[8]. Lo strumento è stato utilizzato come gestore dei device, in particolare per il controllo del processo di lancio dei test sull'emulatore.

Emulator

Emulator è l'emulatore di dispositivi Android incluso nell'sdk di Android Studio[9]. Lo strumento è stato utilizzato per lanciare automaticamente l'emulatore nel processo di automazione dei test. Questo è stato possibile in quanto il tool include un'interfaccia da linea di comando.

AAPT2

AAPT2 (Android Asset Packaging Tool) è uno strumento di compilazione che Android Studio e Android Gradle Plugin utilizzano per compilare e impacchettare le risorse di un'applicazione[10]. Il tool è incluso nell'sdk di Android Studio. Lo strumento è stato utilizzato per ottenere alcune informazioni (necessarie al lancio dei test) direttamente dal file APK.

AVD Manager

AVD Manager è un'interfaccia che può essere avviata da Android Studio e permetta la creazione e la gestione degli AVD. Un Android Virtual Device (AVD) è una configurazione che definisce le caratteristiche di un telefono Android, tablet, sistema operativo Wear, Android TV o dispositivo con sistema operativo automobilistico che si desidera simulare nell'emulatore Android[11]. L'interfaccia è stata utilizzata per la creazione e la gestione degli AVD.

apksigner

Lo strumento apksigner, disponibile nella revisione 24.0.3 e successive di Android SDK Build Tools, consente di firmare gli APK e di confermare che la firma di un APK verrà verificata correttamente su tutte le versioni della piattaforma Android supportate da tali APK[12]. Lo strumento è stato utilizzato per firmare gli APK.

Espresso

È un framework per l'automazione dei test Android. È stato sviluppato da Google e reso disponibile agli sviluppatori e ai tester per effettuare test accurati sulle interfacce utenti delle loro applicazioni[13]. Lo strumento è stato utilizzato per la creazione dei casi di test che dovrebbero riprodurre il bug.

Gradle

Gradle è uno strumento di automazione della build per lo sviluppo di software multi linguaggio. Controlla il processo di sviluppo nelle attività di compilazione e confezionamento fino a test, distribuzione e pubblicazione[14]. Lo strumento è stato utilizzato per la generazione dei file APK.

AndroidTestWithoutSource Project

AndroidTestWithoutSource è un progetto dal quale il collega A. Mendieta ha preso spunto per trovare una soluzione che permetta di effettuare test di applicazioni Android nel caso in cui non si possieda il codice sorgente. La versione originale è presente al seguente indirizzo:

<https://github.com/cmoaciopm/AndroidTestWithoutSource>.

UI Automator Viewer

UI Automator Viewer è uno strumento incluso nell'sdk di Android Studio che fornisce una comoda GUI per scansionare e analizzare i componenti dell'interfaccia utente visualizzati su un dispositivo Android. È possibile utilizzare questo strumento per ispezionare la gerarchia di layout e visualizzare le proprietà dei componenti dell'interfaccia utente visibili in primo piano del dispositivo[15]. Queste informazioni consentono di creare test più dettagliati.

Sitografia

[1] Privacy, che cos'è il Gdpr e perchè ci riguarda

url: <https://www.ilsole24ore.com/art/privacy-che-cos-e-gdpr-e-perche-ci-riguarda-AEYGnchE>

ultimo accesso: 21/10/2021

[2] Protezione dei dati e dalla privacy online

url: https://europa.eu/youreurope/citizens/consumers/internet-telecoms/data-protection-online-privacy/index_it.htm

ultimo accesso: 21/10/2021

[3] Sanzioni privacy, cosa prevede il GDPR

url: <https://www.informazionefiscale.it/sanzioni-privacy-gdpr-cosa-prevede-violazioni-codice-privacy>

ultimo accesso: 21/10/2021

[4] Data masking: cos'è e perché è utile a evitare sanzioni GDPR

url: <https://www.cybersecurity360.it/soluzioni-aziendali/data-masking-cose-e-perche-e-utile-a-evitare-sanzioni-e-salvaguardare-la-reputazione-aziendale/>

ultimo accesso: 21/10/2021

[5] Create UI tests with Espresso Test Recorder

url: <https://developer.android.com/studio/test/espresso-test-recorder>

ultimo accesso: 21/10/2021

[6] IntelliJ IDEA

url: <https://www.jetbrains.com/idea/>

ultimo accesso: 21/10/2021

[7] Android Debug Bridge (adb)

url: <https://developer.android.com/studio/command-line/adb>

ultimo accesso: 21/10/2021

[8] Meet Android Studio

url: <https://developer.android.com/studio/intro>

ultimo accesso: 21/10/2021

[9] Start the emulator from the command line

url: <https://developer.android.com/studio/run/emulator-commandline>

ultimo accesso: 21/10/2021

[10] AAPT2

url: <https://developer.android.com/studio/command-line/aapt2>

ultimo accesso: 21/10/2021

[11] avdmanager

url: <https://developer.android.com/studio/command-line/avdmanager>

ultimo accesso: 21/10/2021

[12] apksigner

url: <https://developer.android.com/studio/command-line/apksigner>

ultimo accesso: 21/10/2021

[13] Espresso

url: <https://developer.android.com/training/testing/espresso>

ultimo accesso: 21/10/2021

[14] Gradle

url: <https://gradle.org/>

ultimo accesso: 21/10/2021

[15] UI Automator

url: <https://developer.android.com/training/testing/ui-automator>

ultimo accesso: 21/10/2021

[16] iOS vs. Android: The Bug Wars

url: <https://www.globalapptesting.com/blog/bug-wars-gold-digger-edition>

ultimo accesso: 21/10/2021