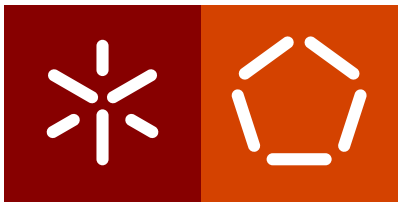
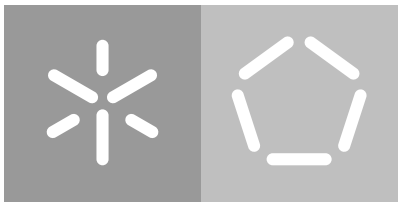


Universidade do Minho
Escola de Engenharia
Departamento de Informática

Pedro Miguel Braga do Vale

Teste baseado em modelos de aplicações Android

October 2018



Universidade do Minho
Escola de Engenharia
Departamento de Informática

Pedro Miguel Braga do Vale

Teste baseado em modelos de aplicações Android

Master dissertation
Master Degree in Computer Science

Dissertation supervised by
José Creissac Campos

October 2018

AGRADECIMENTOS

Gostaria de agradecer, em especial, ao meu orientador, Professor José Creissac Campos, pela oportunidade, apoio e sugestões de desenvolvimento desta dissertação.

Aos meus pais, ao meu avô, à minha namorada, ao meu irmão, aos meus amigos e todos os familiares um obrigado pelo incentivo e apoio incondicional dado.

A todos aqueles que estiverem presentes no decorrer da minha formação, o meu obrigado.

ABSTRACT

With the evolution of smartphones and the growing number of its users, mobile applications are regularly used by more and more people. Because of this growing use of mobile applications, it is critical to ensure their quality. The graphical user interface (GUI) is a very relevant component in these applications, since it enables the user to interact with the available resources. A malfunctioning of it may make it impossible for the application to work properly and, consequently, for the software system to be invalidated. One way to ensure a smooth operation of the system is by performing software tests. Model-based testing (MBT) is a black-box technique that checks whether software has the expected behavior. The MBT focuses on generating and running tests from a system under test (SUT) model.

This dissertation continues the development of an MBT tool called TOM. After validating the Web components of said tool, we have now developed a component of generation and execution of test cases for Android applications. In the course of the dissertation we showcase the various decisions and changes made in the TOM tool during the implementation of this new component, presenting at the end two case studies to prove the operation of the Android component.

RESUMO

Com a evolução dos smartphones e o crescente número dos seus utilizadores, as aplicações móveis são utilizadas regularmente por cada vez mais pessoas. Devido a este crescente uso de aplicações móveis, é fundamental garantir a sua qualidade. A interface gráfica do utilizador (GUI) é uma componente muito relevante nestas aplicações, pois possibilita a interação do utilizador com os recursos disponíveis. Um mau funcionamento da mesma pode impossibilitar o bom funcionamento da aplicação e consequentemente, do sistema de software. Uma forma de garantir um bom funcionamento do sistema é através da realização de testes de software. Os testes baseados em modelos (MBT) são uma técnica *black-box* que verifica se um software tem o comportamento esperado. O MBT foca-se na geração e execução de testes a partir de um modelo do sistema sob teste (SUT).

Esta dissertação continua o desenvolvimento de uma ferramenta de MBT denominada TOM. Após o sucesso apresentado na parte Web por esta ferramenta, desenvolveu-se agora uma componente de geração e execução de casos de teste para aplicações Android. No decorrer da dissertação, encontram-se as varias decisões tomadas e alterações efetuadas na ferramenta TOM durante a implementação desta nova componente, sendo apresentado no final dois casos de estudo para comprovar o funcionamento da componente Android.

CONTEÚDO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE LISTAGEM

ACRÓNIMOS

A

API Application Programming Interface.
APK Android Package.

B

BFS Breadth-First Search.

C

CSS Cascading Style Sheets.

D

DFS Depth-First Search.

F

FSM Finite State Machine.

G

GUI Graphical User Interface.

H

HTML Hypertext Markup Language.
HTTP Hypertext Transfer Protocol.

I

IRIT IRIT Institut de Recherche en Informatique de Toulouse.

J

JSON JavaScript Object Notation.

M

MBT Model-Based Testing.

Q

QoS Quality of Service.

S

SCXML State Chart XML.
SDK Software Development Kit.
SUT System Under Test.

U

UI User Interface.

X

XML eXtensible Markup Language.

1

INTRODUÇÃO

O primeiro *smartphone* foi lançado para o mercado em 1994, criado pela IBM e denominado por Simon[1] (Lewis, 1996; IBM, 1994). Passado cerca de 10 anos começaram a surgir novos modelos, criados por exemplo pela Nokia e a Erickson. Em 2007 a Apple lança o seu primeiro iPhone com o sistema iOS para o mercado (Apple, 2007), tendo conquistado milhões de utilizadores, mas passado um ano, em 2008, a Google lança o sistema Android (Blog, 2008) para o mercado. Embora existam outros sistemas disponíveis (Windowns, BlackBerry, etc) os sistemas operativos Android e iOS afirmaram-se como os mais relevantes no mercado, somando, em conjunto, uma cota de mercado superior a 99% (Vincent, 2017). O sistema Android domina neste momento o mercado dos smartphones, existindo alguns pontos que levam a que seja o preferido dos utilizadores. O ponto mais forte é a Google Play Store que disponibiliza inúmeras aplicações de que os utilizadores podem usufruir de forma maioritariamente gratuita, não contendo grande impedimento para as empresas criarem as suas aplicações e as disponibilizarem aos utilizadores, ao contrario do que acontece na App Store do iOS. Outro ponto forte é o facto do Android ser usado por muitos dos maiores fabricantes de smartphones (como a Samsung, Huawei, LG, etc). Mas nem tudo é positivo no uso de smartphones, em termos de bateria são muito ineficientes, com enumeros processos a correr no *background* do sistema acabando por gastar muita energia. Algumas aplicações disponibilizadas para os utilizadores podem conter software malicioso. A Google nos últimos anos tem vindo a combater este problema. Uma das soluções implementadas é o *Google Play Protect* que permite verificar se todas as aplicações instaladas no dispositivo sao seguras ou não (Brown, 2018) .

Com a evolução dos smartphones e o crescente número de utilizadores, as aplicações móveis são utilizadas regularmente por cada vez mais pessoas, tanto para trabalho como para lazer. Devido ao seu crescente uso, é fundamental garantir a qualidade das aplicações desenvolvidas, pois os utilizadores esperam um bom desempenho, confiabilidade, segurança e usabilidade. Este último aspeto será o foco principal desta dissertação. A interface gráfica do utilizador (*Graphical User Interface* (GUI)) é uma componente muito relevante da

[1] https://www.microsoft.com/buxtoncollection/detail.aspx?id=40 Accessed: 2017-11-14

aplicação, pois possibilita a interação do utilizador com os recursos disponíveis. É crucial que uma aplicação não apresente pontos de falha que possam levar a erros do sistema, prejudicando o bem estar do utilizador.

Neste momento são lançadas, por ano, inúmeras aplicações que não estão devidamente construídas. Não são muitas vezes realizados os testes de software necessários para garantir a qualidade do produto, uma vez que este processo é normalmente caro e demorado.

Os testes de software devem ser considerados para a validação de uma aplicação móvel, sendo a GUI uma das principais preocupações. Um mau funcionamento da GUI pode levar ao mau funcionamento da aplicação e do sistema. A forma mais tradicional de testar a GUI é através de um processo manual, recorrendo a utilizadores reais para testar a aplicação móvel de forma a descobrir erros. O uso destes utilizadores para testar a aplicação leva a que o custo deste processo manual seja elevado. Leva também a que a aplicação não seja bem testada, já que num processo de testes manual podem não se conseguir cobrir todos os casos de testes que deveriam ser realizados. O uso de um processo de teste automático leva a que os teste realizados na interface atinjam uma maior cobertura da mesma. Este processo automático pode ser realizado através de enumeras técnicas, mas a utilizada nesta dissertação é o teste baseado em modelos (*Model-Based Testing* (MBT)). Este tipo de testes consiste na utilização de um modelo abstrato como base (oráculo). Através deste modelo são gerados inúmeros casos de testes que serão posteriormente executados na aplicação. Depois destes serem executados é comparado o seu resultado com a previsão do oráculo (Rodrigues, 2015; Campos et al., 2017).

Na Universidade do Minho foi desenvolvida uma ferramenta chamada TOM, que permite a automatização da realização de testes baseados em modelos para aplicações Web (Campos et al., 2017). Esta ferramenta possui uma arquitetura modular, sendo então possível que cada um dos seus módulos possa operar independentemente. Esta também já tem a capacidade de gerar testes em três tipos de formatos: Web (Pinto, 2017), cenários para a ferramenta Circus do IRIT (Fayollas et al., 2014; Campos et al., 2017) e cenários para a ferramenta PVSio-Web (Masci et al., 2015; Gonçalves, 2017).

1.1 Motivação e Objetivos

Com a constante utilização das aplicações móveis no nosso dia a dia é importante que estas forneçam um bom funcionamento da GUI. O que acontece em algumas aplicações disponíveis para o utilizador é o facto destas terem uma interface de baixa qualidade. Existem causas que levam a este mal funcionamento: uma delas é o facto das aplicações não terem

sido testadas, outra é terem sido mal testadas, isto é, os testes efetuados na aplicação não foram suficientes ou os indicados para encontrar erros na GUI.

Com esta dissertação pretende-se desenvolver uma componente de teste de aplicações Android para a ferramenta TOM, aumentando a qualidade das aplicações móveis mas também o conjunto de cenários para o qual a ferramenta pode ser utilizada.

1.2 ESTRUTURA DO DOCUMENTO

O documento está estruturado da seguinte forma:

- Capítulo 1 - Introdução - Apresenta uma descrição do projeto e o problema que se pretende resolver.
- Capítulo 2 - Testes de software em aplicações móveis - Apresenta uma introdução ao tema de testes de software em aplicações Android para contextualizar o projeto, assim como a comparação de ferramentas para a geração dos testes.
- Capítulo 3 - TOM - Apresenta as funcionalidades e características da ferramenta.
- Capítulo 4 - Geração de testes para Android - Descreve todo o processo de geração Android aplicado.
- Capítulo 5 - Processo de criação dos ficheiros para Android - Descreve todo o processo de criação manual dos modelos para as aplicações Android.
- Capítulo 6 - Caso de estudo - Descreve dois casos de estudo utilizados para verificar o funcionamento da componente Android.
- Capítulo 7 - Conclusão e Trabalho Futuro - Descreve o trabalho realizado e o trabalho futuro.

2

TESTES DE SOFTWARE EM APLICAÇÕES MÓVEIS

Uma abordagem sistemática aos testes de software visa maximizar a deteção de falhas, tornando os resultados reprodutíveis e reduzindo a influência de fatores externos (Muccini et al., 2012). Logo, um teste de software consiste em utilizar um sistema de software por forma a perceber a qualidade do mesmo, isto é, avaliar aspetos pretendidos (como a usabilidade, eficiência, segurança, etc) na aplicação de forma a perceber se existe algum erro ou falha na mesma. Um teste de software é considerado um sucesso quando este encontra um erro (Ammann and Offutt, 2008). É importante perceber que um processo de teste consegue provar a existência de erros, mas não a sua ausência. Torna-se necessário obter o maior número de comportamentos possíveis (caminhos) para conseguir uma maior cobertura sobre o sistema sob teste (*System Under Test* (SUT)), com isto obtemos um grande conjunto de testes podendo assim encontrar o máximo número de problemas na aplicação.

Uma aplicação móvel é um software que corre em dispositivos móveis. O surgimento destes dispositivos levou ao aparecimento de um novo tipo aplicação com características diferenciadoras. Os testes realizados nas aplicações móveis apresentam algumas diferenças que devem ser consideradas comparativamente aos testes de aplicações tradicionais (desktop e web). A existência de sensores que influenciam o comportamento do dispositivo ou da aplicação, os recursos limitados comparativamente aos computadores tradicionais (laptop/desktop), faz com que os testes tenham que considerar um conjunto de fatores que não são relevantes no teste de aplicações tradicionais. Com este pequeno conjunto de aspetos percebemos que existem algumas diferenças e cuidados que devem ser tidos em conta no teste das aplicações móveis.

2.1 TESTES EM APLICAÇÕES MÓVEIS

Existem diferentes tipos de testes que podem ser realizados numa aplicação móvel. De acordo com Muccini et al. (2012) e Kirubakaran and Karthikeyani (2013) estes são:

- **Testes da GUI:** Estes testes focam-se na interação do utilizador com o sistema. É essencial realizar estes testes pois são o único meio pelo qual o utilizador usufrui do

sistema, como tal é necessário que não exista qualquer falha da GUI que leve ao mau funcionamento do sistema.

- **Testes de Segurança:** Os teste de segurança nas aplicações são de extrema importância. Com o acesso constante das aplicações e mesmo do smartphone à Internet é necessário guardar os dados dos utilizadores de forma segura, de maneira a que os utilizadores não tenham a sua informação vulnerável a um ataque.
- **Testes de Desempenho e Confiabilidade:** Servem para testar o desempenho das aplicações sob determinadas condições, depende muito dos recursos e das características dos dispositivos.
- **Testes de memoria e energia:** Devido à energia e recursos limitados que um dispositivo móvel tem, é essencial realizar testes na aplicação por forma a medir a quantidade de recursos e energia utilizados.
- **Testes em vários dispositivos:** Devido ao número de versões de dispositivos lançados para o mercado (principalmente de Android), onde diferentes dispositivos móveis fornecem características e componentes de hardware diferentes, é essencial testar as versões mais utilizadas no mercado, de forma a que a aplicação seja utilizável pelo máximo número possível de dispositivos.

Esta dissertação foca-se nos testes de GUI, na interação do utilizador com o sistema. Estes testes ajudam a minimizar o número de erros na interface de utilizador (*User Interface* (UI)), melhorando o funcionamento do sistema.

2.2 Principais Infraestruturas dos testes móveis

Para a realização dos testes é necessário possuir uma infraestrutura onde os mesmos possam ser executados na aplicação, por forma a obter o seu resultado. Existem diferentes tipos de infraestrutura, cada uma com as suas vantagens e desvantagens, que devem ter sidas em conta quando realizamos testes de software. Dependendo da qualidade de serviço (*Quality of Service* (QoS)) pretendida (desempenho de software, a confiabilidade, a disponibilidade, a escalabilidade e o orçamento disponível) pode-se ter de optar por uma determinada infraestrutura, pois acabam por ser os fatores mais importantes na execução dos testes na aplicação.

Segundo Gao et al. (2014) existem quatro abordagens principais ao teste de aplicações móveis, todas baseadas em arquiteturas cliente servidor:

- **Testes baseados em emulação:** É criada uma maquina virtual com uma versão do dispositivo a ser testado, basicamente um emulador onde os testes serão executados.

Uma grande vantagem destes testes é o facto de serem económicos de realizar, porque apenas é necessário um computador pessoal. Por outro lado, ao utilizarmos um emulador temos recursos mais limitados logo a qualidade de serviço é menor. A maioria dos emuladores tem o suporte de gestos muito limitado o que prejudica os testes na aplicação.

- **Testes baseados em dispositivos reais:** Estes testes usam dispositivos móveis reais onde são executados os testes. Ao contraio dos emuladores, os dispositivos reais oferecem uma maior QoS, permitindo verificar comportamentos e parâmetros de serviço que o emulador não disponibiliza. Em contrapartida estes dispositivos são muito mais caros, logo em caso de testes em grande escala o custo poderá ser proibitivo.
- **Testes na nuvem:** Os testes na nuvem recorrem a dispositivos ou emuladores disponibilizados por terceiros por forma a instalarem aplicações para teste. As aplicações são instalados e os testes executados nos dispositivos ou emuladores que estão acessíveis na nuvem. Esta opção é ideal para quando queremos testar a aplicação em grande-escala, pois é economicamente mais rentável.
- **Testes *Crowd-based*:** Os testes *Crowd-based* usa pessoas reais para testar a aplicação. Esta aplicação irá ser testado de forma diferente por cada pessoa. A grande vantagem de usar esta abordagem é o facto de não ter despesas associadas, mas em contra partida não é garantida a qualidade dos testes.

2.3 ANDROID

2.3.1 *Programação de aplicações Android*

As aplicações Android podem ser escritas em várias linguagens sendo as mais utilizadas o Java (Herbert Schildt, 2017), Kotlin[1] e C++ (Bjarne Stroustrup, 2008). Normalmente em Java é usado o kit de desenvolvimento de software (*Software Development Kit* (SDK))[2] do Android, mas existem outros ambientes de desenvolvimento disponíveis que podem ser usados.

No desenvolvimento de aplicações Android existem componentes que são essenciais (Project, 2018b):

- **Atividades**: As atividades representam o ecrã da aplicação com o qual o utilizador vai interagir. Uma atividade utiliza vários *layouts*, podendo este conter diferentes configurações consoante o tamanho do ecrã do dispositivo, e normalmente são programados

[1] https://kotlinlang.org/docs/kotlin-docs.pdf last accessed: 2018-08-25.
[2] https://developer.android.com/studio/ last accessed: 2017-12-15.

em XML. Os Widgets que compõem o ecrã fornecem elementos visuais, como botões, campos de texto, etc. Estes Widgets são introduzidos num layout formando o conteúdo que o utilizador irá ver. As atividades tem um ciclo de vida representado na Figura 1, passando por diferentes estados em resposta à invocações efetuados pelo sistema através dos diferentes métodos disponíveis:

  - **onCreate():** é o método usado inicialmente para criar a atividade. Inicializa todos os componentes essenciais da atividade (Project, 2018a) passando para o estado *Created*.
  - **onStart():** Após a invocação deste método a atividade fica visível e passa para o estado *Visible*.
  - **onResume():** Quando este método é invocado a atividade fica em primeiro plano no ecrã e tem o foco do utilizador passando para o estado *Active* (Project, 2018a).
  - **onPause():** É invocado pelo sistema quando o utilizador sai da atividade, passanda para o estado *Paused*. Após este método a aplicação tem duas possibilidades de comportamento, caso o utilizador volte à aplicação a mesma irá para o estado *Active* em que estava, caso não volte a atividade vai acabar por ser destruída.
  - **onStop():** O método é invocado quando uma atividade não esta mais visível para o utilizador passando para o estado *Stopped*.
  - **onDestroy():** Depois de invocado a atividade é completamente destruída passando por fim ao estado *Destroy*.

  É muito importante conhecer o ciclo de vida de uma atividade, porque ela ajuda a perceber como não perder informação durante as etapas pelas quais a atividade passa, impedindo que a aplicação possa falhar quando o utilizador sai e entra da aplicação. Uma má implementação deste métodos pode levar a perda de informação.

- **Serviços:** Serviços são componentes executados em segundo plano, as suas tarefas não necessitam de interação com o utilizador, exemplo disso é monitorização do alarme.

- **Provedores de conteúdo:** Estes componentes armazenam e fornecem acesso a dados estruturados armazenados no sistema de arquivos, base de dados, etc.

- ***Broadcast receivers*:** são componentes que permitem às aplicações responder a anúncios de eventos efetuados ao nível do dispositivo. Por exemplo, uma transmissão que anuncia que o ecrã foi desligado ou que a bateria está baixa.

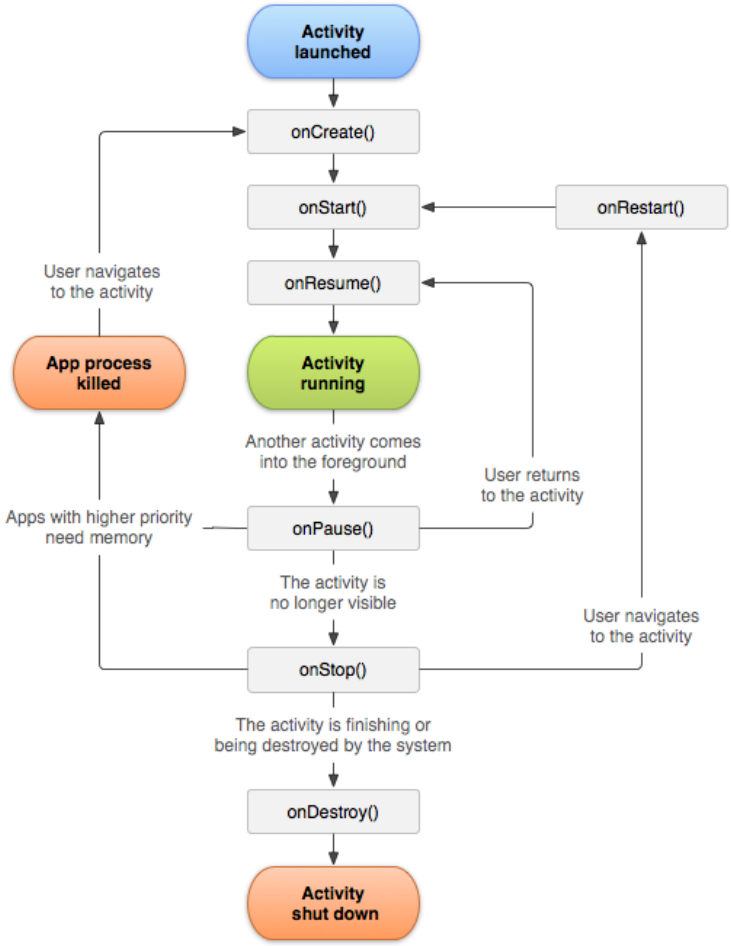


Figura 1.: Ciclo de vida de uma atividade
(Project, 2018a)

2.3.2 *Desafios de testes em aplicações Android*

Como descrito no inicio deste capítulo, os testes em dispositivos móveis apresentam alguns desafios adicionais em relação aos testes tradicionais. No caso dos dispositivos Android segundo Deng et al. (2017) existem sete desafios:

- **Componentes de Android com ciclos de vida únicos:** Como descrito acima todos os principais componentes de aplicações Android devem comportar-se de acordo com um pré-determinado ciclo de vida. Por exemplo, ao termos duas aplicações a executar ao mesmo tempo, caso exista troca de uma para a outra o funcionamento das mesmas deve continuar no ponto em que estávamos quando saímos.

- **Uso intensivo de ficheiros *eXtensible Markup Language* (XML):** Os ficheiros de XML utilizados pelo Android são usados para configuração, construção da interface do utilizador, etc. A maioria dos testes realizados apenas cobre a linguagem utilizada no desenvolvimento da aplicação, deixando de parte os ficheiro XML que podem vir a ter erros inesperados.

- **Características *context-aware:*** As aplicações Android são *context-aware*, isto é, a aplicação sabe o ambiente em que esta inserida e adapta-se a ele. Estas aplicações normalmente recebem como *input* informações do hardware do dispositivo ou do utilizador. Deng et al. (2017) sugere incluir informações vindas do harware nos testes.

- **Dois tipos de orientação de ecrã:** Praticamente todos os dispositivos móveis tem dois tipos de orientação: vertical ou horizontal. Diversas aplicações falham ou apresentam erros quando mudamos a orientação do dispositivo.

- **Programas e botões do sistema baseados em eventos:** As aplicações Android são baseadas em eventos, sendo estas dependentes do *input* utilizador, por exemplo, quando clica ou arrasta um determinado botão. É importante que todos os botões e eventos da aplicação sejam testados para não apresentar falhas indesejadas para o utilizador.

- **Várias conexões de rede:** Algumas aplicações dependem de rede para conseguirem ter o funcionamento correto, mas por vezes uma simples alternância de uma rede wi-fi para os dados móveis faz com que a aplicação falhe.

- **Vida da bateria limitada**: Algumas aplicações consumem muita energia, porque não foram programadas adequadamente, esta dificuldade não representa nenhuma falha da aplicação, mas prejudica o utilizador, por isso deve ser testada a energia que a aplicação gasta de modo a não prejudicar a bateria do dispositivo.

Estamos interessados especialmente em problemas relacionados com a GUI, logo nos eventos da interface e no XML. Vai ser abordado também os testes na rotação do ecrã e nas varias conexões a rede para perceber como o SUT reage a este tipo de testes.

2.4 Testes Baseados em Modelos

Testes baseados em modelos são uma técnica *black-box* que verifica se um software tem o comportamento esperado. O MBT foca-se na geração automática de testes (normalmente funcionais) a partir de um modelo abstrato (denominado por oráculo). Esse modelo é criado com base nos requisitos/especificações do sistema, garantido que os sistemas implementados estejam completos e se comportem como esperado. O modelo deve ser mais abstrato do que o SUT, caso contrario os esforços de validação do modelo são exatamente

os mesmos de validação do SUT (Pretschner et al., 2017). Esta técnica providencia um processo de automatização da geração e execução dos testes nas GUI, diminuindo o custo da criação de testes manualmente.

De uma forma geral, esta técnica consiste na criação de um modelo abstrato do comportamento do SUT e na utilização deste modelo para a geração de um grande número de casos de testes. Estes testes serão então executados no SUT e os resultados comparados com o oráculo criado.

Existem duas maneiras de geração e execução dos testes quando utilizamos MBT (Veanes et al., 2008) :

- **Offline**: Na geração offline os testes são primeiro gerados pela ferramenta de teste baseada em modelo e só depois de todos os testes estarem gerados são executados no SUT.
- **Online**: Na geração online a ferramenta de teste baseada em modelo conecta-se diretamente ao SUT e testa dinamicamente.

Uma das vantagens de MBT, para além do custo e esforço diminuídos na criação do modelo, quando comparado com a criação manual dos casos de teste, é a fácil manutenção dos modelos abstratos (Schieferdecker, 2012). Enquanto num modelo apenas temos de alterar um ou dois comportamentos, no código criado manualmente teríamos de alterar inúmeros casos de teste.

2.4.1 *Processo dos Testes Baseados em Modelos*

Através da Figura 2 conseguimos perceber que o processo de MBT começa com a construção de um modelo abstrato do SUT, que deverá ser validado antes de passar para a geração dos testes. Antes de efetuar a geração deve também ser criado/modificado o critério de cobertura, que pode ser definido em relação a uma determinada funcionalidade do sistema, à estrutura do modelo de teste, etc. (Mark Utting et al., 2012).

O passo seguinte é a geração dos testes a partir do modelo abstrato, sendo o número de testes gerados determinado pelo critério de cobertura definido. Os testes gerados são abstratos contendo apenas sequências de passos definidos em termos do modelo. Desta forma, podem ser depois criados testes concretos em diferentes linguagens, tornando assim o sistema de geração genérico. Depois de criados, os testes concretos são executados e comparados com o oráculo para saber se o resultado obtido é igual ao resultado esperado. Estes resultados podem no fim ser analisados pelo engenheiro de testes.

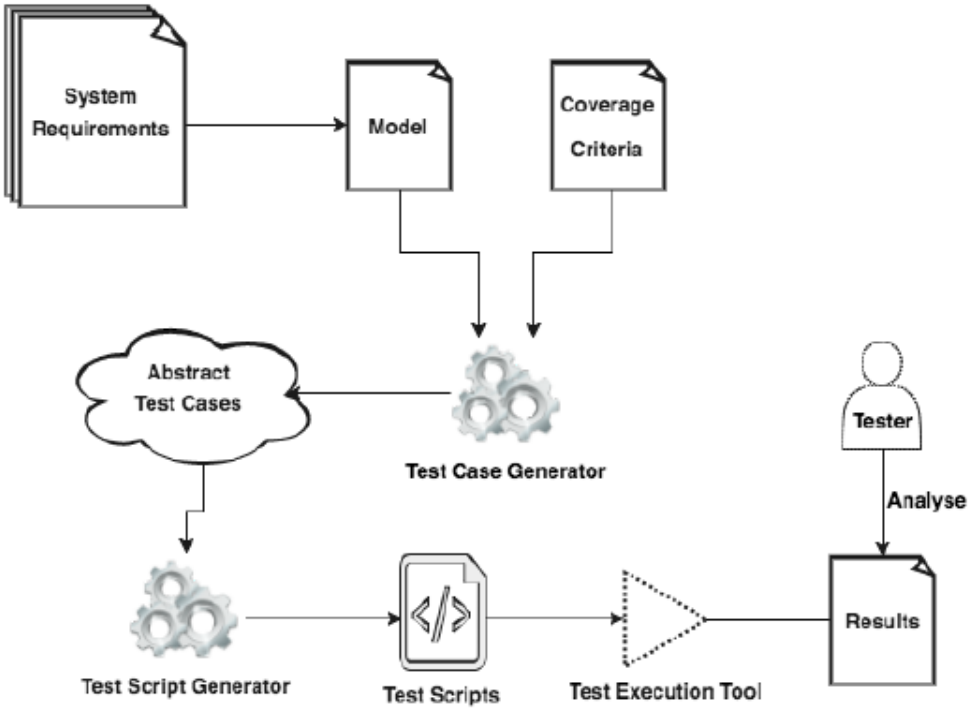


Figura 2.: Processo dos Testes Baseados em Modelos (Gonçalves, 2017)

2.5 Testes baseados em modelos em interfaces de utilizador

Como esta dissertação se foca no MBT em aplicações Android, nesta secção serão abordadas algumas ferramentas existentes que utilizem esta técnica na geração e execução de casos de teste para aplicações móveis. Isto permite-nos analisar as varias estratégias utilizadas por cada ferramenta.

2.5.1 *MobiGuitar*

Um dos projetos desenvolvidos com MBT em GUIs é o MobiGuitar desenvolvido por Amalfitano et al. (2014). O MobiGuitar é uma adaptação do GUITAR (Memon, 2001), uma ferramenta para Web baseada em MBT. Automatiza o processo de teste de GUIs em aplicações Android, usando em tempo de execução, uma abstração dos widgets da GUI. Esta ferramenta tem 3 passos principais:

- ***Ripping*** **:** Neste primeiro passo, é usado uma versão melhorada do AndroidRipper (Amalfitano et al., 2012). A ferramenta atravessa dinamicamente toda a GUI da apli-

cação e cria maquinas de estado. Começando pelo estado inicial, analisa todos os eventos possíveis neste estado colocando-os numa lista de tarefas. Cada tarefa pode levar a um novo estado ou a um estado já representado, se levar a um novo estado são analisados todos os eventos possíveis e adicionados à lista de tarefas. Cada estado é comparado com outros estados já representados para entender se o mesmo está ou não representado. Esta lista de tarefas é executada sequencialmente, começando sempre pela tarefa que está em primeiro lugar.

- ***Generation*** **:** Na geração dos testes é usado o modelo gerado pelo Ripping e o critério pretendido, isto irá gerar os testes necessários para satisfazer o critério de cobertura. De maneira a evitar que seja gerado um grande número de testes, os autores desenvolveram um critério de cobertura intermédio onde todos os pares de transições adjacentes (eventos) devem ser testados. Por exemplo, a2 tem quatro arestas de entrada (e1, e11, e12 e e13) e seis arestas de saída (e7, e8, e9, e10, e11 e e12). Isso cria 4 x 6 = 24 pares para cobrir. O teste (e1, e11, e8) abrange dois desses pares: (e1, e11) e (e11, e8), logo com um teste é possível cobrir 2 pares de teste (Amalfitano et al., 2012).

- ***Execution*** **:** No último passo, os testes gerados são executados no SUT, estes testes podem detetar *crashes* e erros de funcionamento na aplicação.

2.5.2 *iMPAcT Tool*

A iMPAcT Tool é uma ferramenta desenvolvida por Morgado and Paiva (2015) que tal como o MobiGuitar utiliza engenharia reversa. Enquanto o MobiGuitar procura obter uma máquina de estados que descreva a aplicação, aqui a engenharia reversa é utilizada para identificar padrões de UI presentes no SUT, para testar se os mesmo estão bem implementados. Um padrão de UI é uma solução recorrente que resolve um problema comum. Um exemplo de um padrão é o *Orientation Pattern* que permite ao utilizador rodar o ecrã do dispositivo sem que este perca nenhuma informação ou widget.

A ideia principal desta ferramenta está representada na Figura 3. A ferramenta recorrentemente navega pela aplicação móvel à procura de padrões que coincidem com os padrões identificados no catálogo (UI Patterns). Caso seja encontrado um padrão, é utilizada a estratégia implementa para testar se o funcionamento do mesmo está ou não correto. Esta ferramenta esta divida em 4 fases, exploração, correspondência de padrões, teste e artefactos. Na primeira fase é usado a API UiAutomation[3] para recolher a informação do ecrã e a ferramenta UI Automator[4] para simular interação. É construída uma árvore que começa no

[3] https://developer.android.com/reference/android/app/UiAutomation last accessed: 2017-12-20.
[4] https://developer.android.com/training/testing/ui-automator last accessed: 2017-12-20.

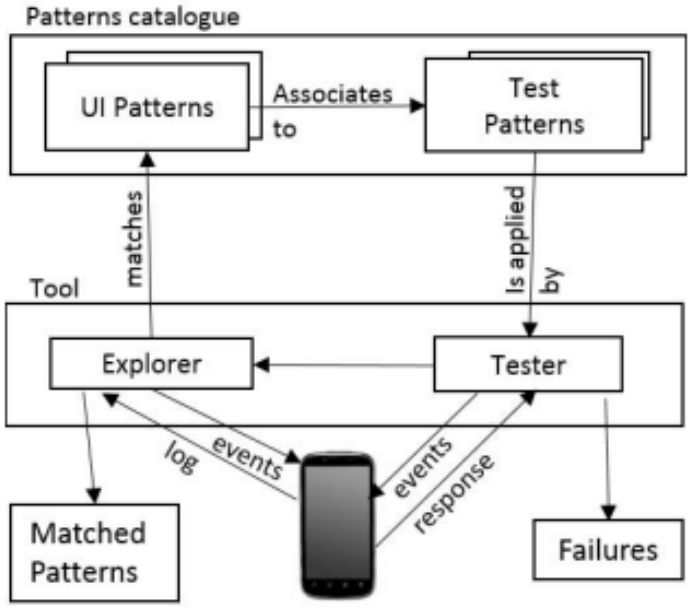


Figura 3.: Arquitetura da abordagem de Morgado and Paiva (2015)

estado inicial da aplicação, a partir do qual se inicia o processo de recolha todos os eventos possíveis de cada estado. Os eventos são escolhidos aleatoriamente para serem ativados. Na segunda fase verifica-se a existência ou não de um padrão depois de ativado o evento escolhido (que causará uma mudança de estado), caso seja identificado algum padrão resultante deste evento é realizada a terceira fase onde é aplicada a estratégia correspondente ao padrão identificado, para saber se o mesmo esta corretamente implementado na aplicação. Na ultima fase são criados dois relatórios. Um com toda a exploração e outro com o modelo da GUI, que toma a forma de uma a Máquina de Estado Finita (*Finite State Machine* (FSM)) e representa todo comportamento da aplicação (Morgado et al., 2014).

2.5.3 *fMBT*

A free Model-Based Testing (fMBT)[5] é uma ferramenta de MBT *open source* desenvolvida pela Intel em AAL/Python. A ferramenta pode ser usada para testes de UI, teste de interface de programação de aplicativos (*Application Programming Interface* (API)) e testes unitários. O gerador é capaz de gerar testes nos modos offline e online. Esta ferramenta esta divida em 2 componentes:

[5] `https://01.org/fmbt/overview` last accessed: 2017-12-28.

- **fmbt-editor:** O primeiro passo é a criação do modelo; com o fmbt-editor é possível visualizar os modelos como máquinas de estado, editar os modelos, editar os parâmetros da geração de testes, gerar diferentes testes e ver a cobertura dos mesmos (Kervinen, 2014).
- **Geração e execução:** Existem diferentes tipos de adaptadores para varias plataformas que podem ser usados, no nosso caso queremos testar para Android pelo que o adaptador a utilizar é o FMBTANDROID (Kervinen, 2013). A geração dos testes é controlada por vários parâmetros sendo os mais importantes: a heurística, que define qual o algoritmo utilizado para escolher a próxima ação a ser testada; a cobertura, que define o que é medido; e as condições finais, que definem quando a geração e a execução do teste devem parar e o que é verificado em cada teste.

2.5.4 *Discussão*

As ferramentas apresentadas anteriormente apresentam algumas limitações que nos levam a querer desenvolver uma componente para Android na ferramenta TOM. Na MobiGuitar, o facto de a construção dos modelos ser totalmente automática, não permite que estes possam ser construídos ou editados manualmente, algo possível no TOM. Na iMPAcT Tool a limitação deve-se ao número restrito de padrões e estratégias que estão implementadas no catálogo, o TOM não apresenta padrões mas acaba por testar toda as funcionalidades da aplicação. A fMBT tem pouca documentação sobre a estrutura dos componentes. Existe um tutorial que apenas contém exemplos com código. Há já algum tempo que a ferramenta não é atualizada o que poderá indicar que o projeto está parado.

Apesar de existirem várias ferramentas disponíveis para a geração de casos de teste todas elas apresentam certas limitações. O TOM, sendo uma ferramenta que já gera casos de teste para Web com sucesso, cria a possibilidade de também o fazer para Android aumentando os cenários que a mesma pode tratar.

2.6 Frameworks para automatização de testes

Tendo sido decidido adicionar suporte para teste de aplicações Android à ferramenta TOM, é importante perceber que ferramentas existem neste momento para suportar a execução automática dos testes.

O Android foi lançado em 2008 e desde então algumas ferramentas tem surgido para ajudar a criar testes automáticos. Neste tópico irão ser abordadas as principais ferramentas

que existem no momento para possibilitar a realização de testes automáticos na UI de aplicações Android.

Espresso (Google, 2017) é a ferramenta oficial adequada para testes funcionais na UI. Esta ferramenta permite realizar testes *white-box* escritos em Java. Algumas das vantagens:

- Fácil de configurar e facilmente extensível.
- É muito rápido.
- Em modo *debugging* fornece uma grande quantidade de informação quando acontece uma falha.
- Um conjunto extenso de APIs de ação para automatizar interações da UI.

Algumas desvantagens desta ferramenta:

- Não suporta versões da Android API inferiores a 8.
- O Espresso espera automaticamente que os eventos sejam concluídos na aplicação SUT usando um *IdlingResource*. Este *IdlingResource* representa um recurso da aplicação em teste que ajuda o expresso a perceber quando um evento está concluído. Algumas vezes este recurso tem de ser inserido manualmente no código para poder ser realizado o teste.
- Não lê conteúdo do ecrã.

UI Automator (Google, 2017) é uma ferramenta oficial também adequada a testes funcionas na UI mas apenas indicada quando é necessário efetuar alterações entre aplicações instaladas e do sistema. Algumas das vantagens:

- Permite visualizar o conteúdo do ecrã, fornecendo a hierarquia de layouts.
- Permite realizar testes entre aplicações.
- Tem acesso ao estado do dispositivo.

Algumas desvantagens desta ferramenta:

- Não suporta versões da Android API inferiores a 18.
- Java é única linguagem suportada.

Robotium[6] surgido em 2010, é uma ferramenta de teste open-source, que permite a escrita de testes automáticos `gray-box` (necessita conhecer um pouco da estrutura interna da aplicação, mas não todo o código-fonte) para aplicações Android. O desenvolvimento dos testes passa pela criação de cenários de testes funcionais, de sistema, de aceitação. etc. Algumas vantagens desta ferramenta:

- Suporta Android Package (é um formato de ficheiro usado para instalar o software no sistema operacional Android, mais conhecido por APKs) e aplicações com código-fonte.
- Suporte total para aplicações Android Nativas e Híbridas.
- Pode ser usado para qualquer versão do Android.
- Contém uma ferramenta que lê o conteúdo do ecrã.

Algumas desvantagens desta ferramenta:

- Sem suporte para aplicações Web.
- Execução dos testes apenas num dispositivo de cada vez.
- Apenas suporta Android.

Appium[7] é uma ferramenta de teste *open-source* que surgiu em 2012. Esta ferramenta usa o *JSON Wire Protocol* para interagir com as aplicações Android e iOS usando *Selenium WebDriver*. Algumas vantagens desta ferramenta:

- Suporta aplicações Nativas, Híbridas e Web.
- Suporta um grande número de linguagens de programação, devido ao facto de usar o *JSON Wire Protocol*.
- Não necessita de acesso ao código do SUT.
- Suporta diferentes ferramentas.
- *Cross-plataform*, o que permite a realização de testes em sistemas Android e iOS.
- Não necessita de utilizar o APK.
- Lê conteúdo do ecrã.

Algumas desvantagens desta ferramenta:

[6] `https://github.com/RobotiumTech/robotium` last accessed: 2018-01-03.
[7] `http://appium.io/` last accessed: 2018-01-03.

- O processo de configuração do Appium é muito demorado, tanto para Android como para iOS.
- Não suporta versões da Android API inferiores a 17.

Calabash[8] é outra ferramenta de testes automáticos. Esta ferramenta foca-se na escrita e execução de testes de aceitação para dispositivos móveis. Algumas vantagens:

- Os testes calabash são baseados na ferramenta Cucumber. São testes que são facilmente criados em linguagem natural.
- *Cross-plataform*.
- Informa sobre o desempenho do dispositivo (memoria, CPU, etc).

Algumas desvantagens desta ferramenta:

- Apenas suporta a linguagem Ruby.
- Não lê conteúdo do ecrã.
- Requer acesso ao código fonte do SUT.

Por último, Selendroid[9] é uma ferramenta de testes automáticos para Android. Esta ferramenta é uma versão do Selenium[10] mas aplicado a Android. Algumas vantagens:

- Suporta aplicações Nativas, Híbridas e Web.
- Não precisa de acesso ao código fonte do SUT.
- Contém uma ferramenta que lê o conteúdo do ecrã.

Algumas desvantagens desta ferramenta:

- A velocidade dos testes é lenta.
- Inutilizável num sistema com menos de 4 GB de RAM.
- Apenas suporta versões da Android API entre 10 a 19.

Salientar que existem mais uma ferramenta oficial da Google para realizar testes automáticos em Android chamada *Android Test Orchestrator* mas apenas foi dado ênfase às duas apresentadas, Expresso e UI Automator, porque são as mais utilizadas e conhecidas.

[8] http://calaba.sh/ last accessed: 2018-01-03.
[9] http://selendroid.io/ last accessed: 2018-01-04.
[10] http://www.seleniumhq.org/ last accessed: 2018-01-04.

2.7 COMPARAÇÃO/ESCOLHA DE UMA FRAMEWORK

Na secção anterior foram apresentadas algumas ferramentas que permitem a programação de testes para Android, cada uma delas com características diferentes. Neste capítulo será efetuada uma comparação das várias ferramentas mencionadas de forma a encontra-se a mais adequada para a nossa geração de testes. A ferramenta selecionada para a implementação dos testes automáticos através do TOM Generator deve ter em conta os seguintes requisitos:

1. Não ser necessário acesso ao código do SUT para a geração dos testes. Como estamos a desenvolver apenas a geração dos testes a partir de um modelo previamente construido não deve ser necessária a utilização do codigo para identificar os elementos.
2. Ser capaz de identificar e modificar os estados dos sensores e serviços do dispositivo. É importante ter acesso a este tipo de serviços e sensores para adicionar certas mutações como por exemplo: rotação do ecrã ou desligar a internet.
3. Ser possível identificar os vários elementos da atividade, através do XPath, id, etc.
4. Ser possível interagir com os elementos selecionados, efetuar clicks, sendKeys, etc.
5. Ter como base o Selenium, uma vez que toda a geração de casos de testes para Web está feita em Selenium era importante conseguir generalizar também para Android aproveitando alguns métodos da geração Web.
6. Segundo Cunningham (2017) a Google para melhorar a segurança e performance da Google Play, decidiu que em novembro de 2018 todas as aplicações irão ser obrigadas a conter o suporte da API superior a 26 e a cada ano que passe esse nível será aumentado. Como tal um dos critérios a avaliar será o suporte da API por parte das ferramentas, isto é, ter suporte para todos os níveis ou então acima de 26.
7. Como toda a geração para Web esta efetuada em Java é importante que estas ferramentas também contenham suporte para Java.

Na Tabela 1 é apresentada a comparação das varias ferramentas em relação aos requisitos necessários para a geração e execução de casos de testes. Apenas o Calash precisa de acesso ao código do SUT. O Espresso e o Robotium não conseguem modificar e identificar os estados dos sensores e serviços no dispositivo. Como era de esperar todas as ferramentas identificam e interagem com os elementos selecionados da atividade. Apenas 3 das 6 ferramentas estão baseadas em Selenium: Robotium, Appium e Selendroid. Somente o Selendroid não tem suporte para APIs superiores a 26, porque tem unicamente suporte para APIs entre 10 e 19. O Calabash apenas pode ser escrito em Ruby não contendo suporte para

a linguagem Java. Nesta analise reparamos que só a ferramenta Appium cumpre todos os requisitos necessários.

Tabela 1.: Comparação das ferramentas através dos requisitos.

Requesitos	Robotium	Appium	Calabash	Selendroid	Espresso	UI Automator
1	X	X		X	X	X
2		X	X	X		X
3	X	X	X	X	X	X
4	X	X	X	X	X	X
5	X	X		X		
6	X	X	X		X	X
7	X	X		X	X	X

Appium, para além dos aspetos comparados nas duas tabelas, fornece outras vantagens que nós levaram à escolha desta ferramenta. Uma delas é ser *cross-plataform*, isto irá permitir que no futuro também seja criado uma componente para IOs que poderá partilhar muito do código do componente Android agora desenvolvido. O facto de esta ferramenta ter suporte também para aplicações Híbridas e Web faz com que a ferramenta TOM consiga vir a ser explorada no futuro com novos cenários a serem desenvolvidos. Outro aspeto muito importante é o Appium selecionar os elementos através do UI Automator View. Este será utilizado para obter a informação sobre os elementos da atividade (Id, XPath, etc) para assim construir os modelos necessários para o TOM.

2.8 sumário

Os testes de software são importantes para aumentar a qualidade das aplicações móveis. As características específicas deste tipo de aplicação, e das aplicações Android em particular, colocam desafios específicos que foram abordados neste capítulo. Estamos particularmente interessados no teste realizado a partir da camada de interface das aplicações.

Os testes baseados em modelos são uma abordagem promissora para automatizar o processo de teste. Foram descritas ferramentas de teste baseados em modelos das interfaces e descritas algumas das ferramentas que poderão ser utilizadas para adaptar este tipo de teste às aplicações móveis.

Durante este capítulo foi apresentada uma lista com as ferramentas existentes que suportam a criação de testes para Android, bem como as principais vantagens e desvantagens de cada uma. Estas ferramentas foram comparadas de forma a escolher a ferramenta mais adequada à solução pretendida. O Appium acabou por ser a ferramenta escolhida para a geração dos casos de teste. O Appium é uma ferramenta *cross-plataform* que permite a realização de testes em aplicações Nativas, Híbridas e Web.

Com esta dissertação pretende-se a criação de uma componente na ferramenta TOM para a geração de casos de teste para aplicações Android. Isto eleva a ferramenta TOM a um novo cenário, passando de 3 para 4 os tipos de teste possíveis de criar através desta ferramenta. No próximo capítulo, a ferramenta TOM será descrita.

3

TOM

TOM é uma ferramenta que permite a automatização da realização de testes baseados em modelos para aplicações Web. Como já descrito em cima, esta ferramenta esta divida em três componentes principais, o TOM Generator, o TOM Editor e o TOM App. Na Figura 4 temos a representação estrutural de todos os componentes da ferramenta. O TOM Editor através do SUT cria os modelos que irão ser usados pelo TOM App e pelo TOM Generator. O TOM Generator pode ser utilizado via web services ou linha de comando.

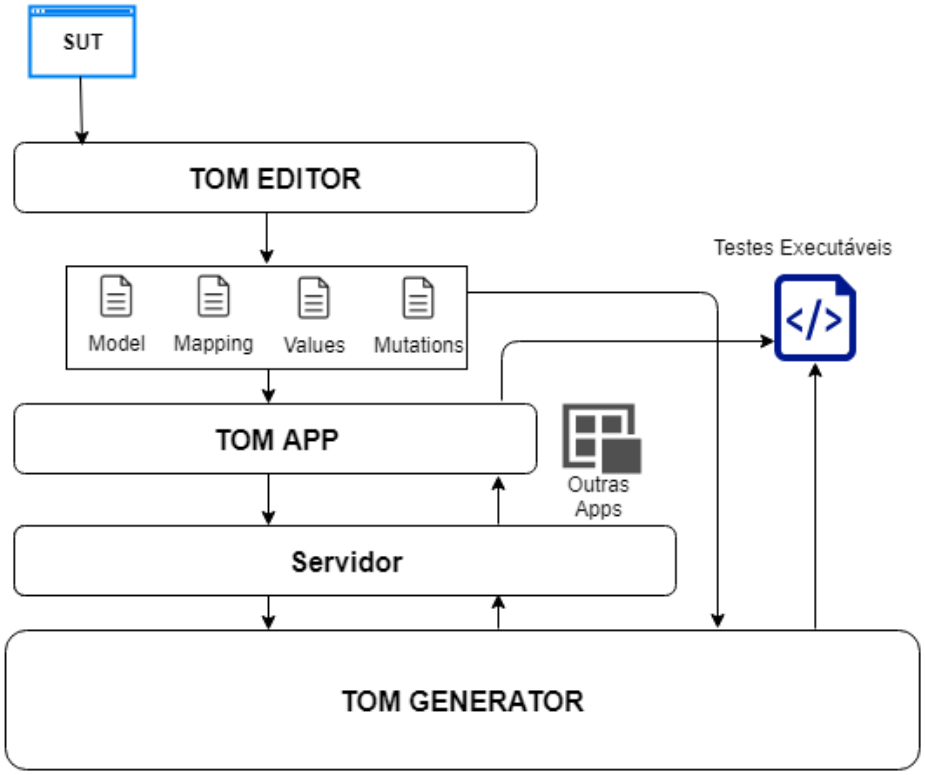


Figura 4.: Estrutura do TOM

3.1 MODELOS

Para a utilização do TOM é necessária a compreensão dos quatro tipos de modelos necessários para a geração dos casos de testes. Esta secção apresenta uma descrição resumida dos formatos. O Anexo C, criado com base em (Pinto, 2017), apresenta uma descrição mais detalhada, servindo de manual para a escrita dos modelos, incluído, em relação a (Pinto, 2017) as especificações adicionadas para suportar o teste de aplicações Android. O TOM Editor cria estes quatro modelos, que o TOM Generator utiliza.

3.1.1 *Modelo do Sistema*

A representação da GUI é efetuada recorrendo a FSMs. Na sua representação é utilizado *State Chart XML* (SCXML), por ser uma linguagem flexível e permitir organizar os dados de uma forma estruturada (Rodrigues, 2015).

Um estado do modelo representa uma página da interface gráfica (uma página Web), dentro do mesmo estado é possível ter outros estados (por exemplo, para representar formulários dentro de uma página). Cada estado pode ter várias transições para outros estados (representando botões, links, etc). Um estado também pode conter validações, por exemplo, verificar se um botão esta visível, se um elemento contém um determinado texto, etc, (Rodrigues, 2015; Gonçalves, 2017).

Na Listagem 3.1 temos um pequeno exemplo da modelação de um login numa página. O estado desta página está representado da linha 1 à linha 11. Dentro do estado temos um sub-estado que é o formulário do login. Este formulário tem 2 campos que é necessário preencher (elementos `<send>`). Depois dos campos preenchidos ao carregar no botão para efetuar o login será efetuado uma transação para um novo estado (identificado pelo identificador 20180001535003), caso o login seja efetuado com sucesso. Temos também duas validações neste estado, a primeira esta representada na linha 2: ao efetuarmos a entrada na pagina (identificador 20180001535011). Depois de realizado o formulário temos uma validação na linha 9 (identificador 20180001535010). Os vários ids e *labels* apresentados na Listagem 3.1 servem para saber como identificar os valores e mapeamento definido nos outros modelos.

```
1 <state id="0">
2   <onentry id="2018001535011" type="displayed?" />
3   <state id="Login" type="form">
4     <send label="s2018001535004" type="required" />
5     <send label="s2018001535005" type="required" />
6     <transition type="form" label="2018001535002">
7       <submit target="2018001535003" />
8     </transition>
9     <onexit id="2018001535010" type="enabled?" />
10  </state>
11 </state>
12 ....
```

Listagem 3.1: Exemplo do Modelo do Sistema

3.1.2 *Mapeamento e Valores*

É necessário um mecanismo que permita mapear as tags representadas na máquina de estados em SCXML para os elementos *Hypertext Markup Language* (HTML) da aplicação Web. Este modelo contém toda a informação necessária para encontrar o elemento HTML da aplicação, bem como informação sobre o tipo de ação a executar. As regras para a construção dos modelos de mapeamento e valores são explicadas em (Pinto, 2017). Voltando ao exemplo, nas linhas 2 a 6 da Listagem 3.2 é definida a *label* s2018001535004, utilizada na linha 4 da Listagem 3.1. Podemos ver que é definido como descobrir o elemento a preencher, neste caso através do atributo name (linha 3), que deverá ter o valor "user_id"(linha 4). Podemos ver ainda que a ação a ser executada é de inserir um texto nesse elemento (linha 5).

```
1 {
2   "s2018001535004": {
3     "how_to_find": "name",
4     "what_to_find": "user_id",
5     "what_to_do": "sendKeys"
6   },
7   "s2018001535005": {
8     "how_to_find": "name",
9     "what_to_find": "password",
10    "what_to_do": "sendKeys"
11  }
12 }
```

Listagem 3.2: Exemplo de um Modelo de Mapeamentos

Como visto acima, algumas das ações e das validações têm valores a si associados (por exemplo o valor com que se vai preencher um *text field*). Como tal, é necessário criar um mecanismo que permita definir os valores a utilizar. Foi criado um modelo para conter toda a informação necessária para o *input* dos campos utilizados no modelo. Na Listagem 3.3 podemos reparar que a *label* s2018001535004 utilizada nos anteriores exemplos vai conter o valor pedro@gmail.com.

```
1 [
2   { "s2018001535004" : "pedro@gmail.com" },
3   { "s2018001535005" : "12345Passe" }
4 ]
```

Listagem 3.3: Exemplo de um Modelo de Valores

3.1.3 *Mutações*

No modelo é representado o comportamento esperado do utilizador, sem erros que este possa cometer no sistema. Estes testes são importantes porque nos permitem garantir que todas as funcionalidades relevantes do sistema são testadas mas não representam comportamentos inesperados do utilizador no sistema. As mutações representam erros que serão introduzidos nos casos de teste para perceber como o sistema irá reagir ao comportamento inesperado do utilizador, com isto aumentamos a qualidade dos testes. As mutações suportadas estão relacionadas com o preenchimento de formulários. Existem as seguintes mutações disponíveis:

- ***Slips*** **:** troca na ordem de execução das ações.
- ***Lapses*** **:** um determinado campo não é preenchido.
- ***Mistakes*** **:** é realizada a modificação de um valor a introduzir num campo do formulário.
- ***Double Click*** **:** é realizado um duplo clique no botão de submissão.

Na Listagem 3.4 o elemento identificado pela *label* s2018001535004 vai sofrer uma mutação do tipo *lapse*, enquanto que o elemento identificado pela *label* s2018001535005 será do tipo *mistake*, ambas as mutações deverão fazer com que o resultado do teste falhe (atributo fail a 1).

```
[
  {
    "type" : "lapse",
    "model_element": "s2018001535004",
    "fail" : "1"
  },
  {
    "type" : "mistake",
    "model_element": "s2018001535005",
    "value" : "passe2",
    "fail" : "1"
  },
  ...
]
```

Listagem 3.4: Exemplo de um Modelo de Mutações

3.2 TOM EDITOR

O TOM Editor é uma extensão de browser desenvolvida por Pinto (2017), que permite a criação de modelos do sistema da interface de uma aplicação Web, à medida que o utilizador vai fazendo a sua navegação ao longo da aplicação a ser testada. Esta extensão possui dois modos de funcionamento:

- **Automático:** vai gravando toda a navegação do utilizador ao longo do sistema.
- **Semi-Automático:** dá alguma liberdade ao utilizador na criação dos modelos, permitindo a introdução de novos estados, validações, formulários, etc.

Um dos principais objetivos desta extensão é facilitar a construção dos modelos, permitindo ao utilizador poupar muito tempo no desenvolvimento dos mesmos, mas também ajudando a evitar erros que tendem a ocorrer nos modelos criados à mão. No final da construção do modelo do sistema são gerados os quatro modelos descritos na secção anterior. Na Figura 5 está representado um pequeno exemplo do funcionamento do TOM Editor na página de login do sistema de elearning da Universidade do Minho. Como se pode ver, foram identificados dois estados, `Login` e `Home`. O estado `Login` tem um sub-estado (`Form Login`) e é possível, indicar o `Username`, a `Password` e transitar para o estado `Home`.

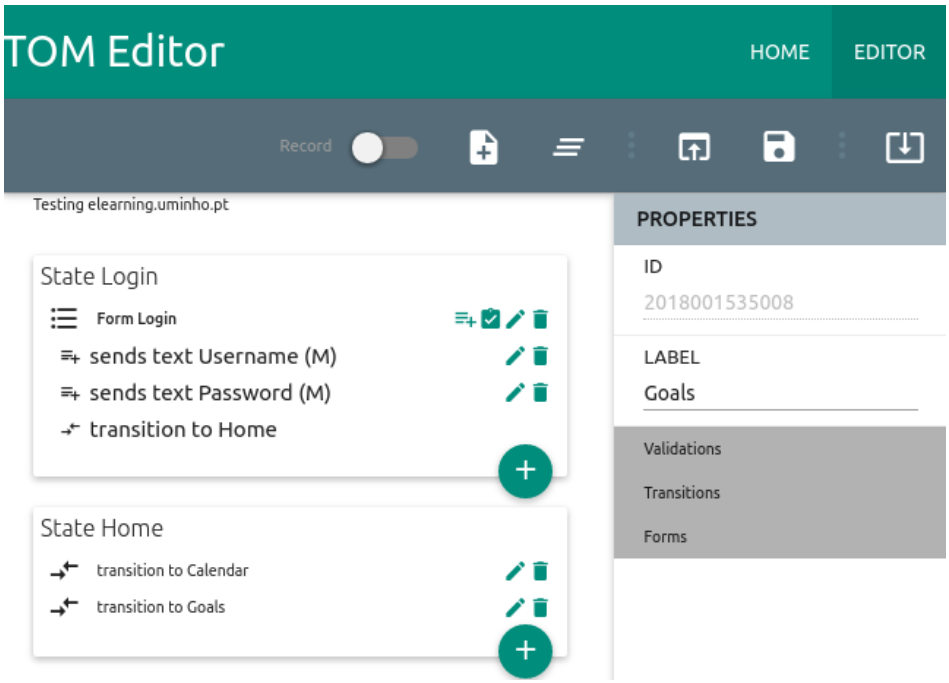


Figura 5.: Exemplo do TOM Editor

3.3 TOM GENERATOR

O TOM Generator é o componente mais importante da ferramenta TOM. É neste componente que os casos de teste são gerados para depois poderem ser executados. O componente segue a abordagem MBT, explicada anteriormente (Secção 2.4), está desenvolvido em Java e disponibiliza um web service com o qual podemos interagir para a geração de casos de teste. Na Figura 6 temos descrito o funcionamento completo deste componente.

Antes de começar a geração dos casos de teste, o componente necessita dos quatro modelos apresentados anteriormente bem como de um ficheiro de configuração. No ficheiro de configuração está descrito que tipo de algoritmo deverá ser utilizado para gerar os testes: *Depth-First Search* (DFS) ou o *Breadth-First Search* (BFS), o número de visitas por vértice e aresta e o tipo de testes a serem gerados. Quando o presente trabalho se iniciou o TOM conseguia gerar testes para Web, Circus (Fayollas et al., 2014) (testes do tipo IRIT) e PVSio-Web (Masci et al., 2015) (testes do tipo JSON).

O modelo do sistema é transformado num grafo que é usado pelo algoritmo escolhido, criando os inúmeros caminhos conforme o número de visitas definido na configuração. Estes caminhos serão depois transformados em testes abstratos que são traduzidos para testes concretos do tipo de geração pretendida (Web, IRIT ou JSON). Durante este processo

de criação de testes concretos podem ser necessários os modelos de mapeamento, de valor e de mutações, assim como o ficheiro de configuração dependendo do tipo de testes que é necessário gerar. No final deste processo obtemos os testes executáveis para serem testados no SUT.

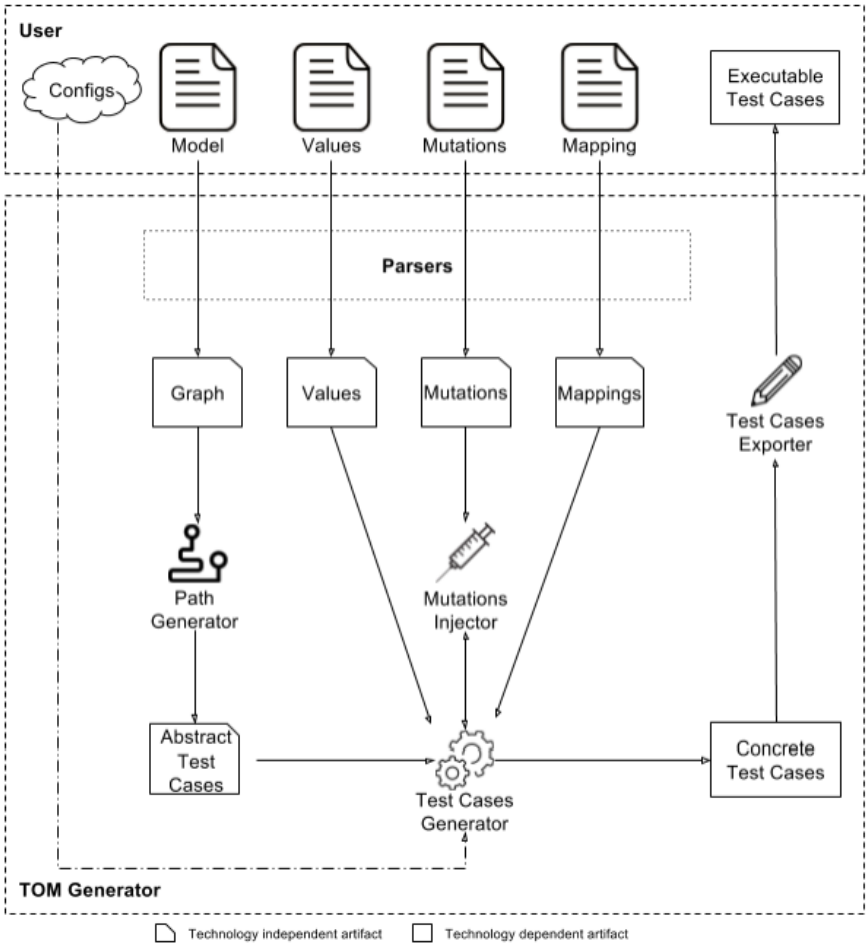


Figura 6.: TOM Generator

A nível de implementação, o TOM Generator utiliza dois padrões. Um dos padrões utilizados foi o *Abstract Factory*, que fornece uma interface para a criação de famílias de objetos relacionadas ou dependentes sem especificar as classes concretas. Este padrão está representado na Figura 7, onde é aplicado aos algoritmos e geradores para ser possível aumentar o leque dos mesmos sem ser necessário realizar grandes alterações. O outro padrão utilizado foi o *Strategy* que define um conjunto de algoritmos encapsulados. Este padrão foi aplicado na criação dos parsers dos modelos do sistema, existindo neste momento dois, o *ParserSCXML* e o *ParserEMDL*.

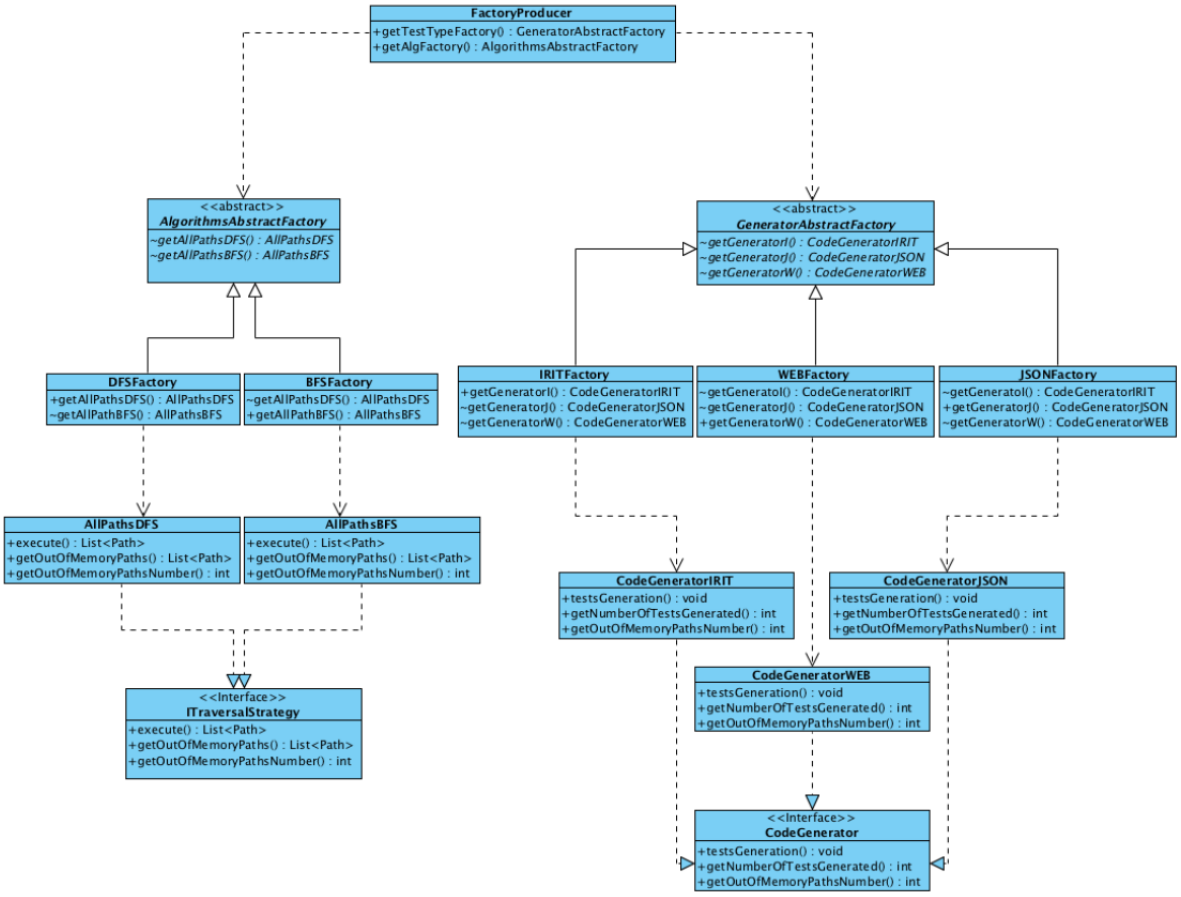


Figura 7.: Padrão *Abstract Factory* aplicado aos algoritmos e geradores (Gonçalves, 2017).

Na Figura 8 está representado o modelo da base de dados utilizada pelo TOM Generator (quando em modo web service) até ao momento do inicio da dissertação. O servidor pode ser acedido por vários utilizadores (tabela `users`), cada um deles pode registar vários projetos (`projects`) e fazer inúmeros pedidos de geração (`generation`). Cada projeto contém um modelo do sistema (`model`) e poderá ter um modelo de valores (`values`), mapeamentos (`mappings`) e mutações (`mutations`) dependendo do cenário. Cada pedido de geração (`generation`) tem as mutações escolhidas (`generation_type_of_mutation`) pelo utilizador e no fim de realizada a geração dos testes (`generation_tests`) os resultados são armazenados. Todos os ficheiros (`files`) são armazenados no sistema de arquivos do servidor onde o TOM Generator é executado. Na base de dados apenas os caminhos para os arquivos são armazenados.

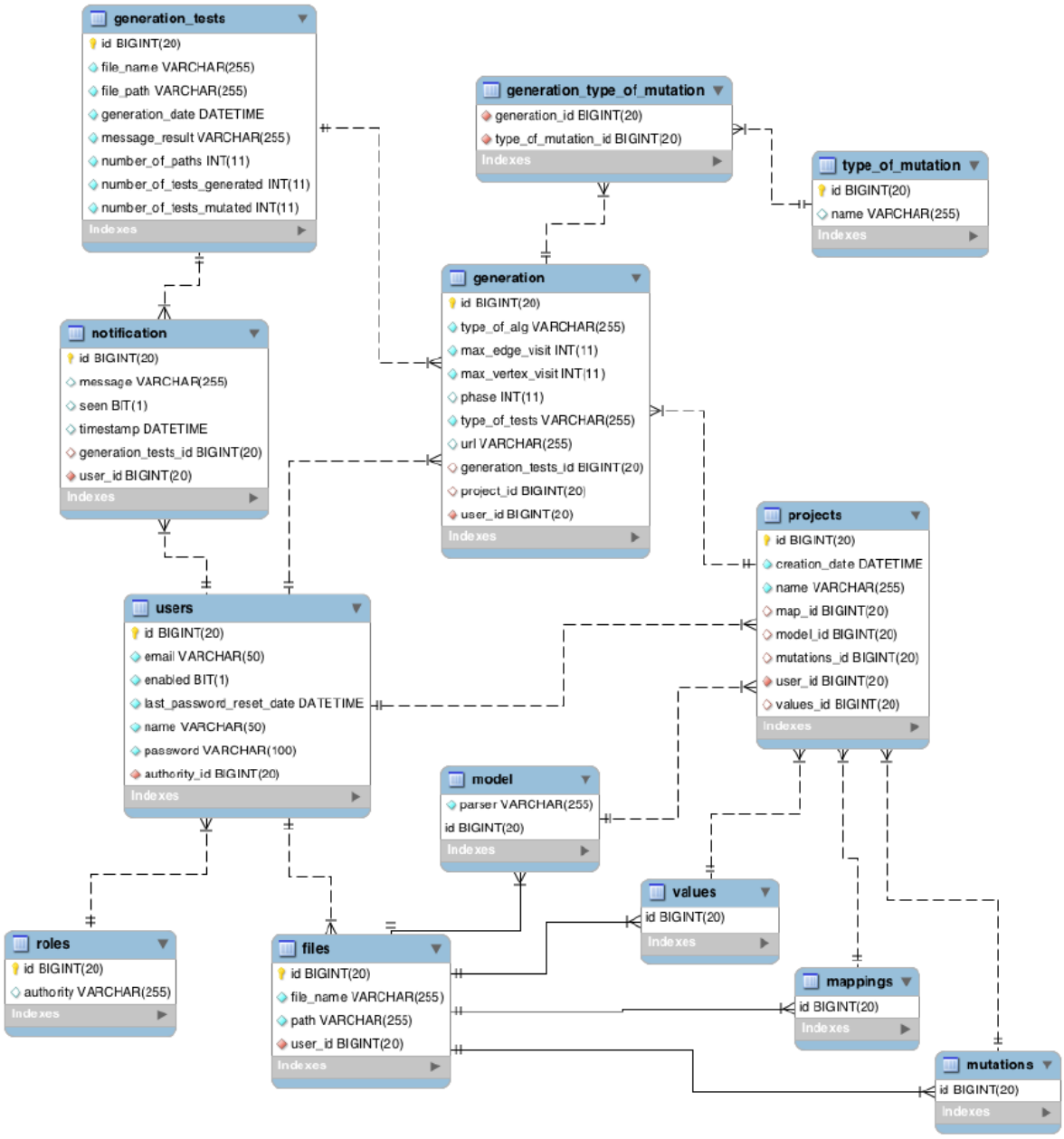


Figura 8.: Modelo da base de dados (Gonçalves, 2017).

3.4 TOM APP

A TOM App é uma aplicação Web que permite aos utilizadores interagirem com a API disponibilizada pelo TOM Generator. Esta aplicação permite, assim, que a comunicação com o TOM Generator para realizar a geração de casos de teste seja mais fácil e prática. Através da Figura 9 conseguimos perceber que a configuração da geração passou a ser mais

fácil e acessível para o utilizador, uma vez que deixa de ser necessário ir constantemente ao ficheiro de configuração mudar os valores que pretendemos utilizar.

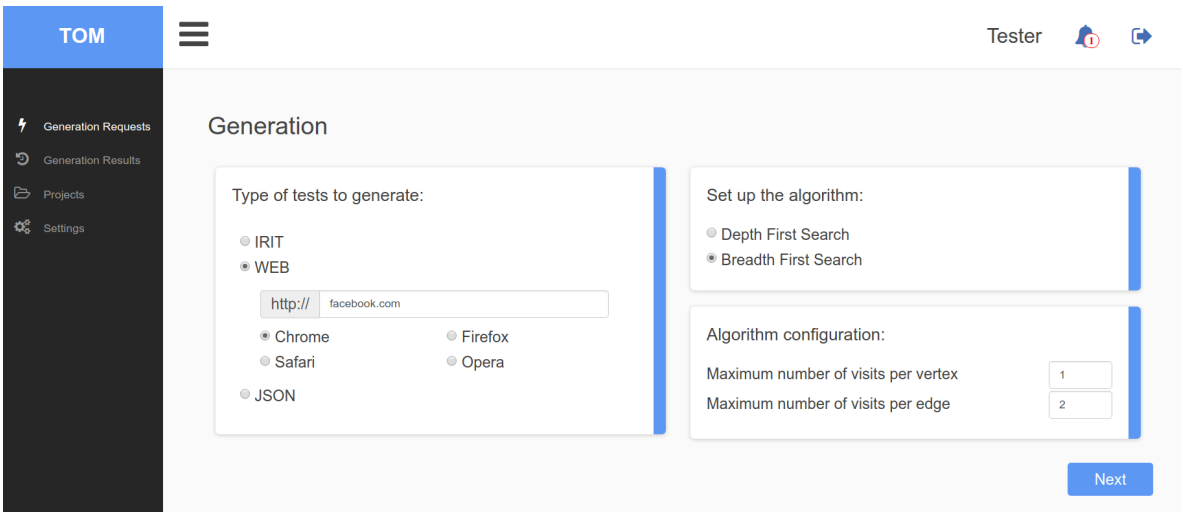


Figura 9.: Exemplo do TOM App

3.5 Sumário

Durante este capítulo foi apresentada a ferramenta TOM. A estrutura da ferramenta foi ilustrada por forma a facilitar a compreensão de como as três principais componentes (TOM Generator, TOM App e TOM Editor) comunicam entre si. Em cada uma das componentes foi explicada o seu funcionamento. Os modelos (sistema, valores, mapeamentos e mutações) dependo do cenário pretendido, são necessários para que TOM Generator consiga gerar os casos de teste. No Anexo C estão representadas todas as possibilidades que podem ser usadas nos modelos apresentados.

Adicionar suporte para Android implica implementar a geração de casos de teste executáveis nessa plataforma. Essa implementação, bem como outras alterações que dela decorreram, serão o tema do próximo capítulo.

4

GERAÇÃO DE TESTES PARA ANDROID

Neste capítulo é abordada a implementação dos gestos relevantes para as várias interações com a aplicação Android através da ferramenta Appium. Estes gestos estão implementados na componente responsável pela geração Android. Esta componente foi criada seguindo a arquitetura do padrão *Abstract Factory* utilizado para os restantes cenários, sendo a mesma responsável pela geração de casos de teste executáveis para Android. Os modelos (sistema e mapeamento) utilizados para a geração de casos de teste para os vários cenários também sofreram alterações devido a implementação dos gestos. Algumas mutações também foram acrescentadas assim com uma melhoria na mutação *slip*. O servidor sofreu alterações numa tabela da base de dados e modificações em alguns métodos para providenciar um pedido de geração Android. O TOM App foi modificado para permitir o pedido de geração para Android.

4.1 INFRAESTRUTURA DE SUPORTE

Com a ferramenta Appium escolhida para a geração dos casos de teste é necessário ter uma estrutura de apoio que permita organizar e executar os testes. Tal como na abordagem utilizada na geração/execução do teste em Web, a ferramenta TestNG vai ser também utilizada, dado permitir a organização dos testes e criação de relatórios de forma a ser mais fácil a visualização de todo o processo de execução e compreensão de possíveis erros ocorridos durante a execução. Os testes são gerados em Java e executados usando Appium e TestNG, as duas ferramentas complementasse uma à outra. Appium fará a interação com o dispositivo e o TestNG a organização dos testes.

4.1.1 *Appium*

Esta secção irá descrever um pouco mais o Appium, a sua arquitetura e o que é necessário configurar inicialmente para um teste começar a ser executado numa determinada aplicação.

Arquitetura

Antes de começar a realização dos testes é necessário entender um pouco da arquitetura da ferramenta. A Figura 10 apresenta o funcionamento do Appium. Os WebDriver Scripts (testes) são executados numa máquina e cada teste comunica os comandos a serem realizados, em formato JSON via *Hypertext Transfer Protocol* (HTTP) ao servidor Appium. O servidor irá invocar as ferramentas especificas para cada plataforma, para executar os comandos no dispositivo. O dispositivo por sua vez retorna as respostas para o servidor e este retorna o resultado do comando executado para a maquina onde os testes estão a ser executados.

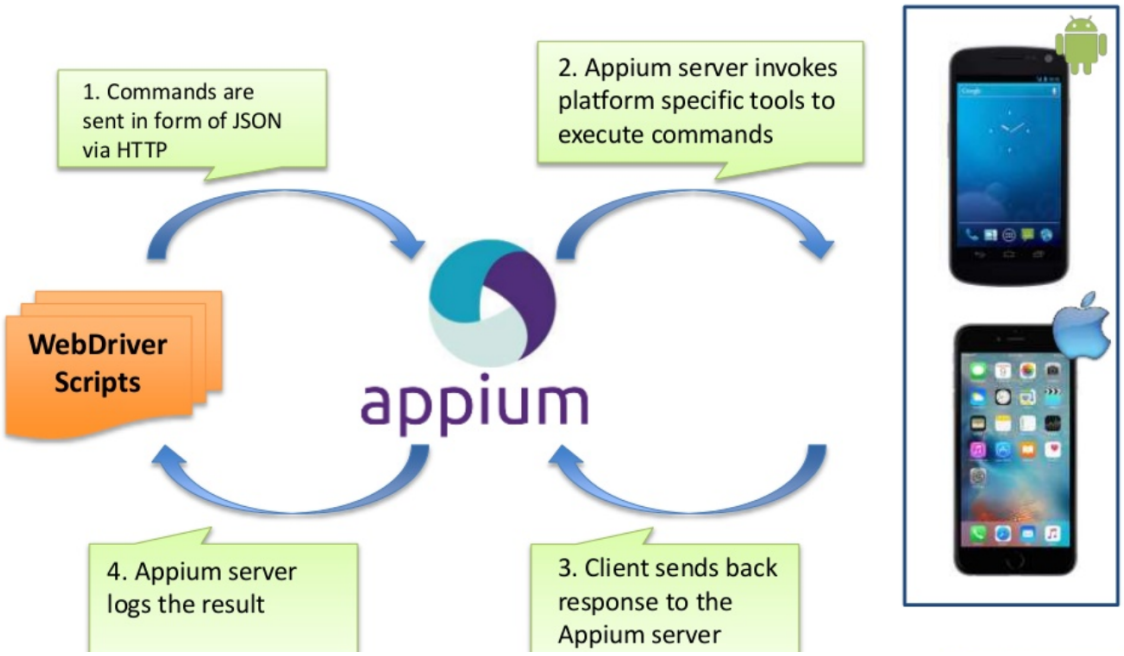


Figura 10.: Arquitetura Appium[1]

Pré-Configuração dos testes

Para haver comunicação entre os testes a serem executados e o dispositivo é necessário fornecer, em todos os testes, alguns parâmetros para o Appium saber em que aplicação e dispositivo deve ser realizado cada teste. Para isso, é necessário fornecer as seguintes informações:

- **deviceName:** Nome do dispositivo, pode ser obtido através do comando adb.
- **platformName:** Nome da plataforma, neste caso é o Android mas pode ser iOS.

[1] https://www.slideshare.net/thinkexist/cross-platform-test-automation-using-appium slide 24 last accessed: 2018-05-08.

- **app:** O caminho local ou o URL http remoto para um arquivo .apk ou .ipa, ou um .zip contendo um deles.
- **appActivity:** A atividade inicial da aplicação.
- **appPackage:** O package da aplicação.

Podemos utilizar apenas o parâmetro app ou então o appAcitvity e o appPackage. O parâmtero app permite o Appium instalar a aplicação no dispositivo e, a partir dai, começa a ser realizado o teste pretendido. A utilização dos outros dois métodos é efetuada caso a aplicação já esteja instalada no dispositivo não sendo necessária a instalação.

Na Listagem 4.1 temos um pequeno exemplo do código para começar a realizar um teste. Como podemos ver, o dispositivo tem o nome 900627246511, é do tipo Android e a aplicação, neste caso uma calculadora, pode ser encontrada através do package com.android.calculator2 e tem a atividade inicial com.android.calculator2.Calculator.

```
1 AppiumDriver driver;
2 DesiredCapabilities caps = new DesiredCapabilities();
3 caps.setCapability("deviceName","900627246511");
4 caps.setCapability("platformName", "Android");
5 caps.setCapability("appPackage","com.android.calculator2");
6 caps.setCapability("appActivity","com.android.calculator2.Calculator");
7 this.driver = new AppiumDriver(new URL("http://0.0.0.0:4723/wd/hub"),caps);
```

Listagem 4.1: Codigo para a comunicação com o servidor Appium.

4.1.2 *TestNG*

De forma a perceber a configuração dos testes com o TestNG e o Appium é apresentado um exemplo, na Listagem 4.2, da estrutura utilizada na geração dos testes. Como podemos ver, para conter o driver é utilizada uma variável global do tipo AppiumDriver. Isto permite que o driver seja inicializado e utilizado pelas funções de teste. Antes da realização de cada teste, neste caso test1, test2 e test3, é inicializado o driver. No inicio de todos os testes é sempre iniciado o driver e no fim é sempre destruído, através das funções initDriver e closeDriver, respectivamente. Estas duas funções, identificadas através das anotações, permitem que este código não seja constantemente repetido nos testes.

```
public class AndroidTests_NORMAL {

    private AppiumDriver driver;

    @BeforeMethod
    public void initDriver()
            throws MalformedURLException
    {
        DesiredCapabilities caps = new DesiredCapabilities();
        caps.setCapability("deviceName","900627246511");
        caps.setCapability("platformName", "Android");
        caps.setCapability("appPackage","com.android.calculator2");
        caps.setCapability("appActivity","com.android.calculator2.Calculator")
            ;
        this.driver = new AppiumDriver(new URL("http://0.0.0.0:4723/wd/hub"),
            caps);
    }
    @Test(invocationCount = 1, groups = "1")
    public void test1()
            throws IOException, InterruptedException
    {
        .
        .
    }
    @Test(invocationCount = 1, groups = "2")
    public void test2()
            throws IOException, InterruptedException
    {
        .
        .
    }
     @Test(invocationCount = 1, groups = "3")
    public void test3()
            throws IOException, InterruptedException
    {
        .
        .
    }
    @AfterMethod
    public void closeDriver() {
        this.driver.quit();
    }
}
```

Listagem 4.2: Exemplo da estrutura utilizada na geração dos testes.

4.2 GESTOS DE ANDROID IMPLEMENTADOS

Sendo o foco principal desta dissertação a geração de casos de testes para aplicações móveis, é necessário ter em consideração as formas de interação características de interfaces por toque e com dispositivos móveis. Na geração de casos de testes foram utilizados os cinco gestos considerados mais relevantes na interação com aplicações móveis. Os mesmos levaram à realização de alterações em alguns modelos utilizados pelo TOM, acrescentando novas variáveis e tipos para que a realização do gesto seja executada. Neste momento é possível realizar cinco tipos de gestos diferentes:

- *Tap*: um simples toque num elemento.
- *DoubleTap*: dois *Taps* no mesmo elemento.
- *LongPress*: pressionar um determinado elemento durante algum tempo.
- *Scroll*: procurar um elemento numa lista e carregar nele.
- *Swipe*: permite trocar um elemento de uma posição com outro elemento ou simplesmente colocar esse elemento noutra posição.

Como alguns destes elementos levaram a que fossem efetuadas diferentes alterações nos modelos utilizados para a geração dos casos de teste, será feita uma analise das alterações que foi necessário fazer.

4.2.1 *Tap/DoubleTap/LongPress/Scroll*

Estes quatro gestos levaram à mesma alteração no modelo de mapeamento, esta alteração ocorre porque a classe TouchAction (classe utilizada no código para executar a ação) realiza as ações dos gestos através de pontos. Estes pontos são coordenadas X e Y no qual o elemento está localizado, estas coordenadas podem ser obtidas de duas formas:

1. O elemento pode ser selecionado da maneira tradicional, isto é, através do id, da classe, do XPath, etc. O que será realizado é a criação de um objeto do tipo MobileElement em que iremos selecionar o elemento e posteriormente através de duas funções do MobileElement retirar as coordenadas X e Y daquele elemento.
2. Dar diretamente as coordenadas do elemento (*OffSet*). Isto permite que o elemento seja selecionado sem ser necessário retirar as coordenadas através da classe MobileElement. Esta opção só deve ser utilizada caso o elemento esteja colocado de forma estática na atividade, pois a ação vai ser sempre realizada nas coordenadas dadas pelo modelo de mapeamento, o que significa que caso existam alterações na atividade o elemento pode não ser encontrado.

Por forma a perceber algumas das alterações realizadas com estes gestos, apresenta-se um exemplo da realização de um *Tap*. Pretende-se realizar um *Tap* no botão tirar uma nota da interface apresentada na Figura 11. Isto fará com que seja possível adicionar uma nova nota ao bloco de notas.

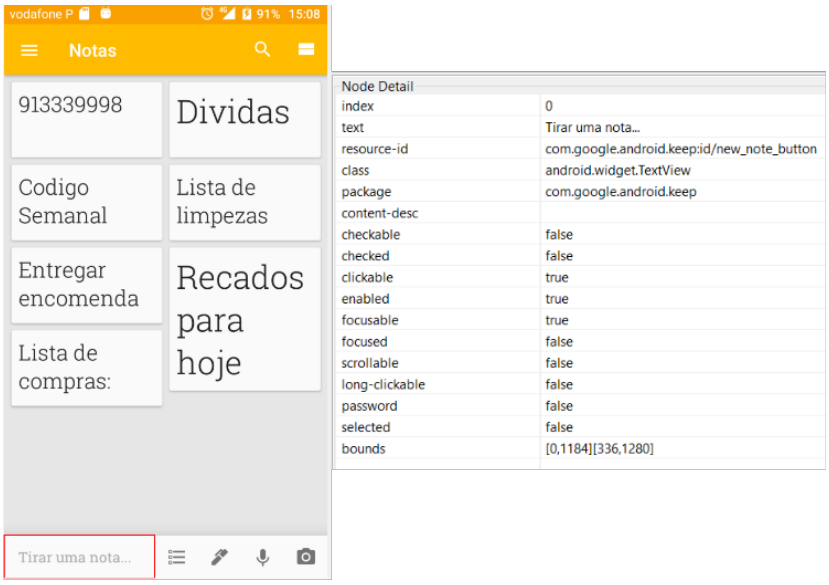


Figura 11.: UI Automator Viewer exemplo do *Tap*

Na Listagem 4.3 é apresentado os mapeamentos efetuados através do id, representado por mapeamentoNormal (também poderia ser o XPath, class, etc.), e *OffSet*, representado por mapeamentoOffset. Num exemplo a ser executado apenas um deles pode ser realizado mas para fins ilustrativos decidiu-se juntar as duas formas de realizar o mesmo *Tap*.

```
[
    "mapeamentoNormal": {
      "how_to_find": "id",
      "what_to_find": "com.google.android.keep:id/new_note_button",
      "what_to_do": "tap"
    },
    "mapeamentoOffset": {
      "how_to_find": "offset",
      "what_to_do": "tap"
      "offset_x" : "250",
      "offset_y" : "1220"
    }
]
```

Listagem 4.3: Mapeamento do *Tap*.

Na Listagem 4.4 temos o código gerado pelo TOM depois de implementada e atualizada a ferramenta para a geração Android. Para a realização de um *Tap* na aplicação através do id, XPath, etc, usando o mapeamentoNormal, necessitamos de selecionar o elemento para obter as suas coordenadas de forma a realizar o *Tap*, enquanto que a realização através do *OffSet*, pelo mapeamentoOffset, é direta como apresentado na Listagem 4.5.

```
1 [
2     mobileElement = (MobileElement) this.driver.findElement(By.id("com.google.
          android.keep:id/new_note_button"));
3
4     new TouchAction(driver).tap(point(mobileElement.getCenter().x,
          mobileElement.getCenter().y)).release().perform();
5 ]
```

Listagem 4.4: Código de um *Tap* através do Id.

```
1 [
2       new TouchAction(driver).tap(250,1220).release().perform();
3 ]
```

Listagem 4.5: Código de um *Tap* através do *OffSet*.

4.2.2 *Swipe*

O *Swipe* levou à realização de várias alterações no modelo de mapeamento. O *swipe* permite trocar um elemento de uma posição com outro elemento ou simplesmente colocar esse elemento noutra posição. Este gesto pode ser realizado de quatro maneiras diferentes, sendo elas:

1. Selecionar o primeiro e o segundo elemento da forma tradicional.
2. Selecionar o primeiro elemento da forma tradicional e o segundo através do *OffSet*.
3. Selecionar o primeiro elemento através do *OffSet* e o segundo da forma tradicional.
4. Selecionar o primeiro e o segundo elemento através do *OffSet*.

Isto levou a que o modelo de mapeamento tivesse quase o dobro das variáveis para ser possível realizar todas as opções.

Na Figura 12 temos um exemplo do *Swipe*. A funcionalidade a ser realizada será colocar a caixa de texto com nome Dividas no fim do bloco de notas.

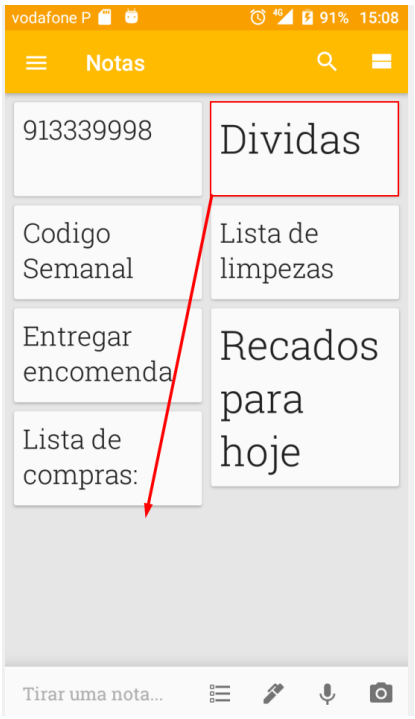


Figura 12.: Exemplo de um *Swipe*

O mapeamento desta funcionalidade está apresentado na Listagem 4.6. Neste caso utilizou-se a forma tradicional no primeiro elemento e o *OffSet* no segundo elemento. Neste mapeamento também estão apresentadas as novas variáveis adicionadas. Mais uma vez, não é necessário conter toda a informação, apenas uma das quatro opções apresentadas anteriormente.

```
[
    "mapeamentoSwipe": {
      "how_to_find": "xpath",
      "what_to_find": "//android.support.v7.widget.RecyclerView[@index='0']/
          android.widget.FrameLayout[@index='1']",
      "what_to_do": "swipe",
      "offset_x" : "",
      "offset_y" : "",
      "what_to_do_second_element": "",
      "what_to_find_second_element": "",
      "how_to_find_second_element": "offset",
      "offset_sencond_x" : "400",
      "offset_sencond_y" : "900"
    }
]
```

Listagem 4.6: Mapeamento do *Swipe*

Na Listagem 4.7 temos o exemplo do código necessário para a execução deste *Swipe*.

```
[
    mobileElement = (MobileElement) this.driver.findElement(By.xpath("//
        android.support.v7.widget.RecyclerView[@index='0']/android.widget.
        FrameLayout[@index='1']"));

    new TouchAction(driver)longPress(point(el2.getCenter().x,el2.getCenter().y
        )).moveTo(point(400,900)).release().perform();
]
```

Listagem 4.7: Codigo do *Swipe*

4.3 MUTAÇÕES

Nesta Secção serão apresentadas duas mutações que foram introduzidas especificamente para a geração de testes Android.

4.3.1 *Rotação*

A Rotação permite que o ecrã rode consoante a posição pretendida, este gesto é de utilização comum por parte dos utilizadores, dependendo do que estejam a fazer ou queiram fazer. O próprio dispositivo acaba por ter a capacidade de rodar automaticamente caso essa opção esteja selecionada pelo utilizador. Existe dois tipos de rotação, mas iremos considerar três opções que podem ser realizadas:

1. Rotação Vertical: mais conhecida por *portrait rotation*.
2. Rotação Horizontal: mais conhecida por *landscape rotation*.
3. Ambas as rotações: rodar primeiro para *landscape rotation* e depois de seguida rodar novamente para *portrait rotation*.

O TOM Generator permite duas formas de realização da rotação: a explícita e a implícita. A explícita dá a liberdade de realizar a rotação a qualquer momento, podendo ser criada no modelo do sistema. Foi aproveitada a utilização da tag *onentry* para o efeito, logo apenas foi necessária a introdução de novos novos valores para o atributo *type* da tag. A presença de uma tag *onentry* com o atributo *type* preenchido com uma das opções da Listagem 4.8, irá causar uma rotação quando entramos no estado. A rotação implícita pretende testar os formulários realizando automaticamente a rotação para verificar se ao rodar existe perca de informação. Foi criada uma mutação com o nome *Rotate*, o que esta

mutação vai fazer é rodar o ecrã no final de todos os formulários antes de carregar no botão de submeter, neste caso será apenas usada a opção de ambas as rotações, para saber se a rotação influencia o que esta já escrito e selecionado no formulário. O implícito acaba por ser igual ao *DOUBLEROTATE* mas será realizado sempre que exista um formulário. Na Listagem 4.9 temos o respetivo código de cada opção de rotação disponível.

```
<onentry id="" type="ROTATELANDSCAPE" />
<onentry id="" type="DOUBLEROTATE" />
<onentry id="" type="ROTATEPORTRAIT" />
```

Listagem 4.8: Modelo do sistema exemplo Rotação

```
[
    // ROTATELANDSCAPE
    driver.rotate(ScreenOrientation.LANDSCAPE);

    // DOUBLEROTATE
    driver.rotate(ScreenOrientation.LANDSCAPE);
    driver.rotate(ScreenOrientation.PORTRAIT);

    // ROTATEPORTRAIT
    driver.rotate(ScreenOrientation.PORTRAIT);
]
```

Listagem 4.9: Código de cada Rotação

4.3.2 *Sensors*

Os sensores são mutações adicionadas que permitem ligar e desligar os sensores do dispositivo durante a realização dos testes na aplicação móvel. Existem cinco possibilidades de realizar esta mutação:

1. WifiInterrupted: desligar e ligar o Wi-Fi.
2. WifiOff: desligar o Wi-Fi.
3. WifiON: Ligar o Wi-Fi.
4. AirPlane: Ligar o modo Avião.
5. Data: Ligar os dados móveis.

Essa mutação tem de ser inserida explicitamente no sistema, do mesmo modo que inserimos as rotações explícitas. Na Listagem 4.10 apresentamos as cinco possibilidades no modelo do sistema e na Listagem 4.11 o seu código para execução no dispositivo.

```
<onentry id="" type="WifiInterrupted" />
<onentry id="" type="WifiOFF" />
<onentry id="" type="WifiON" />
<onentry id="" type="AirPlane" />
<onentry id="" type="Data" />
```

Listagem 4.10: Modelo do sistema exemplo *Sensors*

```
[
        // WifiInterrupted
        driver.setConnection(Connection.NONE);
        driver.setConnection(Connection.WIFI);

        // WifiOFF
        driver.setConnection(Connection.NONE);

        // WifiON
        driver.setConnection(Connection.WIFI);

        // AirPlane
        driver.setConnection(Connection.AIRPLANE);

        // Data
        driver.setConnection(Connection.DATA);
]
```

Listagem 4.11: Código de cada possibilidade *Sensors*

4.4 TOM GENERATOR

Nesta secção será apresentado a criação das classes necessárias para a geração do testes para Android, assim como a alteração na mutação *Slip* e mudanças efetuadas na base de dados e no servidor.

4.4.1 *Geração do Código*

A geração do código para os casos de testes concretos em Android necessitou da criação de várias classes novas no TOM Generator. A compreensão do código do TOM Generator para a criação deste componente de geração acabou por ter o trabalho facilitado devido à implementação de alguns padrões na ferramenta, apresentados na Secção 3.3. A utilização do padrão Abstract Factory aplicado aos algoritmos e geradores permitiu perceber como

e onde deve ser criada a componente. Na Figura 13 temos a criação da componente de geração para Android e Web. De salientar que existe também possibilidade da geração IRIT e JSON.

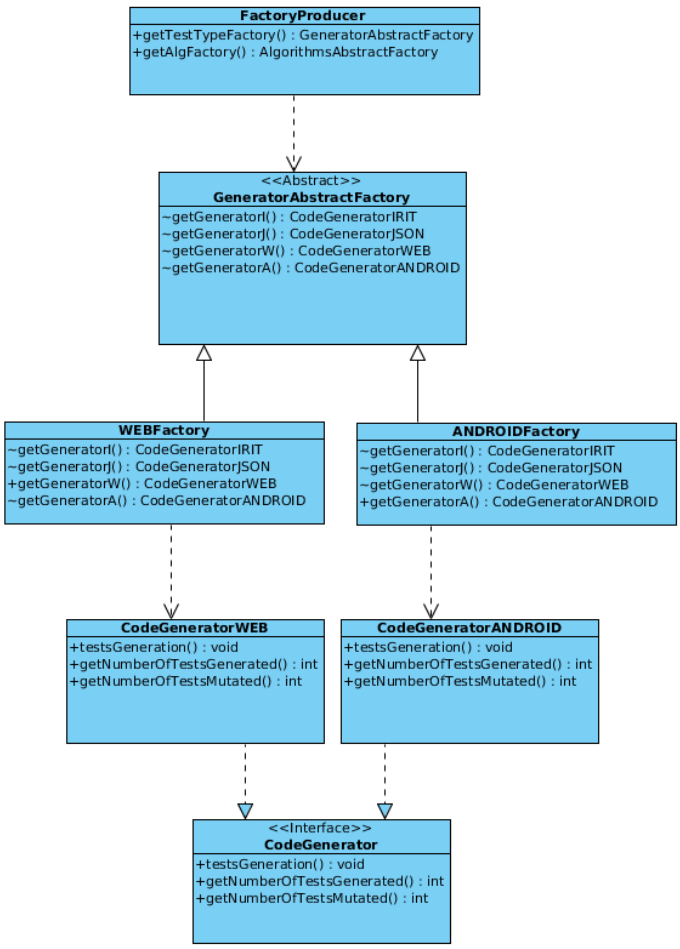


Figura 13.: Criação do gerador para Android no Padrão Abstract Factory aplicado aos algoritmos e geradores

Com a classe CodeGeneratorANDROID criada foi necessário idealizar as restastes classes que irão ajudar na geração do código para Android, utilizando a estrutura usada na geração para Web. Como tal foi necessário a criação de três novas classes:

- TestGeneratorANDROID : Estão concentrados todos os passos necessários para a realização da geração de cada teste, utilizando as restantes classes como apoio a esta geração.

- MutationGeneratorANDROID : Onde é realizada toda geração do código caso surja alguma mutação.
- StaticCodeANDROID : Contém todo o código estático necessario para a execução de cada classe.

Como a ferramenta Appium é baseada em Selenium, durante a criação da geração para Android confirmou-se que alguns dos métodos utilizados na geração do código para Web podiam ser reutilizados para ambos os cenários. Para não termos código duplicado, e de forma a tornar a ferramenta mais genérica, decidiu-se criar uma superclasse chamada TestGenerator com os métodos e variáveis que as classes TestGeneratorWEB e TestGeneratorANDROID tinham em comum. O processo incluiu a introdução de métodos abstratos para garantir uma API igual nas duas classes. A Figura 14 apresenta alguma da arquitetura resultante.

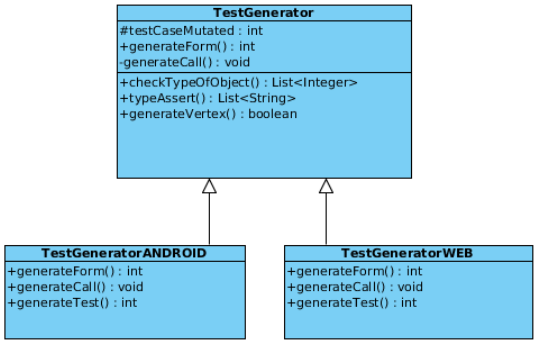


Figura 14.: Representação da superclasse TestGenerator

4.4.2 *Mutação Slip*

A mutação *Slip* realiza uma troca na ordem de execução das ações no formulário. Existem certas interações nas interfaces que são necessariamente sequenciais, como é o caso dos popups. Devido a existência destas ações sequenciais, a mutação Slip acaba por gerar muitos falsos negativos, ao alterar ordens que devem ser sequenciais. Por exemplo, ao gerar testes que tentam carregar primeiro em algo dentro do popup e só depois chamar a ação de abrir o popup. De forma a combater este tipo de falha, foi utilizado o atributo `element` da tag `send` para conter o valor sequenciaClick. Isto permite que o gerador consiga perceber que as interações com aqueles campos tem de ser sequenciais, não podendo haver nenhuma troca com essas ações.

4.4.3 *Servidor*

O servidor criado por Gonçalves (2017) disponibiliza uma API que permite a realização de pedidos para a geração dos vários cenários disponíveis, para a seleção de mutações, registo de utilizadores, etc. Com a adição de suporte para um novo tipo de plataforma na ferramenta TOM é necessário atualizar alguns dos métodos disponíveis pela API. Com esta plataforma novas variáveis foram introduzidas na tabela `generation` da base de dados. Variáveis estas importantes para a realização de pedidos para a geração de testes para Android.

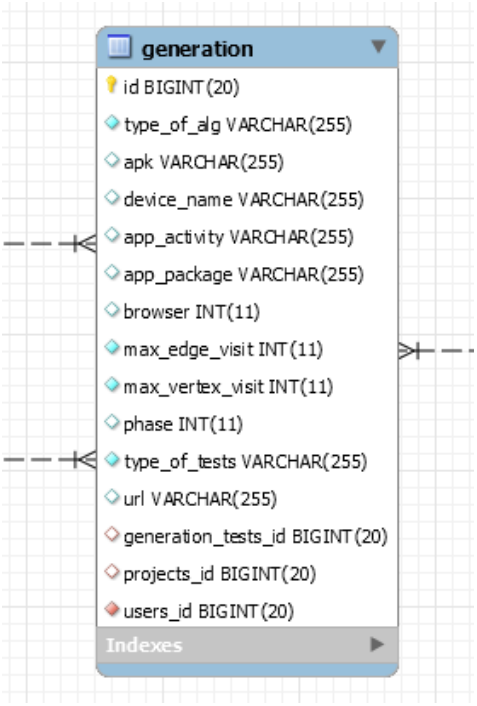


Figura 15.: Representação da tabela `generation`

A Figura 15 contém a estrutura da tabela `generation` onde foram inseridas as variáveis `device_name`, `apk`, `app_package` e `app_activity`. Apesar da adição de 2 novas mutações, não é necessária a inserção de novas variáveis noutras tabelas porque na tabela das mutações a estrutura utilizada apenas utiliza um id e nome da mutação. Como tal, no povoamento da base de dados foram inseridas também as duas novas mutações (*sensors* e *rotate*) no script de povoamento do web services. Toda a estrutura da bases de dados pode ser consultada na Figura 8, apenas foi modificada a tabela apresentada na Figura 15.

Alguns métodos da API sofreram algumas alterações, como é o caso do método `newGenerationRequest` que foi alterado, acrescentado o caso do android. Caso o pedido seja do tipo Android este método verifica se tem os campos necessários para fazer uma geração para Android.

Havia um erro no método `generate` do servidor, este método disponibiliza a geração dos casos de teste de um determinado projeto e no final devolve um zip. Este zip era gerado mas não continha nenhuma informação disponível dentro do mesmo, não sendo então possível depois ter os testes para execução. Como tal, foram efetuadas alterações no método para assim o zip conter todo os casos de teste do projeto selecionado.

4.5 TOM APP

No TOM App a principal alteração ocorreu no acréscimos da geração de testes para Android.

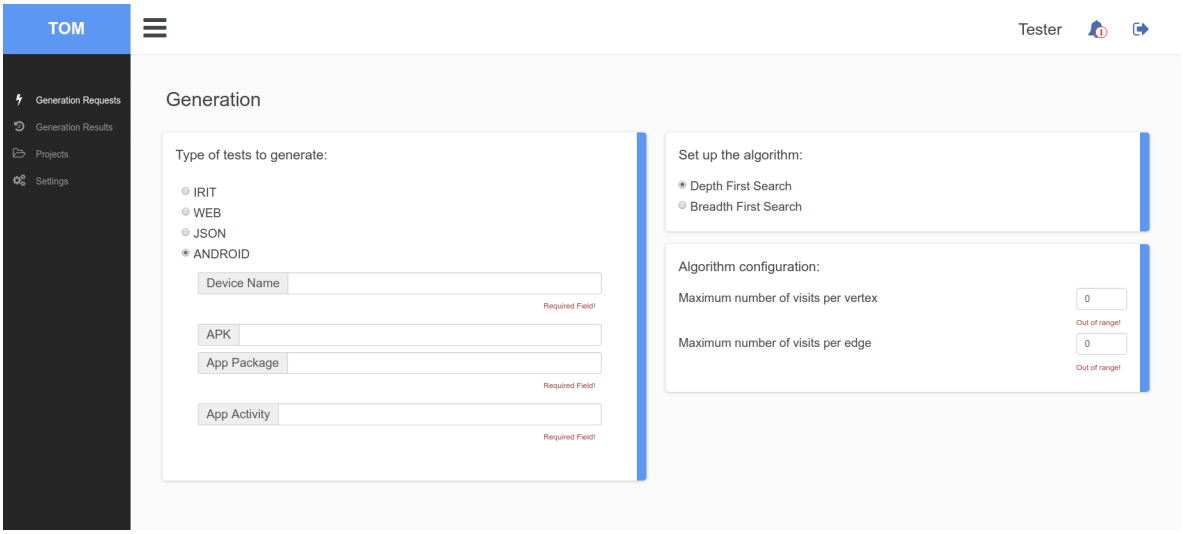


Figura 16.: Pedido de geração Android no TOM App

Na Figura 16 estão representados os quatro campos necessários para essa geração. Esta ferramenta também sofreu algumas alterações de design, tendo ainda sido corrigidos alguns erros menores detetados no HTML e *Cascading Style Sheets* (CSS).

4.6 SUMÁRIO

Neste capítulo fez-se uma descrição da implementação do suporte para Android na ferramenta TOM.

O testNG é utilizado para complementar o Appium, permitindo uma boa organização dos testes e criação de relatórios, tornado mais fácil a visualização de todo o processo e a compreensão de erros durante a execução dos testes.

Foram apresentados os cinco gestos (*Tap, DoubleTap, LongPress, Scroll, Swipe*) utilizados para a interação com as aplicações móveis. Foram ainda descritas as duas novas mutações (*Sensors* e Rotação). Tanto nas mutações como nos gestos, ilustrou-se um exemplo do seu funcionamento, assim como as alterações que foi necessátio efectuar aos vários modelos. Também a mutação *Slip* acabou por sofrer algumas alterações devido ao mau funcionamente que estava a ter.

A criação da componente Android foi ilustrada por forma a perceber o seu enquadramento na ferramenta TOM. No Servidor foram alterados alguns métodos disponibilizados pela API e acrescentadas quatro variáveis (`device_name`, `apk`, `app_package` e `app_activity`) à tabela `generation`. O TOM App sofreu alterações por forma a proporcionar a geração para Android.

5

PROCESSO DE CRIAÇÃO DOS FICHEIROS PARA ANDROID

No Capítulo 3 foi referido que, para a execução do processo de teste baseado em modelos com a ferramenta TOM, é necessária a criação de quatro modelos: modelo do sistema, mapeamentos, valores e mutações. Estes modelos podem ser criados de duas formas, a primeira é a criação manual dos modelos e segunda a criação automática através de uma ferramenta. Para suportar a criação automática seria necessário desenvolver uma ferramenta como o TOM Editor, mas direcionada para o Android, para a extração de informação da aplicação móvel. Apesar de no futuro tal trazer vantagens, devido ao tempo que seria poupado na criação dos modelos, esta opção não pode ser explorada durante esta dissertação, devido ao curto período de tempo disponível. Como tal, será utilizada apenas a criação manual. Neste capítulo, o processo de criação dos modelos é explicado.

5.1 FERRAMENTA UTILIZADA

A criação manual deste tipo de modelos é feita, apesar de tudo, com a ajuda da ferramenta UI Automator Viewer que providencia uma captura de ecrã da atividade a correr no dispositivo. Na ferramenta conseguimos selecionar cada um dos componentes presente naquela atividade, conseguindo obter varias informações sobre os mesmos.

Na Figura 17, apresenta-se uma captura de ecrã da ferramenta UI Automator Viewer. Do lado esquerdo da figura podemos observar a atividade que esta a ser visualizada no ecrã do dispositivo sob monitorização, no canto superior direito está representa toda a hierarquia da atividade e por baixo desta são apresentadas as propriedades (id, class, índice, etc) do objeto selecionado na atividade. Neste caso, foi selecionada a caixa de texto sinalizada a vermelho. Para descrever um resumo do processo da construção dos modelos vai ser utilizada esta mesma atividade como exemplo.

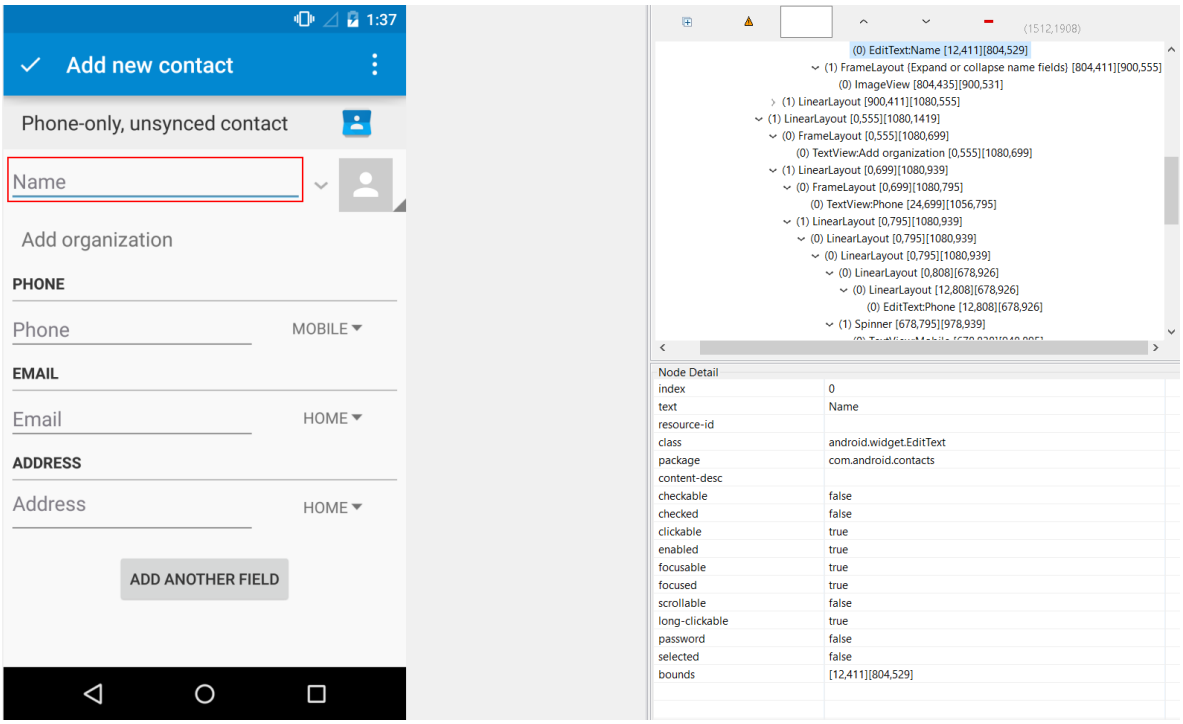


Figura 17.: Propriedades de um elemento/objeto na atividade, utilizando o UI Automator Viewer

5.2 Criação do Modelo de sistema

Para cada atividade da aplicação é criado um novo estado e para cada formulário um sub-estado. Na Figura 18 foram criados dois estados para representar a actividade apresentada na Figura 17. O primeiro, identificado pelo identificador "o" (id="o") representa o estado da atividade. O segundo, identificado por "Add Contact" (id="Add Contact") representa o formulário existente na atividade, logo é sub estado do estado "o". Caso a atividade tivesse vários formulários, seria adicionado um sub estado para cada formulário. Dentro do estado que representa o formulário foram definidas duas *labels*, a primeira representa o nome do contacto a ser adicionado e a segunda o seu número de telefone. No mesmo formulário temos uma transição do tipo `submit` que permite adicionar o contacto, fazendo uma transição para o estado 3 representado na Figura 19. Depois do contacto estar adicionado, vamos verificar (`onexit`) se ele foi adicionado. Isto é efetuado através da validação do conteúdo da *label* apresentada na Figura 19, a vermelho, com o texto "Tester", confirmando assim que o contacto foi adicionado. Nesta fase não precisamos ainda da ferramenta UI Automator Viewer para a idealização do modelo.

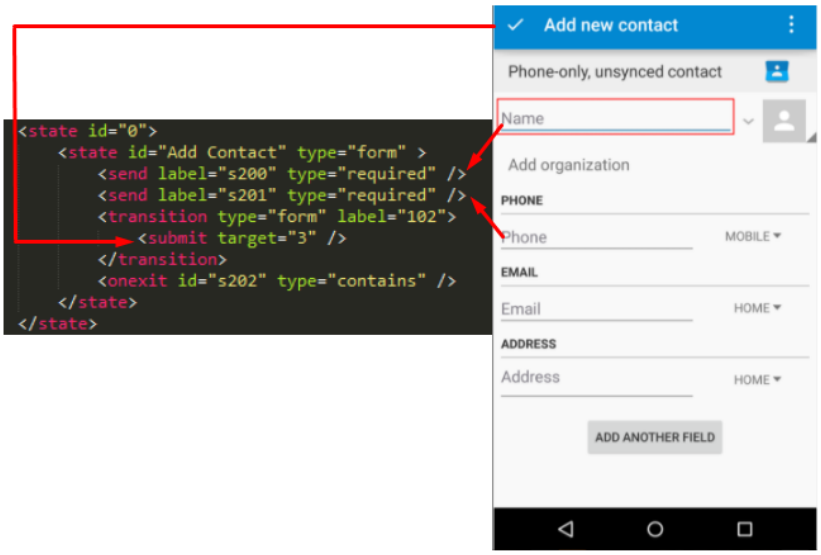


Figura 18.: Exemplo da Criação do Modelo do sistema

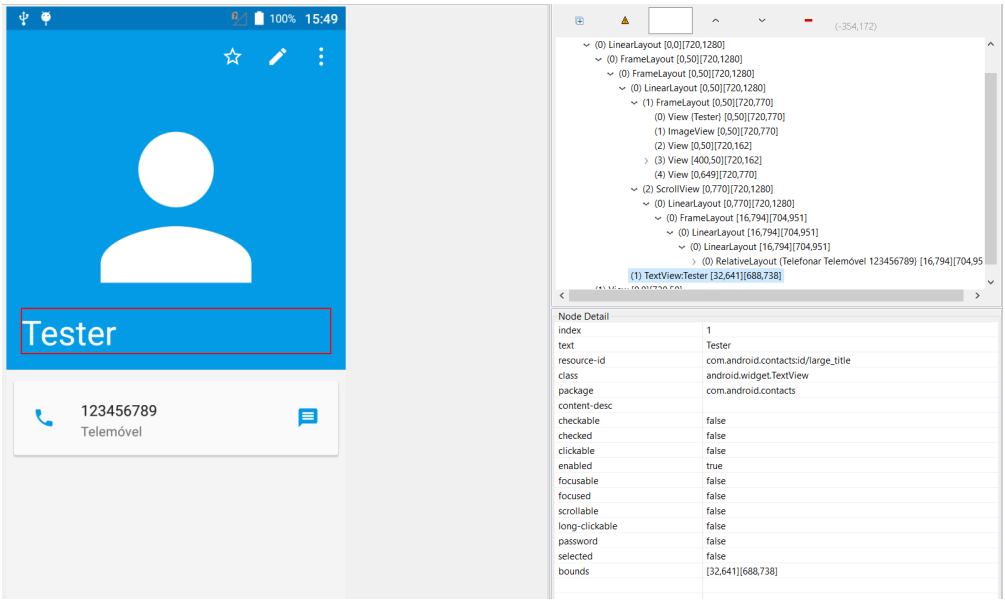


Figura 19.: Estado 3 com a identificação do campo a validar

5.3 Criação do modelo de Mapeamento e Valores

Simultaneamente com a criação do modelo do sistema, é necessário fazer o modelo de mapeamento e valores. O modelo de valores é criado conforme o que é necessário introduzir nos diferentes campos dos formulários. O objetivo é definir valores que permitam garantir uma boa cobertura da aplicação sob teste. Esse aspeto está fora do âmbito da dissertação e

não será abordado aqui.

O modelo de mapeamento é importante para operacionalizar a execução dos casos de teste. Permite definir que ação deve ser realizada e como encontrar os elementos na atividade onde ela deve ser realizada. Como tal, para encontrar esses elementos é necessário a ajuda do UI Automator Viewer. Os elementos podem ser encontrados através do XPath, resource-id, classe, etc, do elemento. Tudo menos o XPath é obtido diretamente das propriedades do elemento (canto inferior direito da Figura 17). Quanto ao XPath pode ser obtido de uma das seguintes formas:

- Através da classe e do texto do atributo. Exemplo: "//android.widget.EditText[@text='Name']", em que o valor "Name" será o texto representado no campo `text` da Figura 20 e o "android.widget.EditText" está no campo `Class`.
- Através da classe e do id. Exemplo: "//android.widget.EditText[contains(@resource-id,'identificador')]", em que o identificador será o valor representado no campo `resource-id`.
- Através da classe, do texto do atributo e do id. Exemplo: "//android.widget.EditText[contains(@resource-id,'identificador') and @text='Name']".
- Através da classe, do texto do atributo e do índice. Exemplo: "//android.widget.EditText[@text= 'Name' and @index='0']", em que o índice esta representado no campo `index`.
- Através do content-desc. Exemplo: "//android.widget.EditText[@content-desc='conteudo']", em que será colocado no conteúdo o que estiver no campo `content-desc`.
- Por recurso à hierarquia de classe pai e filho. Neste caso poderemos usar todos os métodos anteriores para ir descendo na hierarquia até ao elemento que queremos. Exemplo: "//android.widget.LinearLayout[@index='0']/android.widget.EditText[@index='0']".

Na Figura 20 está identificado, com uma seta, o caminho necessário para selecionar o elemento NAME. Continuando com o exemplo utilizado anteriormente na Figura 18, podemos ver que o elemento NAME é selecionado através do XPath criado, neste caso, com base em varias hierarquias de classe pai e filho, utilizando índices e ids. Traduzindo isto em algo concreto, na Listagem 5.1, no atributo `what_to_find` temos a especificação de todo o caminho que é necessário percorrer para selecionar este elemento. No atributo `how_to_find`, a indicação que a identificação é efetuada utilizando XPath e no atributo `what_to_do` a indicação de que deverá ser feito *input* de texto. Por sua vez, a Listagem 5.2 contém o nome a dar ao contacto a adicionar. Ou seja, o valor que vai ser introduzido no elemento NAME.

A associação entre o valor e o elemento onde ele será colocado é feita pelo identificador (neste caso "s200").

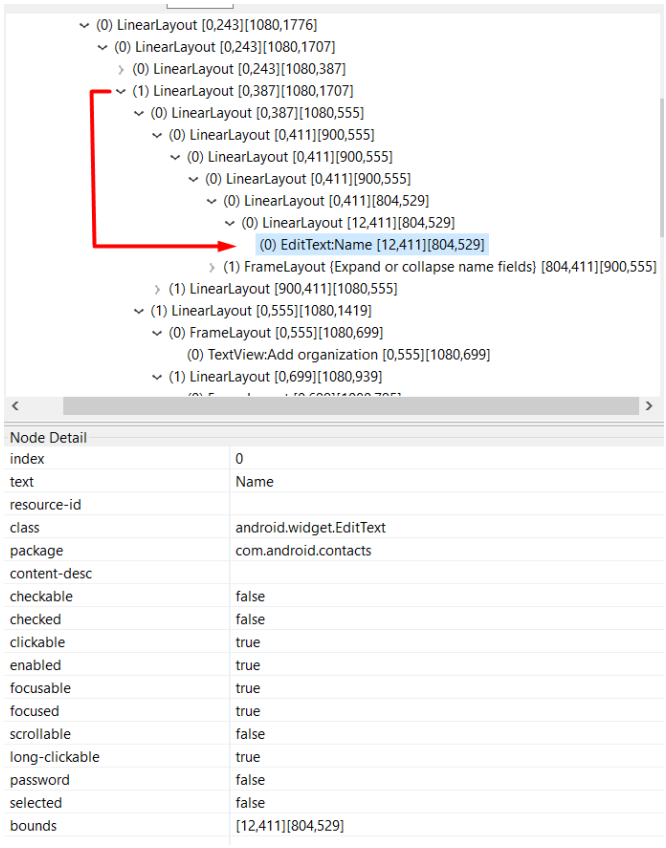


Figura 20.: Exemplo da hierarquia do elemento `NAME`

```
[
  "s200": {
    "how_to_find": "xpath",
    "what_to_find": "//android.widget.LinearLayout[contains(@resource-id,'body
        ')]/android.widget.LinearLayout[@index='0']/android.widget.
        LinearLayout[@index='0']/android.widget.LinearLayout[@index='0']/
        android.widget.LinearLayout[@index='0']/android.widget.LinearLayout[
        @index='0']/android.widget.LinearLayout[contains(@resource-id,'editors
        ')]/android.widget.EditText[@index='0']",
    "what_to_do": "sendKeys"
  }
]
```

Listagem 5.1: Mapeamento do Elemento NAME

```
{ "s200" : "Tester" }
```

Listagem 5.2: Valor do Elemento NAME

Um exemplo de realização do mapeamento através do identificador é o click no botão de adicionar um novo contacto. Na Figura 21 está representada a hierarquia do botão. Como podemos ver na imagem, este elemento tem um identificador único (`resource-id`) que será representado na Listagem 5.3 no campo `what_to_find` como sendo a forma de selecionar esse elemento para realizar o click através do id.

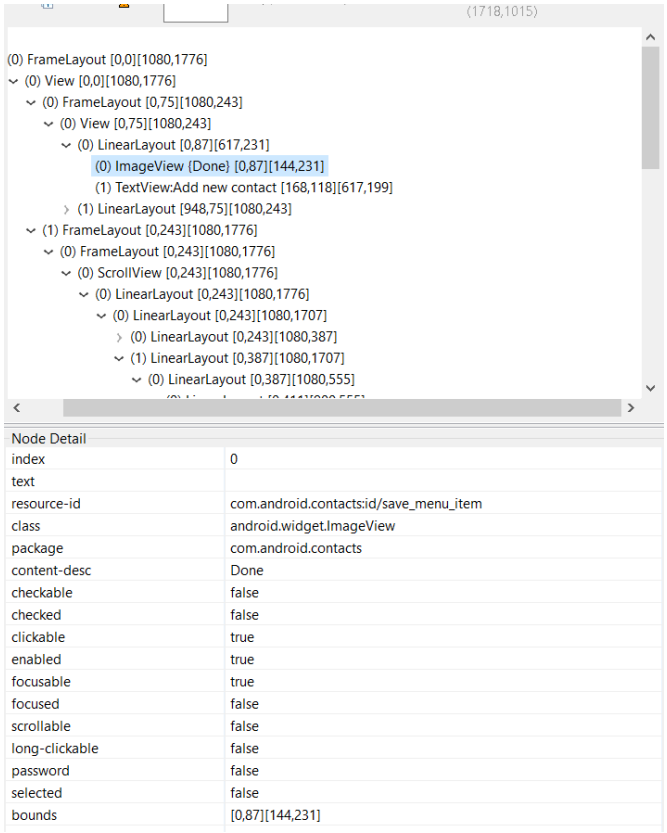


Figura 21.: Exemplo da hierarquia do botão adicionar novo contacto

```
[
  "102": {
    "how_to_find": "id",
    "what_to_find": "com.android.contacts:id/save_menu_item",
    "what_to_do": "click"
    },
]
```

Listagem 5.3: Mapeamento do botão adicionar novo contacto

5.4 CRIAÇÃO DO MODELO MUTAÇÕES

Finalmente, a criação do modelo de mutações é realizada da mesma forma que nas paginas Web, apesar de neste caso ser realizada manualmente. Na Listagem 5.4 temos um exemplo de uma mutação mistake para o elemento NAME. Aplicando esta mutação, o nome a introduzir, em vez de ser "Tester" (valor por omissão definido na Listagem 5.2) passará a ser "Tester12345", não sendo suposto que o teste falhe (atributo fail a zero).

```
[
    {
    "type" : "mistake",
    "model_element": "s200",
    "value" : "Tester12345",
    "fail" : "0"
  },
]
```

Listagem 5.4: Mutação do tipo Mistake no Elemento NAME

5.5 SUMÁRIO

Ao longo deste capítulo explicou-se o processo manual de criação de modelos e mapeamentos que vai ser utilizado para os casos de estudo. Este processo é realizado com a ajuda da ferramenta UI Automator Viewer, que providencia uma captura de ecrã da atividade a correr no dispositivo. Nessa atividade conseguimos obter toda a informação de cada elemento presente na mesma, o que permite então obter o conteúdo necessário para a criação dos vários Modelos.

6

CASO DE ESTUDO

Neste capítulo pretende-se demonstrar a aplicação da ferramenta a aplicações Android. O principal foco nos casos de estudo é verificar se é possível modelar aplicações reais e se a ferramenta consegue criar e executar os testes, bem como detetar inconsistências entre o comportamento 'previsto' e o real. Na aplicação destes casos de estudo também se pretende avaliar as mutações, em particular as mutações *Double Click* e *Slip*, uma vez que estas apresentavam problemas na geração e execução Web. Todos os modelos serão criados manualmente, utilizando-os no TOM Generator para gerar e executar casos de teste, sendo no final analisados e discutidos os resultados obtidos.

Para executar os testes gerados é necessário preparar um ambiente. O ambiente utilizado para a execução dos testes nesta dissertação contém as seguintes configurações:

- Um projeto Gradle (sistema de automatização de builds de compilação).
- Os testes gerados pelo TOM Generator.
- O Appium.
- O TestNG Suite para gerir a execução dos testes.

O dispositivo utilizado para a realização dos testes tem as seguintes características: Huawei P smart com o Android 8.0, contendo um processador 2.36Ghz Octa Core. Foram testadas duas aplicações descarregadas da Play Store da Google. A primeira (Minhas Notas) tinha como objetivo verificar se a ferramenta esta a funcionar. A segunda (Gastos Diários 2) um caso com mais formulários de forma a verificar se todas as mutações estavam a ser aplicadas.

6.1 APLICAÇÃO MINHAS NOTAS

Como referido acima, o primeiro caso de estudo foi efetuado sobre a aplicação Minhas Notas[1], disponibilizada na Play Store, na versão 1.8.6. A aplicação funciona como um bloco de notas mas em formato digital permitindo aos utilizadores apontar as suas notas para mais tarde recordar. Este exemplo é simples, sendo essencialmente utilizado para verificar se a ferramenta está a funcionar como pretendido. Na Figura 22 esta representada a janela inicial da aplicação.

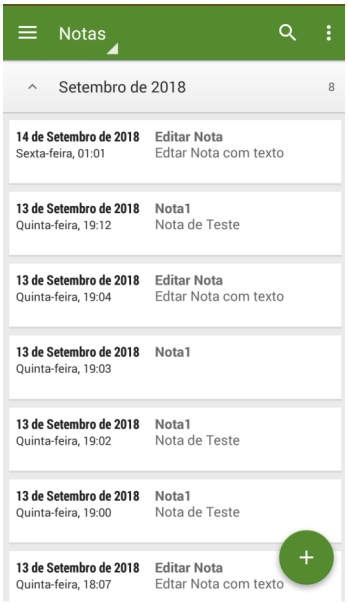


Figura 22.: Janela inicial da aplicação Minhas Notas

6.1.1 *Modelo do Sistema*

A modelação como já referido anteriormente, no Capítulo 5, é criada manualmente. Foi efetuada a navegação pela aplicação e para cada nova atividade foi criado um estado, contendo toda a informação considerada relevante nessa atividade.

A navegação efetuada pela aplicação levou ao desenvolvimento de um modelo de sistema apresentado na Figura 23. O Modelo contém 13 estados, destes apenas 2 contém formulários. Existem validações no modelo que não são apresentadas na imagem para simplificar

[1] https://play.google.com/store/apps/details?id=net.kreosoft.android.mynotes last accessed: 2018-08-17

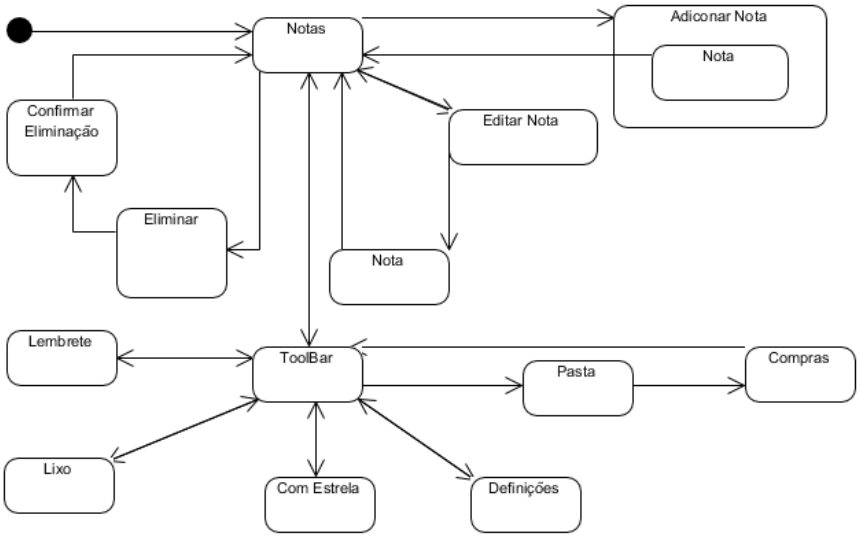


Figura 23.: Modelo do Sistema da aplicação Minhas Notas

a interpretação do mesmo. O modelo foi efetuado manualmente demorando cerca de 15 horas a ser totalmente construído e validado.

6.1.2 *Pedido de Geração*

Construídos os quatro modelos apresentados no Anexo A (Modelo do sistema, mapeamento A.2, valores A.3 e mutações A.4), estes foram então importados para o TOM App para ser efetuado um pedido de geração de testes. Neste pedido de geração utilizaram-se as seguintes configurações:

- Tipo de de teste gerado: Android
- Algoritmo: Depth-First Search
- Configuração do Algoritmo:

  Máximo número de visitas por vértice: 1

  Máximo número de visitas por aresta: 1
- Todo o tipo de mutações foi selecionado.

A Figura 24 apresenta o pedido de geração. Uma vez o pedido de geração feito, a geração dos resultados demorou cerca de 1 segundo a ser concluída. O resultado dessa geração esta apresentado na Tabela 2.

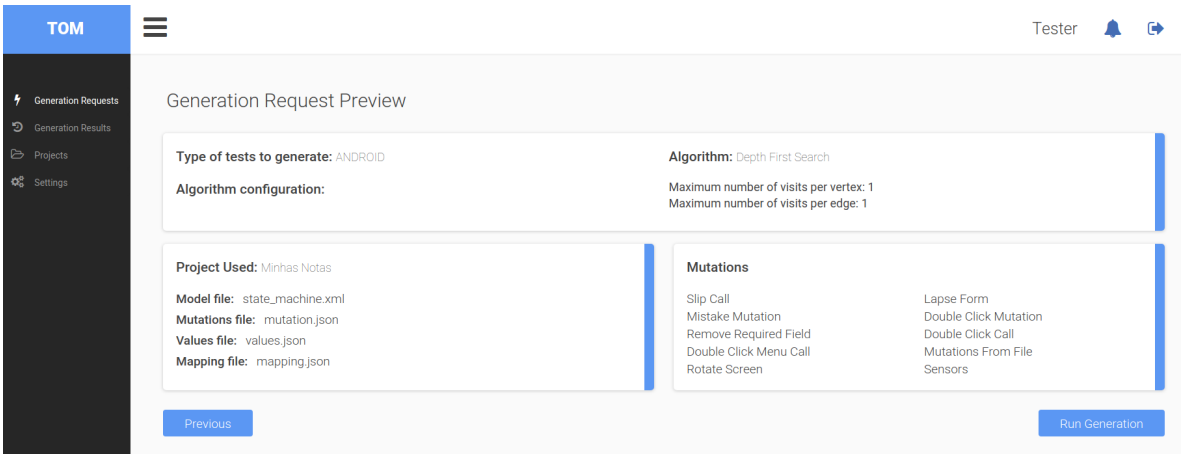


Figura 24.: TOM App: Pré-visualização do pedido de geração

Tabela 2.: TOM App: Resultado da Geração para a aplicação Minhas Notas

Generation Result	
Número de Paths:	10
Número de Teste Normais	10
Número de Testes Mutados	38
Número de Testes Gerados	48
Tempo de Geração em Milisegundos	1115

6.1.3 *Análise de Execução*

Efetuada a geração dos testes os mesmos foram executados. O método usado para executar os testes foi explicado no inicio deste Capítulo. Os testes levaram 55 minutos a serem executados, 6 falhas foram encontradas em 48 testes. A Tabela 3 apresenta os resultados da execução do teste.

Cerca de 88% dos testes foram bem sucedidos, mas é desde logo notório que as falhas ocorridas foram nos testes *Double Click Call* e *Double Click Mutation*. De forma a entender estas falhas os testes foram analisados.

A mutação *Double Click Call* obteve 50% dos testes falhados. Depois de efetuada uma analise dos testes e execução dos mesmo novamente, reparamos que os testes que falharam alguns passaram a funcionar e outros que funcionavam passaram a falhar. Desta análise concluímos que estava a acontecer o mesmo que na parte Web, apresentada no caso de

Tabela 3.: Minhas Notas: Resultados da Execução

Tipo de Teste	Nº de Testes	Nº de Testes Falhados	% de Testes Corretos
Normal	10	0	100%
Lapse	2	0	100%
Mistake	2	0	100%
Slip	2	0	100%
Sensors	4	0	100%
Remove Required Field	2	0	100%
Rotate	4	0	100%
Double Click Call	10	5	50%
Double Click Mutation	10	1	90%
Mutation From File	2	0	100%
Total	48	6	88%

estudo do Gonçalves (2017). O tempo de resposta do dispositivo pode variar, sendo umas vezes mais rápido do que noutras, o que faz com que o segundo click por vezes seja processado de forma errada fazendo o teste falhar, levando então a que os teste tenham falsos negativos. Com isto, podemos ver que este erro, que ocorria já na parte Web, também acaba por acontecer na parte Android.

A mutação *Double Click Mutation* apenas tem um teste falhado. O erro acaba por ser igual ao explicado na mutação *Double Click Call*.

6.1.4 *Conclusão*

No final obteve-se um bom resultado com 88% dos testes corretos, sendo que os *Double Click* acabam por ser falsos negativos que já ocorriam em Web, logo retirando estes falsos negativos acabamos por ter 100% dos testes corretos. Também foi visível que a alteração efetuada no melhoramento da mutação *Slip* acabou por ter o funcionamento esperado e não apresenta mais falsos negativos como acontecia na parte Web. No próximo caso de estudo vamos analisar uma aplicação um pouco mais complexa e que contenha estados de erro, visto que esta aplicação não continha nenhum estado de erro.

6.2 APLICAÇÃO GASTOS DIÁRIOS 2

O presente caso de estudo é efetuado sobre a aplicação Gastos Diários 2[2], também disponibilizada na Play Store, na versão 2.6.62. Esta aplicação permite aos utilizadores controlarem os gastos e despesas diários/mensais que vão ocorrendo no dia a dia. Esta aplicação permite adicionar despesas da gasolina, da renda, etc. e também adicionar ganhos como salário, vendas e empréstimos. Na Figura 25 esta representada a janela principal da aplicação.

Figura 25.: Janela principal da aplicação Gastos Diários 2

6.2.1 *Modelo do Sistema*

Durante a navegação na aplicação foram detetados muitos erros que poderiam originar problemas durante a execução dos testes. O erro mais comum foi a utilização, na mesma atividade, do mesmo identificador (atributo id) em diferentes widgets, o id deve ser único para cada widget da mesma atividade. Uma vez que estes erros foram detetados logo na fase de construção do modelo, e para evitar que tivessem impacto no processo de teste, atrasando-o, decidiu-se nestes casos optar pela identificação dos componentes utilizando XPath ou *Offset*. Acabamos por decidir corrigir já este tipo de erro para ele não aparecer na execução dos testes, mas caso prosseguíssemos com estes erros a própria ferramenta TOM iria, por norma, detetar os erros na fase de execução dos testes, porque iria acabar

[2] https://play.google.com/store/apps/details?id=mic.app.gastosdiarios_clasico last accessed: 2018-08-24.

por tentar efetuar uma ação no elemento errado (Pinto, 2017). Isto mostra que o próprio processo e construção do modelo permite encontrar problemas, sendo este erro detetado com a ajuda do UI Automator Viewer, tal como explicado no Capítulo 5.

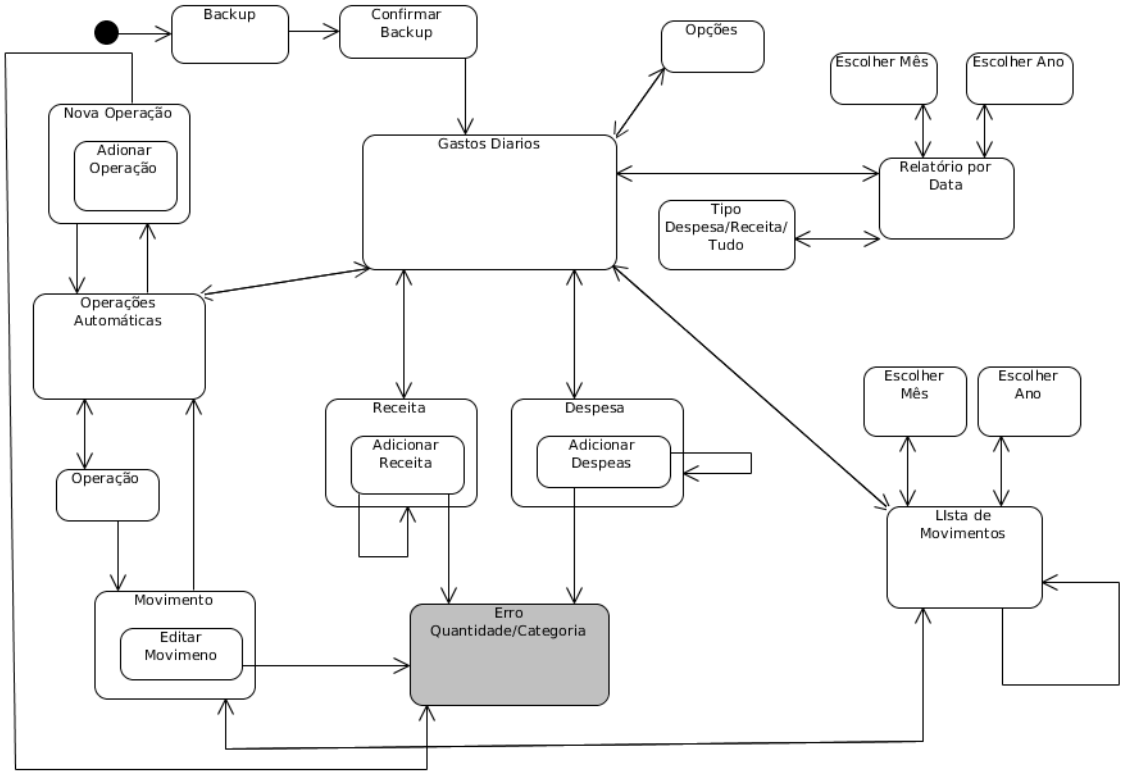


Figura 26.: Modelo do Sistema da aplicação Gastos Diários 2

A Figura 26 apresenta a máquina de estados que descreve o modelo de sistema desenvolvido. Este modelo contém 18 estados, um deles representa um estado de erro (representado a cinzento). O modelo contém 4 sub-estados que representam os formulários e como tal esses têm transições para o estado de erro caso falhem. Existem validações no modelo que não são apresentadas na imagem para simplificar a sua interpretação. Este modelo foi efetuado manualmente demorando 36 horas a ser totalmente construído e validado.

6.2.2 *Pedido de Geração*

Tal como no caso anterior, depois de completados, os quatro modelos apresentados no Anexo B (Modelo do sistema, mapeamento B.2, valores B.3 e mutações B.4) foram importados para o TOM App para executar o pedido de geração de testes. Neste pedido de geração utilizaram-se as mesmas configurações:

- Tipo de de teste gerado: Android
- Algoritmo: Depth-First Search
- Configuração do Algoritmo:

  Máximo número de visitas por vértice: 1

  Máximo número de visitas por aresta: 1
- Todo o tipo de mutações foi selecionado.

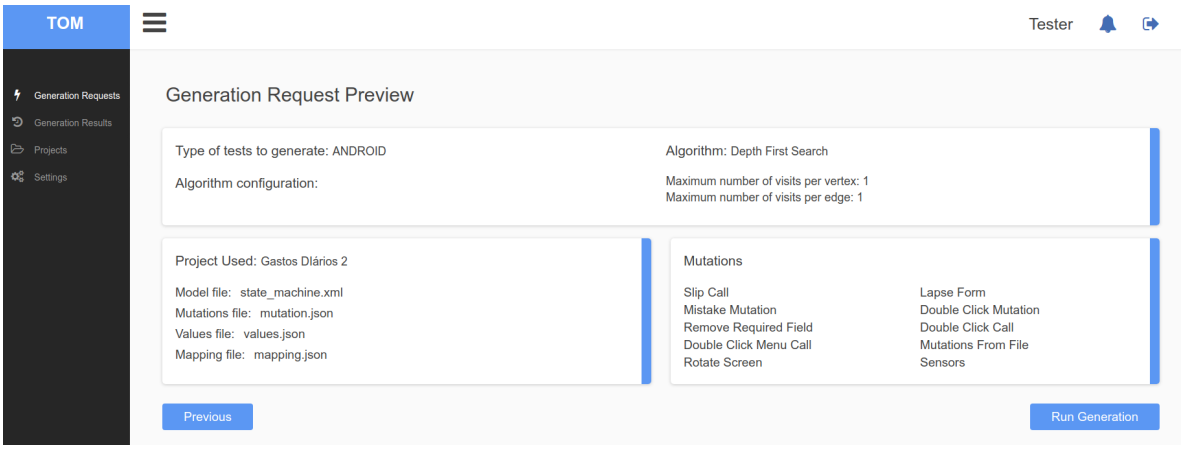


Figura 27.: TOM App: Pré-visualização do pedido de geração

A Figura 27 apresenta o pedido de geração. Uma vez o pedido de geração feito, a geração dos resultados levou cerca de 13 segundos. O resultado dessa geração está apresentado na Tabela 4.

Tabela 4.: TOM App: Resultado da Geração para a aplicação Gastos Diários 2

Generation Result	
Número de Paths:	21
Número de Teste Normais	21
Número de Testes Mutados	82
Número de Testes Gerados	103
Tempo de Geração em Milisegundos	12623

6.2.3 *Análise de Execução*

Após a geração dos testes estar concluída os testes foram executados. Os testes levaram cerca de 2 horas e 45 minutos a serem executados, 33 falhas foram encontradas em 103 testes. A Tabela 5 apresenta os resultados da execução do teste.

Tabela 5.: Gastos Diários 2: Resultados da Execução

Tipo de Teste	Nº de Testes	Nº de Testes Falhados	% de Testes Corretos
Normal	21	0	100%
Lapse	5	3	40%
Mistake	5	2	60%
Slip	5	0	100%
Sensors	5	0	100%
Remove Required Field	5	0	100%
Rotate	10	10	0%
Double Click Call	21	14	33%
Double Click Mutation	21	4	81%
Mutation From File	5	0	100%
Total	103	33	68%

É desde logo vísivel que nenhum dos 21 testes normais falhou. Todas as falhas ocorram nas mutações, sendo o balanço total de 68% dos testes bem sucedidos. Logo, 32% falharam o que acabava por ser uma margem considerável de testes falhados. De forma a perceber o porque de uma percentagem tão alta, os testes foram analisados.

Analisando os testes que falharam com a mutação *Lapse* repara-se que existem 3 falsos negativos, em 2 dos testes trata-se de edições em movimentos de adicionar receita/despesa (por exemplo quando queremos editar uma receita adicionada ontem) e como estamos a editar todos os campos já foram preenchidos anteriormente. Num teste normal esta edição acontece em todos os campos, neste caso como é uma mutação um dos campos para editar é selecionado para não ser efetuado. A ferramenta acabou por escolher um campo opcional para ser retirado do formulário, mas como não existe a mensagem de erro para caso campos opcionais falhem, a ferramenta diz que o teste falha porque esta a espera da mensagem erro. No outro caso é uma edição também mas é escolhido um campo *required* para não ser realizado na edição, mas como este campo já tem um valor preenchido anteriormente

quando adicionamos a receita/despesa é normal não obter erro pois essse campo continua preenchido.

A mutação *Mistake* acaba por ser um falso negativo. Na Listagem 6.1 temos o código de um caso de teste que falhou. Como podemos ver, a mutação *Mistake* foi inserida na linha 4 com o valor 111cebstehcxw que representa o campo da quantidade, normalmente este valor devia dar erro mas a própria aplicação impede que sejam introduzias letras ficando apenas com os números inseridos fazendo o teste passar quando supostamente devia falhar, logo a própria aplicação neste caso tem um mecanismo que só aceita números naquele campo.

```
1 [
2 try {
3         Reporter.log(" Mutation Mistake <br>");
4         driver.findElement(By.id("mic.app.gastosdiarios_clasico:id/editExpense
              ")).sendKeys("111cebstehcxw");
5         driver.findElement(By.id("mic.app.gastosdiarios_clasico:id/
              spinnerCategory")).click();
6         driver.findElement(MobileBy.AndroidUIAutomator("new UiScrollable(new
              UiSelector().scrollable(true).instance(0)).scrollIntoView(new
              UiSelector().textMatches(\"Pessoal\"));")).click();
7         driver.findElement(By.id("mic.app.gastosdiarios_clasico:id/editDetail"
              )).sendKeys("compras da semana");
8         Reporter.log("[Pass]: Form Filled Adicionar Despesa <br>");
9         gonnaFail = 1;
10    }
11 ]
```

Listagem 6.1: Gastos Diários 2: Código gerado pela mutação *Mistake*

Os testes com as mutações *Double Click Call* e *Double Click Mutation* acabam por ter o mesmo problema explicado no caso de estudo anterior.

Os testes com a mutação *Rotate* falharam todos porque a aplicação não permite ao utilizador rodar, como tal, no momento que o Appium obriga a aplicação a rodar a mesma deixa de funcionar.

6.2.4 *Conclusão*

Depois de analisado os resultados, os falsos negativos foram manualmente corrigidos, e os testes novamente corridos. Na Tabela 6 foram apresentados os resultados dos testes executados depois de algumas alterações.

Tabela 6.: Gastos Diários 2: Execução Final

Tipo de Teste	Nº de Testes	Nº de Testes Falhados	% de Testes Corretos
Normal	21	0	100%
Lapse	5	0	100%
Mistake	5	0	100%
Slip	5	0	100%
Sensors	5	0	100%
Remove Required Field	5	0	100%
Rotate	10	10	0%
Double Click Call	21	13	38%
Double Click Mutation	21	4	81%
Mutation From File	5	0	100%
Total	103	27	74%

Como no caso de estudo anterior os *Double Click* têm o problema do tempo de resposta que acontecia já em Web e acabam por ser também falsos negativos, apenas os testes *Rotate* falham completamente. Todas as restantes mutações acabaram por passar assim como os testes normais. No final como ilustrado na Tabela 6, 74% dos testes executados terminaram com sucesso, se consideramos que as mutações do *Double Click* não são testes falhados ficamos com mais de 90% dos testes concluídos com sucesso.

Apesar do número de testes realizados não ser muito grande conseguimos concluir que a geração de testes para Android esta a funcionar. Quantos mais formulários a aplicação tiver maior o número de mutações a ser realizado, porque a maioria das mutações apenas aplica-se aos formulários como tal é necessária uma grande presença dos mesmos. Mas existe uma grande dificuldade a encontrar Aplicações que contenham tantos formulários como as aplicações Web.

6.3 Sumário

Durante este capítulo foi demonstrado a aplicação da ferramenta TOM as aplicações Android. Sendo o objetivo principal verificar se o funcionamento da componente Android está a funcionar como esperado nas aplicações Android. Foi possível mostrar que é possível modelar aplicações e que a ferramenta gerou e executou testes. Também detetou as

inconsistências, embora nesse caso a maioria se devesse ás mutações introduzidas. As mutações *Double Click* continuam com o mesmo problema que acontecia na parte Web, sendo necessário no futuro tentar resolver este problema. Ambas as aplicações testadas não continham nenhum erro imediatamente identificável, mas foi possível detetar dois problemas na segunda aplicação. Por um lado, problemas com a qualidade da implementação, relacionados com os identificadores, por outro, a ausência da possibilidade de rodar a aplicação quando se roda o ecrã.

Relativamente às mutações torna-se claro que a sua geração automática levanta muitos problemas de falsos negativos (e eventualmente de falsos positivos), pela que deverá ser considerada a possibilidade de restringir a sua utilização a cenários de utilização manual.

7

CONCLUSÃO E TRABALHO FUTURO

O trabalho desenvolvido nesta dissertação acrescentou suporte para Android à ferramenta MBT TOM. Permitindo assim, que a ferramenta consiga gerar e executar testes para as aplicações Android.

Este capítulo está divido em duas secções. Na primeira é realizada uma análise ao trabalho desenvolvido no decorrer da dissertação e dos resultados obtidos. Na segunda é abordado o trabalho futuro, com algumas sugestões de melhorias e novos desenvolvimentos que possam futuramente integrar a ferramenta TOM.

7.1 CONTRIBUTOS

O principal objetivo da dissertação era desenvolver uma componente de teste de aplicações Android para a ferramenta TOM, aumentando a qualidade das aplicações móveis mas também o conjunto de cenários nos quais a ferramenta pode ser utilizada.

A arquitetura da ferramenta adota uma solução modular de forma a torná-la mais flexível e adaptável a diferentes contextos. A adição deste novo componente para Android permitiu comprovar que a modularidade existente no TOM Generator é uma mais valia, conseguindo-se uma reutilização bastante substancial das funcionalidades existentes. A componente Android permite que a ferramenta TOM suporte mais um cenário de geração/execução de testes, juntando-se aos três já existentes (IRIT, JSON, Web).

A ferramenta utilizada para a geração dos casos de teste foi o Appium, uma ferramenta *cross-plataform* que permite a realização de testes em aplicações Nativas, Híbridas e Web. Logo, permitirá no futuro expandir o TOM para novos cenários, por exemplo, aplicações iOS, reaproveitando o que neste momento está efetuado para Android. O testNG é utilizado para complementar o Appium permitindo uma boa organização dos testes e criação de relatórios, tornado mais fácil a visualização de todo o processo e a compreensão de erros

durante a execução dos testes.

Na componente Android foram introduzidos cinco gestos (*Tap, DoubleTap, LongPress, Scroll, Swipe*) considerados relevantes na interação com as aplicações móveis. Duas novas mutações (*Sensors* e Rotação) foram adicionadas especificamente para a geração de testes para esta componente. A mutação *Slip* sofreu algumas alterações devido ao mau funcionamento que estava a ter na geração Web. As alterações efetuadas no TOM Generator e no TOM App foram apresentadas ao longo de toda a dissertação.

Por fim, através dos casos de estudo, foi possível mostrar que é possível modelar aplicações Android com a abordagem seguida no TOM e que a ferramenta gerou e executou os testes com sucesso. Nos dois casos de estudo verificou-se que o funcionamento dos testes estava como esperado. No que diz respeito à execução dos testes, os *Double Click*, apesar de tudo, continuam com o mesmo problema encontrado na execução de testes Web, pelo que será necessário no futuro rever também esta mutação. O *Slip* depois da alteração efetuada não apresentou qualquer tipo de erro nos testes realizados nas duas aplicações utilizadas como caso de estudo. Durante a construção dos modelos consegui-se encontrar também erros na definição dos identificadores dos componentes das interfaces, o que nos permite afirmar que tanto na modelação como na geração e execução de testes é possível detetar erros nas aplicações.

7.2 TRABALHO FUTURO

Durante o desenvolvimento desta dissertação foram identificadas algumas limitações e novas funcionalidades que podem ser tidas em conta para o melhoramento da ferramenta TOM. Desta forma, são propostas as seguintes alterações:

- Melhoria das mutações *Double Click* por forma a não serem obtidos tantos falsos negativos.
- Geração semiautomática dos modelos para Android, através da criação de um TOM Editor para Android, permitindo assim reduzir o custo e tempo perdido na modelação dos modelos.
- Melhoramento da versão Web do TOM Editor, por exemplo, os formulários não são identificados se no HTML não estiverem definidos como um *form* e a geração da linha inicial do modelo do sistema deverá sofrer alterações, de forma a identificar o nodo inicial, para assim evitar a necessidade de edição manual desse campo.

- Restringir a geração de mutações de formulário apenas a um formulário por caso de teste. Logo, caso um caso de teste tenha mais que um formulário, será necessário desdobra-lo para gerar um caso de testa para cada formulário mutado.
- Aumentar o número de gestos para Android caso o Appium contenha suporte.
- Como o Appium é uma ferramenta *cross-plataform*, criar uma componente para iOS e também implementar suporte para aplicações Híbridas, Web.

BIBLIOGRAFIA

Domenico Amalfitano, Anna Rita Fasolino, Porfirio Tramontana, Salvatore De Carmine, and Atif M. Memon. Using GUI ripping for automated testing of Android applications. *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering - ASE 2012*, page 258, 2012. ISSN 02705257. doi: http://dx.doi.org/10.1145/2351676.2351717.

Domenico Amalfitano, Anna Rita Fasolino, Salvatore Tramontana, Bryan Dzung Ta, and Atif M. Memon. MobiGUITAR Automated Model-Based Testing of Mobile Apps. *Software, IEEE*, 32(5):53 – 59, 2014. ISSN 0740-7459. doi: http://dx.doi.org/10.1109/MS.2014.55.

Paul Ammann and Jeff Offutt. *Introduction to software testing*. Cambridge University Press, New York, NY, USA, 1 edition, 2008.

Apple. Apple reinvents the phone with iphone, press release, January 9, 2007. Available at: https://www.apple.com/newsroom/2007/01/09Apple-Reinvents-the-Phone-with-iPhone/.

Bjarne Stroustrup. *Programming Principles and Practice Using C++ Second Edition*. Addison-Wesley Professional, 2008. ISBN 9780321992789.

Android Developers Blog. Announcing the android 1.0 sdk, realease 1, September 23, 2008. Available at: https://android-developers.googleblog.com/2008/09/announcing-android-10-sdk-release-1.html.

C. Scott Brown. Google's Play Protect service had huge impact on security in 2017. Android Authority, March 15, 2018. Available at: https://www.androidauthority.com/googles-play-protect-service-huge-impact-security-2017-846008/.

José Creissac Campos, Camille Fayollas, Marcelo Gonçalves, Celia Martinie, David Navarre, Philippe Palanque, and Miguel Pinto. *A "More Intelligent" Test Case Generation Approach through Task Models Manipulation*. Proc. ACM Hum.-Comput.Interact. 1, 1, Article 8, 2017.

Edward Cunningham. Improving app security and performance on google play for years to come, 2017. Available at: https://developer.android.com/topic/libraries/testing-support-library/index.html#setup.

Lin Deng, Jeff Offutt, and David Samudio. Is Mutation Analysis Effective at Testing Android Apps? *2017 IEEE International Conference on Software Quality, Reliability and Security (QRS)*, pages 86–93, 2017. doi: http://dx.doi.org/10.1109/QRS.2017.19.

C. Fayollas, C. Martinie, P. Palanque, Y. Deleris, J. C. Fabre, and D. Navarre. An Approach for Assessing the Impact of Dependability on Usability: Application to Interactive Cockpits. *Proceedings - 2014 10th European Dependable Computing Conference, EDCC 2014*, pages 198–209, 2014. doi: http://doi.org/10.1109/EDCC.2014.17.

Jerry Gao, Xiaoying Bai, Wei-Tek Tsai, and Tadahiro Uehara. Mobile Application Testing: A Tutorial. *Computer*, 7(2):46–55, February 2014. ISSN 0018-9162. doi: http://doi.org/10.1109/MC.2013.445.

Marcelo José Rodrigues Gonçalves. *Model-based Testing of User Interfaces*. Dissertação de Mestrado. Universidade do Minho, 2017.

Google. Testing support library. https://developer.android.com/topic/libraries/testing-support-library/index.html#setup, 2017.

Herbert Schildt. *Java: A Beginner's Guide, Seventh Edition*. McGraw-Hill Education, 2017. ISBN 9781259589317.

IBM. *Simon Says "Here's How!"Users Manual*. ICOM Copr, 1994. URL https://www.microsoft.com/buxtoncollection/a/pdf/Simon%20User%20Manuals.pdf.

Antti Kervinen. fmbt in android ui testing, April 23, 2013. Available at: https://01.org/fmbt/blogs/ask/2013/fmbt-android-ui-testing.

Antti Kervinen. fmbt tutorial, June 2, 2014. Available at: https://github.com/intel/fMBT/wiki/Tutorial.

B. Kirubakaran and V. Karthikeyani. Mobile application testing — Challenges and solution approach through automation. *2013 International Conference on Pattern Recognition, Informatics and Mobile Engineering*, pages 79–84, 2013. doi: http://doi.org/10.1109/ICPRIME.2013.6496451.

James Lewis. *Reaping the Benefits of Modern Usability Evaluation: The Simon Story*. Proceedings of the 1st International Conference on Applied Ergonomics (ICAE '96), 752-757, USA Pub, 1996.

Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing Verification and Reliability*, 22(5):297–312, 2012. ISSN 10991689. doi: http://doi.org/10.1002/stvr.456.

Paolo Masci, Patrick Oladimeji, Yi Zhang, Paul Jones, Paul Curzon, and Harold Thimbleby. *PVSio-web 2.0: Joining* PVS *to* HCI. Proceedings of 27th International Conference on Computer Aided Verification (CAV 2015), Part I, 470–478. Lecture Notes in Computer Science, vol. 9206. Springer., 2015.

Atif M Memon. A Comprehensive Framework for Testing. *Event (London)*, page 139, 2001.

Inês Coimbra Morgado and Ana C.R. Paiva. The iMPAcT tool: Testing UI patterns on mobile applications. *Proceedings - 2015 30th IEEE/ACM International Conference on Automated Software Engineering, ASE 2015*, pages 876–881, 2015. doi: http://doi.org/10.1109/ASE.2015.96.

Inês Coimbra Morgado, Ana C.R. Paiva, and João Pascoal Faria. Automated pattern-based testing of mobile applications. *Proceedings - 2014 9th International Conference on the Quality of Information and Communications Technology, QUATIC 2014*, pages 294–299, 2014. doi: http://doi.org/10.1109/QUATIC.2014.47.

H. Muccini, A. di Francesco, and P. Esposito. *Software testing of mobile applications: Challenges and future research directions*. 7th International Workshop on Automation of Software Test (AST 2012), 29–35., 2012.

Luís Miguel Carvalho Pinto. *TOM Framework: Uma ferramenta de testes baseados em modelos para interfaces gráficas web*. Dissertação de Mestrado. Universidade do Minho, 2017.

Alexander Pretschner, Wolfgang Prenninger, Stefan Wagner, Christian Kjanel, Martin Baumgartner, Bernd Sostawa, Rüdiger Zölch, and Thomas Stauner. One evaluation of model-based testing and its automation. 2017. ISSN 0018-9529. doi: http://doi.org/10.1145/1062455.1062529.

Android Open Source Project. Api guides: Application ativities. Available at: https://developer.android.com/guide/components/activities.html, 2018a.

Android Open Source Project. Api guides: Application fundamentals. Available at: https://developer.android.com/guide/components/fundamentals.html, 2018b.

Raphael Julien Rodrigues. *Testes Baseados em Modelos*. Dissertação de Mestrado. Universidade do Minho, 2015.

Ina Schieferdecker. Model-based testing. *IEEE Software*, 29(1):14–18, Jan 2012. URL https://search.proquest.com/docview/912094932?accountid=39260. IEEE Computer Society Jan/Feb 2012; Document feature - ; Last updated - 2012-01-20; CODEN - IESOEG.

Margus Veanes, Colin Campbell, Wolfgang Grieskamp, Wolfram Schulte, Nikolai Tillmann, and Lev Nachmanson. Model-Based Testing of Object-Oriented Reactive Systems with

Spec Explorer. In *Formal Methods and Testing*, pages 39–76. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008. doi: http://doi.org/10.1007/978-3-540-78917-8_2.

James Vincent. 99.6 percent of new smartphones run Android or iOS - The Verge, February 16, 2017. URL https://www.theverge.com/2017/2/16/14634656/android-ios-market-share-blackberry-2016.

A

APLICAÇÃO 'MINHAS NOTAS'

Neste anexo estão representados todos os modelos (sistema, valores, mapeamento, mutações) da aplicação Minhas Notas necessários para a geração de testes através do TOM Generator.

A.1 MODELO DO SISTEMA

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml" name="Model_v6"
    initial="2">
    <state id="1">
        <onentry id="on1" type="contains" />
        <onentry id="on2" type="default" />
        <transition id="t1" target="2" />
        <transition id="t2" target="3" />
        <transition id="t3" target="4" />
        <transition id="t4" target="5" />
        <transition id="t5" target="6" />
        <transition id="t6" target="7" />
    </state>
    <state id="2">
        <onentry id="on4" type="contains" />
        <onentry id="on3" type="displayed?" />
        <transition id="t8" target="8" />
        <transition id="t7" target="1" />
        <transition id="t9" target="9" />
        <transition id="t11" target="11" />
    </state>
    <state id="3">
        <onentry id="" type="AirPlane" />
        <onentry id="on3" type="displayed?" />
        <onentry id="on5" type="default" />
        <transition id="t7" target="1" />
    </state>
    <state id="4">
        <onentry id="" type="DOUBLEROTATE" />
```

```
28          <onentry id="on6" type="contains" />
29          <onentry id="on3" type="enabled?" />
30          <transition id="t7" target="1" />
31      </state>
32      <state id="5">
33          <transition id="t14" target="13" />
34      </state>
35      <state id="6">
36          <onentry id="" type="WifiInterrupted" />
37          <onentry id="on7" type="contains" />
38          <transition id="t7" target="1" />
39      </state>
40      <state id="7">
41          <onentry id="" type="ROTATELANDSCAPE" />
42          <onentry id="on8" type="contains" />
43          <onentry id="on9" type="default" />
44          <onentry id="on10" type="enabled?" />
45          <transition id="t7" target="1" />
46      </state>
47      <state id="8">
48          <onentry id="on11" type="enabled?" />
49          <onentry id="on12" type="default"/>
50          <state id="Adicionar Nota" type="form">
51              <send label="f1" type="optional" />
52              <send label="f2" type="optional" />
53              <transition type="form" label="notasubmit" >
54                  <submit target="2" />
55              </transition>
56              <onexit id="ex1" type="contains" />
57          </state>
58      </state>
59      <state id="9">
60          <transition id="t10" target="10" />
61          <transition id="t14" target="2" />
62      </state>
63      <state id="10">
64          <onentry id="" type="WifiOFF" />
65          <onentry id="on13" type="enabled?" />
66          <state id="Editar Nota" type="form">
67              <send label="fe1" type="optional" />
68              <send label="fe2" type="optional" />
69              <transition type="form" label="notasubmit" >
70                  <submit target="9" />
71              </transition>
72              <onexit id="ex2" type="displayed?" />
73          </state>
74      </state>
```

```
75      <state id="11">
76          <onentry id="" type="WifiON" />
77          <transition id="t12" target="12"/>
78      </state>
79      <state id="12">
80          <onentry id="on15" type="default"/>
81          <onentry id="on16" type="contains" />
82          <transition id="t13" target="2" />
83      </state>
84      <state id="13">
85          <transition id="t7" target="1" />
86      </state>
87  </scxml>
```

A.2 VALORES

{ "f1": "Nota1"},
{ "f2": "Nota de Teste"},
{ "sente3": "Lindo Dia"},
{ "sente4": },
{ "sente5": },
{ "fe1": "Editar Nota"},
{ "fe2": "Edtar Nota com texto"},
{ "on1": "Minhas Notas"},
{ "on2": "Com estrela"},
{ "on4": "Notas"},
{ "on5": "Com estrela"},
{ "on6": "Lembretes"},
{ "on7": "Lixo"},
{ "on8": "Definições"},
{ "on9": "Premium"},
{ "on12": "Nota..."},
{ "on4": "Notas"},
{ "on15": "Eliminar"},
{ "on16": "Nota será eliminada."},
{ "ex1": "Notas"}

A.3 MAPEAMENTO

```
{
"t1": {
    "how_to_find": "xpath",
    "what_to_find":"//android.widget.ExpandableListView[@index='0']/android.
        widget.LinearLayout[@index='0']",
    "what_to_do": "tap"
  },
  "t2": {
    "how_to_find": "xpath",
    "what_to_find":"//android.widget.ExpandableListView[@index='0']/android.
        widget.LinearLayout[@index='1']",
    "what_to_do": "tap"
  },
  "t3": {
    "how_to_find": "xpath",
    "what_to_find":"//android.widget.ExpandableListView[@index='0']/android.
        widget.LinearLayout[@index='2']",
    "what_to_do": "tap"
  },
  "t4": {
    "how_to_find": "xpath",
    "what_to_find":"//android.widget.ExpandableListView[@index='0']/android.
        widget.LinearLayout[@index='3']",
    "what_to_do": "tap"
  },
  "t5": {
    "how_to_find": "xpath",
    "what_to_find":"//android.widget.ExpandableListView[@index='0']/android.
        widget.LinearLayout[@index='4']",
    "what_to_do": "tap"
  },
  "t6": {
    "how_to_find": "xpath",
    "what_to_find":"//android.widget.ExpandableListView[@index='0']/android.
        widget.LinearLayout[@index='5']",
    "what_to_do": "tap"
  },
  "t7": {
    "how_to_find": "xpath",
    "what_to_find":"//android.view.View[@resource-id='net.kreosoft.android.
        mynotes:id/toolbar']/android.widget.ImageButton[@index='0']",
    "what_to_do": "tap"
  },
  "t8": {
    "how_to_find": "id",
```

```
    "what_to_find":"net.kreosoft.android.mynotes:id/floatingActionButton",
    "what_to_do": "click"
  },
  "f1": {
    "how_to_find": "id",
    "what_to_find":"net.kreosoft.android.mynotes:id/edTitle",
    "what_to_do": "sendKeys"
  },
  "f2": {
    "how_to_find": "id",
    "what_to_find":"net.kreosoft.android.mynotes:id/edContent",
    "what_to_do": "sendKeys"
  },
  "notasubmit": {
    "how_to_find": "xpath",
    "what_to_find": "//android.view.View[@resource-id='net.kreosoft.android.
       mynotes:id/toolbar']/android.widget.ImageButton[@index='0']",
    "what_to_do": "click"
  },
  "t9": {
    "how_to_find": "id",
    "what_to_find":"net.kreosoft.android.mynotes:id/llDate",
    "what_to_do": "click"
  },
  "t10": {
    "how_to_find": "id",
    "what_to_find":"net.kreosoft.android.mynotes:id/miEdit",
    "what_to_do": "click"
  },
  "fe1": {
    "how_to_find": "id",
    "what_to_find":"net.kreosoft.android.mynotes:id/edTitle",
    "what_to_do": "sendKeys"
  },
  "fe2": {
    "how_to_find": "id",
    "what_to_find":"net.kreosoft.android.mynotes:id/edContent",
    "what_to_do": "sendKeys"
  },
  "t11": {
    "how_to_find": "id",
    "what_to_find":"net.kreosoft.android.mynotes:id/llDate",
    "what_to_do": "longPress"
  },
  "t12": {
    "how_to_find": "id",
    "what_to_find":"net.kreosoft.android.mynotes:id/miSearch",
```

```
85      "what_to_do": "click"
86    },
87    "t13": {
88      "how_to_find": "id",
89      "what_to_find":"android:id/button1",
90      "what_to_do": "click"
91    },
92    "on1": {
93      "how_to_find": "xpath",
94      "what_to_find": "//android.view.View[@resource-id='net.kreosoft.android.
          mynotes:id/toolbar']/android.widget.TextView[@index='1']",
95      "what_to_do": "getText"
96    },
97    "on2": {
98      "how_to_find": "xpath",
99      "what_to_find": "//android.widget.ExpandableListView[@index='0']/android.
          widget.LinearLayout[@index='1']/android.widget.RelativeLayout[@index
          ='0']/android.widget.TextView[@index='0']",
100     "what_to_do": "getText"
101   },
102   "on3": {
103     "how_to_find": "id",
104     "what_to_find":"net.kreosoft.android.mynotes:id/floatingActionButton",
105     "what_to_do": "click"
106   },
107   "on4": {
108     "how_to_find": "id",
109     "what_to_find":"net.kreosoft.android.mynotes:id/tvTitle",
110     "what_to_do": "getText"
111   },
112   "on5": {
113     "how_to_find": "id",
114     "what_to_find":"net.kreosoft.android.mynotes:id/tvTitle",
115     "what_to_do": "getText"
116   },
117   "on6": {
118     "how_to_find": "id",
119     "what_to_find":"net.kreosoft.android.mynotes:id/tvTitle",
120     "what_to_do": "getText"
121   },
122   "on7": {
123     "how_to_find": "id",
124     "what_to_find":"net.kreosoft.android.mynotes:id/tvTitle",
125     "what_to_do": "getText"
126   },
127   "on8": {
128     "how_to_find": "xpath",
```

```
129     "what_to_find": "//android.view.View[@resource-id='net.kreosoft.android.
            mynotes:id/toolbar']/android.widget.TextView[@index='1']",
130     "what_to_do": "getText"
131   },
132   "on9": {
133     "how_to_find": "xpath",
134     "what_to_find": "//android.widget.ListView[@resource-id='android:id/list
            ']/android.widget.TextView[@index='0']",
135     "what_to_do": "getText"
136   },
137   "on10": {
138     "how_to_find": "xpath",
139     "what_to_find": "//android.widget.ListView[@resource-id='android:id/list
            ']/android.widget.LinearLayout[@index='3']",
140     "what_to_do": "getText"
141   },
142   "on11": {
143     "how_to_find": "xpath",
144     "what_to_find": "//android.view.View[@resource-id='net.kreosoft.android.
            mynotes:id/toolbar']/android.widget.ImageButton[@index='0']",
145     "what_to_do": "click"
146   },
147   "on12": {
148     "how_to_find": "id",
149     "what_to_find":"net.kreosoft.android.mynotes:id/edContent",
150     "what_to_do": "getText"
151   },
152   "ex1": {
153     "how_to_find": "id",
154     "what_to_find":"net.kreosoft.android.mynotes:id/tvTitle",
155     "what_to_do": "getText"
156   },
157   "t14": {
158     "how_to_find": "xpath",
159     "what_to_find": "//android.view.View[@resource-id='net.kreosoft.android.
            mynotes:id/toolbar']/android.widget.ImageButton[@index='0']",
160     "what_to_do": "click"
161   },
162   "ex2": {
163     "how_to_find": "id",
164     "what_to_find":"net.kreosoft.android.mynotes:id/tvTitle",
165     "what_to_do": "click"
166   },
167   "on13": {
168     "how_to_find": "xpath",
169     "what_to_find": "//android.view.View[@resource-id='net.kreosoft.android.
            mynotes:id/toolbar']/android.widget.ImageButton[@index='0']",
```

```
    "what_to_do": "click"
  },
  "on15": {
    "how_to_find": "id",
    "what_to_find":"android:id/alertTitle",
    "what_to_do": "getText"
  },
  "on16": {
    "how_to_find": "id",
    "what_to_find":"android:id/message",
    "what_to_do": "getText"
  },
  "t15": {
    "how_to_find": "xpath",
    "what_to_find":"//android.widget.ExpandableListView[@index='0']/android.
        widget.LinearLayout[@index='4']",
    "what_to_do": "tap"
  }
}
```

A.4 MUTAÇÕES

```
[
  {
    "type" : "mistake",
    "model_element": "fe1",
    "value" : "Nota Mutada",
    "fail" : "0"
  },
  {
    "type" : "Lapse",
    "model_element": "fe2",
    "fail" : "0"
  },
  {
    "type" : "Lapse",
    "model_element": "f1",
    "fail" : "0"
  },
  {
    "type" : "slip",
    "model_element": "f2",
    "fail" : "0"
  }
]
```

B

APLICAÇÃO 'GASTOS DIÁRIOS 2'

Neste anexo estão representados todos os modelos (sistema, valores, mapeamento, mutações) da aplicação Gastos Diários 2 necessários para a geração de testes através do TOM Generator.

B.1 MODELO DO SISTEMA

```
<scxml version="1.0" xmlns="http://www.w3.org/2005/07/scxml" name="Model_v6"
    initial="100">
    <state id="100">
        <transition id="130" target="101" />
    </state>
    <state id="101">
        <transition id="131" target="102" />
    </state>
    <state id="102">
        <transition id="132" target="0" />
    </state>
    <state id="0">
        <onentry id="on1" type="contains" />
        <onentry id="on2" type="default" />
        <onentry id="on3" type="displayed?" />
    <transition id="100" target="1" />
    <transition id="101" target="2" />
    <transition id="102" target="3" />
    <transition id="103" target="4" />
    <transition id="104" target="5" />
    <transition id="105" target="6" />
  </state>
  <state id="1">
        <onentry id="" type="WifiON" />
        <onentry id="on4" type="contains" />
        <onentry id="on5" type="default" />
        <transition id="120" target="0" />
    <state id="Adicionar Receita" type="form">
```

```
28        <send label="s200" type="required" />
29        <state id="categoria" type="form">
30          <send label="s201" type="required" element="sequentialClick"/>
31                  <state id="categoria2" type="form">
32                      <send label="s202" type="required" element="
                            sequentialClick"/>
33                  </state>
34        </state>
35        <send label="s203"/>
36        <transition type="form" label="1002">
37          <submit target="1" />
38                  <error target="16" />
39        </transition>
40              <onexit id="ex1" type="contains" />
41              <onexit id="ex2" type="enabled?" />
42      </state>
43    </state>
44    <state id="2">
45          <onentry id="" type="WifiOFF" />
46          <onentry id="on6" type="contains" />
47          <onentry id="on7" type="displayed?" />
48      <transition id="120" target="0" />
49          <state id="Adicionar Despesa" type="form">
50              <send label="s500" type="required" />
51              <state id="categoria" type="form">
52                  <send label="s501" type="required" element="sequentialClick"/>
53                  <state id="categoria2" type="form">
54                      <send label="s502" type="required" element="
                            sequentialClick"/>
55                  </state>
56              </state>
57              <send label="s503" type="optional" />
58              <transition type="form" label="1003">
59                  <submit target="2" />
60                  <error target="16" />
61              </transition>
62              <onexit id="ex3" type="contains?" />
63          </state>
64    </state>
65    <state id="3">
66          <onentry id="on8" type="default" />
67          <onentry id="on9" type="enabled?" />
68      <transition id="120" target="0" />
69          <transition id="350" target="8" />
70          <transition id="352" target="9" />
71          <transition id="354" target="3" />
72          <transition id="355" target="3" />
```

```
73          <transition id="356" target="3" />
74          <transition id="357" target="13"/>
75    </state>
76    <state id="4">
77          <onentry id="" type="DOUBLEROTATE" />
78          <onentry id="on10" type="contains" />
79          <onentry id="on11" type="displayed?" />
80          <transition id="400" target="10" />
81          <transition id="402" target="11" />
82          <transition id="404" target="12" />
83      <transition id="120" target="0" />
84    </state>
85    <state id="5">
86          <onentry id="" type="WifiInterrupted" />
87          <onentry id="on12" type="contains" />
88          <onentry id="on13" type="contains" />
89      <transition id="120" target="0" />
90          <transition id="140" target="7" />
91          <transition id="190" target="15" />
92    </state>
93    <state id="6">
94          <onentry id="" type="ROTATELANDSCAPE" />
95          <onentry id="on14" type="contains"/>
96      <transition id="120" target="0" />
97    </state>
98      <state id="7">
99          <onentry id="" type="Data" />
100         <onentry id="on15" type="displayed?" />
101         <onentry id="on16" type="enabled?" />
102         <onentry id="on17" type="contains" />
103         <onentry id="on18" type="contains" />
104         <transition id="160" target="5"/>
105         <state id="nova operacao" type="form">
106             <send label="s300" type="optional" element="sequentialClick"/>
107             <state id="repetir" type="form">
108                 <send label="s301" type="optional" element="sequentialClick"/>
109             </state>
110             <send label="s302" type="optional" element="sequentialClick"/>
111             <state id="data" type="form">
112                 <send label="s303" type="optional" element="sequentialClick"/>
113                 <send label="s304" type="optional" element="sequentialClick"/>
114             </state>
115             <send label="s311" type="optional" element="sequentialClick"/>
116             <state id="repetir" type="form">
117                 <send label="s312" type="optional" element="sequentialClick"/>
118             </state>
119             <send label="s305" type="optional" element="sequentialClick"/>
```

```
120             <state id="cada" type="form">
121                 <send label="s306" type="optional" element="sequentialClick"/>
122             </state>
123             <send label="s307" type="required"/>
124             <send label="s308" type="required" element="sequentialClick"/>
125             <state id="categoria" type="form">
126                 <send label="s309" type="required" element="sequentialClick"/>
127             </state>
128             <send label="s310" type="optional"/>
129             <transition type="form" label="180">
130                 <submit target="5" />
131                 <error target="16" />
132             </transition>
133             <onexit id="ex4" type="contains" />
134         </state>
135     </state>
136     <state id="8">
137         <onentry id="on19" type="contains" />
138         <onentry id="on20" type="contains" />
139         <transition id="351" target="3"/>
140     </state>
141     <state id="9">
142         <onentry id="on21" type="enabled?" />
143         <transition id="353" target="3" />
144     </state>
145     <state id="10">
146         <onentry id="on22" type="displayed?" />
147         <transition id="401" target="4" />
148     </state>
149     <state id="11">
150         <onentry id="on23" type="contains" />
151         <onentry id="on24" type="enabled?" />
152         <transition id="403" target="4" />
153     </state>
154     <state id="12">
155         <onentry id="on25" type="contains" />
156         <transition id="405" target="4" />
157     </state>
158     <state id="13">
159         <onentry id="on26" type="enabled?" />
160         <onentry id="on27" type="contains" />
161         <transition id="420" target="14" />
162     </state>
163     <state id="14">
164         <onentry id="" type="AirPlane" />
165         <onentry id="on28" type="displayed?" />
166         <onentry id="on29" type="enabled?" />
```

```
167         <onentry id="on30" type="contains" />
168         <state id="Editar Movimentos" type="form">
169             <send label="s530" type="required" />
170             <state id="categoria" type="form">
171                 <send label="s531" type="required" element="sequentialClick"/>
172                 <state id="categoria2" type="form">
173                     <send label="s532" type="required" element="
                            sequentialClick"/>
174                 </state>
175             </state>
176             <send label="s533" type="optional" />
177             <transition type="form" label="1003">
178                 <submit target="3" />
179                 <error target="16" />
180             </transition>
181             <onexit id="on31" type="contains" />
182         </state>
183     </state>
184     <state id="15">
185         <onentry id="on32" type="contains" />
186         <onentry id="on33" type="contains" />
187         <onentry id="on34" type="contains" />
188         <transition id="191" target="7"/>
189     </state>
190     <state id="16">
191         <onentry id="on36" type="contains" />
192         <onentry id="on37" type="displayed?" />
193     </state>
194 </scxml>
```

B.2 VALORES

{ "s200": "555"},
{ "s203": "test12345"},
{ "s310": "test2345"},
{ "s307": "222"},
{ "s500": "111"},
{ "s503": "compras da semana"},
{ "s530": "9"},
{ "s533": "blabla"},
{ "on1": "Gastos Diários"},
{ "on2": "Adicionar receita"},
{ "ex1": "Salvar"},

{ "on4": "Adicionar receita"},
{ "on5": "Categoria:"},
{ "on6": "Despesa:"},
{ "on7": "Descrição:"},
{ "on8": "Dinheiro"},
{ "on10": "Relatório por data"},
{ "on12": "Operações automáticas"},
{ "s502": "Pessoal"},
{ "on13": "Novo operação"},
{ "on14": "Opções"},
{ "on17": "Valor:"},
{ "on18": "Categoria:"},
{ "on19": "Janeiro"},
{ "ex3": "Despesa:"},
{ "on20": "Setembro"},
{ "on23": "Por dia"},
{ "on25": "2018"},
{ "on27": "Copiar"},
{ "on30": "Edição de dados"},
{ "on31": "Lista de movimentos"},
{ "on32": "Excluir"},
{ "on33": "Visualização de registros criado"},
{ "on34": "Modificar"},
{ "ex4": "Operações automáticas"},
{ "on36": "Atenção"}

B.3 MAPEAMENTO

```
{
 "100": {
   "how_to_find": "id",
   "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonAddIncome",
   "what_to_do": "click"
 },
 "101": {
   "how_to_find": "id",
   "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonAddExpenses",
   "what_to_do": "click"
 },
 "102": {
```

```
13      "how_to_find": "id",
14      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonMovementList",
15      "what_to_do": "click"
16    },
17    "103": {
18      "how_to_find": "id",
19      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonReportsByDate",
20      "what_to_do": "click"
21    },
22    "104": {
23      "how_to_find": "id",
24      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonFrequentRecords",
25      "what_to_do": "click"
26    },
27    "105": {
28      "how_to_find": "id",
29      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonSettings",
30      "what_to_do": "click"
31    },
32    "120": {
33      "how_to_find": "id",
34      "what_to_find": "mic.app.gastosdiarios_clasico:id/imageApp",
35      "what_to_do": "click"
36    },
37    "s200": {
38      "how_to_find": "id",
39      "what_to_find": "mic.app.gastosdiarios_clasico:id/editIncome",
40      "what_to_do": "sendKeys"
41    },
42    "s201": {
43      "how_to_find": "id",
44      "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerCategory",
45      "what_to_do": "click"
46    },
47    "s202": {
48      "how_to_find": "xpath",
49      "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
            RelativeLayout[@index='2']",
50      "what_to_do": "click"
51    },
52    "s203": {
53      "how_to_find": "id",
54      "what_to_find": "mic.app.gastosdiarios_clasico:id/editDetail",
55      "what_to_do": "sendKeys"
56    },
57      "s204": {
58        "how_to_find": "id",
```

```
      "what_to_find": "mic.app.gastosdiarios_clasico:id/textDate",
      "what_to_do": "click"
    },
    "s205": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/r4c5",
      "what_to_do": "click"
    },
    "s206": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonOk",
      "what_to_do": "click"
    },
  "1002": {
    "how_to_find": "id",
    "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonSave",
    "what_to_do": "click"
  },
    "140": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonNewOperation",
      "what_to_do": "click"
    },
    "160": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonCancel",
      "what_to_do": "click"
    },
    "180": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonSave",
      "what_to_do": "click"
    },
    "s300": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerFrequent",
      "what_to_do": "click"
    },
    "s301": {
      "how_to_find": "xpath",
      "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
          LinearLayout[@index='5']",
      "what_to_do": "click"
    },
    "s307": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/editAmount",
```

```
105       "what_to_do": "sendKeys"
106     },
107     "s308": {
108       "how_to_find": "id",
109       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerCategory",
110       "what_to_do": "click"
111     },
112     "s309": {
113       "how_to_find": "xpath",
114       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              RelativeLayout[@index='1']",
115       "what_to_do": "click"
116     },
117     "s302": {
118       "how_to_find": "id",
119       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerStartDate",
120       "what_to_do": "click"
121     },
122     "s303": {
123       "how_to_find": "id",
124       "what_to_find": "mic.app.gastosdiarios_clasico:id/r3c3",
125       "what_to_do": "click"
126     },
127     "s304": {
128       "how_to_find": "id",
129       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonOk",
130       "what_to_do": "click"
131     },
132     "s305": {
133       "how_to_find": "id",
134       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerEach",
135       "what_to_do": "click"
136     },
137     "s306": {
138       "how_to_find": "xpath",
139       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='3']",
140       "what_to_do": "click"
141     },
142     "s310": {
143       "how_to_find": "id",
144       "what_to_find": "mic.app.gastosdiarios_clasico:id/editDetail",
145       "what_to_do": "sendKeys"
146     },
147     "s311": {
148       "how_to_find": "id",
149       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerRepeat",
```

```
150       "what_to_do": "click"
151     },
152     "s312": {
153       "how_to_find": "xpath",
154       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='3']",
155       "what_to_do": "click"
156     },
157     "350": {
158       "how_to_find": "id",
159       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerMonth",
160       "what_to_do": "click"
161     },
162     "351": {
163       "how_to_find": "xpath",
164       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='3']",
165       "what_to_do": "click"
166     },
167     "352": {
168       "how_to_find": "id",
169       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerYear",
170       "what_to_do": "click"
171     },
172     "353": {
173       "how_to_find": "xpath",
174       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='1']",
175       "what_to_do": "click"
176     },
177     "354": {
178       "how_to_find": "id",
179       "what_to_find": "mic.app.gastosdiarios_clasico:id/imageOrder",
180       "what_to_do": "click"
181     },
182     "355": {
183       "how_to_find": "id",
184       "what_to_find": "mic.app.gastosdiarios_clasico:id/imageIncome",
185       "what_to_do": "click"
186     },
187     "356": {
188       "how_to_find": "id",
189       "what_to_find": "mic.app.gastosdiarios_clasico:id/imageExpense",
190       "what_to_do": "click"
191     },
192     "357": {
193       "how_to_find": "xpath",
```

```
194       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='0']",
195       "what_to_do": "click"
196     },
197     "130": {
198       "how_to_find": "id",
199       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonOk",
200       "what_to_do": "tap"
201     },
202     "131": {
203       "how_to_find": "id",
204       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonOk",
205       "what_to_do": "click"
206     },
207     "132": {
208       "how_to_find": "id",
209       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonOk",
210       "what_to_do": "click"
211     },
212     "400": {
213       "how_to_find": "id",
214       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerType",
215       "what_to_do": "click"
216     },
217     "401": {
218       "how_to_find": "xpath",
219       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='0']",
220       "what_to_do": "click"
221     },
222     "402": {
223       "how_to_find": "id",
224       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerPeriod",
225       "what_to_do": "click"
226     },
227     "403": {
228       "how_to_find": "xpath",
229       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='5']",
230       "what_to_do": "click"
231     },
232     "404": {
233       "how_to_find": "id",
234       "what_to_find": "mic.app.gastosdiarios_clasico:id/textDate",
235       "what_to_do": "click"
236     },
237     "405": {
```

```
238       "how_to_find": "id",
239       "what_to_find": "mic.app.gastosdiarios_clasico:id/textSimple",
240       "what_to_do": "click"
241     },
242     "s500": {
243       "how_to_find": "id",
244       "what_to_find": "mic.app.gastosdiarios_clasico:id/editExpense",
245       "what_to_do": "sendKeys"
246     },
247     "s501": {
248       "how_to_find": "id",
249       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerCategory",
250       "what_to_do": "click"
251     },
252     "s502": {
253       "how_to_find": "textMatches",
254       "what_to_do": "scroll"
255     },
256     "s503": {
257       "how_to_find": "id",
258       "what_to_find": "mic.app.gastosdiarios_clasico:id/editDetail",
259       "what_to_do": "sendKeys"
260     },
261     "1003": {
262       "how_to_find": "id",
263       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonSave",
264       "what_to_do": "click"
265     },
266     "420": {
267       "how_to_find": "xpath",
268       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              RelativeLayout[@index='0']",
269       "what_to_do": "click"
270     },
271     "s530": {
272       "how_to_find": "id",
273       "what_to_find": "mic.app.gastosdiarios_clasico:id/editAmount",
274       "what_to_do": "sendKeys"
275     },
276     "s533": {
277       "how_to_find": "id",
278       "what_to_find": "mic.app.gastosdiarios_clasico:id/editDetail",
279       "what_to_do": "sendKeys"
280     },
281     "s531": {
282       "how_to_find": "id",
283       "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerCategory",
```

```
284       "what_to_do": "click"
285     },
286     "s532": {
287       "how_to_find": "xpath",
288       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              RelativeLayout[@index='1']",
289       "what_to_do": "click"
290     },
291     "190": {
292       "how_to_find": "xpath",
293       "what_to_find": "//android.widget.ListView[@index='1']/android.widget.
              LinearLayout[@index='0']",
294       "what_to_do": "longPress"
295     },
296     "191": {
297       "how_to_find": "xpath",
298       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              RelativeLayout[@index='0']",
299       "what_to_do": "click"
300     },
301     "on1": {
302       "how_to_find": "id",
303       "what_to_find": "mic.app.gastosdiarios_clasico:id/textTitle",
304       "what_to_do": "getText"
305     },
306     "on2": {
307       "how_to_find": "id",
308       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonAddIncome",
309       "what_to_do": "getText"
310     },
311     "on3": {
312       "how_to_find": "id",
313       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonFrequentRecords"
              ,
314       "what_to_do": "click"
315     },
316     "ex1": {
317       "how_to_find": "id",
318       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonSave",
319       "what_to_do": "getText"
320     },
321     "on4": {
322       "how_to_find": "id",
323       "what_to_find": "mic.app.gastosdiarios_clasico:id/textTitle",
324       "what_to_do": "getText"
325     },
326     "on5": {
```

```
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/text3",
      "what_to_do": "getText"
    },
    "ex2": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonCategories",
      "what_to_do": ""
    },
    "on6": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/text1",
      "what_to_do": "getText"
    },
    "on7": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/text4",
      "what_to_do": "getText"
    },
    "ex3": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/text1",
      "what_to_do": "getText"
    },
    "on8": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/textAccountName",
      "what_to_do": "getText"
    },
    "on9": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerMonth",
      "what_to_do": "getText"
    },
    "on10": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/textTitle",
      "what_to_do": "getText"
    },
    "on11": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/spinnerType",
      "what_to_do": "getText"
    },
    "on12": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/textTitle",
```

```
374       "what_to_do": "getText"
375     },
376     "on13": {
377       "how_to_find": "id",
378       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonNewOperation",
379       "what_to_do": "getText"
380     },
381     "on14": {
382       "how_to_find": "id",
383       "what_to_find": "mic.app.gastosdiarios_clasico:id/textTitle",
384       "what_to_do": "getText"
385     },
386     "ex4": {
387       "how_to_find": "id",
388       "what_to_find": "mic.app.gastosdiarios_clasico:id/textTitle",
389       "what_to_do": "getText"
390     },
391     "on15": {
392       "how_to_find": "id",
393       "what_to_find": "mic.app.gastosdiarios_clasico:id/imageIncome",
394       "what_to_do": "getText"
395     },
396     "on16": {
397       "how_to_find": "id",
398       "what_to_find": "mic.app.gastosdiarios_clasico:id/imageExpense",
399       "what_to_do": "getText"
400     },
401     "on17": {
402       "how_to_find": "id",
403       "what_to_find": "mic.app.gastosdiarios_clasico:id/text4",
404       "what_to_do": "getText"
405     },
406     "on18": {
407       "how_to_find": "id",
408       "what_to_find": "mic.app.gastosdiarios_clasico:id/text6",
409       "what_to_do": "getText"
410     },
411     "on19": {
412       "how_to_find": "xpath",
413       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='0']/android.widget.TextView[@index='0']",
414       "what_to_do": "getText"
415     },
416     "on20": {
417       "how_to_find": "xpath",
418       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='8']/android.widget.TextView[@index='0']",
```

```
419       "what_to_do": "getText"
420     },
421     "on21": {
422       "how_to_find": "xpath",
423       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='1']",
424       "what_to_do": "getText"
425     },
426     "on22": {
427       "how_to_find": "xpath",
428       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='0']/android.widget.TextView[@index='0']",
429       "what_to_do": "getText"
430     },
431     "on23": {
432       "how_to_find": "xpath",
433       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='0']/android.widget.TextView[@index='0']",
434       "what_to_do": "getText"
435     },
436     "on24": {
437       "how_to_find": "xpath",
438       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              LinearLayout[@index='5']/android.widget.TextView[@index='0']",
439       "what_to_do": "getText"
440     },
441     "on25": {
442       "how_to_find": "id",
443       "what_to_find": "mic.app.gastosdiarios_clasico:id/textSimple",
444       "what_to_do": "getText"
445     },
446     "on26": {
447       "how_to_find": "xpath",
448       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              RelativeLayout[@index='0']/android.widget.TextView[@index='1']",
449       "what_to_do": "getText"
450     },
451     "on27": {
452       "how_to_find": "xpath",
453       "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
              RelativeLayout[@index='2']/android.widget.TextView[@index='1']",
454       "what_to_do": "getText"
455     },
456     "on28": {
457       "how_to_find": "id",
458       "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonCancel",
459       "what_to_do": "getText"
```

```
    },
    "on29": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/buttonSave",
      "what_to_do": "getText"
    },
    "on30": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/titleDialog",
      "what_to_do": "getText"
    },
    "on31": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/textTitle",
      "what_to_do": "getText"
    },
    "on32": {
      "how_to_find": "xpath",
      "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
          RelativeLayout[@index='1']/android.widget.TextView[@index='1']",
      "what_to_do": "click"
    },
    "on33": {
      "how_to_find": "xpath",
      "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
          RelativeLayout[@index='2']/android.widget.TextView[@index='1']",
      "what_to_do": "click"
    },
    "on34": {
      "how_to_find": "xpath",
      "what_to_find": "//android.widget.ListView[@index='0']/android.widget.
          RelativeLayout[@index='0']/android.widget.TextView[@index='1']",
      "what_to_do": "click"
    },
    "on36": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/titleDialog",
      "what_to_do": "getText"
    },
    "on37": {
      "how_to_find": "id",
      "what_to_find": "mic.app.gastosdiarios_clasico:id/imageView",
      "what_to_do": "getText"
    }
}
```

B.4 MUTAÇÕES


```
[
  {
    "type" : "mistake",
    "model_element": "s307",
    "value" : "odd",
    "fail" : "1"
  },
  {
    "type" : "mistake",
    "model_element": "s533",
    "value" : "Pedro",
    "fail" : "0"
  },
  {
    "type" : "Lapse",
    "model_element": "s310",
    "fail" : "0"
  },
  {
    "type" : "Lapse",
    "model_element": "s307",
    "fail" : "1"
  },
  {
    "type" : "slip",
    "model_element": "s503",
    "fail" : "0"
  },
  {
    "type" : "slip",
    "model_element": "s500",
    "fail" : "1"
  },
  {
    "type" : "Lapse",
    "model_element": "s203",
    "fail" : "0"
  },
  {
    "type" : "mistake",
    "model_element": "s310",
    "value" : "Tentativa",
    "fail" : "0"
  },
  {
```

```
46      "type" : "mistake",
47      "model_element": "s203",
48      "value" : "Teste",
49      "fail" : "0"
50    },
51    {
52      "type" : "Lapse",
53      "model_element": "s200",
54      "fail" : "1"
55    },
56    {
57      "type" : "Mistake",
58      "model_element": "s200",
59      "value" : "Teste",
60      "fail" : "1"
61    }
62  ]
```

C

MANUAL DE REFERÊNCIA PARA MODELOS TOM

Este anexo serve de manual de referência para a criação dos modelos necessários para a geração de testes no TOM.

<state id="" type=""> ... </state>	
id	Atributo obrigatório que é utilizado para identificar cada estado.
type	Atributo utilizado quando se está a preencher um formulário na interface gráfica, tomando o valor "form". Caso contrário, não é definido.

Tabela 7.: Tabela com os atributos do elemento state.

<transition id="" label="" />	Para Transições simples entre páginas.
id	Atributo obrigatório que se utiliza como identificador da ação.
type	O id do estado para o qual se pretende transitar.

Tabela 8.: Tabela com os atributos do elemento transition.

<transition type="" label="" > <submit target=""/> <error target="" /> </transition>	Género de transição que é utilizado quando se está a modelar o preenchimento de um formulário na interface gráfica. O elemento error é opcional, e usa-se em caso de mutação de falha.
label	Serve para identificar o tipo de transição da página Web, depois da submissão do formulário. Assume o valor de "form" quando se preenche um formulário normal, "alert" quando surge uma alert window e "ajax" para eventos assíncronos.
type	Atributo utilizado para representar identificar a ação de submissão de um formulário.
element	O id do estado para o qual se pretende transitar.

Tabela 9.: Tabela com os atributos do elemento transition nos formulários.

<send label="" type="" element="" />	Correspondem à introdução de informação nos formulários.
label	É utilizado como identificador da ação.
type	Indica se a informação a ser introduzida no formulário é obrigatória ou opcional. "required" ou "optional" são os valores possíveis.
element	Este atributo é somente utilizado quando a informação que se vai adicionar ao formulário é do tipo checkbox, selectbox ou sequencialClick, sendo estes os valores possíveis de entrada.

Tabela 10.: Tabela com os atributos do elemento send.

<onentry id="" type=""> <onexit id="" type=""/>	Corresponde às validações que são realizadas quando se entra ou sai de uma página Web.
id	Utilizado como identificador da ação.
type	O tipo de validação que pode ser efetuada: DISPLAYED?, NOT DISPLAYED?, IS SELECTED / IS NOT SELECTED ENABLED? / DISABLED?, ATTRIBUTE , CSS, CONTAINS, REGEX, URL, DEFAULT, Data ROTATELANDSCAPE, DOUBLEROTATE, AirPlane ROTATEPORTRAIT, WifiInterrupted, WifiOFF, WifiON.

Tabela 11.: Tabela com os atributos dos elementos onentry e onexit.

{type: "", model element: "", value: "", fail: "" }, ...]	
type	É utilizado para representar qual o tipo de mutação que se pretende realizar. Os valores aceites para este atributo são: lapse, slip, mistake, doubleClick.
model_element	Identifica o elemento do modelo onde se introduz a mutação.
value	Este atributo só é necessário quando o tipo de mutação é mistake. Neste sentido, significa que o valor original do ficheiro de valores será trocado por este.
fail	Permite determinar se com a introdução desta mutação o caso de teste deverá falhar, gerando um teste inválido. Caso o valor seja definido como "1" o caso de teste irá conter falhas no teste, se o valor for "0" significa que esta mutação não irá interferir no resultado do caso de teste.

Tabela 12.: Tabela com a descrição dos elementos de mutação.

”#id” : {
how to find: ””,
what to find: ””,
what to do: ””,
type of action: ””
}
	O formato genérico utilizado na mapeamento entre a interface gráfica e o modelo do sistema.

how_to_find	O localizador que se deve utilizar para encontrar o elemento na interface gráfica. Pode tomar os seguintes valores: id, XPath, cssSelector, className, linkText, name, tagName, partialLinkText.
what_to_find	Valor do tipo de elemento a ser encontrado.
what_to_do	O tipo de ação que será executada neste mapeamento. É um atributo opcional e os próximo valores são os únicos válidos neste campo: sendKeys, click, submit, moveToElemetn, getText, accept, tap, doubleTap, longPress, scroll, swipe.
type_of_action	Auxilia na determinação do tipo de elemento HTML. É um atributo opcional que pode ser utilizado com um dos seguintes valores: textBox, selectBox, checkBox, ou com um elemento HTML.
offset_x	A coordenada X do elemento a ser encontrado.
offset_y	A coordenada Y do elemento a ser encontrado.
what_to_do_ second_element	O localizador que se deve utilizar para encontrar o segundo elemento na interface gráfica
what_to_find_ second_element	Valor do tipo do segundo elemento a ser encontrado.
how_to_find_ second_element	O tipo de ação que será executada neste mapeamento.
type_of_action_ second_element	Auxilia na determinação do tipo do segundo elemento HTML.
offset_sencond_x	A coordenada X do segundo elemento a ser encontrado.
offset_sencond_y	A coordenada Y do segundo elemento a ser encontrado.

Tabela 13.: Tabela com a descrição dos elementos de mapeamento do modelo.

[{ ”#id” : ”#value” } ...]	
#id	Identifica a ação do utilizador no ficheiro de valores.
#value	O valor que será utilizado para preencher os formulários ou nas validações que usem texto para comparação.

Tabela 14.: Tabela descritiva da estrutura do ficheiro de valores.