FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Geração e execução de casos de teste baseados em padrões de interação

Isabel Alexandra Silva Barbosa

Mestrado Integrado em Engenharia Informática e Computação

Orientadora: Ana Paiva (PhD)

Fevereiro 2012

Geração e execução de casos de teste baseados em padrões de interação

Isabel Alexandra Silva Barbosa

Mestrado Integrado em Engenharia Informática e Computação

Aprovado em provas públicas pelo júri:

Presidente: José Magalhães Cruz (PhD)

Vogal Externo: José Creissac Campos (PhD)

Orientadora: Ana Paiva (PhD)

Fevereiro 2012

Resumo

Atualmente, as interfaces gráficas estão presentes na grande maioria das aplicações e têm um papel bastante importante no seu desempenho.

Devido à sua complexidade, criada principalmente devido ao elevado número de combinações ações que estas possuem, é praticamente impossível realizar os testes às GUIs de forma manual. A automatização deste processo é, por isso, uma necessidade urgente.

Neste momento, já existem algumas ferramentas que permitem realizar testes a GUIs automaticamente mas estas ainda possuem algumas limitações ou problemas. Uma das mais reconhecidas e vantajosas é o teste de GUIs baseado em modelos.

Esta técnica passa pela construção de um modelo a partir da estrutura da interface a testar e os casos de teste são gerados a partir desse modelo. O principal problema desta abordagem reside na construção do modelo. Este processo pode ser demorado e dispendioso e, em alguns casos, pode ser bastante complicado.

A abordagem seguida nesta dissertação passa por construir um modelo baseado em padrões de interação. Os padrões representam pequenos elementos das interfaces que se comportam de maneira semelhante. O objetivo é encontrar os padrões mais comuns nas interfaces, definir uma estratégia de teste para cada um e, a partir daí, gerar e executar os testes.

Esta é uma abordagem que já se encontra a ser desenvolvida, para a qual já existe um protótipo de ferramenta, que torna este processo possível para um pequeno conjunto de padrões.

Nesta dissertação optou-se por implementar uma funcionalidade bastante recorrente nas GUIs: a pesquisa. Para isso, após uma pesquisa de todos os padrões relacionados com essa funcionalidade, definiu-se um padrão de teste para testar a pesquisa. Esse padrão de teste define uma estratégia de teste baseada em partição em classes de equivalência. Por último, implementou-se o padrão de teste na ferramenta já existente, a PETTool, e verificou-se a sua eficácia através de alguns casos de estudo.

Abstract

Nowadays, graphical interfaces are present in most applications and have a very important role in their performance.

Due to its complexity, mainly because of the high number of combined actions, it is virtually impossible to perform tests on a GUI by hand. The automation of this process, is therefore an urgent need.

At this moment, there are some tools that allow the testing of a GUI using an automated approach but they still have some limitations or problems. One of most recognized and with a higher number of advantages is the model-based GUI testing.

This technique involves the construction of a model from the structure of the interface under testing, and the test cases are generated from that model. The main problem with this approach lies in the construction of the model. This process can be time consuming and in some cases it can also be quite complicated.

The approach taken in this dissertation is to build a model based on interaction patterns. The patterns represent small elements of the interfaces that behave in a similar manner. The objective is to find the patterns which are more common in the interfaces, define a test strategy for each one and from there, generate and execute the tests.

This is an approach that is being developed, for which there is a prototype of the tool, which makes this process possible for a small set of patterns.

In this dissertation, we chose to implement a frequent feature in a GUI: search. To do so, after a research of all patterns related to this feature, we have defined a testing pattern for testing search. This testing pattern defines a test strategy based on partition into classes of equivalence.

For last, we have implemented the testing pattern in the existing tool, the PETTool, and verified its effectiveness through some case studies.

Agradecimentos

Em primeiro lugar, um agradecimento à minha orientadora, professora Ana Paiva, por todo o apoio e acompanhamento prestado durante o desenvolvimento do projeto.

A minha família, sem dúvida, foi também uma parte importante deste processo, por estar disponível sempre que precisei. Por isso quero agradecer o apoio e compreensão dos meus pais, do meu marido e do meu irmão. Quero também agradecer, de uma forma especial, à minha avó.

Por último, mas não menos importante, o meu obrigada a todos os meus amigos que me ajudaram, aconselharam e distraíram quando precisei, em especial à Inês Carvalho, Patrícia Nogueira e Bernardo Almeida.

Isabel Barbosa

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

API	Application Programming Interface
FAQ	*Frequently Asked Questions*
FEUP	Faculdade Engenharia Universidade Porto
GUI	*Graphical User Interface*
HTML	*HyperText Markup Language*
IDE	Integrated Development Environment
LST	*Labelled Transition Systems*
LSTS	*Labelled State Transiton Systems*
MIEIC	Mestrado Integrado Engenharia Informática e Computação
PBGT	*Pattern Based GUI Testing*
PETTool	*PattErn Testing Tool*
UI	*User Interface*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
WPF	*Windows Presentation Foundation*
WWW	*World Wide Web*

Capítulo 1

Introdução

Este trabalho enquadra-se na área de testes de software. O seu principal objetivo é garantir a qualidade do software.

Apesar de não serem ainda reconhecidos como tal pela maioria dos profissionais da área, os testes são uma componente indispensável do desenvolvimento de aplicações. Isto deve-se ao facto de estes permitirem a detecção de possíveis falhas no software, ou seja, determinam se este se apresenta em conformidade com os requisitos.

Nos últimos tempos esta área tem vindo a crescer e a interessar cada vez mais as empresas devido às provas de sucesso que têm sido dadas.

Os testes podem ser realizados por um elemento ou uma equipa de elementos, preferencialmente não envolvidos no desenvolvimento da aplicação a ser testada, para assim obterem melhores resultados. Estes podem ser realizados ao longo do processo de desenvolvimento do software ou no final.

Existem vários tipos de técnicas de teste que têm vindo a ser desenvolvidas ao longo dos tempos para dar resposta às diferentes necessidades das aplicações.

A automatização do processo de testes é um fator que tem sido levado em conta visto que, quando a dimensão do software é considerável, torna-se impraticável testá-lo manualmente.

1.1 Contexto

Visto esta ser ainda uma área em crescimento existem ainda zonas pouco ou nada exploradas. As GUIs (*Graphical Users Interfaces*) são um exemplo disso.

As interfaces gráficas estão presentes na grande maioria das aplicações atuais. São a barreira de comunicação entre o utilizador e a aplicação e, como tal, devem corresponder a alguns requisitos.

A boa usabilidade e a correta funcionalidade são fundamentais. Um mau funcionamento de uma GUI pode facilmente levar à não utilização do software. Por essa razão é importante testar exaustivamente as interfaces gráficas, de modo a reduzir ao máximo os erros destas.

Mas testar interfaces não é, de todo, uma tarefa fácil. A sua complexidade, criada pelo elevado número de possíveis ações presentes nas GUIs, é uma das principais dificuldades a ultrapassar. Neste caso, a automatização dos testes é imprescindível.

Neste momento existem algumas abordagens que tentam solucionar o problema de testes em interfaces gráficas. Umas passam por adaptar outras técnicas de testes, outras foram criadas exclusivamente para as GUIs, mas ainda assim existem alguns problemas. Um dos principais problemas é o elevado custo das ferramentas e do pessoal necessário para as usar, assim como o tempo despendido no processo.

1.2 Objetivos

O principal objetivo deste projeto passa por identificar uma solução para os testes de interface gráfica. Esta solução tem que apresentar alternativas às limitações apresentadas pelas já existentes. Deve por isso ser o mais automatizada possível, de fácil e rápida utilização e permitir o teste de todo o tipo de aplicações (generalização).

Para atingir esse objetivo pretende-se dar continuidade a um projeto já iniciado na Faculdade de Engenharia da Universidade do Porto no ano lectivo 2009/2010 (PETTool). A abordagem de testes utilizada é a baseada em padrões - PBGT (*Pattern-Based GUI Testing*).

O processo passa por identificar comportamentos recorrentes nas GUIs - os padrões - e, para cada um, identificar soluções de teste baseadas em técnicas já existentes.

A ferramenta criada deverá permitir a identificação do padrão na interface gráfica, a configuração de pequenos elementos do padrão e, por fim, deve gerar e executar automaticamente os casos de teste para esse padrão.

Neste projeto em específico iremos focar a atenção num componente da GUI em particular: a Pesquisa.

Os objetivos do projeto são:

- Encontrar todos os padrões relacionados com a pesquisa;
- Selecionar os padrões relevantes para testar, definir a estratégia de teste para cada um e definir possível relações entre os padrões;
- Implementar os testes na ferramenta PETTool.

1.3 Estrutura da Dissertação

Para além da introdução esta dissertação contém mais 5 capítulos.

No capítulo 2 é apresentado o estado da arte. É feita uma breve descrição das técnicas de testes existentes e das abordagens a testes de interfaces gráficas. São também apresentados os padrões de desenho e a ferramenta PETTool.

No capítulo 3 é descrita a abordagem a seguir no projeto. São discutidos quais os padrões a implementar, as estratégias de teste a usar e possíveis relações entre os padrões.

O capítulo 4 refere-se à implementação do projeto. Apresenta uma descrição das tecnologias utilizadas e descreve o funcionamento da ferramenta.

No capítulo 5 são apresentados alguns casos de estudo que comprovam o funcionamento do que foi implementado e descrito no capítulo anterior.

O capítulo 6 apresenta as conclusões retiradas deste projeto e o trabalho a desenvolver no futuro.

Capítulo 2

Estado da arte

Neste capítulo é apresentado o estado da arte. Nele são abordados dois temas principais relacionados com o trabalho a realizar: as técnicas e ferramentas que têm vindo a ser utilizadas nos testes de interfaces gráficas, identificando as suas principais vantagens e desvantagens; Os padrões de interação, apresentando a sua definição, alguns exemplos e explorando em especial os padrões que serão implementados neste projeto. Será também abordada a ferramenta PETTool descrevendo-se as suas funcionalidades e limitações.

2.1 Testes de GUIs

O uso de interfaces gráficas nas aplicações tem vindo a sofrer um enorme crescimento. É rara a aplicação atualmente que não faz uso de uma GUI visto esta facilitar bastante a interação entre os utilizadores e o computador. É de notar também que, ao longo dos anos, a sua complexidade é cada vez mais elevada. Podemos fazer a comparação entre editores de texto da linha de comandos - que atualmente caíram em desuso - e o *Microsoft Word* 2011- um dos mais usados - e facilmente se encontram as diferenças.

A detecção de erros é algo que pode ocorrer em qualquer aplicação, sendo que, quanto mais complexa esta for, maior será a probabilidade de esta conter pequenas falhas. No caso das interfaces gráficas os testes são cruciais, devido à sua interação direta com o utilizador, que tornam os erros um problema grave podendo levar à desistência da utilização da aplicação em causa por parte do utilizador.

Apesar da sua importância, o teste de GUIs não é algo que os programadores apreciem devido à sua dificuldade podendo até levar à desistência. E são vários os factores que tornam estes testes difíceis de realizar.

Devido à sua complexidade, o número de eventos que podem ser realizados nas GUIs é muito elevado o que torna a realização dos testes manualmente muito demorada e dispendiosa. Existe a necessidade fulcral de automatizar esse processo.

Outros dos problemas de testes a interfaces gráficas é a localização da origem dos erros. Este problema ocorre pois, para além de existir um elevado número de sequências de eventos, é ainda possível que diferentes sequências levem a resultados iguais.

Estes fatores tornam ainda complicado garantir uma cobertura alargada do código através dos testes realizados. A simulação de um conjunto de eventos integrado e a sua memorização são factores que não podem ser esquecidos na realização dos testes [RP08] [CPFA10] [dMC10].

2.1.1 Estratégias de teste

Existem dois tipos de estratégias de teste básicas - Caixa Preta e Caixa Branca - das quais resultam variados métodos de teste.

2.1.1.1 Testes de Caixa Preta

Os testes de Caixa Preta são realizados sem conhecimento da estrutura da aplicação, do seu código, apenas são conhecidas as suas funcionalidades e os seus resultados esperados. O teste é elaborado com base nessas informações, é executado com os valores de entrada, e os resultados são comparados com os valores de saída esperados. Esta estratégia é eficaz a encontrar falhas de requisitos e especificação.

Partição em classes de equivalência O método de partição em classes de equivalência permite que os domínios de entrada e saída - normalmente aplicado nos domínios de entrada - sejam divididos em classes, sendo que cada uma dessas classes contém valores com um comportamento semelhante. Isto é, se um determinado valor da classe de equivalência fizer o teste falhar, todos os outros também falham e no caso de sucesso verifica-se a mesma situação. Neste caso, para cada classe de equivalência é apenas necessário testar um dos seus valores, o que diminui bastante o número de casos de teste a implementar. Este processo aumenta a probabilidade de encontrar defeitos pois cobre uma grande área do software. É de grande importância que sejam criadas classes de equivalência para valores válidos, mas também para valores inválidos, para que os dois cenários possam ser testados.

Para exemplificar o uso de classes de equivalência vamos pensar num campo de entrada de dados, por exemplo um campo de idade num formulário, que só aceita valores entre 0 e 100. Para este caso existem 3 classes de equivalência:

- CE1: Valores entre 0 e 100, classe válida;

- CE2: Valores menores que 0, classe inválida;
- CE3: Valores maiores que 100, classe inválida.

Análise de Valores Fronteira Este método pode ser visto como um complemento ao método de partição em classes de equivalência, visto que requer uma análise aos valores fronteira das classes, ou seja, para além do valor da classe a testar é necessário testar os valores de fronteira e os valores imediatamente acima e abaixo de cada um deles.

No exemplo em cima mencionado, se escolhermos um valor para cada classe podemos ter:

- CE1: 20;
- CE2: -10
- CE3: 123.

Mas aplicando o método de análise de valores fronteira acrescentamos os seguintes casos de teste:

- Valores fronteira: 0 e 100;
- Valores imediatamente acima a abaixo: -1 e 1, 99 e 101.

Esta técnica poderá permitir encontrar mais erros, pois é muitas vezes nestas fronteiras que se encontram mais falhas, por parte dos programadores - uma simples troca de sinais (< e <=) pode causar problemas no software.

Grafo Causa-Efeito Apesar das vantagens do método de partição em classes de equivalência, este não permite a combinação de condições, ou seja, cada condição tem de ser tratada individualmente sendo que até poderá vir a ser possível testar duas condições com o mesmo caso de teste.

Para contornar esse problema foi criado o método do Grafo Causa-Efeito, em que é construído um grafo que posteriormente será convertido numa tabela decisão, de onde serão gerados os casos de teste. Para criar este grafo é necessário identificar as causas (as entradas) e os efeitos (as saídas) e relacioná-los. Na tabela final cada coluna representa um caso de teste a implementar.

Este método é bastante pois faz uma análise completa à especificação do software. Os problemas podem surgir quando o número de causas é elevado, visto que o número de casos de teste gerados é também muito elevado.

Teste de Transição de Estados O método de transição de estados pretende que seja possível a descoberta de erros que não são possíveis de descobrir com os métodos anteriores.

Para isso faz-se a representação da aplicação num grafo de transição de estados, em que os nós representam os estados e as setas representam as transições (que ocorrem através de inputs/eventos) que se encontram devidamente legendadas. Cada transição irá originar os casos de teste necessários. Pode até ser possível conseguir cobrir todas as transições com apenas um caso de teste, dependendo do grafo.

Se a especificação for bem representada, é possível testar mais profundamente as aplicações mais complexas com este método.

Adivinhar Erros Adivinha erros é um método que, apesar de não seguir nenhum tipo de regras, pode ser bastante produtivo. Aqui tudo se obtém através da experiência e do conhecimento das pessoas que testam o software. Quer experiência de projetos semelhantes em que estiveram envolvidos anteriormente, quer o conhecimento do projeto em questão.

Neste caso, não existe nenhuma técnica específica, cabe apenas à pessoa responsável por testar a aplicação a tarefa de tentar adivinhar onde podem estar "escondidos"alguns erros.

2.1.1.2 Testes de Caixa Branca

Os testes de Caixa Branca são realizados com base na estrutura interna da aplicação, quer seja o código, ou alguma representação do mesmo (grafos de fluxo, diagramas, etc...). Este tipo de estratégia é usada para detectar erros de design, de lógica, de inicialização, entre outros. Nas subsecções seguintes são apresentadas algumas das técnicas de teste deste método.

Visto que o utilizador da ferramenta poderG̃ nŃo ter conhecimento do código das GUIs, esta estratégia não se aplica no nosso estudo.

Análise de fluxo de contolo

- Cobertura de instruções: cada instrução do programa é executada pelo menos uma vez;
- Cobertura de decisões: cobrir todas as possibilidades de uma decisão pelo menos uma vez - num *if*, testar *true* e *false*;
- Cobertura de condições: nas decisões compostas (várias condições) testar os vários resultados possíveis de cada condição;
- Cobertura modificada de condições e decisões: mostrar que cada condição numa decisão afeta o resultado da decisão;

- Cobertura de condições múltiplas: gerar casos de teste para todas as combinações de condições;
- Cobertura de caminhos: executa todas as possíveis sequências de instruções do programa.

Análise de fluxo de dados Adiciona-se os dados ao grafo de fluxo de controlo e são gerados os casos de teste que executem todos os pares de escrita/leitura de instruções para cada variável.

2.1.2 Ferramentas de teste

Nos últimos tempos e face ao grande número de dificuldades encontradas nos testes de GUIs, têm vindo a ser desenvolvidas várias ferramentas de teste que são a combinação de diferentes técnicas de teste existentes adaptadas ao teste de interfaces gráficas.

De seguida são apresentadas as ferramentas mais utilizadas, assim como as suas vantagens e desvantagens.

2.1.2.1 *Capture/Replay*

As ferramentas de *Capture/Replay* ou *Record/Playback* gravam uma primeira sequência de passos (dados de entrada e saída) que o utilizador/programador executa e volta a executar esses mesmos passos, usando o script que guardou no início. Quando uma pequena alteração é efectuada, é possível correr os scrips de testes que a ferramenta guardou e verificar se essa alteração modificou ou não o resultado esperado através da comparação dos dados de saída das duas etapas. A nível de interfaces gráficas, guardam todos os passos do utilizador, tais como: cliques e movimentos do rato, uso do teclado, imagens do ecrã. Este tipo de ferramentas é muito útil nos testes de regressão - testes usados na comparação de diferentes versões da mesma aplicação. [Bur03]

Tem como principais vantagens permitir que um grupo de testes seja repetido automaticamente as vezes que forem necessárias (economização de tempo) e a facilidade com que é possível encontrar a falha e a sua origem.

Apesar das suas vantagens estas ferramentas, ao representarem a aplicação de um modo bastante específico e não abstracto, tornam-se bastantes sensíveis a pequenas alterações. Por exemplo, uma pequena mudança de layout de componentes pode levar à necessidade de execução de um grande conjunto de testes manualmente perdendo assim a sua principal vantagem. [UBL07] O investimento inicial neste tipo de ferramentas é bastante elevado, visto que, para além dos custos das ferramentas é ainda necessário investir na formação dos seus utilizadores para que estes obtenham todas as vantagens destas ferramentas.

Algumas das ferramentas usadas neste tipo de testes: *Jacareto* -[Jac08], *Marathon* -[Kar].

2.1.2.2 Testes unitários

Os testes unitários, tal como o próprio nome indica, são testes realizados a unidades de código. "Uma unidade é o componente de software mais pequeno possível de testar". No caso de código, uma unidade pode ser um método ou uma classe (programação orientada a objetos). Neste caso, são criados testes que permitem verificar as funcionalidades de uma determinada unidade do código, sendo que se torna bem mais fácil do que testar todas as funcionalidades da aplicação no mesmo teste. [Bur03] Existem variadas ferramentas de testes unitários disponíveis para a maior parte das linguagens de programação.

No caso das GUIs, não é possível identificar o código que representa a interface, então é necessário conseguir identificar as suas componentes no ecrã (por exemplo, uma *combobox* ou uma *checkbox*). Existem algumas *frameworks* que permitem identificar esses componentes e desenvolver (programar) casos de teste para testar o seu comportamento: *Abbot* (*Java*) [Wal02], *jfcUnit* (*Java Swing*) [MC04], *UI Automation(.Net)* [Mic]. Mais informações sobre as ferramentas podem ser encontradas nas respectivas referências.

Um dos principais problemas neste tipo de testes consiste no facto de a construção dos casos de teste também não ser um processo automatizado, necessitando de algum esforço de programadores. Por último, existe também a dificuldade de detectar os erros da aplicação que são devidos a sequências de interação pouco comuns ou não previstas, visto que o número de açoes (e combinações) possíveis é muito elevado.

2.1.2.3 Testes aleatórios

Ferramentas de testes aleatórios, também conhecidos por testes do macaco, são ferramentas que simulam os dados de entrada a testar na aplicação - escolhem aleatoriamente os valores a introduzir - a partir de um domínio de valores de entrada pré-definidos [Bur03].

São ferramentas que são fáceis de implementar, de custos reduzidos e que podem encontram bastantes erros permitindo assim poupar tempo e esforço. Estas ferramentas são mais eficazes numa fase inicial da aplicação [Bur03] [HPW09].

Apesar de tudo isto, não há nenhum tipo de controlo sobre as acções das ferramentas nem dos erros que esta encontra. Para superar estas falhas foi criado o *Smart Monkey* em oposição ao *Dumb Monkey* que aqui foi explicado. Nesta técnica, ao contrário do *Dumb Monkey*, a ferramenta possui algum conhecimento do funcionamento da aplicação - tabela de estados ou modelo de *software*. Nesse caso, a aplicação escolhe aleatoriamente um estado para começar e, seguidamente, escolhe uma das opções válidas para esse estado. Assim sendo, existe algum controlo das ações efectuadas pela ferramenta de teste. Em contrapartida, o uso de *Smart Monkeys* tem um custo bastante elevado [Nym00].

Um dos exemplos de ferramentas mais conhecidos de testes aleatórios é a *Rational TestFactory* [Cor01].

2.1.2.4 Baseada em modelos

A ferramenta de testes baseados em modelos permite a geração de casos de testes a partir de um modelo. Este modelo pode ser de vários tipos: domínio, ambiente, comportamento e testes abstractos.

No caso das interfaces gráficas é mais comum aplicar-se a técnica baseada no modelo de comportamento. O modelo é construído com base nos requisitos da aplicação e compara os resultados com os valores esperados. O modelo tem de ser capaz de relacionar os valores de entrada e saída da aplicação, assim como ter o conhecimento prévio do comportamento esperado da aplicação. Assim sendo, esta técnica permite iniciar a geração dos casos de teste numa fase inicial da sua construção (não é necessário acesso ao código), coisa que não acontece com as anteriores.

O funcionamento desta técnica consiste na criação de um modelo que terá de representar a aplicação a testar e terá de ser o mais pequeno e detalhado possível. Este modelo pode ser criado pela equipa de testes a partir da aplicação, ou pode ser reutilizado o modelo construído pela equipa de desenvolvimento da aplicação. No segundo caso, o modelo não deverá ser utilizado na sua totalidade mas apenas como referência, pois normalmente não estará completo [UBL07].

De seguida é utilizada a ferramenta baseada em modelos para gerar o conjunto de casos de teste abstractos, que são uma sequência de operações relacionadas com os valores de entrada e os valores de saída esperados. Para a transformação em scripts de testes executáveis são usadas alguns templates e tabelas de transformações [UBL07] [ATa10].

Atualmente existem várias implementações de testes de GUIS baseados em modelos, sendo que a sua principal diferença reside na técnica utilizada para a construção do modelo e para a geração de casos de teste a partir dele. Seguidamente são apresentadas algumas das abordagens mais utilizadas com uma breve explicção, assim como algumas ferramentas utilizadas.

Máquina de estados finita A máquina de estados finita é um modelo representado por um conjunto de nós (que representam estados) e setas direcionadas (que representam transições). Este modelo garante que, num dado momento, a aplicação está num estado e que quando ocorre um evento ou ação a aplicação muda de estado. Pode-se afirmar que este modelo tem um comportamento sequencial. O número de estados assim como o número de transições tem obrigatoriamente de ser finito e não deve ser muito elevado pois poderá causar problemas na construção do modelo [UBL07] [EFW01].

Na modelação de GUIs, cada estado representa o conjunto de janelas e objetos num determinado momento e as transições são provocada por eventos. O número de estados da mesma interface gráfica varia conforme a sua definição: quanto mais detalhada for a construção do modelo e a definição de estado, maior o número de estados e maior a complexidade do modelo. É por isso importante escolher um critério de definição adequado, suficientemente detalhado para a geração dos casos de teste mas não demasiado complexo [YM10] [HJ09].

Uma das principais desvantagens desta abordagem é a possível explosão de estados, visto as interfaces gráficas serem bastante complexas com um número elevado de possíveis sequências de eventos. Uma solução passa por aumentar o nível de abstração do modelo, tornando a máquina de estados mais pequena ou então criar um conjunto de máquinas de estado mais simples que representem as diferentes partes da interface.

Um exemplo de ferramenta baseada em máquinas de estados finitas é a *GuiTam* (*GUI Test Automation Model*). O objetivo da *GuiTam* é automatizar todo o processo de geração e execução de casos de testes a interfaces gráficas [YM10]. A máquina de estados utilizada é constituída pelos seguintes elementos: um conjunto (não-nulo) de eventos, um conjunto finito de estados, um estado inicial e um conjunto de relações estado/transição(ões).

A ferramenta permite a interação automática com a interface e utiliza um algoritmo específico para gerar o modelo respectivo. Depois do modelo criado, é possível gerar os casos de testes a partir da máquina de estados. Assim sendo, todo este processo é automatizado.

Modelo de fluxo de eventos Este tipo de modelo só pode ser aplicado a interfaces gráficas que possam ser definidas como uma sucessão de eventos pré-definidos, em que os eventos provocam alterações ao estado da interface. O estado de uma interface, num dado momento, é representado pelos seus objetos e respetivas propriedades.

O principal objectivo do modelo é o de representar todas as possíveis ligações entre os eventos da GUI. Um evento é uma ação que leva à mudança de um determinado estado para outro. Cada um destes eventos é definido através de um conjunto de pré-condições (estado em que o evento deverá ser executado) e efeitos (alterações provocadas pela sua execução).

Nesta abordagem, a interface gráfica é decomposta em vários diálogos restritos, ou seja, em cada um destes diálogos existe uma janela restrita (limita a ação do utilizador a esta janela enquanto esta não for fechada) e um conjunto de janelas não restritas, sendo que o diálogo só termina quando se fechar a janela restrita. Esta divisão é possível visto que não existe interações entre eventos de diálogos restritos diferentes. Deste modo, cada diálogo é convertido num grafo de fluxo de eventos e no final é construída uma árvore de integração que representa as possíveis interações entre os diferentes diálogos. No que diz

respeito aos grafos, os seus vértices representam os eventos e as arestas representam as possíveis transições entre eles.

A versatilidade deste modelo baseado em eventos permite a sua utilização num número alargado de estratégias de exploração. Outra das suas vantagens é a capacidade de gerar um número elevado de casos de teste rapidamente e o facto de ser reutilizável.

A sua maior dificuldade centra-se na restrição de possíveis interações entre eventos, visto que quanto maior o número de eventos presentes na GUI, maior se torna o modelo [Mem07].

GUITAR é o nome de um software criado para testar interfaces gráficas baseado em modelos. Esta ferramenta permite, entre outras coisas, gerar e analisar os grafos de fluxo de eventos e gerar automaticamente casos de testes através de algoritmos pré-definidos. [JS11]

UML - *Unified Modeling Language* UML é uma linguagem de modelação que é utilizada como apoio ao desenvolvimento de aplicações. Esta linguagem permite especificar os requisitos da aplicação através de variados diagramas o que permite ao programador um melhor entendimento do problema. Os diagramas de classes, de casos de uso e de atividades são dos mais utilizados.

Nesta abordagem o modelo será o(s) diagrama(s) de UML criados para representar o comportamento e as especificações da aplicação a testar. A geração automática dos casos de teste é efetuada através dos diagramas. O tipo de diagramas a utilizar depende dos objetivos dos testes bem como da aplicação em causa [EFW01].

Um exemplo de abordagem é o de Vieira, Marlon [MV06], que usa diagramas de casos de uso e de atividades para modelar a aplicação. O primeiro é usado com o objetivo de descrever a relação entre os atores e os casos de uso, já o segundo é utilizado para definir como cada um dos casos de estudo será testado. De modo a tornar a cobertura de testes mais abrangente, é possível acrescentar algumas notas pré-definidas aos diagrama de atividades. Os testes são gerados a partir de cada um dos casos de uso, pois é considerado que para cada caso de uso existe pelo menos um diagrama de atividades.

Uma das principais desvantagens desta abordagem é a construção dos modelos: a sua possível complexidade e o tempo despendido nessa tarefa. Por outro lado, torna-se bastante menos demorado efetuar alguma alteração aos casos de teste previamente gerados.

Labelled Transition Systems **(LTSs)/***Keyword, Action word* O modelo desenvolvido nesta abordagem resulta de uma combinação de duas técnicas: Sistemas de transição rotulados (LTS) e o conjunto de *Keyword* e *Action word*.

Um LTS é uma máquina de estados constituída por um conjunto de estados, um conjunto de transições, um conjunto de ações e um estado inicial. As diferenças em relação à

máquina de estados finita são: a inexistência de estado final e de restrição do número de estados e transições (pode não ser finito).

A *Action word* representa um evento, ações de alto nível que simulam os atos do utilizador, que necessitam de ser implementadas por ações de mais baixo nível - *Keywords*. Uma *Action word* deve ser implementada por um conjunto sequencial de *Keywords* relacionadas com esta. A diferença entre *Action words* e *Keywords* nem sempre é fácil de identificar, sendo que os principais aspectos a ter em conta são o nível de abstração do evento e a sua utilização.

A combinação resulta na construção de sistemas de transição para as ações de alto-nível (*Action words*) e sistemas de transição para as ações de baixo-nível (*Keywords*). O último passo é a composição paralela, ação em que os diferentes LTSs são compilados de modo a criar um modelo com as ações possíveis a executar em cada estado [AK06].

O TEMA *project* tem vindo a ser desenvolvido ao longo dos últimos anos na *Tampere University of Technology, Department of Software Systems* (TUT/DSS) [oT11]. O projeto é composto por um conjunto de ferramentas de teste de interfaces gráficas baseadas em modelos para plataformas *mobile*, mais propriamente, os LTSs e as *Action words e Keywords*. Recentemente introduziram também o conceito de LSTS (*Labelled State Transition Systems*), em que difere do LST apenas na possibilidade de guardar informações (etiquetas) acerca dos estados e das transições.

A modelação é divida em quatro categorias: máquinas de ação, máquinas de refinamento, máquinas de lançamento e máquinas de inicialização. As duas primeiras são as mais importantes na definição do modelo, visto que as máquinas de ação modelam a aplicação ao nível de *Action words* e as de refinamento definem as *Keywords* a implementar para cada ação. As restantes máquinas apenas definem funções de apoio. No final os modelos são compilados em modo de composição paralela. Este processo serve para verificar a ligação entre os vários modelos e especificar quais as ações que podem ser efetuadas num determinado estado, tendo em conta todas as dependências entre os diferentes modelos analisados.

Este tipo de abordagem, segundo os autores, está mais orientada e preparada para o mundo empresarial do que as restantes, visto esta ser uma abordagem de domínio específico, logo menos generalizada. Uma dificuldade que advém deste facto, é a implementação de casos de teste que variam um pouco do domínio e podem ser importantes no que diz respeito aos testes [AJ08] [Sat06].

As técnicas de teste de GUIs baseadas em modelos existentes são imensas e são uma mais-valia nesta área. Permitem a geração de um número elevado de testes em comparação com outras técnicas, de forma mais rápida e eficiente. Outra das suas vantagens é a fácil manutenção. Quando ocorre uma mudança, basta alterar a parte do modelo respetiva e tudo o resto é tratado.

Ainda assim, existe muita relutância por parte do ambiente empresarial em adoptar estas ferramentas. Na origem disto encontram-se algumas das limitações que estas podem oferecer. Por exemplo, se o modelo gerado tiver erros, é provável que os testes terão erros também. Existe então a necessidade de ter funcionários especializados a utilizar as ferramentas disponíveis o que pode trazer custos elevados: quer monetário, quer de tempo (formações).

Baseada em padrões A abordagem baseada em padrões está a ser desenvolvida com o objetivo de possibilitar a geração e execução de casos de teste para GUIs com o menor esforço possível, ou seja, automaticamente e com um aumento de reutilização. A ideia é decompor a interface gráfica em pequenos componentes e não em eventos, como têm sido feito anteriormente. Com isto, pretende-se contornar o problema da complexidade das GUIs, tornando o processo de testes bastante simples e intuitivo. Nas seguintes secções são apresentados os padrões e o protótipo de ferramenta que existe nesta abordagem e que serve de base para este projeto.

2.2 Padrões

Os padrões podem ser usados de modo a repartir um problema de grandes dimensões em vários problemas mais pequenos e de solução facilitada. É esta a sua principal vantagem e a razão pela qual têm vindo a ser bastante exploradas as suas possíveis utilizações.

Em primeiro lugar, nesta secção, é explicado o que é um padrão e apresentam-se alguns exemplos. Seguidamente foca-se a atenção numa categoria específica de padrões de interação: a pesquisa. Por último, é apresentada a ferramenta PETTool.

2.2.1 Definição

Os padrões de interação são componentes com comportamentos semelhantes que pretendem ser uma solução para problemas recorrentes das GUIs. Os comportamentos podem ser semelhantes quer a nível de estrutura quer de comportamento e estão presentes em interfaces gráficas. O uso destes componentes no desenho de interfaces é uma mais valia, pois são uma solução eficaz para problemas de interação e comunicação [Bin99] [Tid10].

Os principais benefícios de um padrão são: o facto de serem soluções comprovadas para um determinado problema; poderem ser reutilizados para diferentes situações e a sua estrutura - a sua constituição pode ser consultada no próximo parágrafo - permite apresentar soluções complexas de forma simples e elegante [Osm11].

Os padrões são tipicamente constituídos pelos seguintes elementos: [EGRHRJ94]

- Nome: Identificador do padrão;

- Problema: Explicação do problema e do contexto;
- Solução: Elementos do padrão e a relação entre eles;
- Consequências: Resultados da aplicação do padrão.

Estas soluções permitem generalizar os problemas (das interfaces gráficas), tornando-se aqui um fator a ter em conta na hora de testar as interfaces. Isto é, se facilitam no desenho da interface, podem também ajudar no teste da mesma, simplificando esse processo. Neste capítulo serão apresentados alguns exemplos de padrões de interação com o utilizador e será também explicada qual a abordagem que se pretende seguir no desenvolvimento da ferramenta - a perspectiva de solução [EGRHRJ94] [Bin99] [Osm11].

O número de padrões existentes é bastante elevado, muitos deles facilmente identificáveis por um simples utilizador, outros um pouco mais complexos e difíceis de identificar. No que diz respeito a padrões de desenho, existem várias abordagens em que cada autor cria a sua própria linguagem de padrões, sendo que é possível encontrar padrões semelhantes com nomes diferentes, dependendo da fonte.

Os padrões que se seguem são alguns dos mais comuns que se podem encontrar nas interfaces gráficas e já se encontram implementados na ferramenta PETTool [dMC10].

- Campo de entrada de dados: Qualquer elemento da interface que se destina à recolha de informação introduzida pelo utilizador. Podem ser caixas de texto ou de seleção. Como exemplo temos todos os campos num formulário de registo. Na figura 2.1 é apresentado o formulário de registo do Facebook;

Figura 2.1: Formulário de registo do Facebook

- Autenticação: Elemento que permite ao utilizador identificar-se à aplicação. É composto por dois campos de entrada de dados (nome de utilizador e palavra-passe) e um botão de submissão. A imagem 2.2 representa o formulário de autenticação do Gmail;

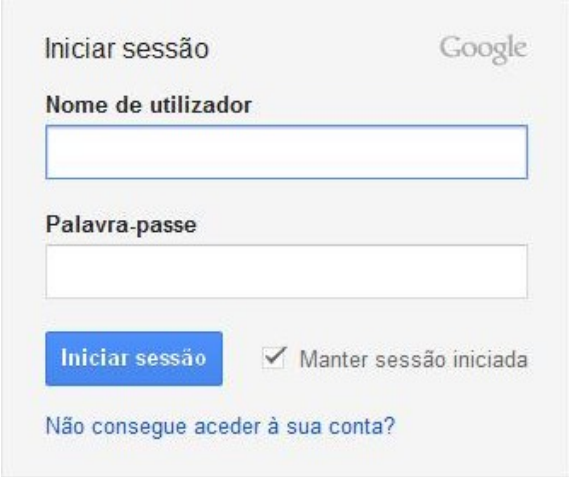

Figura 2.2: Autenticação no Gmail

- Mestre/Detalhe: É um padrão que representa uma relação entre dois campos de dados, sendo que a alteração de um desencadeia a alteração dos valores do outro. O componente que provoca a alteração é o Mestre e o que sofre essa mudança é o Detalhe. Um dos exemplos mais utilizados são as *comboboxs* de seleção de Distrito e Concelho, como demonstrado na figura 2.3;

Figura 2.3: Pesquisa de imóveis na página www.era.pt

- Campo calculado: Elemento cujo valor depende do valor de outros campos. Pode ser calculado a partir do valor de um ou mais campos. Um dos casos em que é utilizado é o conversor de moedas, como pode ser consultado na imagem 2.4;

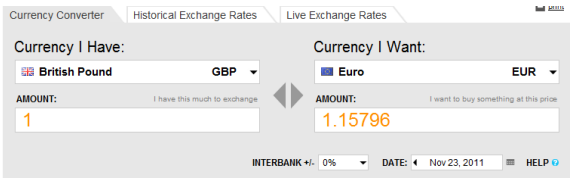

Figura 2.4: Conversor de moeda em www.oanda.com

As interfaces gráficas possuem uma enorme variedade de componentes, o que faz com que exista um número elevados de padrões de interação. Os que aqui se encontram são uma pequena amostra do que existe.

Na secção seguinte é apresentada uma característica bastante presente na maioria das aplicações: a pesquisa. Nesse sentido existe uma definição dessa funcionalidade bem como alguns dos padrões com ela relacionados.

2.2.2 Pesquisa

Este projeto focou o estudo numa funcionalidade específica das interfaces gráficas: a pesquisa. A pesquisa está presente numa parte significativa das aplicações o que faz com que se torne uma prioridade testar o seu desempenho.

A sua importância no desempenho das aplicações obriga a que esta funcione sem problemas. Quando o desempenho de uma pesquisa não é o esperado, o utilizador pode perder o entusiasmo e evitar o uso de toda a aplicação. É vital evitar este cenário, sendo este outro motivo da necessidade de testar a pesquisa.

O principal objetivo da pesquisa é o de encontrar algo, a informação desejada e para ser bem sucedida é necessário que esta seja simples, rápida e eficaz. Mas a complexidade desta ação é maior do que aparenta, pois existem vários fatores que complicam o processo.

Uma pesquisa pode ser tão simples como colocar uma questão e esperar uma resposta (os resultados esperados), mas o tipo de informação pesquisada, o número de resultados obtidos, a estratégia a utilizar para os encontrar, a clareza da questão colocada vão influenciar cada pesquisa de diferentes formas. Por exemplo, existem pesquisas simples constituídas apenas por uma caixa de texto e um botão de submissão e existem pesquisas em que para além desses dois itens é possível acrescentar filtros e/ou categorias à pesquisa, reduzindo o domínio da pesquisa.

Esta sua complexidade torna o processo de desenvolvimento muito complicado, razão pela qual têm vindo a ser criados vários padrões de modo a facilitar o processo. Obviamente o processo de testar a pesquisa é também um problema, sendo que para automatizar este processo se irá fazer uso dos padrões de pesquisa.

2.2.2.1 Padrões de pesquisa

Na pesquisa podemos encontrar dois tipos de padrões: os de comportamento e os de desenho [PM10] [vW08].

Os padrões de comportamento, tal como o próprio nome indica, são relativos a comportamentos efectuados numa pesquisa. Alguns desses padrões encontram-se descritos em baixo.

Sair - *Quit* Abandonar uma pesquisa assim que esta acaba, seja com sucesso ou não.

Reduzir - *Narrow* Reduzir número de resultados retornados na pesquisa, através de filtros.

Expandir - *Expand* Aumentar número de resultados retornados na pesquisa.

Este tipo de padrão não é relevante para o projeto em estudo.

Os padrões de desenho são aqueles que ajudam no desenvolvimento da aplicação, que facilitam o processo de criar a pesquisa e de a tornar um sucesso. São estes os padrões que nos ajudarão a criar um modelo genérico de testes para a pesquisa.

Este tipo de padrões varia em número e em tipo dependendo do autor da publicação onde estão inseridos. Os padrões que se seguem são uma seleção dos mais importantes e relevantes para este projeto [PM10] [vW08].

Dos padrões selecionados podemos destacar três categorias gerais em que estes se inserem: definição da pesquisa, navegação e resultados.

Na definição da pesquisa encontram-se os seguintes padrões:

Caixa de pesquisa - *Search box* É constituída pela caixa de pesquisa e o botão de submissão. Pode ou não conter um campo de filtragem de dados (Figura 2.5).

Dicas de pesquisa - *Search tips* Pequena explicação para o preenchimento do campo. Em alguns casos existe um caso de demonstração. Pode ser visto um exemplo na figura 2.6, no lado direito de 3 dos campos de pesquisa existe uma hiperligação com o nome "dica". Quando esta é selecionada aparece uma caixa de texto com a respetiva dica.

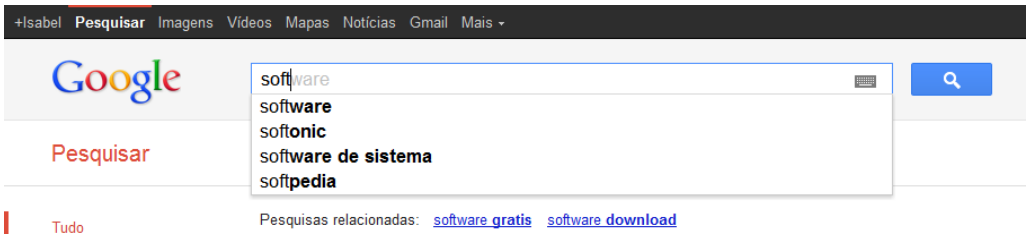

Figura 2.5: Pesquisa Simples no Google, com demostração do Auto-completar

Pesquisa avançada - *Advanced search* Pesquisa através da *Search Box* com elementos adicionais auxiliares à pesquisa. Especialmente útil quando o número de elementos a pesquisar é bastante elevado. A pesquisa pode ser melhorada em três componentes: Correspondência do termo; Delimitação do âmbito; Controlo do output - paginação, ordenação. Figura 2.6.

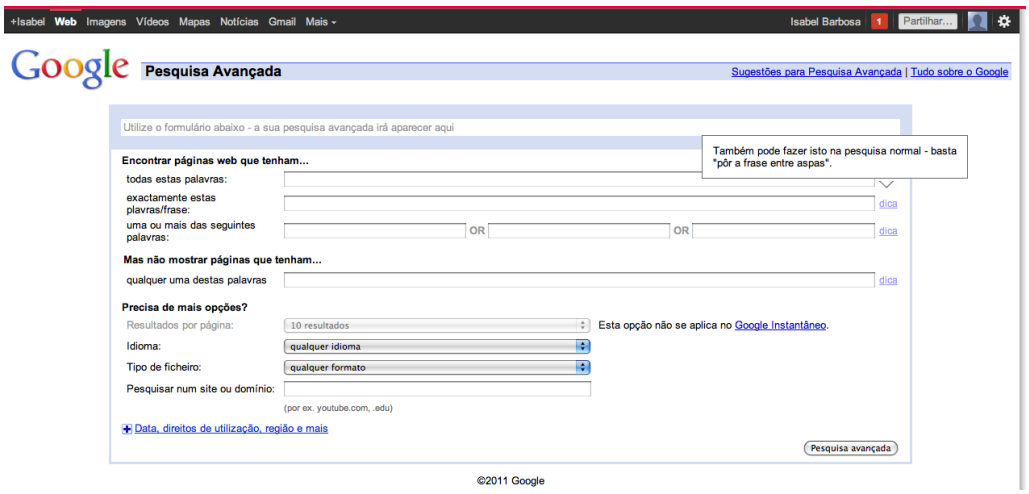

Figura 2.6: Pesquisa avançada no Google

Área de pesquisa - *Search area* Área dedicada à pesquisa que é constituída pela Caixa de pesquisa e possíveis elementos adicionais que apoiam a pesquisa: Dicas de pesquisa, Mapa do sítio, Índice do sítio e Pesquisa avançada (Figura 2.7).

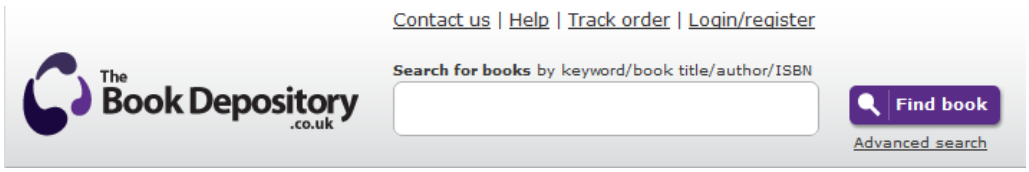

Figura 2.7: Área de pesquisa em http://www.bookdepository.co.uk/

Auto-completar - *Autocomplete* À medida que o utilizador escreve no campo de texto, as possíveis soluções são antecipadas, aparecendo uma lista em que se pode selecionar uma delas, se for o caso pretendido, como demonstrado na figura 2.5.

Salto para item - *Jump to item* Quando um utilizador escreve o nome do elemento numa árvore ou tabela, vai diretamente para o elemento que corresponde à informação introduzida e seleciona o item.

Os seguintes padrões referem-se à navegação:

FAQ - *Frequently Asked Questions* Conjunto de questões e respostas a problemas frequentemente encontrados pelo utilizador (Figura 2.8).

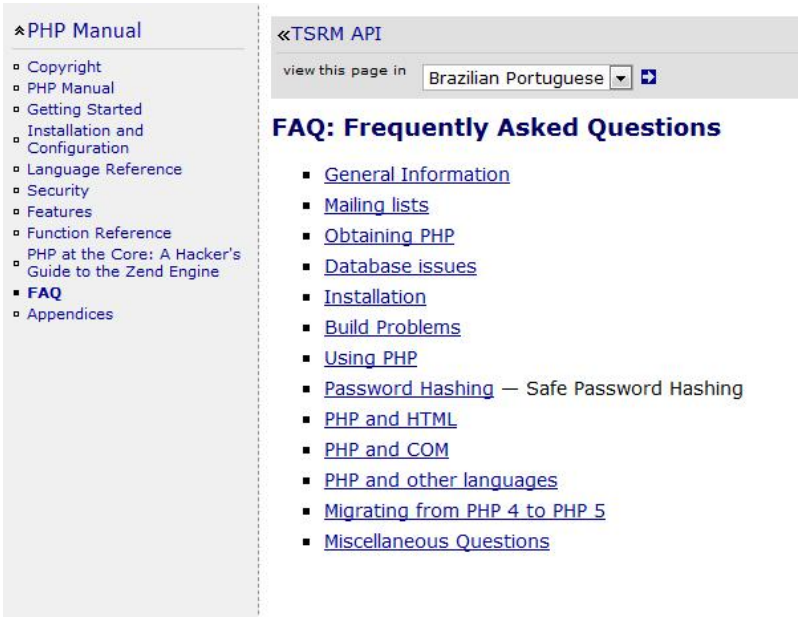

Figura 2.8: Área de perguntas frequentemente colocadas em pt2.php.net

Assistente de ajuda - *Help wizard* Assistente que fornece apoio ao utilizador, numa área especificada por este. Apresenta um número de resultados bastante reduzido.

Índice do sítio - *Site index* Apresentação de resultados divididos por índice: alfabético ou categoria.

Mapa do sítio - *Sitemap* Estrutura hierárquica do sítio numa página. Mostra pelo menos 2 níveis e todos os elementos de cada nível. Deve ser acessível a partir de qualquer página. A página a partir da qual acedeu deve estar assinalada no mapa.

Mapa do sítio em rodapé - *Footer sitemap* Mapa do sítio ou lista das principais páginas do website (dividas em categorias) no fundo da página (Figura 2.9).

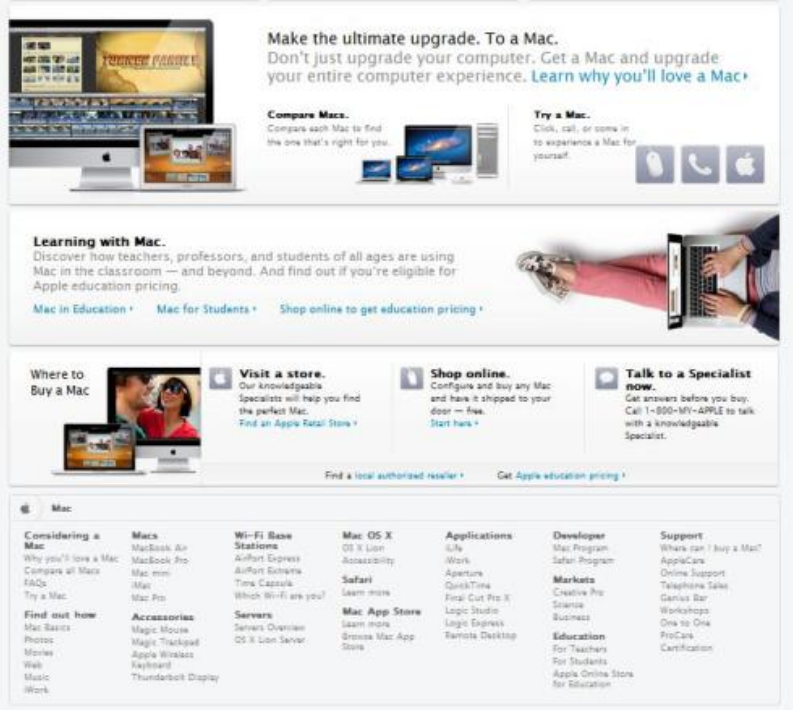

Figura 2.9: Mapa do sítio em rodapé em www.apple.com

Nuvem de tags - *Tag cloud* Lista de tags mais usadas e mais populares no sítio. Elementos ordenados alfabeticamente. Itens mais populares em destaque (Figura 2.10).

Figura 2.10: Nuvem de tags no Amazon

Páginas de tópicos - *Topic pages* Páginas com informação relevante sobre um determinado tema (mais popular/importante) com ligações para os documentos/artigos mais relevantes.

Paginação - *Pagination* Divisão dos resultados esperados em páginas, quando o seu número é demasiado elevado. O número máximo de elementos por página é variável (normalmente entre 10 e 20). A figura 2.11 apresenta um exemplo de paginação.

Por último definimos os padrões que estão relacionados com os resultados da pesquisa:

Resultados das pesquisa - *Search results* Apresentação dos resultados obtidos após uma pesquisa. Lista de resultados, em que cada resultado tem uma pequena descrição (2/3 linhas), como pode ser observado na figura 2.11.

Figura 2.11: Lista de resultados e paginação no Google

Melhores primeiro - *Best first* Os primeiro elementos resultantes da pesquisa devem ser os mais relevantes. Os 3 primeiros elementos retêm 80% de atenção.

Navegação facetada - *Faceted navigation* Pesquisa simples (palavra-chave), cujos resultados podem ser reduzidos através de filtros, após a pesquisa inicial.

Personalização - *Personalization* Uso de informações pessoais (histórico, informação social, localização) para obter resultados mais indicados para o utilizador. É um elemento de ajuda nos padrões Auto-completar e Melhor primeiro.

Resultados estruturados - *Structured results* Elementos resultantes da pesquisa apresentados com novos elementos (imagens, mapas, gráficos, vídeos, entre outros). Proporcionam uma melhor perceção desse elemento e até podem permitir ao utilizador suprimir mais alguns passos até chegarem à informação desejada.

Figura 2.12: Exemplo de apresentação de resultados estruturados na pesquisa do Google

Estes são alguns dos padrões mais relevantes no que diz respeito à pesquisa. No próximo capítulo será analisada o seu contributo possível no que diz respeito à geração automática de testes.

A pesquisa é uma ação bastante complexa e ainda é muito complicado implementá-la e consequentemente testá-la com sucesso.

O uso de padrões de interação pode ser um grande aliado nestes campos. De seguida apresentam-se algumas informações importantes sobre uma ferramenta que permite a geração e execução de testes para interfaces gráficas com base em padrões de desenho.

2.2.3 PETTool

A PETTool é uma ferramenta de teste a padrões (*PattErn Testing Tool*). Esta ferramenta tem como grande objetivo a geração e execução de testes automaticamente a partir de padrões de interação.

Esta ferramenta está numa fase inicial do seu desenvolvimento, sendo que foi iniciado por um aluno da FEUP em 2010, no seu projeto de dissertação. A PETTool é ainda um protótipo que se pretende expandir e melhorar para poder entrar no mercado como uma ferramenta inovadora no que diz respeito aos testes a interfaces gráficas.

O número de padrões que é possível testar é ainda bastante reduzido mas ainda assim já é possível comprovar a sua eficácia nos seguintes padrões: Formulário, Campo de Entrada de Dados, Campo Calculado, Mestre/Detalhe e Autenticação.

A PETTool foi desenvolvida em C#, e foi utilizada a *framework UI Automation* (Explicação no próximo capítulo) para permitir a interação automática com a interface gráfica.

2.2.3.1 Funcionamento da Ferramenta

A ferramenta permite a criação de vários padrões assim como a configuração de testes relativamente a cada um, dos quais resultará a geração de casos de testes específicos para cada padrão consoante a especificação dada pelo utilizador.

Inicialmente o utilizador deve selecionar um padrão do tipo formulário e o botão de submissão respetivo. O passo seguinte passa por criar os padrões desejados que se encontram no formulário previamente guardado. É possível criar vários formulários e cada um pode ter vários padrões associados, desde que estes se encontrem dentro da janela ou secção do formulário.

Ao criar o padrão deve selecionar o(s) campo(s) respectivo(s), se for o caso deve selecionar outros padrões relacionados com o atual (ver explicação do padrão Autenticação) e configurar os testes como desejar. A configuração varia de padrão para padrão, mas no geral cada padrão tem um conjunto de comportamentos esperados e são esses comportamentos que serão configurados pelo utilizador. Estes comportamentos representam o que é suposto acontecer em cada caso - no caso de um Campo de Entrada de Dados existem três: valor válido, valor inválido e valor em branco. A estratégia de teste definida para cada padrão é o que define os comportamentos esperados de cada um.

Quando todos os testes estão configurados, a ferramenta executa os testes e verifica se os resultados obtidos estão ou não em conformidade com os resultados esperados. O relatório final apresentado refere para cada padrão quais os testes que sucederam, os que falharam e os que não foram realizados explicando a razão.

Para clarificar um pouco mais este processo vamos demonstrar um exemplo: a Autenticação.

Este padrão é constituído por um campo de nome de utilizador, um campo de palavra-passe e um botão de submissão. O botão já se encontra definido no formulário. Os restantes campos são Campos de Entrada de Dados. Assim sendo é necessário definir primeiro estes dois padrões. Na figura 2.13 é visível um exemplo de definição do Campo de Entrada de Dados correspondente ao nome de utilizador.

Agora já é possível definir o padrão Autenticação. Selecionam-se os respetivos padrões para o nome de utilizador e password e configuram-se os valores esperados para valores válidos e inválidos (Figuras 2.14 e 2.15).

Neste momento é possível correr os testes e analisar os resultados, observando o relatório gerado. A figura 2.16 apresenta um relatório gerado para o padrão de Autenticação [dMC10].

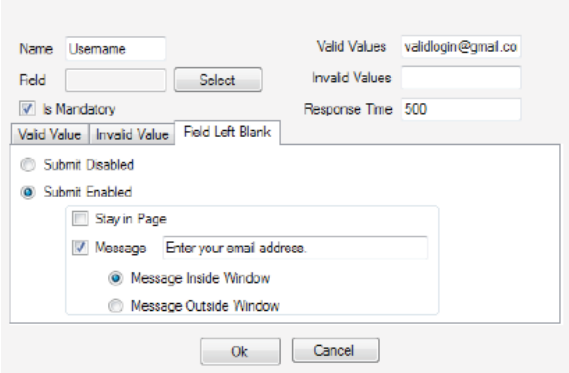

Figura 2.13: Exemplo de configuração do campo nome de utilizador

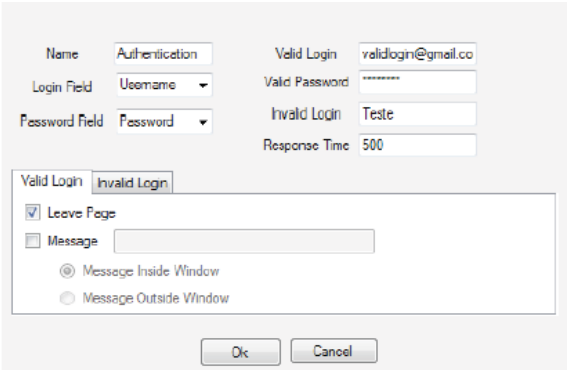

Figura 2.14: Configuração para valores válidos do padrão Autenticação

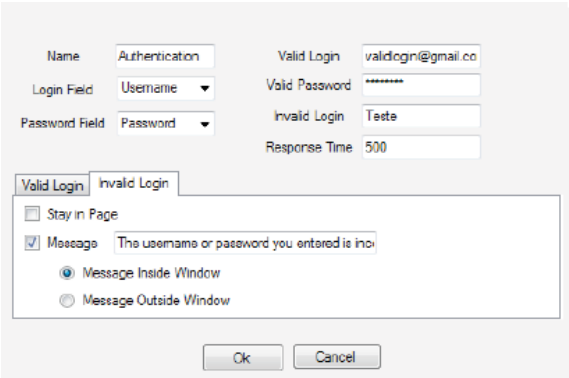

Figura 2.15: Configuração para valores inválidos do padrão Autenticação

A grande vantagem desta abordagem é que o tempo despendido para gerar os testes não depende do número de testes, mas sim do número de padrões, visto que cada padrão pode originar vários casos de teste. Isto significa que é possível realizar um maior número de casos de teste num período de tempo mais reduzido [CPFA10].

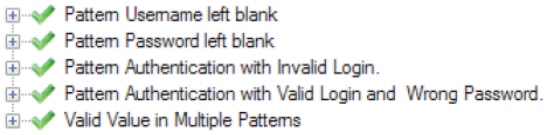

Figura 2.16: Relatório gerado pela PETTool

A ferramenta pode ser ainda melhorada em vários aspectos, como por exemplo no número de padrões que permite testar, a perda de informação quando a aplicação é fechada, entre outros problemas [dMC10].

2.3 Conclusões

A presença constante de interfaces gráficas nas aplicações faz com que seja indispensável testá-las exaustivamente. Esse processo torna-se impossível de ser feito manualmente visto as GUIs serem de um funcionamento um pouco complexo, com um conjunto enorme de sequências de eventos.

Nos últimos tempos têm sido várias as tentativas de criar soluções para este problema. Apesar de já ter sido percorrido um longo caminho e ser possível automatizar um pouco o processo, através de ferramentas de *Capture/Replay* ou abordagens baseadas em modelos, existe ainda pouco interesse das empresas em investir nesta área. Os problemas vão desde gastos elevados nos recursos até à complexidade de algumas das opções.

No sentido de ultrapassar esta falha nos testes de software, a solução que pretendemos apresentar passa por criar uma ferramenta que permita a geração e execução de casos de testes baseada em padrões de interação - cujos benefícios foram explicados anteriormente. Já está em fase de desenvolvimento uma ferramenta com essas características e o objetivo deste projeto é o de dar continuidade à PETTool, inserindo a configuração dos padrões de pesquisa, visto esta ser uma presença assídua na maioria das GUIs.

No próximo capítulo serão apresentados os padrões a implementar na ferramenta, assim como as possíveis relações entre eles e as estratégias de teste a implementar.

Capítulo 3

Abordagem

Após o levantamento de toda a informação relevante sobre os padrões de interação e do trabalho desenvolvido na área de teste de interfaces gráficas, é necessário estudar os problemas levantados e definir as soluções a seguir.

Sendo que a utilização da técnica baseada em padrões é a solução escolhida, através da continuação do desenvolvimento da PETTool, é importante agora definir quais os padrões a implementar e qual a estratégia a seguir para cada um.

Como também já foi referido no capítulo 2, este projeto irá focar-se numa das componentes mais importantes das interfaces gráficas: a pesquisa.

Neste capítulo serão definidos quais os padrões da pesquisa que serão implementados, assim como a estratégia a usar nos seus testes. Serão ainda apresentadas as possíveis relações entre esses padrões e aqueles que já se encontram implementados na ferramenta.

3.1 Padrões de teste a implementar

A pesquisa é um dos elementos mais comuns nas GUIs, visto ser uma grande apoio ao utilizador. Quando a quantidade de informação disponível pela aplicação é elevada, a pesquisa é o meio mais rápido e eficaz para se encontrar o que se deseja. Dada a sua importância e a sua presença em vários tipos de aplicações, a pesquisa precisa de ser bastante flexível. É por isso que ela existe das mais variadas formas, adaptando-se ao ambiente em que está inserida.

Os padrões de interação para elementos de pesquisa refletem isso mesmo: a variedade. Como apresentado na secção 2.2.2.1, os padrões de pesquisa também podem ser divididos em categorias. Por exemplo, os autores [PM10] [vW08] consideraram que a navegação entre conteúdos, tal como, um mapa do sítio *Web* ou um assistente de ajuda são formas de pesquisa.

Neste contexto, decidiu-se que os padrões mais relevantes para o teste de GUIs são os que dizem respeito à definição da pesquisa.

O principal objetivo da ferramenta será testar a eficácia da pesquisa, ou seja, se esta encontrou ou não determinado elemento ou quantos elementos encontrou e, por isso, nem todos os padrões da categoria de definição da pesquisa são relevantes.

A área de pesquisa refere-se apenas a um conjunto de outros padrões de pesquisa num determinado espaço, tornando-se irrelevante.

O mesmo se pode dizer das dicas da pesquisa, do auto-completar e do salto para o item visto estes não interferirem no ato da pesquisa em si. Servem apenas como um apoio ao utilizador, para este pesquisar com mais eficácia.

Os padrões de pesquisa que realmente interessam testar são apenas dois: a caixa de pesquisa e a pesquisa avançada.

Interessa agora verificar quais os seus comportamentos, as diferenças e semelhanças entre os dois e definir qual a estratégia de teste relativa a cada um.

3.1.1 Definição do padrão de teste

A caixa de pesquisa é constituída por uma caixa de texto - onde é introduzido o texto a pesquisar - e o botão de submissão - que inicia a pesquisa.

A pesquisa avançada pode ser vista como uma caixa de pesquisa com parâmetros adicionais que controlam e limitam a pesquisa a efetuar. Os parâmetros podem ser caixas de texto, ou caixas de seleção de opções - *comboboxs*, *checkboxs*, *radio buttons*, entre outros, e o número de parâmetros varia de aplicação para aplicação. Podemos limitar a pesquisa em vários aspetos: categorias, idiomas dos resultados, palavras/frases contidas ou não. Estes são apenas alguns exemplos do que pode ser encontrado.

Se pensarmos nestes dois padrões desta forma, podemos pensar que são o mesmo padrão variando apenas no número de parâmetros.

Temos então um padrão a implementar, ao qual chamamos **Pesquisa**, com as seguintes características:

- Nome: Pesquisa;
- Problema: O utilizador tem a necessidade de encontrar um item ou um conjunto de itens num conjunto de informação;
- Solução: Oferecer uma área de pesquisa composta por um ou mais campo de entrada de dados - que representam os parâmetros da pesquisa - e por um botão de submissão - elemento que inicia a pesquisa;
- Consequências: A aplicação deste padrão pretende gerar uma lista de elementos (que pode até ser vazia) que correspondam às necessidades do utilizador.

Esta simplificação de todos os padrões em apenas um, leva a uma generalização bastante importante que facilitará a implementação dos testes e a utilização da ferramenta em qualquer tipo de aplicação, sendo esta uma das mais-valias desta abordagem.

Depois de definido o padrão a implementar é necessário definir qual a estratégia de testes a aplicar.

3.1.2 Definição da estratégia de teste

Em primeiro lugar é importante permitir ao utilizador da ferramenta fazer algumas verificações aos resultados obtidos. Como foi dito anteriormente, o objetivo é verificar se encontrou um elemento específico, ou o número de elementos encontrados. A única maneira de fazer essas verificações é pelo texto apresentado na página de resultados.

As **verificações a implementar** são as seguintes:

- Existe uma das palavras - O utilizador introduz um conjunto de palavras, separadas por espaço e o software procura cada uma das palavras individualmente, retornando verdadeiro se encontrar pelo menos uma das palavras;
- Existe a expressão - O utilizador introduz uma expressão e o software procura a expressão completa e não as palavras em separado, retornando verdadeiro se a encontrar;
- Não existe a expressão - Mesmo que o anterior mas, ao invés desse, retorna verdadeiro se não encontrar essa expressão.

Com este conjunto de verificações é possível testar um número variado de fatores.

Por exemplo, a presença ou não de uma palavra ou expressão pode permitir a verificação da existência de um determinado resultado esperado, ou simplesmente de um elemento do resultado. Isto é, imaginando que o utilizador faz uma pesquisa na página do Google, este pode verificar a existência de um determinado resultado através do título da página que pretende. Pode, no entanto, verificar a existência do endereço de URL, o que significa que está a verificar a existência de um dos elementos do resultado: o endereço.

Outro exemplo é a verificação do número de resultados obtidos. Esta verificação não pode ser feita através da contagem de resultados, visto que numa página *Web* só existe acesso aos resultados apresentados na página visualizada e não ao número de resultados total. Assim, a alternativa encontrada para verificar o número de resultados é verificar a existência do número pretendido em toda a página, visto ser habitual existência dessa informação. Pode ser colocado apenas o número, por exemplo "200", ou então, para ser mais específico, a expressão completa "Cerca de 5.830.000.000 resultados"(exemplo do sítio da Google).

Uma verificação a fazer na pesquisa está relacionada com o comportamento da aplicação. É importante verificar se os resultados aparecem na mesma página ou não.

Para além de efetuar as verificações, existe ainda a necessidade de tornar o teste mais completo. De modo a garantir uma área de cobertura abrangente com o menor número de casos de teste possíveis, utilizou-se uma das técnicas abordadas na secção 2.1.1.1 do capítulo anterior.

A partição em classes de equivalência foi a técnica que mais se adequou ao padrão em estudo. A partição foi aplicada em função do número de resultados apresentados, ou seja, ao domínio de saída.

As **classes de equivalência** encontradas são as seguintes:

- Retorna 0 resultados;
- Retorna 1 resultado;
- Retorna N resultados.

Na tabela 3.1 pode ser encontrado um exemplo de configuração dos testes para um padrão Pesquisa. O utilizador pode definir quantas verificações de texto desejar para cada uma das classes (até nenhuma se for esse o caso). Em cada classe é efetuada uma verificação de comportamento. No exemplo, existem 3 verificações para a classe Retorna 0, 1 para a Retorna 1 e 2 para a Retorna N. O texto entre parêntesis representa o texto que se pretende verificar (a sua existência ou não). No exemplo não se encontra demonstrada a verificação do comportamento da pesquisa.

Retorna 0	**Retorna 1**	**Retorna N**
Existe uma das palavras (*Model-Based Testing*)	Existe a expressão (*GUI Testing*)	Existe a expressão (*Model-Based Testing*)
Existe a expressão (A sua pesquisa não encontrou nenhum documento)		Existe a expressão (*Pattern-Based GUI Testing*)
Não existe a expressão (Cerca de 33.700 resultados)		

Tabela 3.1: Exemplo de conjunto de testes para um padrão Pesquisa

Para garantir uma boa cobertura dos testes, o utilizador terá de fazer pelo menos uma verificação para cada uma das classes apresentadas. A escolha das verificações a efetuar em cada caso é de total responsabilidade do utilizador da ferramenta, o que pode levar à não detecção de algumas falhas da aplicação em teste.

Outro fator importante a ter em conta no teste baseado em padrões são as relações que existam entre diferentes padrões. Na secção seguinte esse tema é discutido.

3.2 Relações entre padrões

Os padrões de desenho são pequenos componentes de interfaces gráficas que definem comportamentos recorrentes. Mas estes componentes, ao contrário do que se possa pensar, não estão isolados. Eles podem interagir entre si e até existem alguns padrões mais complexos que são compostos por um ou mais padrões mais simples.

Estas relações entre os padrões podem ser determinantes para os testes de GUIs. Interessa então aqui mostrar as relações existentes entre os padrões já existentes na PETTool e identificar e adicionar as relações com o padrão pesquisa.

A relação entre os padrões presentes na PETTool pode ser analisada na figura 3.1.

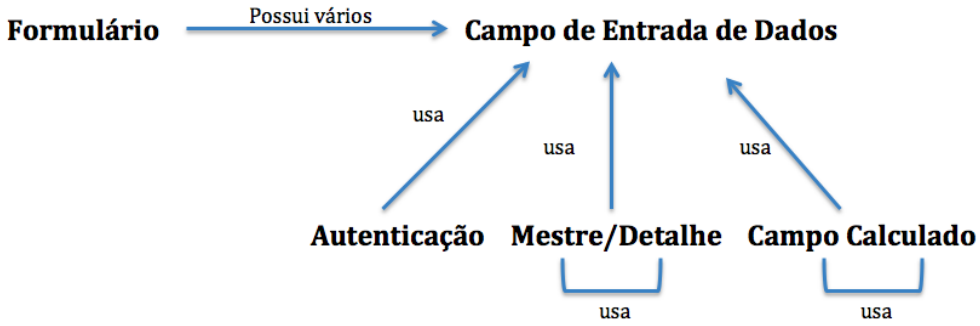

Figura 3.1: Relações entre os padrões antes da Pesquisa

As relações mais fáceis de identificar são sem dúvida as relações com o Campo de Entrada de Dados. Todos os padrões presentes utilizam um ou mais campos de dados na sua estrutura. Por exemplo, no padrão Autenticação existem dois: o campo de nome de utilizador, e o campo da palavra-passe.

Por outro lado a relação entre os mesmos padrões - Mestre/Detalhe usa Mestre/Detalhe ou Campo Calculado usa Campo Calculado - não é tão trivial.

No caso do Mestre/Detalhe representa a possibilidade de existir o encadeamento entre padrões. Um encadeamento acontece quando o campo Detalhe de um padrão é Mestre noutro. Por exemplo, no caso do sítio da ERA Portugal figura 2.3, existe um padrão Mestre/Detalhe para o conjunto Distrito/Concelho e um para Concelho/Freguesia. Este tipo de dependência não pode ser ignorada e os padrões não podem ser testados separadamente.

No caso do Campo Calculado acontece o mesmo. Pode existir dependência entre os padrões quando um dos valores a entrar no cálculo é um Campo Calculado também.

A Pesquisa apresenta algumas relações com os restantes padrões, através dos seus parâmetros. Os parâmetros poderão ser Campos de Entrada de Dados. Existe também a possibilidade de os parâmetros da pesquisa serem do tipo Mestre/Detalhe, como é visível também na figura 2.3. Existe ainda a possibilidade de um dos campos de pesquisa ser

um Campo Calculado. A nova estrutura de relações entre os padrões está apresentada na figura 3.2.

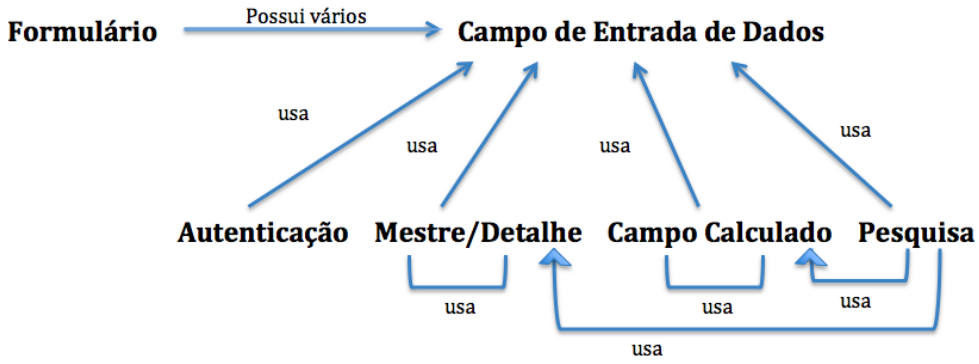

Figura 3.2: Relações entre os padrões depois da Pesquisa

Em termos de teste é importante assegurar que os valores introduzidos na pesquisa estão válidos, sendo ideal testar primeiro os padrões existentes: o Mestre/Detalhe e o Campo Calculado.

No caso do Mestre/Detalhe é necessário verificar se o valor introduzido para o campo Detalhe existe no grupo de Detalhes do valor introduzido no campo Mestre. Esta verificação pode ser efetuada com a definição do padrão Mestre/Detalhe já existente na ferramenta.

Com o Campo Calculado será mais difícil verificar, visto que é necessário identificar também quais os campos utilizados para o cálculo, assim como a operação a executar.

No caso de não ser possível efetuar estas verificações, o utilizador deve ter o cuidado de introduzir sempre valores válidos, de modo a não causar problemas na pesquisa.

3.3 Conclusões

A diversificação dos tipos de pesquisa existentes está relacionada com a sua constante presença no mais variado tipo de aplicações. O elevado número de padrões, que resulta dessa variedade, necessitou de uma análise de modo a identificar quais os padrões relevantes para testar a pesquisa.

A análise efetuada acabou por registar apenas um padrão de teste ao qual foi atribuído o nome de Pesquisa. A Pesquisa é definida por um conjunto de campos de entrada de dados que irão definir os parâmetros e um botão de submissão para executar a pesquisa.

A estratégia de teste definida para o padrão utiliza a partição em classes de equivalência, sendo que para cada uma das classes o utilizador poderá realizar uma série de verificações.

Por último, foram identificadas as relações existentes entre a Pesquisa e os restantes padrões já implementados na PETTool: Campo de Entrada de Dados, Mestre/Detalhe e Campo Calculado.

No próximo capítulo irão ser apresentados pormenores relativos à implementação, tais como, as tecnologias utilizadas e o funcionamento da ferramenta.

Capítulo 4

Implementação

A implementação do padrão Pesquisa na ferramenta tem de ter em conta dois fatores: a estrutura e tecnologias que a ferramenta já possui e as necessidades tecnológicas e estruturais para o novo padrão.

Neste capítulo serão apresentadas as tecnologias usadas na PETTool - as existentes e as novas -, assim como as suas vantagens e limitações.

É ainda descrita neste capítulo a estrutura da ferramenta e o seu modo de funcionamento.

4.1 Tecnologias utilizadas

A automatização da aplicação implica a interação automática com a interface sem a intervenção do utilizador.

Para isso, é necessário recorrer ao uso de ferramentas com essas capacidades. No desenvolvimento da PETTool optou-se pelo uso da *framework UI Automation* que, apesar das suas limitações, foi considerada a melhor opção.

Na implementação deste novo padrão verificou-se que o uso dessa *framework* não era suficiente e foi necessário recorrer ao uso de uma outra ferramenta: o *Selenium*.

Nesta secção serão apresentadas as duas ferramentas, assim como as suas vantagens e limitações e as razões para o uso das duas.

4.1.1 *UI Automation*

UI Automation é uma *framework* desenvolvida pela *Microsoft* para interagir com as interfaces gráficas. Com esta *framework* é possível aceder à maioria dos elementos presentes na interface. É compatível com todos os sistemas operativos que suportem *Windows Presentation Foundation* (WPF).

A sua estrutura é representada por uma árvore em que cada elemento da interface é um objeto do tipo *AutomationElement*. A raiz da árvore é o ambiente de trabalho (*desktop*) e os seus filhos são todas as janelas existentes no desktop nesse momento. Para cada janela, os seus elementos serão os seus filhos e assim sucessivamente.

O *AutomationElement* contém a informação relativa ao elemento em causa, nomeadamente as propriedades do objeto. Uma das mais importantes é o *ControlType* que se refere ao tipo de elemento - por exemplo, uma *combobox*, ou um botão.

Outra das principais características do *UI Automation* é que cada *AutomationElement* pode ter um *Pattern*[1] associado através do qual é possível introduzir dados nos elementos e/ou obter informações extra. Cada *Pattern* representa uma funcionalidade específica que pode ser usado por vários *AutomationElements* e cada *AutomationElement* pode ter mais do que um *Pattern* associado.

Entre as suas vantagens, podemos destacar o facto de funcionar com qualquer tipo de interface gráfica e as variadas opções de pesquisa de elementos (pesquisa na árvore através dos nós pai/filhos, propriedades e até posição absoluta).

Apesar das suas vantagens, a framework apresenta também algumas desvantagens. Entre elas está a possível perda de referências por parte dos *AutomationElements* quando a página é carregada (em interfaces na web) e a falta de suporte de *Patterns*, isto é, existem ainda muitos elementos na maioria das interfaces que não suportam os *Patterns* que deveriam para se executarem determinadas ações.

Para superar algumas destas desvantagens a PETTool contém uma classe denominada *MyAutomationElement*. Esta classe contém algumas funções que permitem contornar a falta de *Patterns* associados aos elementos e a perda da referência por parte dos *AutomationElements*.

No que diz respeito aos *Patterns*, foram criadas funções que permitem realizar invocações, introduzir texto em caixas de texto e selecionar elementos de listas, tudo isto usando outros tipos de *Patterns* ou, em último caso, simulando ações do teclado.

No segundo caso, foram criadas algumas funções que permitem a obtenção do caminho de um determinado elemento e a obtenção de um *AutomationElement* a partir de um caminho, para possibilitar o acesso aos *AutomationElements*.

Apesar de todos os esforços realizados para ultrapassar as dificuldades, a utilização da *framework UI Automation* não é bem sucedida num grande número de aplicações.

Foi então necessário procurar uma alternativa viável para testar convenientemente as interfaces gráficas.

[1] É importante referir que o *Pattern* presente no *UI Automation* não se encontra relacionado com os padrões referidos anteriormente: padrões de interação e padrões de teste.

4.1.2 *Selenium*

A escolha recaiu sobre o *Selenium*, um conjunto de ferramentas que permite automatizar os *browsers*. Um dos principais objetivos dessa automatização é a realização de testes, mas o seu uso não está restringido a esse fator.

As principais ferramentas constituintes do *Selenium* são o *Selenium IDE* e o *WebDriver*, cada uma com os seus objetivos distintos.

4.1.2.1 *Selenium IDE*

O *Selenium IDE* é uma ferramenta que permite a rápida criação de scripts de testes.

É uma ferramenta de *Capture/Replay* que permite gravar, editar, reproduzir e exportar casos de teste muito facilmente. Está implementado como uma extensão para o Firefox.

As ações gravadas podem ser executadas várias vezes e podem ainda ser alteradas manualmente. A exportação do script de testes pode ser efetuada em várias linguagens como HTML, *Ruby*, C#, entre outras.

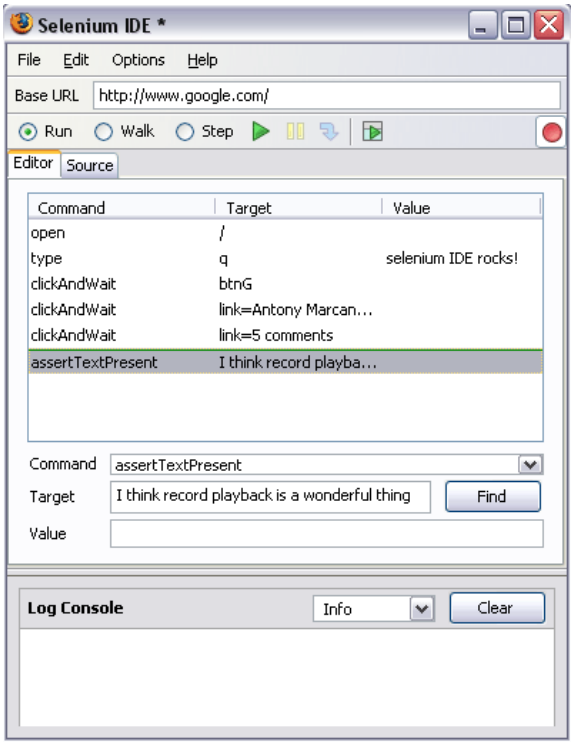

Figura 4.1: *Selenium IDE*

Esta ferramenta é utilizada principalmente por novos utilizadores que pretendem criar testes simples e rápidos. Quando o objetivo é criar testes complexos e robustos, esta não é a ferramenta indicada. Para tal existem outras ferramentas, como o *Selenium 2*.

4.1.2.2 *Selenium 2*

Selenium 2 ou *Selenium WebDriver* é uma ferramenta utilizada para gerar testes automáticos a aplicações *Web* e verificar se os resultados estão de acordo com o esperado. Esta ferramenta surgiu da junção de duas ferramentas existentes: *Selenium 1* e *WebDriver*. Esta junção permitiu ultrapassar as limitações das duas, originando uma ferramenta mais poderosa. Enquanto a primeira estava limitada pelo uso de *JavaScript*, a segunda não era suportada por um grande número de navegadores.

O *Selenium 2* passa a ser uma ferramenta suportada por vários navegadores que contacta diretamente com o navegador, sem o uso de *Javascript*.

É uma ferramenta que pode ser utilizada em várias linguagens e é suportada pelo *Firefox*, *Google Chrome*, *Internet Explorer* e *Opera*. É ainda possível utilizar o *Selenium 2* em ambiente mobile, mais especificamente em *Android* e *IPhone*.

A ferramenta usa uma interface, que é denominada de *Driver*, onde serão executados todos os passos selecionados pelo programador. Existem diferentes *Drivers* para cada um dos navegadores suportados, por exemplo o *FirefoxDriver()* para o *Firefox* e o *ChromeDriver()* para o *Google Chrome*.

Através da API do *WebDriver* é possível executar um variado número de ações relativas ao *Driver*. É possível abrir o *Driver* no URL indicado, procurar elementos presentes no *Driver*, obter o HTML respetivo, entre outras operações.

O *Driver* é composto por vários elementos que são denominados *WebElements*. Estes elementos correspondem aos elementos presentes no código HTML da página. A pesquisa dos elementos pode ser efetuada através de alguns parâmetros: nome, id, tag name, *XPath*, entre outros.

Um *WebElement* contém um conjunto de ações pré-definidas que podem ser executadas sobre esse elemento, tal como um clique ou a introdução de texto (através do método *SendKeys*). Na figura 4.2 está um exemplo de utilização do *Selenium 2*. Demonstra a criação de um *WebDriver* no navegador *Firefox*. É aberta a página da Google da qual é selecionado o primeiro botão e efetua um clique sobre ele.

```
//Exemplo
driver = new FirefoxDriver();

driver.Navigate().GoToUrl("www.google.pt");

IWebElement OkButton = driver.FindElement(By.TagName("button"));

OkButton.Click();
```

Figura 4.2: Exemplo de código utilizando o *Selenium 2*

Uma das vantagens do *Selenium 2* é permitir o teste a aplicações *Web Mobile*, que

é um dos mercados em maior crescimento atualmente. Outra vantagem realmente importante para este projeto é o facto de as ações sobre os elementos não terem restrições, independemente da página a testar. O facto de ser suportada pela maioria dos navegadores é também uma mais-valia.

A principal desvantagem do *Selenium* é, sem dúvida, estar apenas disponível para aplicações *Web*. Outra desvantagem, comparativamente com o *UI Automation*, é o facto de não ser possível localizar os elementos por posição com um clique do rato no elemento. Existem também algumas restrições na utilização de alguns comando em browsers diferentes. Por exemplo, o uso de *XPath* não é suportado no *Internet Explorer*.

Apesar de não suportar a interação com interfaces de aplicações não *Web*, a sua utilização traz mais vantagens que o *UI Automation*. Assim sendo, optou-se por continuar a utilizar a *framework UI Automation* para testar interfaces de aplicações não *Web*, as quais denominamos *Desktop*, e para testar as aplicações *Web* utiliza-se o *Selenium 2*.

Na secção seguinte serão apresentados os pormenores mais relevantes da implementação da Pesquisa na PETTool, evidenciando as diferenças entre as aplicações *Web* e *Desktop*, causadas pela utilização destas duas ferramentas distintas.

4.2 Pesquisa na PETTool

A implementação do padrão de teste Pesquisa na ferramenta tem de seguir as regras já existentes sempre que possível e acrescentar, caso seja necessário, outras funcionalidades.

Esta secção apresenta a estrutura da ferramenta após a inclusão do novo padrão e também o funcionamento da ferramenta relativo ao teste do padrão da pesquisa.

4.2.1 Estrutura da ferramenta

O acrescentar deste novo padrão não causou mudanças significativas na arquitetura da ferramenta. A única alteração foi a adição de um novo elemento, a Pesquisa, que corresponde a um padrão de comportamento. A arquitetura da PETTool encontra-se representada na figura 4.3.

Com esta arquitetura podemos verificar que a ferramenta suporta dois tipos de padrões: estrutura e comportamento. Como referido anteriormente, o padrão Pesquisa é um padrão de comportamento.

Cada padrão gera um conjunto de testes que, no final, é comparado com o comportamento esperado desse mesmo padrão.

4.2.2 Funcionamento da Pesquisa na ferramenta

A pesquisa em aplicações *Desktop* e *Web* apresenta algumas diferenças, em grande parte devido ao uso de duas tecnologias diferentes na sua implementação.

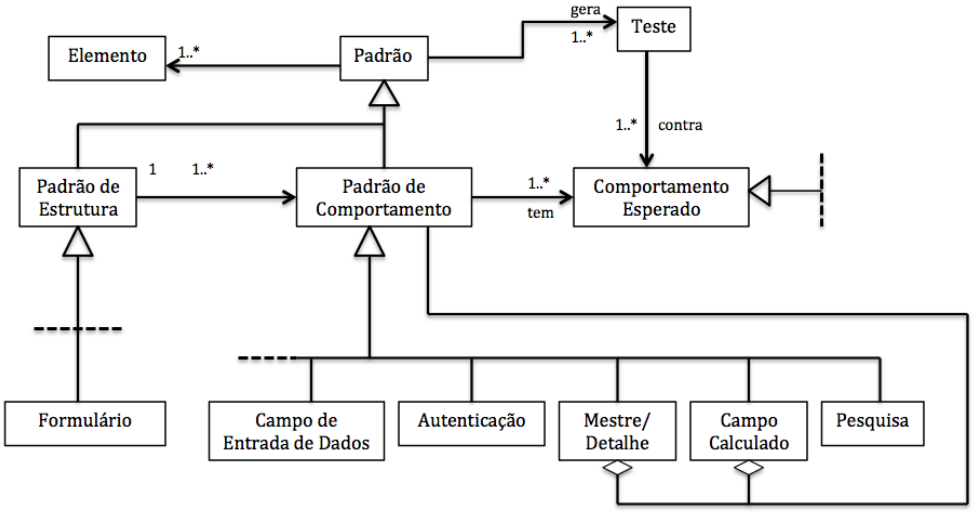

Figura 4.3: Arquitetura da ferramenta PETTool

De seguida é explicada a forma de funcionamento das duas variantes, demonstrando todos os passos necessários para utilizar a ferramenta corretamente.

4.2.2.1 Aplicações *Desktop*

Criar formulário Em qualquer uma das Pesquisas, tal como é efetuado nos outros padrões, o primeiro passo passa por identificar o formulário onde o padrão Pesquisa se encontra. Nas aplicações *Desktop* é necessário selecionar a janela de pesquisa da aplicação e o botão de submissão deverá ser o botão a clicar para iniciar a pesquisa, como apresentado na figura 4.4.

Criar padrão pesquisa De seguida é necessário criar um padrão do tipo Pesquisa, selecionando da lista de padrões existentes. Por defeito, ao selecionar o padrão Pesquisa, o tipo de padrão que aparece é o de aplicações *Desktop*, sendo que apenas é necessário indicar um nome para o padrão.

Introduzir valores de entrada O próximo passo passa por clicar no botão *Get Fields* que irá obter todos os campos de entrada de dados presentes no formulário indicado anteriormente e que irão ser apresentados numa *DataGridView*, no painel *Input Values* - figura 4.5. Cada um dos campos é representado através do seu nome, id e tipo de controlo. Os campos de entrada de dados selecionados são os *AutomationElement* que se encontram no formulário selecionado (ou seja, são os filhos do *AutomationElement* que representa o formulário) e cujo *ControlType* é dos seguintes tipos: *Edit, Combobox, Checkbox, RadioButton e List*. Se o campo for do tipo *Edit*, é necessário escrever o texto que se pretende

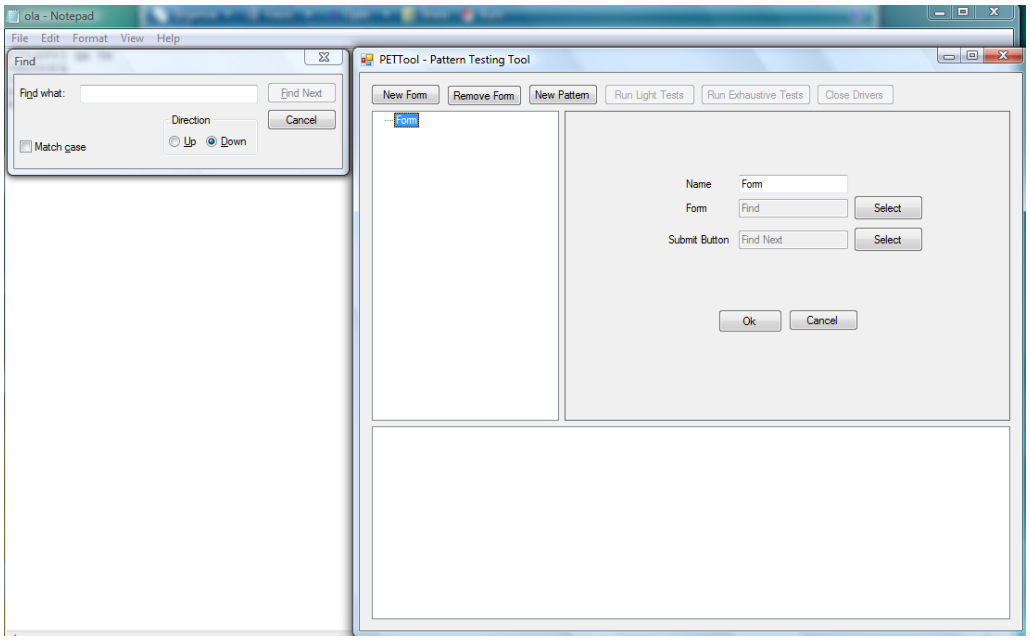

Figura 4.4: Criação do padrão *Form* na PETTool

introduzir. No caso de ser *Combobox* ou *List* deve-se indicar o nome exato do elemento que se pretende selecionar, e nos restantes casos deve-se deixar o valor em branco se não se pretender selecionar o elemento ou escrever algo no caso contrário.

Nessa matriz, para cada campo existem 3 colunas vazias onde o utilizador irá introduzir os inputs desejados para cada um dos casos de teste já referenciados no capítulo 3: Retorna 0, Retorna 1 e Retorna N. Se desejar eliminar todos os inputs introduzidos num determinado caso de teste (uma coluna), basta selecionar uma célula qualquer dessa coluna e pressionar o botão *Delete Test Case*.

Configuração dos testes Seguidamente, no painel *Test Configurations*, o utilizador terá que introduzir as verificações a realizar em cada um dos casos de teste. Para isso, deve selecionar o caso de teste na caixa de combinação que se encontra no canto superior direito, introduzir as verificações necessárias para esse caso e no final guardar os testes - botão *Save test configuration*. Deve repetir esta operação para os 3 casos de teste.

Quando desejar eliminar uma verificação, deve selecionar a linha desejada e clicar no botão *Delete*. Se não introduzir nenhuma das verificações, não será realizado nenhuma comparação. As verificações podem ser as verificações de conteúdo, assim como a verificação *Same Page*, sendo que esta se apresenta *False* por defeito.

Neste tipo de aplicações, a verificação *Same Page* refere-se apenas ao elemento onde serão feitas as verificações, isto é, se a verificação estiver com o valor *True*, as verificações de conteúdo serão efetuadas na janela de resultados especificada, senão as verificações serão efetuadas noutra janela. Este tipo de verificação é necessária porque existem casos

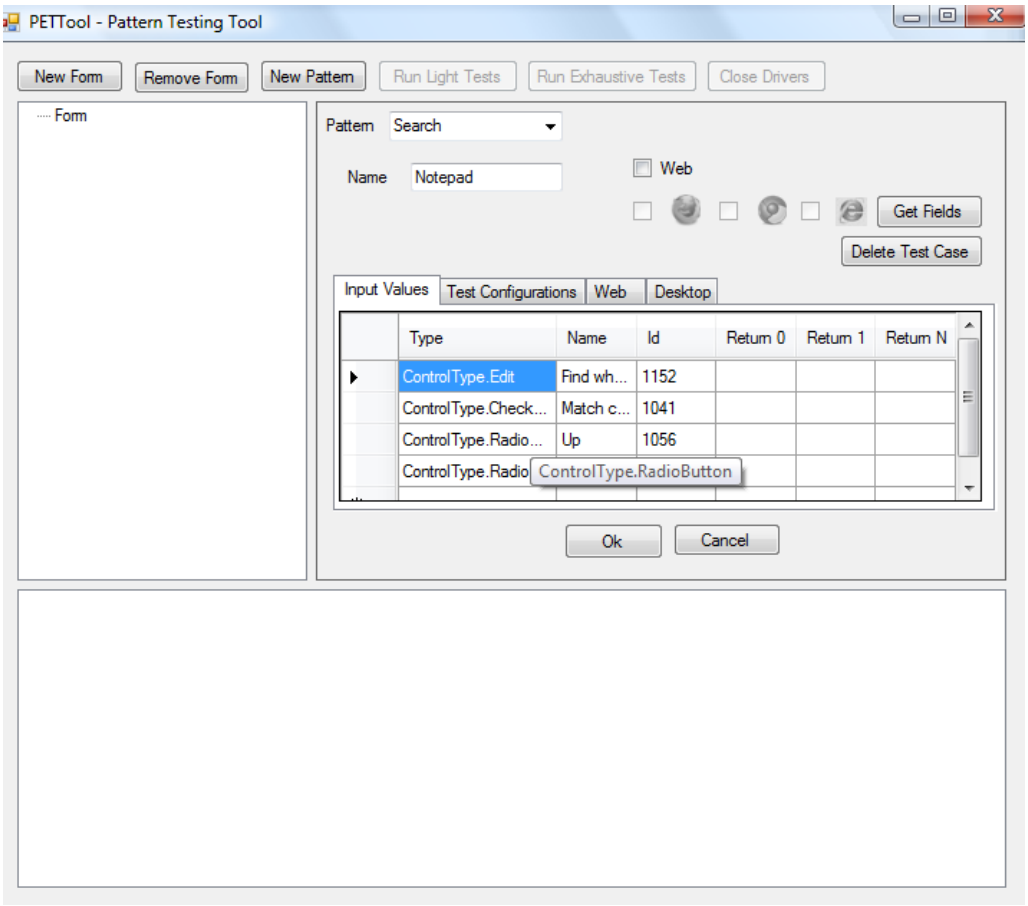

Figura 4.5: Padrão Pesquisa - aplicação *Desktop*, painel *Input Values*

em que são lançadas mensagens de erro em janelas distintas, como é o caso do *Notepad* quando não encontra nenhum resultado.

Desktop O último passo da configuração diz respeito à localização do resultado, ou seja, o elemento onde serão realizadas as verificações especificadas anteriormente. Este passo é necessário pois, normalmente, em aplicações *Desktop*, o formulário de pesquisa é apresentado numa janela diferente de onde se encontra o conteúdo a ser pesquisado. É então necessário indicar qual o elemento da aplicação onde será efetuada a pesquisa. Para isso, no painel *Desktop*, existe um botão *Results*, que permite identificar o elemento do conteúdo. Basta clicar no botão e seguidamente clicar no elemento pretendido, tal como acontece na identificação do formulário, no primeiro passo.

Assim que a configuração seja finalizada, clica-se no botão *Ok*, no fundo da página e a aplicação irá guardar todas as informações.

Execução dos testes Para executar os testes na aplicação, seleciona-se o formulário onde se encontra definido o padrão, apresentado na árvore presente no lado esquerdo do ecrã,

e clica-se no botão *Run Light Tests*. A partir daqui todo o processo é efetuado automaticamente pela aplicação, sendo que esta introduz os valores de entrada especificados pelo utilizador, efetua a pesquisa e verifica os resultados obtidos. No final gera um relatório que para cada caso de teste apresenta o resultado: indica se o teste passou ou falhou - através de uma mensagem e de símbolos de sucesso e falha - e, no caso de não haver verificações efetuadas ou de não ter sido possível efetuar o teste apresenta um símbolo de aviso e a respetiva mensagem.

Para efetuar as verificações definidas pelo utilizador, a aplicação procura o texto no elemento selecionado - elemento dos resultados caso a caixa de seleção *Same Page* esteja ativa e outras janelas no caso contrário - e, no segundo caso, fecha essa janela. A procura do texto é efetuada através do uso de várias propriedades da ferramenta utilizada. Primeiro é pesquisado nos elementos com o *ControlType Text* ou *Edit*, senão encontrar pesquisa ainda através do *Pattern TextPattern*, caso este se encontre disponível no elemento a pesquisar.

4.2.2.2 Aplicações *Web*

Existem vários fatores comuns na configuração dos padrões de aplicações *Desktop* e *Web*, mas a utilização do *Selenium* em substituição do *UI Automation* no segundo caso, causa algumas diferenças que serão explicadas de seguida.

Criar formulário Tal como no caso anterior, é necessário criar um formulário, mas este não tem necessariamente de estar relacionado com a aplicação que vamos testar, pois nenhum dos seus elementos vai ser utilizado.

Criar padrão pesquisa Quando selecionamos o padrão *Search*, é necessário verificar a caixa de seleção *Web*, de modo a permitir o aparecimento de alguns campos que estão relacionados apenas com a pesquisa em aplicações *Web*.

De seguida, é necessário indicar o URL da página onde se pretende efetuar a pesquisa, assim como o navegador pretendido. Neste momento, apenas é possível realizar os testes em *Firefox* e *Google Chrome* visto o *Internet Explorar* não suportar o uso de *XPath* na localização de elementos.

Introduzir valores de entrada De seguida, é efetuado o clique no botão *Get Fields* que irá obter os campos de entrada de dados presentes na página indicada, tal como no passo anterior. A diferença, neste caso, é o facto de o utilizador não precisar ter a aplicação aberta quando inicia a configuração. Ao clicar no *Get Fields* é a aplicação que lança um *WebDriver* no navegador e URL especificados anteriormente. Os campos de entrada

selecionados são: os elementos com a *tag input* - excluindo alguns tipos não relevantes (*hidden, image, reset* e *file*) - e com a *tag select*.

O próximo passo é o de introduzir os valores para cada caso de teste. No caso do elemento ser um *select*, o valor a introduzir deverá ser o valor exato do elemento que se pretende selecionar, visto que este não está a ser verificado. No caso de elementos *input* do tipo *text* introduz-se o texto pretendido, mas se forem do tipo *checkbox* ou *radio* basta introduzir um valor qualquer para selecionar o elemento. Se o valor estiver vazio o elemento não é selecionado, caso contrário, seja qual for o valor introduzido, o elemento é selecionado.

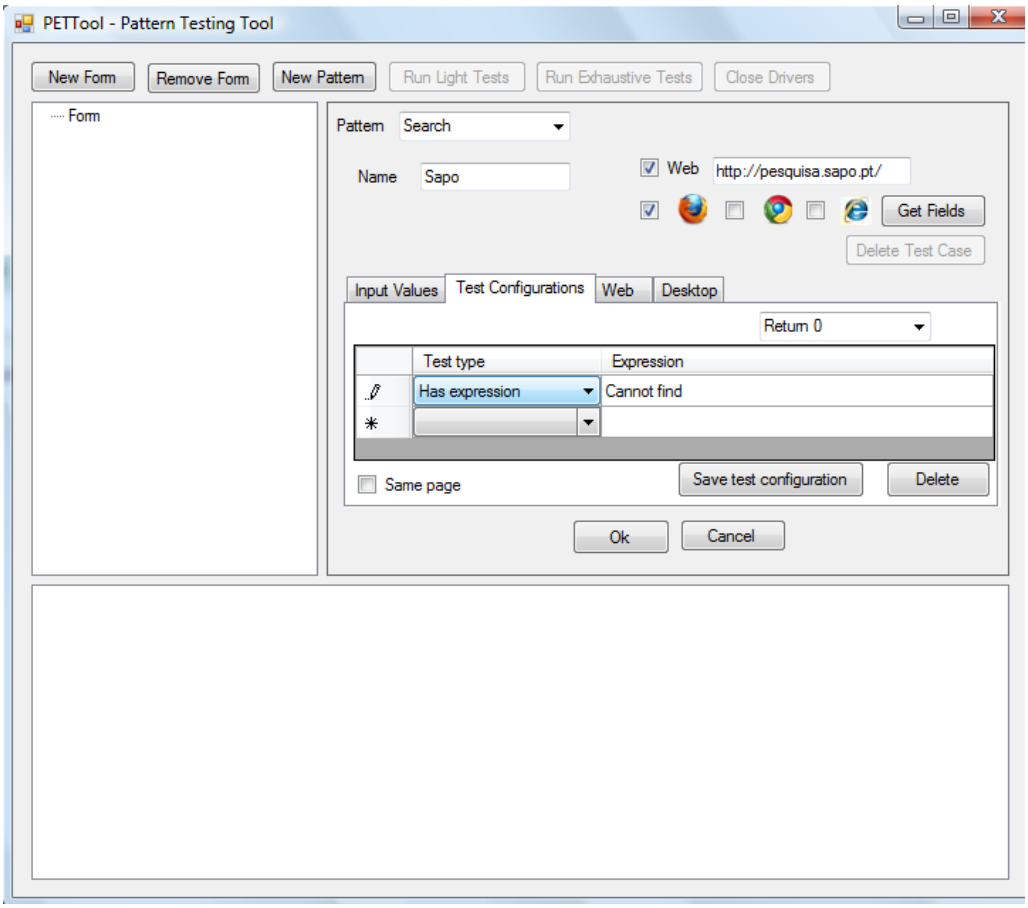

Figura 4.6: Padrão Pesquisa - aplicação *Web*, painel *Test Configurations*

Configuração dos testes No painel *Test Configurations*, o procedimento é o mesmo descrito anteriormente, sendo que a única diferença diz respeito à verificação *Same Page*. Podemos ver um exemplo disso na figura 4.6

Neste caso, esta verificação é relativa à página indicada inicialmente pelo utilizador e a página apresentada após a pesquisa. Isto é, a verificação efetuada é a comparação entre o URL indicado pelo utilizador e o URL da página após a pesquisa. Se a caixa de seleção

estiver ativa, a verificação retorna sucesso se os dois URLs forem iguais e retorna falha se estes forem diferentes. No caso de a caixa de seleção não estar selecionada, acontece o contrário.

Web Por último, no painel *Web*, é possível selecionar o botão de submissão da pesquisa. Inicialmente, encontra-se definido que o *Enter* será a forma de iniciar a pesquisa, sendo que este é efetuado sobre o último elemento a ser preenchido com os valores de entrada. Esta abordagem não funciona em todos os casos tornando-se necessário indicar qual o elemento que deve ser clicado. Para isso basta Clicar no botão *Change* que irá gerar uma lista de todos os elementos que possam realizar essa operação e selecionar o pretendido. Os elementos que aparecem são: *tag button, tag img* com atributo *onclick* definido, *tag link* e *tag input* com tipo *submit* ou *image*.

Execução dos testes Ao clicar no botão *Ok*, todas as configurações são guardadas e o driver é fechado. Para correr os testes usa-se a estratégia que foi especificada anteriormente.

Aqui, os testes decorrem da seguinte forma: A ferramenta abre um driver no URL e *browser* especificados, introduz os valores, efetua a pesquisa e verifica os resultados. Para realizar o caso de teste seguinte, o driver navega até à página inicial - o URL introduzido na configuração - e efetua as mesmas operações. No final, fecha o *driver* e gera o relatório.

Visto não ser possível selecionar elementos por localização com o *Selenium 2*, a verificação do texto é efetuada em toda a página, não estando limitada à área de pesquisa. Assim, para verificar a presença ou não de determinado texto, é efetuado o levantamento de todos os elementos presentes na página e, em cada um desses elementos, é verificada a presença ou não - dependendo da verificação definida - do texto. A pesquisa dos elementos é efetuada a partir de uma expressão de *XPath*, razão pela qual o teste deste padrão ainda não é suportado pelo navegador *Internet Explorer*.

4.3 Conclusões

Apesar da PETTool utilizar a *framework UI Automation* para realizar automaticamente as interações com a interface gráfica, esta apresenta algumas limitações que impossibilitam a realização dos testes para o padrão Pesquisa.

Foi então necessário procurar uma outra solução para resolver este problema. A melhor solução encontrada até ao momento foi a utilização do *Selenium 2*. Ainda assim o *Selenium 2* não resolve todos os problemas, sendo que esta ferramenta apenas pode ser utilizada em aplicações *Web*.

A solução encontrada para conseguir realizar os testes necessários foi o uso das duas ferramentas em conjunto: *UI Automation* para aplicações *Desktop* e *Selenium 2* para aplicações *Web*.

O funcionamento da ferramenta é parecido nos dois tipos de aplicações, mas existem algumas diferenças, principalmente devido ao facto referido anteriormente.

No capítulo seguinte são demonstrados alguns exemplos de utilização da ferramenta para testar o padrão Pesquisa.

Capítulo 5

Casos de estudo

Este capítulo tem como objetivo principal apresentar alguns casos de estudo que foram utilizados na realização dos testes da nova funcionalidade da ferramenta.

Pretende-se mostrar, a partir de aplicações reais, que a ferramenta é capaz de testar a pesquisa eficazmente. Para isso, foram utilizadas aplicações *Web* e *Desktop* com diferentes características e configurações de testes.

As aplicações selecionadas são a pesquisa no *Windows Notepad* - aplicação *Desktop* - e a pesquisa no sítio do Sapo e no sítio ERA Portugal - aplicações *Web*.

De seguida serão descritos os passos utilizados na configuração de cada um dos casos, assim como, os resultados obtidos.

5.1 *Windows Notepad*

O *Notepad* é um editor de texto bastante simples, que está presente em todos os computadores que utilizem o *Windows* como sistema operativo.

A sua principal utilidade é a criação e edição de ficheiros de texto, mas é comum ser utilizado por programadores, que o consideram uma boa ferramenta para desenvolver código em determinadas linguagens de programação.

Tal como em qualquer outro editor de texto, a pesquisa é uma ferramenta bastante importante. No *Notepad*, para realizar uma pesquisa sobre o texto, o utilizador tem duas opções: ou utiliza o menu *Edit -> Find* ou o atalho *Ctrl + F*. Esta ação lança uma nova janela onde poderá ser efetuada a pesquisa, como pode ser observado na figura 5.1.

5.1.1 Primeiro cenário

Este exemplo pretende demonstrar um conjunto de configurações que retornam casos de sucesso, em que o comportamento esperado é igual ao comportamento obtido.

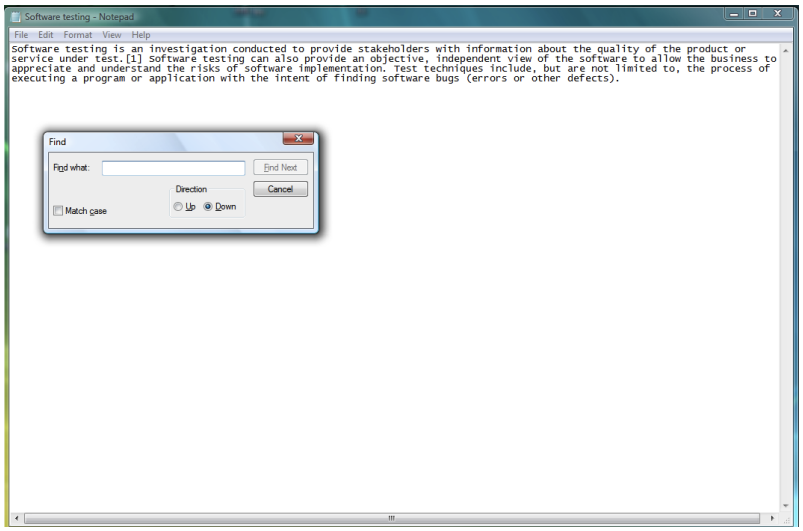

Figura 5.1: Pesquisa no *Windows Notepad*

Depois de ter aberto a janela de pesquisa, é necessário iniciar a PETTool e criar o formulário associado à Pesquisa do *Notepad*, como pode ser verificado na imagem 4.4, presente no capítulo anterior.

Na imagem 5.2, podemos verificar a introdução de valores de entrada para os 3 casos de teste a realizar.

No primeiro caso de teste - Return 0 - foi introduzido o valor "erros"na caixa de texto. No segundo caso - Return 1 - foi introduzido o valor "Software"no mesmo local e o botão de opções com o valor "Up"foi selecionado. O valor "or"foi introduzido na caixa de texto no último caso de teste.

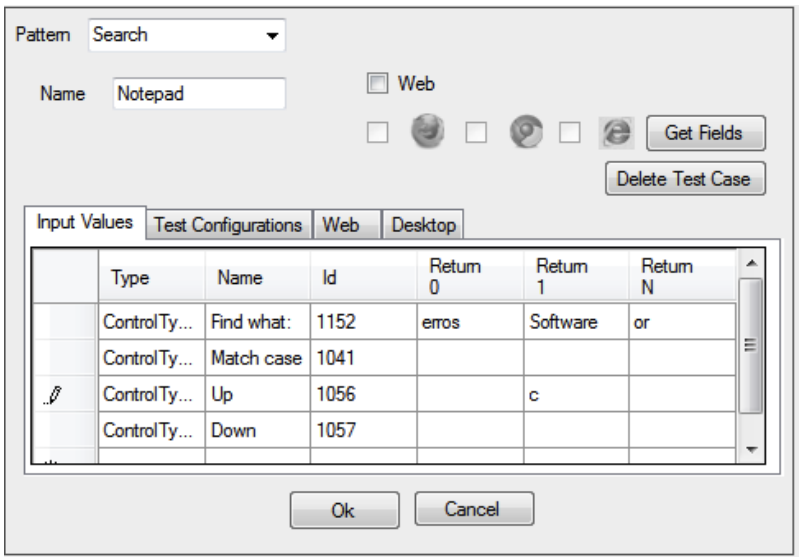

Figura 5.2: Introdução de valores de entrada no padrão Pesquisa do *Notepad*

A tabela 5.1 apresenta as verificações a efetuar a este padrão e os exemplos destas mesmas verificações estão demonstrados nas imagens 5.3, 5.4 e 5.5.

Retorna 0	**Retorna 1**	**Retorna N**
Existe a expressão (*Cannot find* "erros")	Existe uma das palavras (software)	Não existe a expressão (Testes de software)
Not Same Page	*Same Page*	*Same Page*

Tabela 5.1: Conjunto de testes para o padrão Pesquisa da aplicação *Notepad*

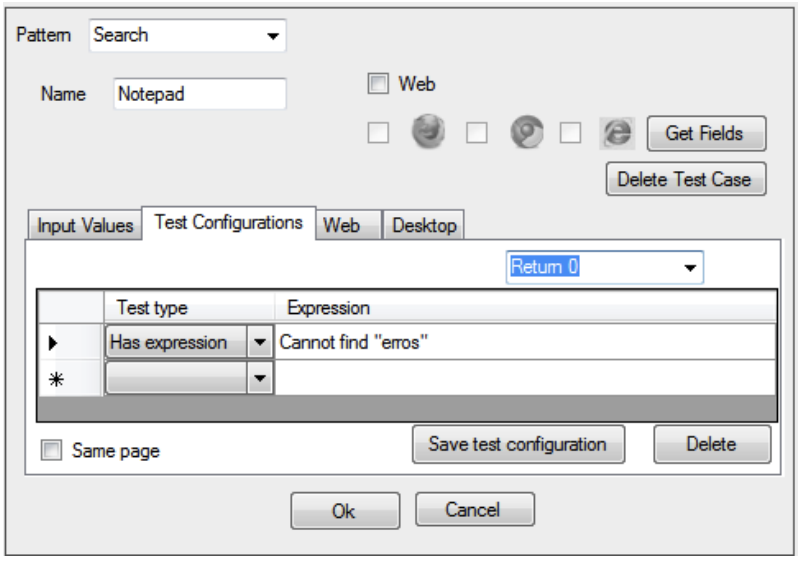

Figura 5.3: Configuração do primeiro caso de teste do *Notepad*

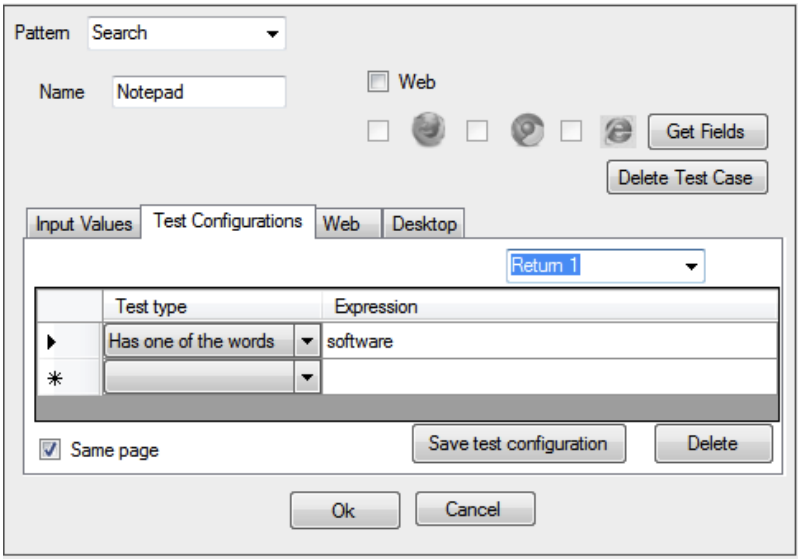

Figura 5.4: Configuração do segundo caso de teste do *Notepad*

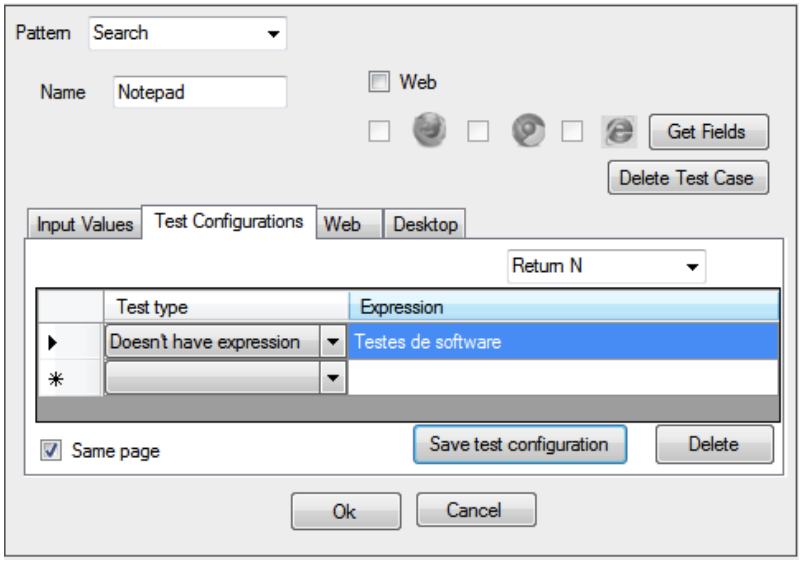

Figura 5.5: Configuração do terceiro caso de teste do *Notepad*

O passo seguinte é a seleção do elemento dos resultados que pode ser consultada na imagem 5.6.

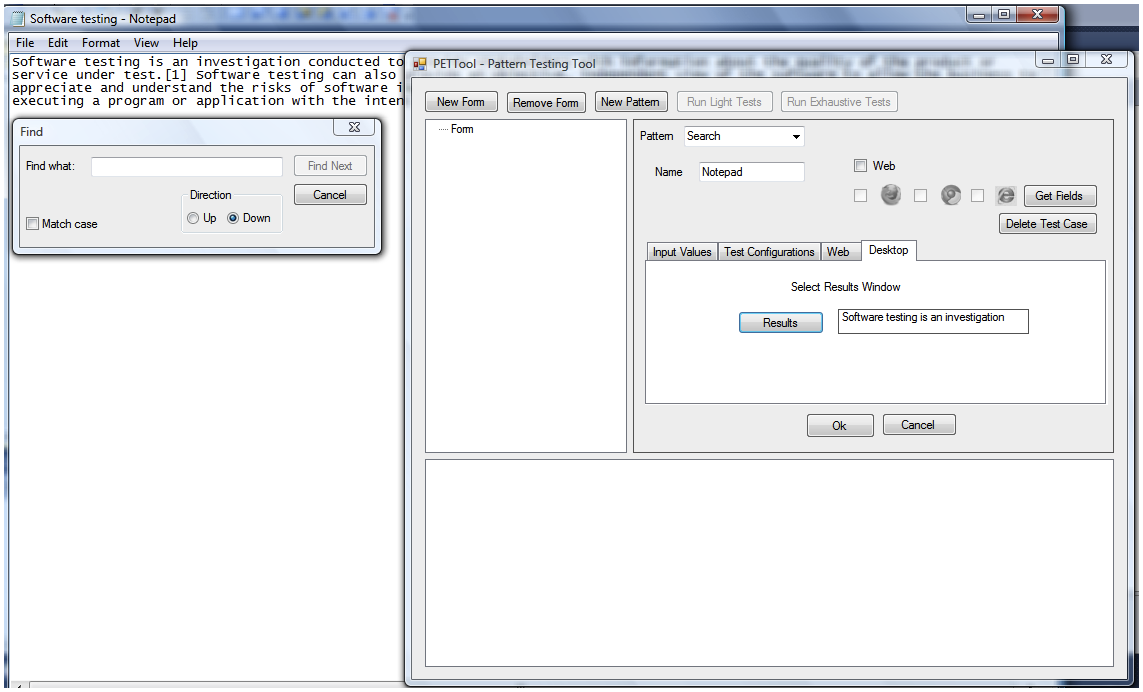

Figura 5.6: Seleção do elemento dos resultados do *Notepad*

Todos os testes definidos anteriormente devem ser bem sucedidos. O relatório gerado 5.7 confirma isso mesmo.

O relatório apresenta os resultados de cada caso de teste separadamente, sendo que, em cada caso, indica o nome do padrão e respetivo caso de teste no formato: *Testing Pattern* <Nome padrão> -> *Test Case*: <Nome do caso de teste>.

Para cada caso de teste, são apresentadas as mensagens relativas às verificações efetuadas, sendo que neste caso todas as verificações foram bem sucedidas, o que significa que a aplicação se comportou de acordo com o esperado inicialmente.

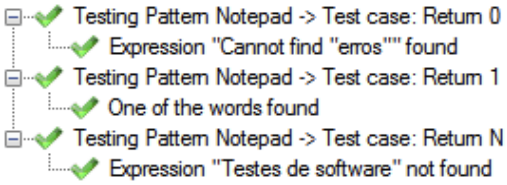

Figura 5.7: Relatório gerado pela PETTool, após a geração e execução dos testes no *Notepad*

5.1.2 Segundo cenário

Para garantir que a aplicação também deteta falhas, foi criado um outro conjunto de casos de teste, com diferentes configurações para o mesmo documento testado no caso anterior. As imagens de introdução de dados não são aqui apresentadas, visto estas serem bastante semelhantes às apresentadas anteriormente.

Os dados de entrada desta vez foram: "testes", "quality"e "or"na caixa de texto de cada um dos casos de teste. As configurações introduzidas podem ser consultadas na tabela 5.2.

Retorna 0	**Retorna 1**	**Retorna N**
Existe a expressão (*Cannot find* "testes")	Não existe a expressão (*an investigation*)	
Not Same Page	*Same Page*	

Tabela 5.2: Conjunto de testes para o padrão Pesquisa da aplicação *Notepad* - Segundo cenário

Neste caso, e como o relatório da figura 5.8 comprova, o primeiro caso de teste é bem sucedido, pois a mensagem é encontrada numa janela diferente da selecionada. O caso 2 falha, pois a expressão "an investigation"está presente no documento. O caso de teste 3, devido a não lhe ter sido atribuída nenhuma configuração, emite um aviso alertando para o facto do teste poder estar incompleto. Este aviso ocorre pois, ao não efetuar testes para todas as classes de equivalência, é provável que existam áreas da aplicação que não foram testadas.

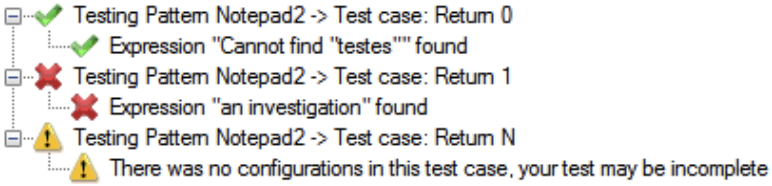

Figura 5.8: Relatório gerado pela PETTool, após a geração e execução dos testes no *Notepad* - Segundo cenário

5.2 Pesquisa no Sapo

O Sapo é um sítio *Web* português que apresenta um variado conjunto de serviços. O número de serviços oferecido é bastante elevado e variam desde as notícias, informações do tempo, disponibilização de conteúdo até à criação de blogs, de emails, uma sala de conversação (Sapo Messenger), entre muitos outros. O endereço da sua página principal é http://www.sapo.pt/.

A pesquisa é um dos serviços que o Sapo oferece, no endereço http://pesquisa.sapo.pt/ (ver figura 5.9), e é nela que nos vamos focar. Esta página apresenta uma caixa de texto onde se introduz o texto que se pretende pesquisar, o botão de execução da pesquisa, e outros elementos que permitem definir a pesquisa.

A escolha do tipo de resultados que pretendemos encontrar pode variar entre conteúdos da *Web*, imagens, vídeos e outros mais. Neste exemplo iremos apenas pesquisar conteúdos da *Web*.

Figura 5.9: Página de pesquisa do Sapo

A introdução dos valores de entrada - 3eresw, 12qqqqqqqqqq, software testing -, assim como as verificações e informações relativas ao padrão de aplicações *Web* podem ser consultadas na imagem 5.10. Os primeiros dois valores são algo incomuns mas são

alguns dos poucos valores que retornam 0 e 1 resultados. Normalmente, neste tipo de pesquisa, o número de resultados encontrados é sempre elevado.

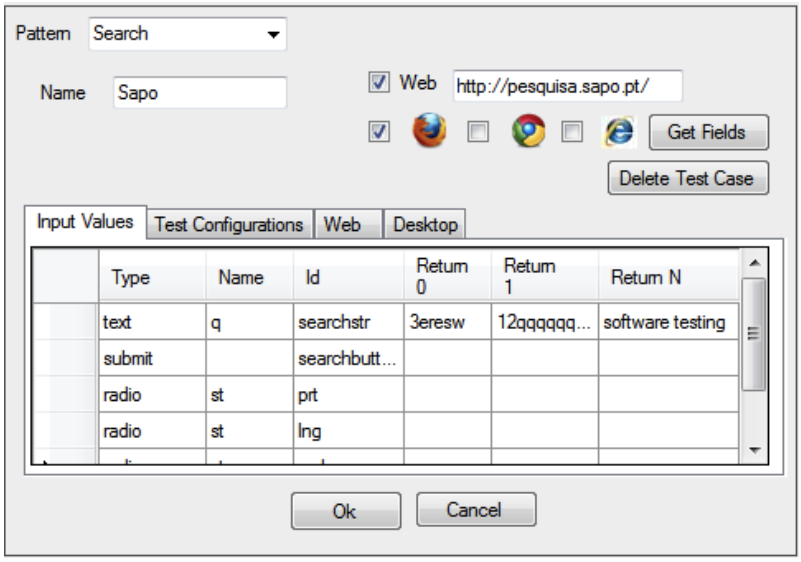

Figura 5.10: Valores de entrada na pesquisa do Sapo

Retorna 0	**Retorna 1**	**Retorna N**
Existe a expressão (Não encontramos resultados para a pesquisa por 3 eresw)	Existe a expressão (Resultados 1-1 de 1)	Existe a expressão (en.wikipedia.org/wiki/Software_testing.)
		Não existe a expressão (Não encontramos resultados para a pesquisa)
Not Same Page	*Not Same Page*	*Not Same Page*

Tabela 5.3: Conjunto de testes para o padrão Pesquisa do Sapo

As configurações dos casos de teste estão indicadas na tabela 5.3. Como o método de preenchimento é semelhante em qualquer tipo de aplicação não existe a necessidade de incluir aqui as imagens da ferramenta.

Visto que o clique na tecla *Enter* funciona na pesquisa no Sapo, não houve a necessidade de alterar o botão de submissão.

O relatório que é gerado (figura 5.11) demonstra que todos os testes executados geram os comportamentos esperados.

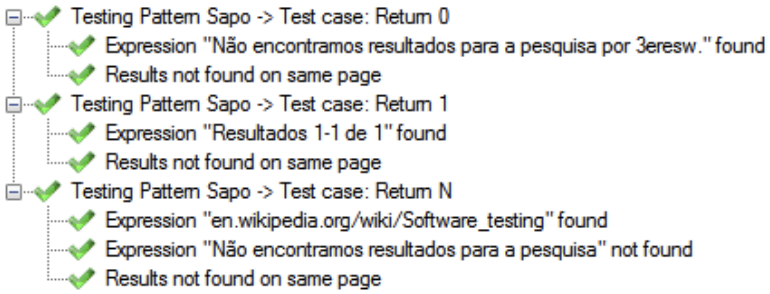

Figura 5.11: Relatório gerado na pesquisa do Sapo

5.3 ERA Portugal

A ERA Portugal é uma imobiliária que dispõe de um sítio na *Web* onde é possível pesquisar imóveis. Esta pesquisa é efetuada à medida de cada utilizador, visto que existem vários critérios de filtragem relativos ao tipo de imóveis e à sua localização.

Os valores de entrada usados neste caso são apresentados na tabela 5.4.

	Retorna 0	**Retorna 1**	**Retorna N**
Tipo de imóvel	Garagem	Quinta	Apartamento
Distrito	Ilha do Pico	Porto	Lisboa
Localidade		Maia	

Tabela 5.4: Valores de entrada no padrão de Pesquisa da ERA

A utilização desta pesquisa como caso de estudo serve para demonstrar algumas diferenças em relação ao caso do Sapo.

Para além de existirem mais campos de entrada de dados nesta página, a utilização do clique na tecla *Enter* para executar a pesquisa não está disponível. Neste caso é necessário selecionar o botão para efetuar essa ação. Nas figuras 5.12 e 5.12 é demonstrado esse processo.

O botão selecionado é um elemento do código HTML da página, do tipo *input*, sendo "DASSPA"o seu identificador.

Retorna 0	**Retorna 1**	**Retorna N**
Existe a expressão (Lamentamos, mas não foram encontrados imóveis para a sua pesquisa)	Existe a expressão (2.200.000 €)	Existe uma das palavras (Porto)
		Existe a expressão (de 1015)
Not Same Page	*Same Page*	*Not Same Page*

Tabela 5.5: Conjunto de testes para o padrão Pesquisa da ERA

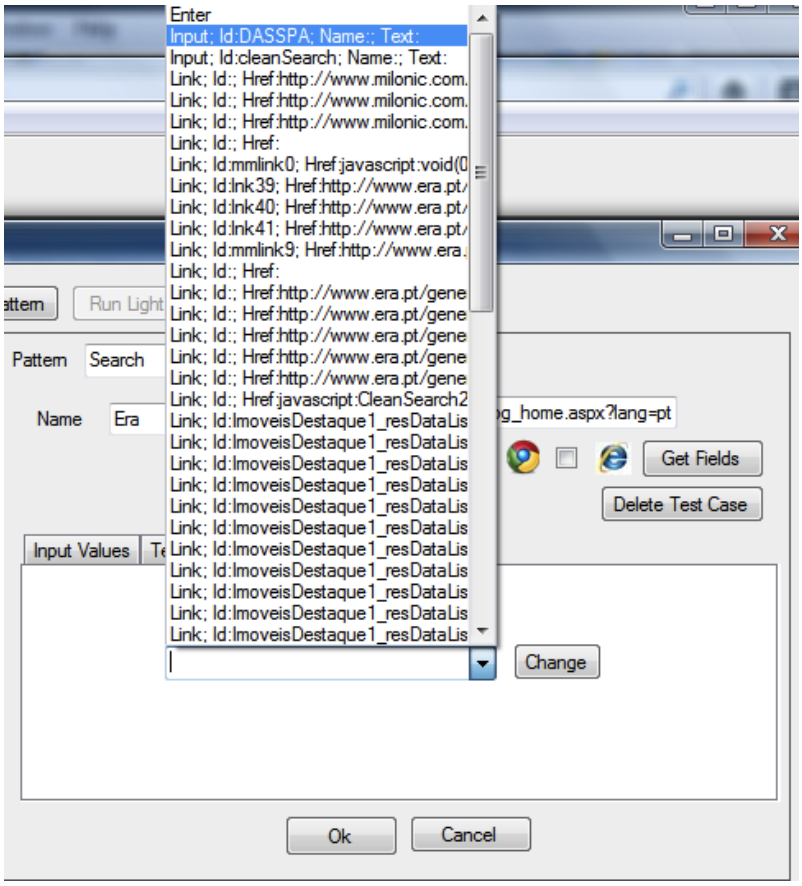

Figura 5.12: Escolha de botão de execução da Pesquisa no ERA

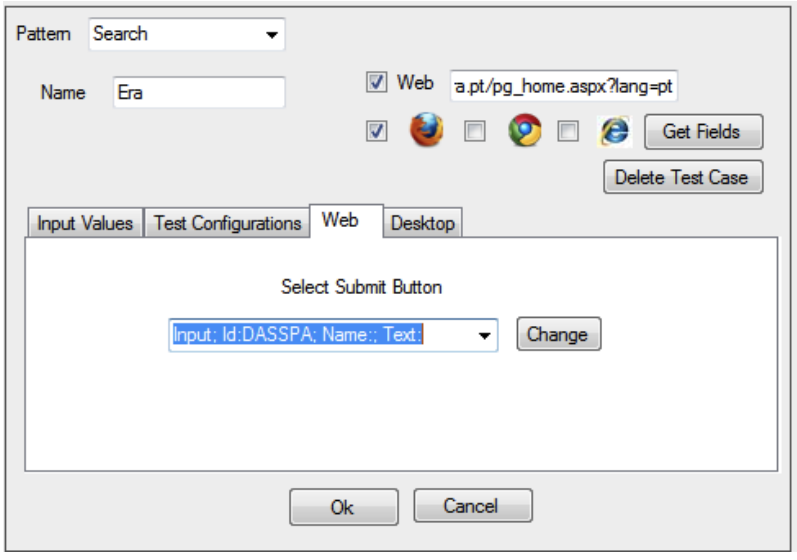

Figura 5.13: Botão de execução selecionado no ERA

Os resultados apresentados no relatório5.14 estão de acordo com as configurações definidas na tabela 5.5.

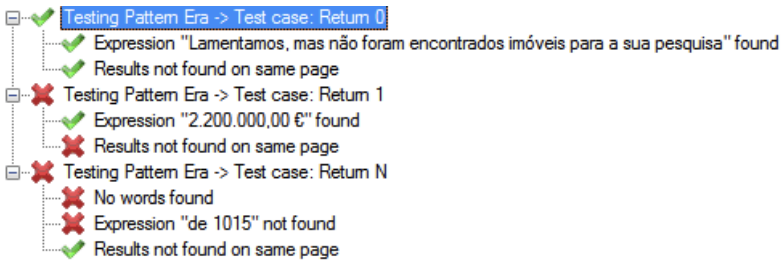

Figura 5.14: Relatório gerado pelo padrão pesquisa da página ERA

O primeiro caso de teste não encontra nenhum imóvel e retorna uma mensagem de erro numa página diferente. No segundo caso de teste, a aplicação encontra o valor do imóvel que resultou da pesquisa mas esse resultado encontra-se numa página diferente da inicial, o que torna o teste falhado. No último caso, tanto o número de resultados como o Distrito especificados não são encontrados na página de resultados.

5.4 Conclusões

Uma das soluções para comprovar a eficácia de uma determinada a aplicação é a utilização de casos de estudo. A ideia é utilizar alguns exemplos e analisar os resultado obtidos, para determinar se esta se encontra de acordo com os objetivos estipulados.

Neste capítulo recorreu-se ao uso de algumas aplicações com as mais variadas características de modo a perceber se os testes do padrão Pesquisa na PETTool funcionam corretamente.

Apesar de ainda existirem algumas limitações, pelo facto da *framework UI Automation* não reconhecer todos os objetos de interação de todas as aplicações de *software*, verificou-se que a Pesquisa do *Windows Notepad* pode ser testada corretamente. A ferramenta consegue procurar conteúdos dentro ou fora dos documentos em questão e ainda gera um aviso no caso de não terem sido especificadas verificações para algum caso de teste.

Nas aplicações *Web* não existem grandes limitações. Provou-se que é possível testar padrões de Pesquisa dos mais simples aos mais complexos, com mais ou menos valores de entrada e verificações de comportamentos. A PETTool verifica corretamente os comportamentos esperados, quer no que diz respeito aos conteúdos quer na localização dos resultados na página inicial ou numa diferente.

Normalmente, o tempo de geração e execução de testes não é muito elevado mas este depende bastante do número de verificações a efetuar e, no caso de aplicações *Web*, depende também do tempo de carregamento das páginas. Isto é, quanto mais lento for o sítio *Web* a testar, mais lento será o processo de teste.

Capítulo 6

Conclusões e Trabalho Futuro

Ao longo dos tempos, a utilização de interfaces gráficas nas aplicações tem vindo a crescer bastante. Hoje em dia são raras as aplicações que não possuem GUI, visto estas facilitarem bastante a interação entre o utilizador e a aplicação.

A área de testes de software também tem sofrido um aumento de interesse por parte dos profissionais da área. Esta é uma área bastante importante que ainda se encontra em crescimento, com muitas vertentes a explorar. O teste de interfaces gráficas é, sem dúvida, uma área onde ainda existem muitas dificuldades e onde é preciso investir devido à sua importância.

As técnicas e ferramentas existentes de suporte para este tipo de teste apresentam ainda bastantes limitações. Entre elas podemos destacar o baixo de nível de automação, a dificuldade na sua utilização e os custos elevados que resultam da sua utilização e manutenção. O desenvolvimento de uma ferramenta que permita testar interfaces gráficas automaticamente, de uma forma rápida e eficaz, é por isso essencial.

Após a análise de várias das implementações já existentes, concluiu-se que a mais vantajosa era a técnica baseada em modelos. Mas esta técnica apresenta algumas dificuldades, com principal destaque na construção do modelo.

A solução que se pretendeu implementar tem como principal objetivo utilizar os padrões de interação para auxiliar a construção do modelo. Isto é, pretende-se desenvolver uma ferramenta que gere e execute casos de teste para interfaces gráficas baseados em padrões de interação. A generalização dos padrões permite desenvolver soluções genéricas de teste para cada um e permite, assim, testar separadamente os componentes recorrentes das GUIs.

Este projeto já foi iniciado, existindo neste momento um protótipo de uma ferramenta, a PETTool, que já possibilita o teste de alguns padrões de interação: Autenticação, Mestre/Detalhe, Campo de Entrada de Dados, Formulário e Campo Calculado.

6.1 Satisfação dos Objectivos

O objetivo deste projeto passa por dar continuidade ao trabalho já desenvolvido, acrescentando novos padrões de teste à ferramenta.

Visto a pesquisa ser uma componente que marca presença num número significativo de aplicações, optou-se por explorar essa funcionalidade.

Os objetivos definidos inicialmente eram a identificação dos padrões de teste associados à pesquisa, a definição de estratégias de teste para cada um e a sua implementação na ferramenta.

Após uma análise dos padrões de pesquisa foi definido um padrão de teste, ao qual se deu o nome de Pesquisa, que é representado por todos os campos de entrada de dados utilizados na definição da pesquisa e o botão que permite executá-la.

A estratégia de teste definida foi a partição em três classes de equivalência: Retorna 0 resultados, Retorna 1 resultado e Retorna N resultados. Para cada classe, existe a possibilidade de efetuar verificações de existência de conteúdos e de comportamento - mudança de página.

Na implementação, devido a limitações da tecnologia utilizada, foi necessário distinguir entre pesquisa a aplicações *Web* e *Desktop*, visto ter sido utilizada a ferramenta *Selenium 2* na primeira e a *Microsoft UI Automation* na segunda.

A aplicação é capaz de gerar e executar automaticamente os testes que o utilizador introduz e, no final, apresenta um relatório que descreve o sucesso ou não de cada teste efetuado.

Apesar de existirem ainda algumas melhorias a efetuar, pode-se dizer que a os objetivos deste projeto foram cumpridos.

6.2 Trabalho Futuro

Como foi dito anteriormente, existem alguns pontos ainda a melhorar, no que diz respeito à implementação do padrão Pesquisa, mas também no funcionamento da ferramenta em geral.

Em primeiro lugar, será necessário reavaliar a utilização da *framework UI Automation*, visto que não é compatível com todas as aplicações tornando o teste impossível de se realizar em alguns casos.

O uso do *Selenium 2* também não é o ideal, visto só ser compatível com aplicações *Web*. O ideal seria encontrar uma ferramenta que funcionasse de igual modo em todas as aplicações, mas sem limitações.

Outro dos assuntos que precisa de ser melhorado é a verificação dos campos de entrada de dados da pesquisa. Apesar de terem sido identificadas as relações existentes entre o padrão Pesquisa e outros já implementados (Campo de Entrada de Dados, Mestre/Detalhe,

Campo Calculado), não foi implementado nenhum tipo de verificação da validade do valor introduzido pelo utilizador. É importante referir que a identificação, um por um, de todos os elementos de entrada de dados em pesquisas com bastantes campos se torna bastante demorada, pelo que deixou de ser considerada como uma opção.

A ferramenta ainda não guarda os testes que foram configurados, sendo que esse é também um ponto que precisa de ser elaborado.

Por último é importante referir que, assim que as dificuldades encontradas sejam superadas, o trabalho passará por identificar mais padrões recorrentes nas GUIs e repetir todo o processo efetuado para o padrão implementado neste projeto.

Referências

[AJ08] Antti Kervinen Antti JŁŁskelŁinen, Mika Katara. Model-based testing service on the web. *TESTCOM/FATES*, pages 38–53, 2008.

[AK06] Mika Maunumaa Antti Kervinen. Model-based testing through a gui. pages 16–31, 2006.

[ATa10] F. Abbors e D. Tru? andcan. Approaching performance testing from a model-based testing perspective. In *Advances in System Testing and Validation Lifecycle (VALID), 2010 Second International Conference on*, pages 125 –128, aug. 2010.

[Bin99] Robert V. Binder. *Testing object-oriented systems: Models, Patterns and Tools*. Addison-Wesley, Fifth edition, 1999.

[Bur03] Ilene Burnstein. *Pratical software testing*. Springer, 2003.

[Cor01] Rational Software Corporation. *Using Rational TestFactory*. Rational Software Corporation, 2001.

[CPFA10] M. Cunha, A.C.R. Paiva, H.S. Ferreira e R. Abreu. Pettool: A pattern-based gui testing tool. In *Software Technology and Engineering (ICSTE), 2010 2nd International Conference on*, volume 1, pages V1–202 –V1–206, oct. 2010.

[dMC10] Marco André da Mota Cunha. *Padrões de Teste para Interfaces Gráficas*. PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2010.

[EFW01] Ibrahim K. El-Far e James A. Whittaker. Model-based software testing. *Encyclopedia on Software Engineering*, 2001.

[EGRHRJ94] John Vlissides Erich Gamma Richard Helm Ralph Johnson. *Design Patterns: Element of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[HJ09] Shuo Wang Hu Jin. Finite state machine for automatic gui testing. 2009.

[HPW09] B. Hofer, B. Peischl e F. Wotawa. Gui savvy end-to-end testing with smart monkeys. In *Automation of Software Test, 2009. AST '09. ICSE Workshop on*, pages 130 –137, may 2009.

[Jac08] JacaretoWiki. Jacareto, Agosto 2008. http://jacareto.sourceforge.net/wiki/index.php/Main_Page, acedido a última vez em 10 Julho de 2011.

[JS11] Atif Memon Jonathan Sternberg. Guitar home page, 2011. http://sourceforge.net/apps/mediawiki/guitar/index.php?title=GUITAR_Home_Page, acedido a última vez em 10 Julho de 2011.

[Kar] Dakshinamurthy Karra. Marathon - gui acceptance test runner. http://sourceforge.net/projects/marathonman/, acedido a última vez em 10 Julho de 2011.

[MC04] Kevin Wilson Matt Caswell, Vijay Aravamudhan. Introduction to jfcunit, Dezembro 2004. http://jfcunit.sourceforge.net/, acedido a última vez em 10 Julho de 2011.

[Mem07] Atif M. Memon. An event-SSow model of gui-based applications for testing. *Wiley InterScience*, pages 137–157, Janeiro 2007.

[Mic] Microsoft. Ui automation overview. http://msdn.microsoft.com/en-us/library/ms747327.aspx, acedido a última vez em 10 Julho de 2011.

[MV06] Bill Hasling Marlon Vieira, Johanne Leduc. Automation of gui testing using a model-driven approach. 2006.

[Nym00] Noel Nyman. Using monkey testing tools. *STQE - Software Testing and Quality Engineering Magazine*, pages 18–21, Janeiro, Fevereiro 2000.

[Osm11] Addy Osmani. *Essential JavaScript Design Patterns For Brginners, Volume 1*. 2011.

[oT11] Tampere University of Technology. Tema, 2011. http://tema.cs.tut.fi/, acedido a última vez em 16 de Janeiro de 2012.

[PM10] Jeffery Callender Peter Morville. *Search Patterns*. O'Reilly Media, Inc., first edition, 2010.

[RP08] A. Ruiz e Y.W. Price. Gui testing made easy. In *Practice and Research Techniques, 2008. TAIC PART '08. Testing: Academic Industrial Conference*, pages 99 –103, aug. 2008.

[Sat06] Mikko Satama. *Event Capturing Tool for Model-Based GUI Test Automation*. PhD thesis, Tampere University of Technology, 2006.

[Tid10] Jenifer Tidwell. *Designing Interfaces*. O'Reilly Media, Inc., Second edition, 2010.

[UBL07] Mark Utting e Bruno Legeard. *Pratical model-based testing*. Morgan Kaufmann Publishers, First edition, 2007.

[vW08] Martijn van Welie. Patterns in interaction design, 2008. http://www.welie.com/patterns/index.php, acedido a última vez em 16 de Janeiro de 2012.

[Wal02] Timothy Wall. Getting started with the abbot java gui test framework, 2002. http://abbot.sourceforge.net/doc/overview.shtml, acedido a última vez em 10 Julho de 2011.

[YM10] Xuebing Yang Yuan Miao. An fsm based gui test automation model. pages 137–157, Dezembro 2010.