FACULDADE DE ENGENHARIA DA UNIVERSIDADE DO PORTO

FEUP

Padrões de Teste para Interfaces Gráficas

Marco André da Mota Cunha

Mestrado Integrado em Engenharia Informática e de Computação

Orientadora: Ana Cristina Ramada Paiva Pimenta (Professora Auxiliar)

28 de Junho de 2010

Padrões de Teste para Interfaces Gráficas

Marco André da Mota Cunha

Mestrado Integrado em Engenharia Informática e de Computação

Aprovado em provas públicas pelo Júri:

Presidente: Raul Fernando de Almeida Moreira Vidal (Professor Associado)
Vogal Externo: José Francisco Creissac Freitas Campos (Professor Auxiliar)
Orientadora: Ana Cristina Ramada Paiva Pimenta (Professora Auxiliar)

---

22 de Julho de 2010

Resumo

Hoje em dia o uso de aplicações informáticas tende a ser feito usando Interfaces Gráficas com o Utilizador (GUIs). A qualidade das aplicações, aos olhos do utilizador, está muito dependente da qualidade da GUI com que este interage. Torna-se portanto necessário testar GUIs de forma a melhorar a sua qualidade. Este teste enfrenta várias dificuldades, sendo que estas fazem com que o processo de teste leve frequentemente muito tempo e acarrete elevados custos. Existe então a necessidade de arranjar formas de testar GUIs mais simples e automatizadas, reduzindo custos e aumentando a rapidez do processo de testes. No entanto, as GUIs possuem alguns comportamentos recorrentes, ou seja, que se processam de forma semelhante e produzem resultados semelhantes. Exemplos destes comportamentos são formulários de autenticação e de registo. Estes comportamentos recorrentes denominam-se Padrões e existe hoje em dia muito conhecimento a seu respeito.

Este trabalho de investigação tem como objectivo avaliar a possibilidade da utilização desse conhecimento para testar as GUIs de uma forma automatizada. Esta automatização espera-se que seja feita ao nível quer de geração, quer de execução de casos de teste.

Inicialmente foi feito o levantamento do Estado da Arte relativo ao teste de GUIs. Foi identificado um conjunto de estratégias como a Partição de Classes de Equivalência e um conjunto de ferramentas como *Record/Playback*. No âmbito desta etapa foram ainda identificados vários Padrões existentes em interfaces gráficas com o utilizador. Relativamente a um subconjunto destes Padrões foi estudado o contexto em que se inserem, os seus objectivos e qual a estratégia de teste mais apropriada para cada um deles. Este estudo permitiu identificar um conjunto de relações existentes entre vários Padrões bem como uma estratégia genérica de teste a eles associada.

Para validar a estratégia de teste definida na Abordagem, foi criada uma ferramenta chamada PETTool. A PETTool foi desenvolvida recorrendo à *Framework UIAutomation*. Esta *Framework* foi escolhida devido a vários factores mas um dos mais relevantes foi a possibilidade de execução de acções de forma automatizada quer em aplicações locais quer em sítios *Web*. Estas acções não se encontram sempre disponíveis pelo que tiveram que ser implementados mecanismos alternativos de as realizar para tornar a ferramenta mais genérica. De forma a validar a abordagem foram efectuados dois grupos de testes. O primeiro grupo destinou-se a verificar se a PETTool permite especificar o comportamento de uma aplicação, gerar e executar testes. O segundo grupo de testes foi usado para avaliar a capacidade de detecção de erros por parte da PETTool. Nesse sentido foram introduzidos alguns erros em aplicações e estas foram testadas com a ferramenta.

Os resultados obtidos sugerem que a abordagem é de facto capaz de realizar as tarefas esperadas e que o esforço necessário não é muito elevado. Concluiu-se portanto que esta é uma abordagem válida e que, dada a existência de um elevado número de Padrões, o seu desenvolvimento é justificado e pode trazer mais-valias no processo de teste de GUIs.

Abstract

Nowadays, software tends to be used through its Graphical User Interface (GUI). Software's quality, as it is perceived by the final user, is dependent on the quality of its GUI so, it is necessary to test it. This process, however, faces many difficulties that increase the time spent and lead to higher costs. Therefore, there is the need to find ways to test GUIs in a simpler and more automatic way, reducing costs and improving the speed of the whole test process. GUIs have some recurrent behaviours such as authentication or registration forms. Such behaviours are very similar and produce similar results. Those behaviours are called Patterns and there is much knowledge about them.

This thesis has the purpose to evaluate the possibility to use that knowledge to test GUIs in an automatic way. This automation is expected to be both at the test generation and test execution levels.

Initially, the state of the art in what concerns to GUI testing was analyzed. Some strategies such as Equivalence Class Partitioning and some tools like Record/Playback were identified. In this process, Patterns were also identified in GUIs. Some of those Patterns were studied in order to understand when they are used, their purpose and the test strategy which is most appropriate to each one of them. This study led to the identification of many relations among Patterns as well as a test strategy for them.

In order to validate the test strategy identified, a tool called PETTool was developed. This tool was developed using the UIAutomation Framework. This Framework was chosen due to many advantages but one of the most relevant was the capability to execute some actions automatically both in local applications and in browsers. These actions however are not always available so there was the need to implement some alternative mechanisms in order to keep the tool as generic as possible. Two groups of tests were executed to validate the approach. The first one tested if PETTool was capable to specify a GUI's behaviour, to generate and execute tests for that behaviour. The second group of tests was used to evaluate PETTool's error detection capability. In those tests, some errors were injected in applications which were tested using the tool.

The test results suggest that this approach is in fact capable of executing the expected tasks and the required effort is acceptable. It was concluded that this approach is valid and, due to the existence of many Patterns, its development is justified and it can bring advantages for GUI testing process.

Agradecimentos

À minha orientadora, professora Ana Paiva, por ter estado sempre disponível para me receber, para me aconselhar e orientar ao longo de toda esta Dissertação.

Aos professores Hugo Ferreira, Rui Abreu e Ana Paiva pela preciosa colaboração na escrita do artigo científico e por todas as dicas que só a experiência sabe ensinar.

Aos meus amigos, principalmente à Ana Catarina, Ana Sofia e Pedro, por terem estado sempre disponíveis para mim, sempre prontos a discutir ideias e dar conselhos. Muito obrigado pelas vezes em que, simplesmente por ouvirem o que eu tinha a dizer, me ajudaram a resolver os meus problemas.

Aos meus avós, que sempre acreditaram em mim e que me acompanharam desde pequeno.

Por fim, e não menos importante, obrigado aos meus Pais. Por sempre me terem dado a liberdade que precisava e por estarem sempre disponíveis para me aconselhar, para me ouvir e para me ajudar no que podem. Obrigado por me terem transmitido os valores que fazem de mim aquele que sou hoje.

Um grande Obrigado a todos.

Marco Cunha

Conteúdo

Lista de Figuras

Lista de Tabelas

Abreviaturas e Símbolos

ASP	*Active Server Pages*
GUI	*Graphical User Interface*
GUITAR	*GUI Testing frAmewoRk*
NaN	*Not a Number*
PETTool	*PattErn Testing Tool*
UML	*Unified Modeling Language*
XML	*eXtensible Markup Language*

Capítulo 1

Introdução

O teste de *software* nunca foi uma área simples no ramo da informática. Esta área torna-se particularmente complexa quando o teste recai sobre interfaces gráficas com o utilizador (GUIs). Estas componentes colocam dificuldades adicionais ao teste que não podem ser ignoradas. Deste modo, é importante encontrar novas estratégias que permitam não só tornar o seu teste mais eficaz, como mais rápido e automatizado.

As GUIs possuem alguns comportamentos recorrentes, ou seja, que se processam de forma semelhante e produzem resultados semelhantes. Exemplos destes comportamentos são formulários de autenticação e de registo. Estes comportamentos recorrentes denominam-se Padrões e existe hoje em dia muito conhecimento a seu respeito. Segundo [MH03], "Um Padrão (...) descreve um problema bem conhecido, com pistas para o detectar num dado contexto, possíveis soluções e quais as forças relacionadas".

Neste sentido, começa a haver interesse em utilizar Padrões para avaliar a qualidade das interfaces. É portanto esperado que a identificação de Padrões nas GUIs permita realizar um teste mais eficaz das mesmas.

1.1 Contexto

Actualmente, praticamente todo o *software* desenvolvido possui uma interface gráfica com o utilizador (GUI). Dado que a utilização do *software* se faz a partir dessa interface, a qualidade da GUI é crucial para o sucesso do *software*.

Dada a importância da qualidade das GUIs, é fundamental testá-las para avaliar o seu funcionamento. No entanto, esta é uma tarefa complexa. Problemas relacionados com o elevado número de possíveis combinações de acções do utilizador tornam praticamente impossível, na maioria dos casos, efectuar um teste exaustivo à GUI.

Apesar de existir um conjunto de estratégias e ferramentas que visam testar as GUIs,

estas possuem algumas limitações. As principais limitações baseiam-se no tempo dispendido, nos custos associados e, em alguns casos, na reduzida possibilidade de automatização do processo.

Torna-se portanto importante encontrar formas de testar GUIs que procurem reduzir de alguma forma os problemas mencionados.

1.2 Objectivos

Com esta tese de mestrado pretende-se avaliar se o conhecimento existente ao nível Padrões de comportamento e desenho das interfaces gráficas com o utilizador pode ser utilizado para as testar.

Para tal pretende-se identificar os principais Padrões existentes em interfaces gráficas com o utilizador. Com estes Padrões, pretendem-se determinar erros típicos na sua implementação (do ponto de vista de comportamento, não de desempenho da aplicação) e, com isso, definir estratégias de teste especificamente a eles dirigidas.

Um exemplo de um Padrão de uma GUI é um campo de preenchimento obrigatório num formulário. Um erro comum na implementação deste Padrão, seria permitir enviar o formulário sem que o campo estivesse preenchido. Uma estratégia de teste possível seria a utilização de partição em classes de equivalência, testando o comportamento da GUI para as duas classes: campo obrigatório preenchido e campo obrigatório não preenchido.

Após se ter obtido um conjunto de Padrões de comportamento e desenho de GUIs e se terem definido as estratégias de teste para cada Padrão, pretende-se desenvolver uma ferramenta capaz de gerar os testes para uma GUI de uma forma automática, através da instanciação dos Padrões. Ou seja, após o programador identificar uma determinada caixa de texto como sendo um campo obrigatório, a ferramenta geraria os dois casos de teste descritos acima, executaria os testes e indicaria quais os erros detectados.

1.3 Estrutura da Dissertação

A dissertação encontra-se dividida em seis capítulos. No primeiro capítulo é feita a introdução ao tema da dissertação, definindo o contexto em que se insere e os seus objectivos. No segundo capítulo é feita a apresentação do estado da arte relativo ao teste de *software* (principalmente orientado para as interfaces gráficas com o utilizador) e aos Padrões de comportamento das interfaces. O terceiro capítulo apresenta, do ponto de vista teórico, a abordagem utilizada e que procura satisfazer os objectivos da tese. Os principais detalhes de implementação encontram-se detalhados no capítulo quatro. No quinto capítulo são apresentados vários casos de estudo que visam avaliar do ponto de vista prático, a abordagem descrita no terceiro capítulo. Por fim, no sexto capítulo são

tiradas conclusões ao trabalho realizado e são identificados os principais aspectos que podem ser melhorados de forma a dar continuidade ao trabalho efectuado.

Capítulo 2

Estado da Arte

Neste capítulo será descrito o estado da arte relativo aos testes de *software* em geral e ao teste de GUIs em particular. Serão descritas em primeiro lugar as dificuldades que o teste de GUIs acarreta e quais as estratégias e ferramentas actualmente utilizadas. Por fim serão enumerados os principais Padrões de Comportamento numa GUI.

2.1 Dificuldades do Teste de GUIs

Testar uma Interface Gráfica com o Utilizador (GUI) não é de todo trivial. Existem vários factores a ter em conta neste processo e que, devido à sua complexidade, tornam praticamente impossível garantir a correcção de uma determinada GUI.

Em primeiro lugar, é notório o elevado número de eventos que podem ser despoletados. No limite, cada *pixel* do ecrã pode representar uma acção diferente e se for tida em conta a combinação entre sequências de cliques e movimentações do rato, estes valores crescem muito rapidamente, tornando impossível testar todos os traços de execução em tempo útil. Alguns dos traços de execução referidos anteriormente, podem simplesmente não ter sido pensados pelo programador. [Ger97]

A definição de critérios de cobertura, ou seja, os critérios que nos permitem dizer se um determinado conjunto de testes cobre de forma apropriada um pedaço de código, quando aplicada ao teste de GUIs, é um processo complexo. Por exemplo, um determinado estado de uma GUI pode ser alcançado por muitos traços de execução distintos. Um exemplo típico desta característica pode ser a gravação de um ficheiro aberto numa determinada aplicação. O utilizador pode aceder com o rato à opção num menu de atalhos ou na barra de menus. O critério de cobertura usado determina se basta testar uma das opções ou se é necessário testar ambas, sendo que esta última pode, dependendo da implementação, levar a testes redundantes. [Ger97, Mem02]

Uma das principais dificuldades inerentes ao teste de GUIs baseia-se na impossibilidade de distinguir uma interface lenta de uma interface bloqueada. Assim sendo, qualquer

valor arbitrário estipulado para a duração máxima de um teste, pode assinalar um erro caso a interface demore mais tempo a responder, mesmo que seja, por exemplo, uma base de dados que não responda em tempo útil.

Tendo em consideração que os requisitos mudam rapidamente e os utilizadores frequentemente gostam de alterar a posição dos controlos (caixas de texto, botões, etc.) dentro de uma GUI, torna-se necessário evitar que os testes sejam feitos utilizando a posição absoluta desses mesmos controlos. Deste modo é possível que, caso um botão troque de posição dentro da GUI, o teste que implicava um clique nesse mesmo botão continue a funcionar. [Mem02]

Dependendo da tecnologia utilizada, um outro problema que surge da utilização de GUIs consiste no fecho das janelas da aplicação. Caso uma janela esteja responsável por uma acção, por exemplo um conjunto de escritas numa base de dados, caso a janela seja fechada antes de concluir a sua tarefa, é difícil conseguir saber se o estado da aplicação é consistente com o esperado. No entanto, este tipo de sincronização entre elementos externos e/ou outras janelas da aplicação é, por vezes, difícil de detectar. [Ger97, LHRM07]

Um outro problema reside na avaliação de quais os testes que passaram, os que falharam e identificar a localização do erro. Por exemplo, assumindo que um determinado teste possui cinco passos, caso o segundo passo falhe, é quase certo que os restantes também falhem. Como tal, é necessário haver mecanismos que indiquem explicitamente onde surgiu o erro e parem o teste de modo a que não sejam reportados erros nos passos seguintes. [Mem02]

2.2 Estratégias de Teste

Perante o grande número de dificuldades que o teste de GUIs enfrenta, têm sido utilizadas, ao longo dos anos, várias estratégias que procuram testar da melhor forma possível as GUIs. Todas as estratégias possuem vantagens e limitações, logo não existe uma estratégia perfeita de teste, apenas várias estratégias que se complementam.

As várias estratégias de teste organizam-se em dois grandes grupos: caixa preta e caixa branca. De forma a prestar um maior auxilio às equipas de teste, foram ainda desenvolvidas ao longo dos anos vários tipos de ferramentas que permitem automatizar a geração e/ou execução de casos de teste para GUIs.

2.2.1 Testes de Caixa Preta

As estratégias de caixa preta destinam-se a avaliar o comportamento das aplicações sem conhecer a sua estrutura interna, apenas os resultados esperados. Deste modo, algumas destas estratégias ficam por vezes dependentes da experiência do responsável pela concepção dos testes.

2.2.1.1 *Checklists* manuais

As *checklists* são listas de aspectos aos quais uma GUI deve obedecer. Contém itens como a localização dos botões, consistência de nomes, tamanho da janela, etc. Dada a sua simplicidade, esta técnica é relativamente eficaz e facilmente documentada já que a lista é a sua própria documentação. Contudo, a verificação dos itens presentes na lista pode levar bastante tempo. Como tal, é util sempre que possível efectuar alguma automatização em alguns itens, como por exemplo a verificação do texto das *label* ou botões. [Ger97, Mem03, CHE10]

2.2.1.2 Partição em Classes de Equivalência

Esta estratégia, após identificar os vários *inputs* possíveis, divide-os em classes de equivalência. Estas classes devem obedecer a três princípios:

- A aplicação funciona de uma forma análoga para todos os elementos da classe;
- Testar a aplicação com um elemento da classe é suficiente para ter confiança que com os restantes elementos a aplicação também funciona correctamente;
- Um erro encontrado num elemento da classe deveria ser encontrado por qualquer outro elemento da classe caso este fosse utilizado no teste.

Após ser feita a divisão dos *inputs* em classes de equivalência é então escolhido para cada classe um elemento que a represente e este elemento serve como caso de teste para a aplicação. A escolha de apenas um ou dois elementos de cada classe de equivalência reduz o número de testes a ser feito o que os torna mais rápidos. [GTWJ03, Bur03]

2.2.1.3 Análise dos valores fronteira

Segundo [Bei90], "*Bugs lurk in corners and congregate at boundaries*". Esta estratégia tem portanto como objectivo detectar os erros nos valores fronteira. Por exemplo, considerando uma aplicação que deve obter um valor inteiro entre 0 e 20 inclusive, segundo esta estratégia devem ser feitos os seguintes testes:

- Um valor menor que 0 arbitrário. Ex: -5;
- Um valor maior que 20 arbitrário. Ex: 30;
- Um valor entre 0 e 20 arbitrário. Ex 13;
- Os valores na fronteira: 0 e 20;
- Valores imediatamente acima e abaixo dos valores na fronteira: -1, 1, 19, 21.

Tabela 2.1: Casos de teste pela estratégia Grafo Causa-Efeito

Causa									
Num1 > 0	Sim	Sim	Sim	Não	Não	Não	Não	Não	Não
Num1 = 0	Não	Não	Não	Sim	Sim	Sim	Não	Não	Não
Num1 < 0	Não	Não	Não	Não	Não	Não	Sim	Sim	Sim
Num2 > 0	Sim	Não	Não	Sim	Não	Não	Sim	Não	Não
Num2 = 0	Não	Sim	Não	Não	Sim	Não	Não	Sim	Não
Num2 < 0	Não	Não	Sim	Não	Não	Sim	Não	Não	Sim
Efeito									
Valor Positivo	Sim	Não	Não	Não	Não	Não	Não	Não	Sim
Valor Negativo	Não	Não	Sim	Não	Não	Não	Sim	Não	Não
Zero	Não	Sim	Não	Sim	Sim	Sim	Não	Sim	Não

Esta estratégia, além dos benefícios da partição em classes de equivalência, aumenta ainda a confiança no código visto que, para além de analisar as classes de equivalência, analisa os valores que estão no limite dessas classes. Por exemplo, se uma aplicação tiver dois comportamentos distintos para valores menores que 5 e para maiores ou iguais a 5, do ponto de vista da Partição em Classes de Equivalência pode ser suficiente testar com os valores 0 e 10. No entanto, se o programador cometer um erro e usar o operador <= em vez de <, o erro não é detectado. Com a Análise dos Valores Fronteira, estes problemas já são detectados.[GTWJ03, Bur03]

2.2.1.4 Grafo causa-efeito

Nesta estratégia é necessário, em primeiro lugar, determinar que tipo de causas (*inputs*) e efeitos (*outputs*) podem existir na aplicação. Perante essas causas e efeitos é então criada uma tabela que mostra a relação entre as diferentes causas e efeitos. Cada coluna da tabela representa um caso de teste que deve ser testado. Considerando, por exemplo, uma aplicação que leia dois inteiros e indique se o seu produto é negativo, positivo ou zero, na tabela 2.1 é possível ver os vários testes que teriam que ser feitos e qual a principal limitação desta estratégia: o número de casos de teste tende a crescer muito rapidamente com o número de *inputs*.

Apesar de haver apenas dois valores de *input* os quais podem ser divididos em apenas três categorias, estes levam à necessidade de serem feitos nove testes. Neste cenário, o número de testes é aceitável mas num cenário mais complexo, o número de testes a realizar facilmente se torna excessivo. [GTWJ03, Bur03]

2.2.1.5 Adivinhar Erros

A estratégia de adivinhar erros é a menos estruturada, mas no entanto consegue em certos casos ser muito eficiente. Nesta estratégia, quando perante uma aplicação, o res-

ponsável pelos testes "adivinha" onde podem estar os erros e verifica se a aplicação se comporta devidamente nessas componentes.

Esta estratégia depende muito da intuição e experiência do responsável pelos testes, não devendo ser por isso utilizada como única estratégia de teste. [GTWJ03, Bur03]

2.2.2 Testes de Caixa Branca

As estratégias de caixa branca destinam-se a avaliar o comportamento das aplicações tendo por base a estrutura interna do código. Existem várias estratégias de caixa branca sendo algumas destas estratégias de utilização obrigatória em certas aplicações, nomeadamente as aplicações cujo funcionamento é crítico. Estas estratégias possuem vários níveis de teste:

- Cobertura de Instruções: Garantir que todas as instruções são executadas nos testes;
- Cobertura de Decisões: Para todas as decisões, existem cenários de teste que cobrem todas as alternativas. (Num "*if*" haver testes cuja condição resulta em verdadeiro e falso);
- Cobertura de Condições: Para todas as decisões, cada condição que a compõe é testada com valores que a tornem verdadeira e falsa (Num "*If*" com duas condições, testar cada uma delas com valores verdadeiros e falsos);
- Cobertura Modificada de Condições e Decisões: Para todas as decisões, é provado que o resultado final da decisão é afectado directamente pelo resultado de cada uma das condições (Num "*If*" com duas condições, testar cada uma das condições a verdadeiro e falso, colocando a outra condição com um valor neutro, verdadeiro caso a operação seja um "*and*" e falso caso a operação seja um "*or*" por exemplo);
- Cobertura de Condições Múltiplas: Verificar todas as possíveis combinações de valores de condições numa decisão;
- Cobertura de Caminhos: Verificar todos os caminhos de execução de código da aplicação.

Dado que numa GUI o seu código encontra-se por vezes protegido, estas estratégias não podem ser sempre aplicadas neste cenário de teste.[GTWJ03, Bur03]

2.2.3 Ferramentas de Teste

De forma a tornar mais eficiente o teste de GUIs, existem actualmente vários tipos de ferramentas que suportam este tipo de teste.

2.2.3.1 *Record/playback*

As ferramentas de *Record/Playback* permitem gravar as acções do utilizador (*Record*) e mais tarde repetir essas acções (*Playback*) e verificar se a GUI se comporta da forma esperada. Estas ferramentas permitem automatizar a fase de *Playback* tornando-a fácil de executar. No entanto, a fase de *Record* fica muito dependente da experiência do programador, do estado de desenvolvimento da aplicação e pode levar muito tempo o que acarreta elevados custos.

Um outro problema comum para os utilizadores destas ferramentas consiste na posição dos controlos no ecrã. Caso os controlos sejam especificados com posições absolutas, os testes ficam difíceis de manter pois os controlos podem facilmente trocar de sítio.

Um dos principais problemas destas ferramentas reside ainda no facto de estas executarem os passos sequencialmente sem tempos de espera. Isto leva a que por vezes estas ferramentas assinalem erros simplesmente porque a aplicação demora mais tempo a responder do que a ferramenta espera. [Mem02]

Actualmente existem várias ferramentas deste tipo disponíveis, como por exemplo Marathon[Sys10], Badboy[Sof10], WinRunner[Ltd10], etc. Estas ferramentas possuem âmbitos de utilização diferentes. Enquanto o Marathon se destina a aplicações Java, Badboy está orientado para sítios *Web* e WinRunner pode funcionar para aplicações locais ou sítios *Web*. Um aspecto que estas ferramentas têm em comum é que permitem que não sejam utilizadas posições absolutas nos comandos a realizar. Outra característica importante reside na forma de armazenamento dos comandos. Estas ferramentas permitem que o utilizador altere manualmente a sequência de testes gravados o que permite alterar pequenos aspectos sempre que necessário. Estes aspectos são muito importantes para a eficácia do uso destas ferramentas.

2.2.3.2 Testes aleatórios

Os testes aleatórios baseiam-se num princípio muito simples: “Se seis macacos escreverem em seis máquinas de escrever de forma aleatória durante um milhão de anos, irão eventualmente escrever as obras de Isaac Asimov.”.[Nym00]

Estas ferramentas têm portanto como modo de funcionamento a geração de uma sequência de eventos aleatória que vá, eventualmente, encontrar os erros da aplicação.

Estas ferramentas são fáceis de implementar, rápidas e com reduzidos custos, boas a encontrar falhas de sistema e problemas de performance. Por outro lado, os testes ficam muito dependentes do factor sorte, a forma de execução é imprevisível, torna-se difícil reproduzir um determinado erro específico e este não é o tipo de ferramenta mais adequado para encontrar alguns tipos de erros.

De forma a colmatar alguns dos problemas da utilização desta estratégia, algumas ferramentas deste tipo, em vez de gerarem apenas eventos aleatórios (macacos pouco

inteligentes) registam quais os eventos gerados e possuem mecanismos para reduzir a aleatoriedade de eventos (macacos inteligentes), aproximando-se um pouco da utilização esperada da aplicação.

Esta estratégia é utilizada em algumas ferramentas como por exemplo a Rational TestFactory e o Director de Testes da Microsoft afirmou que, em alguns projectos, foram encontrados dez a vinte porcento de erros usando estas ferramentas. [Nym00]

2.2.3.3 Baseada em Modelos

As ferramentas de Teste Baseado em Modelos tem por base a utilização de um modelo do sistema a ser testado e esse modelo é usado para avaliar se o sistema se comporta de forma apropriada. Para tal, é verificada a consistência entre o modelo e o sistema, ou seja, é verificado se ao executar uma acção sobre o modelo a correspondente acção sobre sistema conduz a um estado equivalente ao do modelo. Por exemplo, considerando um modelo do funcionamento de uma pilha que indica que a operação de esvaziar a pilha deve levar a uma pilha vazia. Se a operação for executada no sistema e for obtida uma pilha com elementos, é assinalado um erro.

Esta estratégia acarreta no entanto algumas dificuldades. Em primeiro lugar é necessário criar o modelo o que acarreta custos. Outro problema típico desta estratégia reside na explosão de estados e existência de ciclos. Dado que a interface possui vários eventos possíveis em cada momento e que facilmente existem ciclos (por exemplo escolher uma opção e de seguida cancelar e repetir o processo), é necessário tomar medidas para que a exploração da máquina de estados não entre num ciclo infinito. Tal como a própria implementação da GUI, o modelo também se encontra sujeito a erros. Deste modo, para cada erro detectado por esta estratégia, é importante avaliar se o erro se encontra na GUI ou se encontra no modelo.[UL06]

Existem actualmente algumas ferramentas de Teste Baseado em Modelos como por exemplo o Guitar[GUI10] e o SpecExplorer[Cor10a]. Estas ferramentas tem no entanto um modelo de funcionamento bastante diferente.

Guitar é um conjunto de ferramentas sendo que cada uma tem uma tarefa específica. Em primeiro lugar, o JFC GUI Ripper encarrega-se de converter uma GUI num ficheiro XML que a representa. De seguida, o GUI Structure to Graph Converter converte o ficheiro XML para um grafo que representa a GUI. Após ter sido obtido o grafo, o Test Case Generator gera um conjunto de testes que tem por base a exploração do grafo obtido anteriormente. Por fim, os testes são executados pelo JFC Replayer.

O SpecExplorer é menos automático que o Guitar tendo o modelo que ser construído manualmente ou convertido a partir de um modelo UML. À semelhança do Guitar, o SpecExplorer converte o modelo para uma máquina de estados e gera testes em função da

exploração dessa máquina de estados. Uma das dificuldades da utilização de SpecExplorer reside na definição de quais os eventos possíveis em cada estado da GUI dado que estes podem ser demasiados e por vezes pouco óbvios. Outra dificuldade relevante do SpecExplorer é que este não suporta a execução dos eventos na GUI tendo para isso que ser utilizadas extensões como por exemplo a GUI Mapping Tool[PFTV05] que faz o mapeamento entre as transições entre estados e os eventos na GUI. [PFV07]

2.2.3.4 Testes Unitários

Testes unitários são testes feitos a pequenas componentes de uma aplicação, tipicamente um método. No entanto, existem algumas ferramentas que possibilitam que este tipo de testes seja feito a componentes de GUIs. Para criar este tipo de testes é necessário que, de uma forma manual, sejam definidos os testes. Esta definição requer que sejam simuladas as acções do utilizador. Para isso podem ser usadas ferramentas como por exemplo Jemmy, Abbot[Wal10], ou UIAutomation[Cor10b]. Após a execução das acções é possível verificar se o comportamento obtido por parte da GUI corresponde ao definido nos testes.

2.3 Padrões das Interfaces

Nos dias que correm, em todas as aplicações que possuem interfaces gráficas com o utilizador, é possível encontrar alguns componentes comuns a muitas outras interfaces. Estes componentes, apesar de poderem ter aspectos diferentes e eventualmente uma ou outra característica que varia, possuem funcionalidades semelhantes e são utilizados para o mesmo fim. Por exemplo, grande parte dos sítios *Web* possuem um formulário de autenticação sendo que este possui tipicamente dois atributos, um de nome do utilizador e outro de palavra-passe.

A utilização destes componentes é tão comum que os utilizadores, sem grande esforço, percebem rapidamente como funcionam e, muitas vezes, até ficam surpreendidos se estes não existirem em determinados contextos.

Assim sendo, a identificação destes aspectos recorrentes (padrões) e a determinação de quais os erros típicos resultantes da sua implementação, pode levar a testes mais eficazes das interfaces com o utilizador nas quais estes padrões existem. Na tabela 2.2 é possível ver uma lista de alguns destes padrões. Estes foram obtidos a partir de livros e sítios Web dedicados à reunião de informação relativa a Padrões e foram escolhidos por terem sido os que pareceram mais relevantes para este trabalho de investigação.

Tabela 2.2: Padrões das Interfaces

Padrão	Descrição
Mestre/Detalhe [Tid05, Laa10, vW10]	Um campo Mestre pode assumir vários valores (exemplo: *combobox*). Consoante o valor escolhido para o Mestre, o Detalhe(exemplo: uma lista) é actualizado
Campo Obrigatório [Oy10]	Campo de um formulário que deve obrigatoriamente ser preenchido
Campo Calculado	Campo de uma janela que é calculado em função dos valores presentes em outros campos. Os valores presentes nesses campos são tipicamente inseridos pelo utilizador.
Campo Automático	Campo de uma janela que é gerado automaticamente não devendo por isso ser alterado (Exemplo: Número de uma nova factura)
Campo de entrada de dados	Campo de uma janela que possui um determinado tipo de dados. É necessário verificar se os dados inseridos estão de acordo com o tipo especificado
Campos activos mediante condição [Tid05]	Campos que só estão activos caso uma condição se verifique (por exemplo uma *checkbox* marcada)
Anular [Tid05, Laa10, Oy10]	Acção que anula a última alteração feita. Pode ser multi-nível até uma profundidade especificada
Repetir [Tid05, Laa10]	Acção que repete a última alteração anulada. Pode ser multi-nível até uma profundidade especificada
Navegação de Registos	Operações de avançar, retroceder, ir para o primeiro ou último registo numa janela de apresentação de dados
Filtro [Laa10, vW10, Oy10, Tox10]	Perante um conjunto de linhas, o utilizador insere que informação pretende aceder e as linhas que não estejam relacionadas com essa informação desaparecem
Formulário [vW10]	Página com vários campos que devem ser preenchidos pelo utilizador
Autenticação [vW10]	Par de caixas de texto "Nome do Utilizador" e "Palavra-passe" que permitem ao utilizador autenticar-se na aplicação/sítio *Web*

Padrão	Descrição
Pesquisa [vW10]	Pesquisa de um determinado conjunto de caracteres num bloco de texto
Lista Dupla [Laa10, vW10]	O utilizador possui duas listas e passa valores de uma lista para a outra usando botões com esse efeito. Tipicamente existem botões para passar um elemento e todos os elementos
Separadores [Tid05, vW10, Oy10, Tox10]	Elemento semelhante aos separadores de papel. Permitem dividir a informação por várias secções.
Paginação [vW10, Tox10]	Divisão da informação num conjunto de páginas. Tipicamente existe a operação de ir para a primeira página, página anterior, página seguinte e última página
Árvores de opções [Tid05, Laa10, vW10]	Conjunto de opções organizadas em árvore (por exemplo sistema de ficheiros) em que operações de mais alto nível permitem aceder a operações de mais baixo nível
Valores por omissão [Tid05, Oy10, Tox10]	Valor presente num determinado campo caso nenhum outro valor seja explicitamente definido
Ordenar por Coluna [Tid05, vW10, Tox10]	Operação que ordena a informação presente numa tabela em função dos dados de uma das suas colunas
Instruções passo a passo [Tid05, vW10, Oy10, Tox10]	Conjunto de janelas nas quais é pedido um conjunto de indicações (por exemplo, a instalação de uma aplicação)
Dropdown chooser [Tid05]	Controlo que, quando nele é feito um clique do rato, se expande mostrando um conjunto de opções.
Indicador de Progresso [Tid05, Oy10]	Controlo que indica o estado do progresso de uma determinada tarefa (por exemplo, percentagem de instalação de uma aplicação
Tabela Editável [Laa10]	Conjunto de informações apresentadas de forma tabular e o utilizador pode editar directamente
O escudo [vWvdVE00]	Pedido de confirmação após a invocação de uma acção que não pode ser anulada (exemplo: eliminação permanente de um ficheiro)
Navegação Global [Tid05, vW10]	Pequena secção da aplicação ou sítio *Web* que fornece um conjunto de atalhos para as áreas mais relevantes
Adicionar Elementos a Lista [vW10]	O utilizador adiciona uma nova linha a um conjunto de linhas de informação existentes

2.4 Conclusões

O teste de GUIs, quando comparado com outros tipos de teste, apresenta várias dificuldades adicionais. Algumas destas dificuldades consistem, por exemplo, na quantidade enorme de cenários a serem testados e no tempo e custo associados aos testes. Isto pode levar a que uma GUI não seja devidamente testada, o que poderá reduzir a sua qualidade e consequentemente a qualidade da aplicação aos olhos do utilizador final.

As estratégias de teste utilizadas para outros tipos de teste geralmente não se aplicam (pelo menos directamente) no caso das GUIs. Assim sendo, são necessários mecanismos direccionados para este tipo de teste. Ferramentas do tipo *Record/Playback* e de testes aleatórios procuram estar mais direccionadas a este tipo de teste mas mesmo estas possuem limitações.

No entanto, é bastante comum hoje em dia encontrar pequenos comportamentos comuns a várias GUIs. Estes comportamentos são designados Padrões. Espera-se que esta recorrência de comportamento possa ser usada para melhorar o teste de GUIs. No capítulo seguinte será apresentado, do ponto de vista teórico como esta utilização pode ser feita.

Capítulo 3

Abordagem

Os Padrões identificados na secção 2.3 são comuns a várias GUIs. No entanto, apesar desta recorrência, existem algumas variações de implementação para implementação. Um exemplo simples de um problema recorrente consiste num formulário de autenticação. A solução para este problema, do ponto de vista de estrutura, é normalmente muito semelhante: duas caixas de texto, uma para o nome do utilizador e outra para a sua palavra-passe e um botão para validar a informação introduzida. No entanto, do ponto de vista da resposta do sistema a uma tentativa de autenticação, existem vários tipos de resposta que se verificam. Alguns sítios *Web* como o GMail[1] respondem a uma tentativa de autenticação inválida apresentando uma mensagem na página. O Webmail[2] utilizado na FEUP[3] apresenta igualmente uma mensagem na página em caso de dados inválidos mas, no caso de dados em falta, apresenta uma mensagem fora da página.

Estes tipos de resposta, apesar de poderem variar de implementação para implementação tendem a ser poucos. Os tipos de resposta mais comuns numa aplicação são geralmente:

- Mensagem dentro da página
- Mensagem fora da página
- Inactivação de um botão
- Saída da página
- Permanência na página

Para além da recorrência de problemas e de comportamentos associados a cada problema, existe também uma recorrência nas estratégias de teste a utilizar para cada problema, estratégias essas que foram referidas na secção 2.2.

[1] `http://www.gmail.com` Acedido em Junho de 2010
[2] `http://webmail.fe.up.pt` Acedido em Junho de 2010
[3] `http://www.fe.up.pt` Acedido em Junho de 2010

Neste capítulo será portanto apresentado de que forma o conhecimento adquirido, do ponto de vista do design de GUIs, bem como relativo às estratégias de teste de *software*, pode ser utilizado para testar de uma forma automatizada uma determinada GUI.

3.1 Padrões

As soluções genéricas para problemas recorrentes são vulgarmente designadas por "Padrões". Tal como foi visto na secção 2.3, existem vários Padrões que hoje em dia se podem encontrar em várias GUIs.

De forma a avaliar se é possível utilizar o conhecimento existente ao nível dos Padrões de comportamento para testar GUIs, apenas um subconjunto desses Padrões será utilizado. Este subconjunto possuí portanto alguns Padrões bastante recorrentes nas interfaces com o utilizador nomeadamente:

- Formulário
- Campo de Entrada de Dados
- Autenticação
- Campo Calculado
- Mestre/Detalhe

De seguida será apresentada uma descrição mais detalhada para cada um dos Padrões referidos.

3.1.1 Formulário

Um formulário consiste essencialmente num conjunto de campos que permite a introdução de informação por parte do utilizador. Este Padrão é a base que suporta os outros Padrões uma vez que os restantes existem em formulários. O Formulário possui tipicamente um botão que permite desencadear uma determinada acção estando esta dependente dos dados nele inseridos.

3.1.1.1 Estratégia de Teste

Um Formulário, por si só, não possui uma estratégia de teste associada. A estratégia de teste a usar está dependente dos restantes Padrões que existam no Formulário. Para testar um Formulário torna-se portanto necessário avaliar quais os restantes Padrões nele contidos e definir a estratégia de teste mediante esses Padrões.

É no entanto recomendável que os Padrões contidos num Formulário sejam testados de duas formas distintas, individualmente e em combinação. Testar os Padrões individualmente permite que seja avaliado se um erro num determinado Padrão é detectado correctamente (por exemplo, determinar se um campo que apenas deve aceitar valores positivos está a aceitar valores negativos). Testar os Padrões em combinação permite detectar erros que resultam de dependências entre os diversos padrões. Este último tipo de teste demora mais tempo uma vez que o número de casos de teste cresce muito rapidamente com o número de Padrões existentes.

3.1.1.2 Exemplo de utilização

Os Formulários são utilizados para as mais diversas finalidades como, por exemplo, pesquisa de um determinado conteúdo num documento (Figura 3.1).

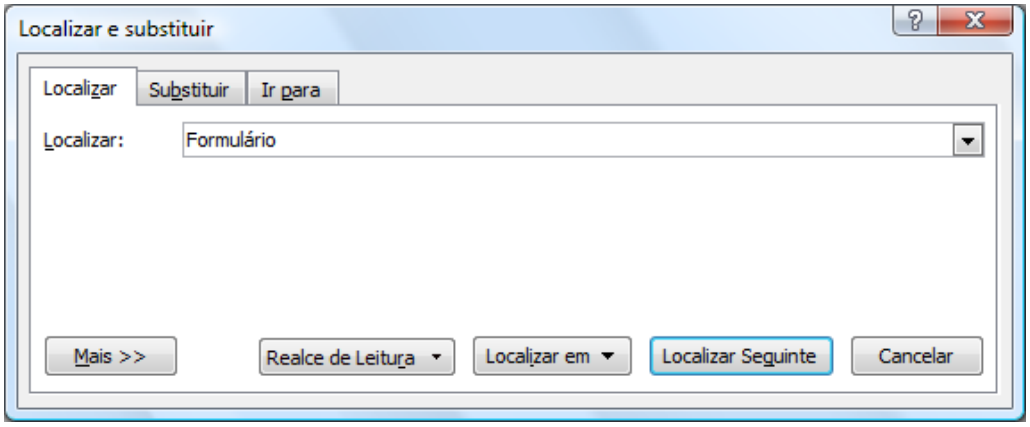


Figura 3.1: Formulário para a pesquisa de uma palavra num documento no Microsoft Word 2007

3.1.2 Campo de Entrada de Dados

Os Campos de Entrada de Dados são a principal forma de recolher informação fornecida pelo utilizador. Esta recolha pode ser feita usando caixas de texto, listas, *combobox*, *checkbox*, etc. De forma a simplificar a utilização deste Padrão, optou-se por apenas contemplar os casos em que o Padrão se apresenta sob a forma de caixa de texto, lista ou *combobox*. Outra característica importante dos Campos de Entrada de Dados reside no facto que estes podem ser obrigatórios consoante a informação que recolhem.

3.1.2.1 Estratégia de Teste

Para testar este padrão, a estratégia mais apropriada é a Análise dos Valores Fronteira (2.2.1.3), caso o campo tenha uma gama de valores permitidos ou simplesmente a Partição em Classes de Equivalência (2.2.1.2), se o campo apenas tiver como limitação um

determinado tipo de dados.

Mediante as estratégias indicadas, para testar este campo é necessário um conjunto de testes. Este conjunto é constituído por:

- Um valor válido
- Um valor inválido, mas do tipo de dados correcto
- Um valor de um tipo de dados inválido
- Valores limite admitidos pelo campo (superior e inferior)
- Valores imediatamente acima e abaixo dos limites superior e inferior (quando aplicável)

Uma vez que este Padrão se encontra normalmente relacionado com outros Padrões mais complexos, a definição de quais os valores a testar neste Padrão pode revelar-se complicada. Nesse sentido, optou-se nesta abordagem por deixar os valores a serem testados neste Padrão ao cargo do responsável pelos testes, em vez de estes serem gerados automaticamente.

3.1.2.2 Exemplo de utilização

Os Campos de Entrada de Dados são utilizados por exemplo em Formulários de registo para a criação de uma conta de e-mail no GMail[4] (Figura 3.2). Nestes casos é frequente alguns dos campos serem obrigatórios como, por exemplo, a palavra-passe escolhida.

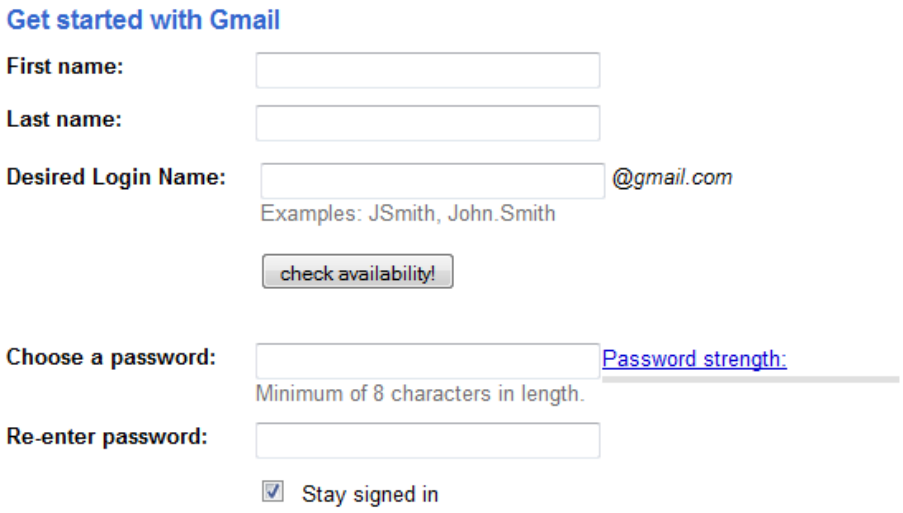


Figura 3.2: Formulário para a criação de uma conta no GMail.

[4] https://www.google.com/accounts/NewAccount?service=mail. Acedido em Junho de 2010

3.1.3 Autenticação

A Autenticação é um mecanismo que permite que um utilizador prove que ele é quem diz ser. É tipicamente utilizada quando o acesso à informação é restrito ou quando a informação a apresentar é diferente de utilizador para utilizador. Este Padrão é frequentemente implementado recorrendo a duas caixas de texto e um botão.

3.1.3.1 Estratégia de Teste

A estratégia de teste para testar uma Autenticação é a de partição em classes de equivalência (secção 2.2). Para este Padrão, três classes podem ser definidas o que conduz a três testes:

- Nome do utilizador e palavra-passe válidos
- Nome do utilizador válido e palavra-passe inválida
- Nome do utilizador inválido

3.1.3.2 Exemplos de utilização

A Autenticação pode ser encontrada em vários sítios *Web* por exemplo para aceder a contas de e-mail do GMail[5] (Figura 3.3) ou a serviços bancários.

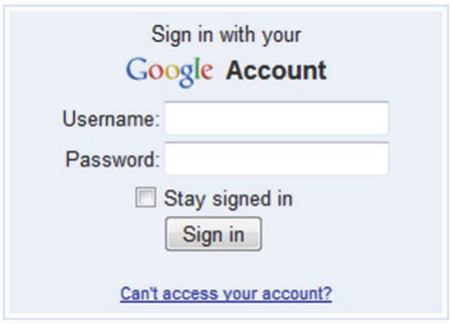


Figura 3.3: Formulário de Autenticação no GMail.

3.1.4 Campo Calculado

Um Campo Calculado é um campo cujo valor depende de outros campos. O seu valor pode ser o resultado de uma operação sobre um determinado campo (por exemplo, converter um valor monetário em Euros para o correspondente valor em Dólares) ou o resultado de uma operação sobre vários campos (por exemplo, a soma de um conjunto de valores). As operações podem ser executadas em valores numéricos ou conjuntos

[5] http://gmail.com/. Acedido em Junho de 2010

de caracteres. Assim sendo, e de forma a simplificar a utilização deste Padrão, apenas serão contemplados os casos em que a fórmula de cálculo contenha somas, subtracções, produtos ou divisões no caso de valores numéricos e igualdade ou concatenação no caso de cadeias de caracteres.

3.1.4.1 Estratégia de Teste

Para avaliar o comportamento deste Padrão, a estratégia de teste partição em classe de equivalência com análise dos valores fronteira pode ser utilizada (secção 2.2).

À semelhança do que se verifica com o Campo de Entrada de Dados, para este Padrão também é necessário que o responsável pelos testes indique quais os valores a utilizar em cada campo. Isto acontece porque, uma vez que existe relação entre vários campos, torna-se difícil avaliar que combinação de valores leva, por exemplo, a que o valor do resultado saia fora da gama de valores válidos. Deste modo é solicitado o conjunto de valores que deverão ser utilizados em cada campo que contribua para o resultado final.

3.1.4.2 Exemplos de utilização

Os Campos Calculados podem ser usados, por exemplo, em verificação de palavras-passe. Nestes casos, o utilizador deve inserir a palavra-passe duas vezes para se certificar que não há enganos, sendo aqui usada uma expressão de igualdade entre duas cadeiras de caracteres. Um outro exemplo pode ser encontrado em conversores de moedas como o existente no sítio Yahoo!Finance[6] (Figura 3.4), sendo neste caso utilizada uma fórmula de calculo numérica simples.

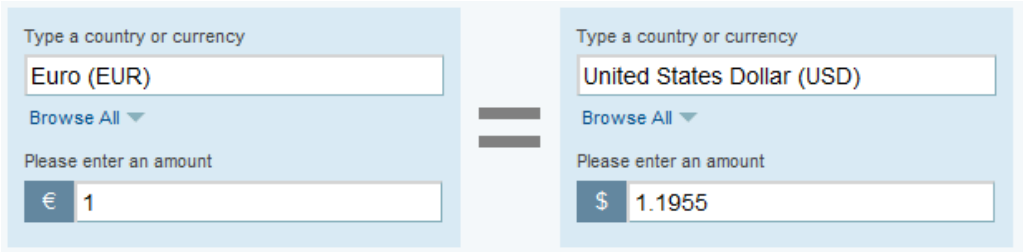


Figura 3.4: Conversor de moedas online.

3.1.5 Mestre/Detalhe

Um Mestre/Detalhe consiste em dois campos que estão relacionados entre si. Nestes campos, quando o valor de um deles muda, a gama de valores associada ao outro muda automaticamente. O primeiro designa-se Mestre e o segundo Detalhe.

[6] http://finance.yahoo.com/currency-converter/. Acedido em Junho de 2010

Tipicamente o Mestre consiste numa *combobox* ou lista enquanto o Detalhe pode ser uma *combobox*, lista ou mesmo uma *label*. Existem ainda outros cenários, no entanto estes não foram contemplados nesta abordagem.

3.1.5.1 Estratégia de Teste

De forma a testar um Mestre/Detalhe pode ser utilizada a estratégia de partição em classes de equivalência (secção 2.2).

Neste caso existem três condições que podem ser avaliadas para cada Mestre:

- O detalhe contém um conjunto de valores entre outros
- O detalhe não contém um conjunto de valores
- O detalhe contém um determinado conjunto de valores (nem mais, nem menos)

Estas condições têm como objectivo avaliar se para um dado valor definido no Mestre, o Detalhe respectivo está de acordo com os critérios especificados.

Cada uma das condições apresentada possui as suas limitações o que pode em alguns casos deixar escapar erros. A primeira e a segunda condições não garantem que não existam mais valores que os esperados. Esse problema pode levar a que o utilizador possa escolher valores inválidos apesar de a lista de valores apresentada obedecer às restrições especificadas. A terceira condição é a que oferece mais segurança mas também é a que implica mais esforço por parte pelo responsável pelos testes. Cabe portanto ao responsável pelos testes decidir a melhor forma de usar estas condições para cada caso específico.

3.1.5.2 Exemplo de utilização

Um exemplo da utilização de um Mestre/Detalhe consiste na escolha de um subúrbio em sítios *Web* de agências imobiliárias como por exemplo da Remax[7] (Figura 3.5). Uma vez que existem muitos subúrbios, normalmente são apresentadas duas *combobox* em que a primeira permite escolher uma região e, mediante a escolha feita, a segunda *combobox* apresenta os subúrbios apenas da região escolhida.

[7] http://www.remax.com.au/. Acedido em Junho de 2010

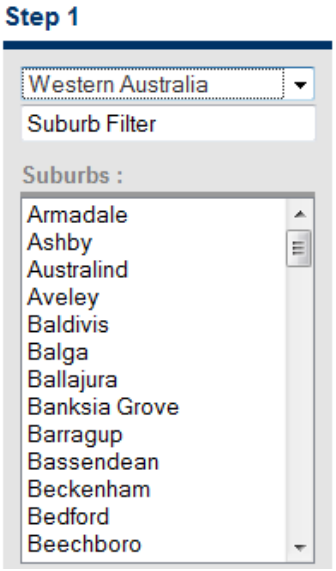


Figura 3.5: Exemplo de um Mestre/Detalhe.

3.2 Relações entre os Padrões

Todos os Padrões referidos anteriormente possuem características distintas. No entanto, os Padrões possuem várias relações entre si. Essas relações estão ilustradas na figura 3.6.

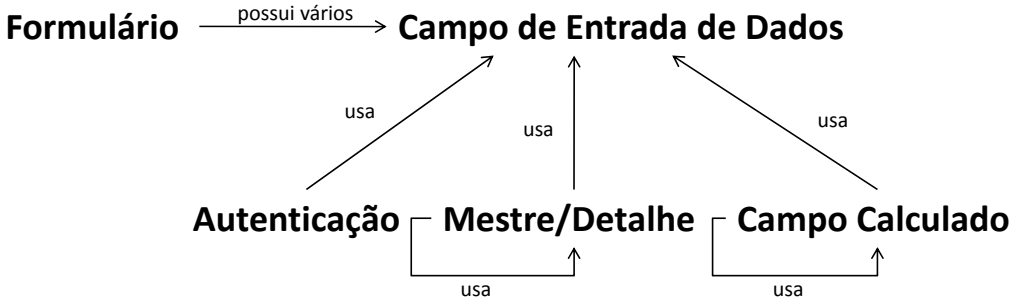


Figura 3.6: Relações existentes entre os vários Padrões

Em primeiro lugar, o Formulário possui vários Campos de Entrada de Dados. Uma vez que os Campos de Entrada de Dados são indispensáveis para que o utilizador introduza informação, estes são a base para o funcionamento dos restantes Padrões. Assim sendo, uma Autenticação usa dois Campos de Entrada de Dados (nome do utilizador e palavra-passe), um Mestre/Detalhe usa dois Campos de Entrada de Dados (Mestre e Detalhe) e um Campo Calculado usa vários Campos de Entrada de Dados, um por cada valor não constante da sua fórmula de cálculo.

Estas relações são as mais elementares para poder definir de uma forma simples alguns Padrões. No entanto, no caso do Mestre/Detalhe e do Campo Calculado, existem relações adicionais que têm que ser tidas em conta.

É frequente existir um encadeamento de Mestre/Detalhe, como pode ser visto na figura 3.7.

Figura 3.7: Relação entre dois Mestre/Detalhe

Neste caso, é possível verificar que Distrito e Concelho constituem um Mestre/Detalhe. Por outro lado, Concelho e Freguesia constituem um segundo Mestre/Detalhe. Nestes casos é importante ter em atenção que o segundo Mestre/Detalhe não pode ser testado de forma independente do primeiro. Assim sendo, caso seja pretendido testar, por exemplo, se o Concelho "Matosinhos" possui uma Freguesia chamada "Lavra", é necessário primeiro seleccionar o Distrito "Porto" para que "Matosinhos" possa ser seleccionado.

O Padrão Campo Calculado pode possuir igualmente relações entre vários Campos Calculados. Na figura 3.8, que representa a estrutura habitual de uma factura, é possível verificar que em cada linha existe um Campo Calculado. Este campo resulta do produto da quantidade pelo preço unitário de cada produto sendo designado "Total Parcial".

Quantidade	Artigo	Preço Unitário	Total Parcial
9	Artigo A	0.5	4.5
2	Artigo B	0.42	0.84
6	Artigo C	0.67	4.02
4	Artigo D	1.24	4.96
		Total	14.32

Figura 3.8: Exemplo de vários Campos Calculados Relacionados

O total da factura resulta do somatório dos vários totais parciais. Uma vez que os vários totais parciais não podem tipicamente ser editados manualmente, por forma a testar o valor do total final, torna-se necessário preencher quantidades e preços unitários de produtos. Este facto leva a que o Campo Calculado correspondente ao total final não possa ser testado independentemente dos restantes totais parciais.

3.3 Execução de Testes

Uma vez analisados os Padrões a serem testados, suas estratégias de teste individuais e relações entre si, torna-se necessário avaliar de que forma estes podem ser usados para testar uma GUI genérica.

O Padrão de Formulário como foi dito anteriormente não possui nenhum teste associado servindo como base para os restantes Padrões. Uma vez que o teste de Mestre/Detalhe tipicamente não requer uma acção de envio do formulário este Padrão pode ser o primeiro a ser testado. O primeiro passo dos testes consiste portanto em avaliar quais as instâncias do Padrão Mestre/Detalhe existentes e quais as restrições que devem ser observadas. Para cada restrição (contém, não contém ou contém apenas) é importante avaliar se existe alguma dependência relativamente a outra instância do Mestre/Detalhe (como foi referido na secção 3.2) e definir o valor necessário para o seu funcionamento.

Tendo em conta que os Campos de Entrada de Dados são a base para os restantes Padrões, o passo seguinte consiste em testar este Padrão. Este teste visa avaliar se o campo causa um erro quando um valor inválido é introduzido. Uma vez que outros Padrões podem ser baseados neste, e dado que estes tipicamente restringem a gama de valores válidos, o caso de sucesso será testado posteriormente. Desde modo, para cada valor inválido de um Campo de Entrada de Dados, esse valor será utilizado no campo e nos restantes campos serão colocados valores válidos para avaliar se o erro é detectado.

O passo seguinte consiste em testar os restantes Padrões. A estratégia é semelhante à utilizada no Campo de Entrada de Dados: introduzem-se valores inválidos num Padrão e válidos nos restantes e verifica-se que o erro é detectado.

Uma vez testados os casos em que os Padrões devem assinalar erro, resta analisar o comportamento em caso de serem introduzidos valores válidos. Para isso, é necessário colocar valores válidos em todos os Padrões e verificar que a aplicação se comporta de acordo com o esperado. No entanto, ao efectuar este passo é necessário ter em conta que a introdução de um valor num Campo de Entrada de Dados depende dos Padrões existentes na GUI. Assim sendo, caso exista um Padrão que tenha por base este Campo, ele apenas deve tomar os valores válidos de acordo com esse Padrão (por exemplo, Autenticação), caso contrário, pode tomar os valores válidos definidos no próprio Campo de Entrada de Dados.

Em cada etapa da execução de testes cada Padrão foi testado individualmente. Uma alternativa a esta estratégia, e que permite detectar erros que se ocultam mutuamente, consiste em testar erros em subconjuntos de campos. Deste modo, seriam introduzidos erros em dois ou mais campos de modo a avaliar se estes são detectados. Esta estratégia leva contudo a um crescimento rápido do número de testes o que conduz a um maior esforço na sua execução. Por exemplo, assumindo a existência de três Campos de Entrada de Dados, cada um com apenas um valor inválido, a estratégia apresentada levaria à execução de três

testes enquanto que esta nova estratégia levaria à execução de sete testes (três testes em que é introduzido erro em apenas um campo, três testes em que são introduzidos erros em pares de campos e um teste em que é introduzido erro em todos os campos).

3.3.1 Exemplo prático

De forma a ilustrar como a execução de testes descrita na secção anterior funciona na prática será usado, como exemplo, o caso de uma Autenticação.

A Autenticação, tal como foi dito anteriormente, possui dois Campos de Entrada de Dados obrigatórios e uma instância do Padrão Autenticação.

Tal como foi dito na secção anterior, o primeiro passo consiste portanto em testar os Campos de Entrada de Dados (uma vez que neste exemplo não existe nenhum Mestre/Detalhe). Dado que cada campo aceita cadeias de caracteres, apenas existe um caso de insucesso que consiste em deixar o campo em branco. Desta forma, existem dois testes a realizar:

- Nome do utilizador em branco e palavra-passe introduzida
- Nome do utilizador introduzido e palavra-passe em branco

Caso fosse seguida a estratégia de combinar vários erros seria necessário realizar um teste adicional: ambos os campos em branco.

Passando à etapa seguinte, é possível verificar que existe apenas um outro Padrão, ou seja, o Padrão Autenticação. Para este Padrão, dois testes devem ser executados para testar dados inválidos:

- Nome do utilizador inválido e qualquer palavra-passe
- Nome do utilizador válido e palavra-passe inválida

Por fim, resta testar o caso em que todos os Padrões possuem valores válidos. Neste caso, uma vez que ambos os Campos de Entrada de Dados fazem parte do Padrão Autenticação, não é necessário inserir valores válidos nestes campos de forma independente visto que o Padrão Autenticação é responsável por definir os valores válidos para o campo. Neste caso, apenas mais um teste deverá ser executado: nome do utilizador e palavra-passe válidos.

3.4 Conclusões

Neste capítulo foram apresentados alguns padrões de comportamento de GUIs como, por exemplo, o Campo de Entrada de Dados ou o Campo Calculado. Cada um destes Padrões possuí um objectivo e é usado num determinado contexto. Isto permite que seja

definida uma estratégia de teste dirigida a cada padrão. A própria forma de responder a uma acção realizada é também normalmente semelhante, sendo encontradas tipicamente mensagens escritas dentro da própria página ou fora desta.

Os Padrões não se encontram no entanto isolados. Existem várias relações entre si. Estas relações podem ser baseadas em composição, quando um Padrão é composto por outros ou dependência, quando um Padrão se encontra dependente de uma acção realizada em outro.

Tendo sido identificadas as principais relações entre os Padrões e o modo de testar cada um, foi identificada uma estratégia de teste generalizada que permite realizar o teste a uma GUI que possua vários Padrões de uma forma automatizada.

No próximo capítulo serão apresentados os principais detalhes relativos à implementação desta abordagem.

Capítulo 4

Aspectos Tecnológicos

A abordagem definida no capítulo anterior encontra algumas dificuldades no momento de a colocar em prática. Existe um conjunto de ferramentas que permite executar acções sobre GUIs. Muitas destas ferramentas são no entanto de *Record/Playback* o que implica que, na maioria dos casos, os testes sejam executados manualmente na fase de *Record* o que reduz a automatização da abordagem. Outras ferramentas como o Jemmy ou Abbot apenas permitem testar GUIs desenvolvidas em Java. Nesse sentido, e uma vez que a *framework* UIAutomation resolve alguns desses problemas, esta foi escolhida para ser usada no desenvolvimento de uma ferramenta que suporte a abordagem do capítulo anterior.

Esta *framework* possui contudo, apesar das suas vantagens, algumas limitações. Neste capítulo será apresentada esta *framework*, as suas características e de que forma foi usada para criar a ferramenta que suporta a referida abordagem e que foi nomeada de PET-Tool[1] bem como a sua arquitectura.

4.1 UIAutomation

A *framework* UIAutomation foi desenvolvida pela Microsoft e está disponível em todos os sistemas operativos que suportem Windows Presentation Foundation. Esta *framework* permite o acesso aos vários elementos existentes no Ambiente de Trabalho do computador (o qual daqui em diante será designado *Desktop*) e permite que sejam realizadas acções sobre eles. Cada um destes elementos é designado por *AutomationElement*. *AutomationElements* podem ser caixas de texto, botões, janelas, caixas de diálogo, etc. Os *AutomationElements* estão estruturados em árvore como apresentado na figura 4.1.

[1] Acrónimo para PattErn Testing Tool

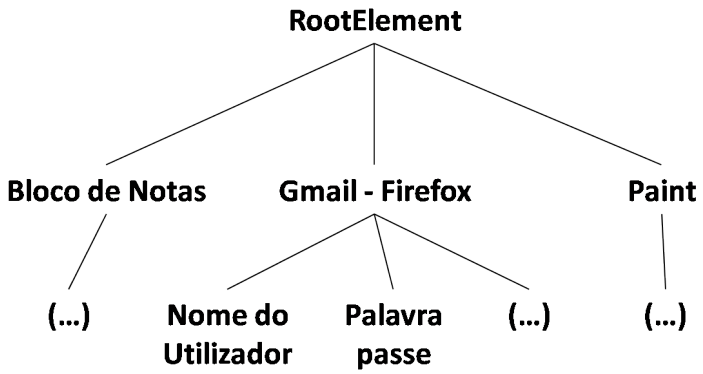


Figura 4.1: Estrutura dos *AutomationElements* no Ambiente de Trabalho

Na raiz da árvore encontra-se um elemento especial: o *RootElement*. Este elemento é único no *Desktop* e dele descendem todos os restantes *AutomationElements*. Cada filho do *RootElement* representa uma janela diferente. Na figura 4.1 encontram-se representadas três janelas. Cada janela possui uma sub-árvore de controlos própria. No caso da janela "Gmail - Firefox" é possível verificar que existe um *AutomationElement* para "Nome do Utilizador" e um para "Palavra-passe", entre outros. Esta estrutura permite, navegando dentro de cada sub-árvore, aceder aos elementos de uma janela específica. Serão apresentadas de seguida algumas vantagens e desvantagens do uso desta *framework*.

4.1.1 Vantagens

Tal como foi visto na secção anterior, cada elemento do *Desktop* é visto como um *AutomationElement*. Desta forma, esta *framework* pode ser utilizada, para testar aplicações .Net, Java, em *browser*, etc. Isto é uma mais-valia uma vez que a mesma ferramenta e estratégia pode ser usada para testar qualquer tipo de aplicações que possuam interface gráfica.

De forma a auxiliar a pesquisa de elementos no *Desktop*, existe a opção de pesquisar filhos e descendentes (filhos, filhos dos filhos, etc.) de um elemento, bem como aceder ao seu pai. A pesquisa pode ser feita usando vários critérios como, por exemplo, *AutomationID*. Um *AutomationID* é um identificador de cada *AutomationElement*. Outros critérios que podem ser utilizados incluem o tipo de elemento (*ControlType*) a pesquisar e o nome (*Name*) do elemento. O caso da pesquisa por nome torna-se bastante útil quando se pretende procurar um determinado texto num formulário, uma vez que a propriedade *Name* de uma *Label* geralmente é o próprio texto da *Label*.

Para além da pesquisa de um elemento por *AutomationID* e por *Name* é possível pesquisar um elemento pela sua posição absoluta no *Desktop*. Deste modo, a selecção de um elemento pode ser feita facilmente recorrendo a um clique do rato. Naturalmente que

se existir uma janela sobreposta à que contém o elemento, o elemento que esta estratégia selecciona é o que se encontra na janela de cima.

Dado que cada elemento do *Desktop* é um objecto do tipo *AutomationElement*, a sua posição dentro da janela não importa, podendo este ser utilizado de forma semelhante, mesmo que mude de posição. Este aspecto permite também que a automatização de certas operações (por exemplo, cliques) possa ser realizada mesmo que exista uma janela em frente à janela onde deverá ser feito o clique.

Por fim, associado a cada *AutomationElement* existem um conjunto de acções pré-definidas chamadas *Patterns*. Estes *Patterns* permitem, por exemplo, simular o clique num botão (*InvokePattern*), alterar o conteúdo de uma caixa de texto (*ValuePattern*) entre outras. Cada *AutomationElement*, dependendo do seu tipo, suporta um ou mais destes *Patterns*. Na secção 4.1.3 serão descritos em melhor detalhe os principais *Patterns* existentes.

4.1.2 Desvantagens

Uma das principais dificuldades no uso da *framework* UIAutomation reside nas referências para os *AutomationElements*. Tal como foi dito, a cada elemento do *Desktop* corresponde um *AutomationElement*. No entanto, para testar as GUIs é necessário realizar acções. Quando estas acções são realizadas em aplicações locais, tipicamente a página não muda em caso de dados inválidos, o que faz com que as referências para os *AutomationElements* continuem a ser válidas. No entanto, quando se realizam acções num *browser*, tipicamente a página é recarregada. Quando isto acontece, todas as referências para *AutomationElements* anteriormente pesquisadas são perdidas, o que torna necessário obter novas referências, mesmo que a página não tenha sofrido alterações.

Uma solução que permite teoricamente facilitar a recuperação das referências consiste em pesquisar o elemento de novo usando o seu *AutomationID* ou *Name*. No entanto, estas duas propriedades não garantem unicidade uma vez que, quer uma quer outra, podem ficar a branco. Nesse sentido, esta estratégia apenas pode ser usada em algumas aplicações. Uma vez que é pretendido com esta abordagem uma solução genérica, esta estratégia não é solução. Na secção 4.2 será descrito de que modo se procurou contornar esta limitação.

Outra desvantagem do uso desta *framework* em *browser* reside na quantidade de informação presente nas páginas. Em aplicações locais, a informação apresentada tende a estar limitada pelo tamanho do ecrã. Nos sítios *Web*, dada a necessidade de apresentar muita informação, opta-se geralmente por permitir que a página possa conter informação para além do tamanho do ecrã. Como consequência, o número de *AutomationElements* presente aumenta o que causa que as pesquisas sejam mais demoradas, aumentando o tempo de execução de várias acções.

Apesar de os tipos de elementos nas interfaces funcionarem de forma similar quer em *browser* quer fora, no caso das *combobox* existem por vezes algumas diferenças. Assumindo que é necessário seleccionar uma *combobox* recorrendo a um clique do rato, no caso de aplicações locais, o clique em qualquer ponto da *combobox* selecciona o elemento correctamente, visto que a *combobox* é apenas constituída por um *AutomationElement* (Figura 4.2, lado esquerdo). No entanto, em *browser*, por vezes a *combobox* é constituída por uma caixa de texto ou *label* e um botão (Figura 4.2, lado direito), apesar de visualmente ser idêntica a uma *combobox* utilizada numa aplicação local. Deste modo, é necessário tomar medidas para que a selecção incida de facto sobre a *combobox* como um todo e não apenas sobre a caixa de texto ou sobre o botão.

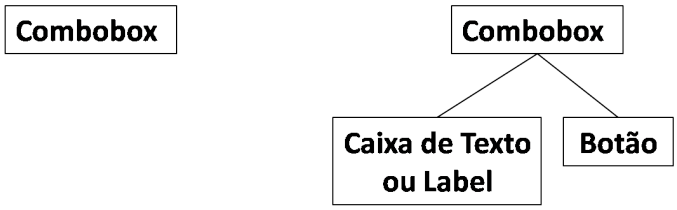


Figura 4.2: Estrutura de duas *combobox*

Na secção anterior foi referida como sendo uma vantagem a existência dos *Patterns*. Contudo, a sua utilização não é aplicável em todos os cenários. Muitas vezes os *AutomationElements* de uma certa GUI simplesmente não permitem a utilização de um determinado *Pattern*. Nesses casos torna-se necessário arranjar uma forma alternativa de realizar a acção, o que por vezes não é trivial.

Na secção 4.2 será descrita de que forma se tentou minimizar os problemas apresentados.

4.1.3 *Patterns*

Tal como foi referido, os *Patterns* são objectos que permitem realizar uma determinada acção sobre um determinado *AutomationElement* que a suporta. É necessário no entanto verificar sempre se um *AutomationElement* suporta o *Pattern* antes de o usar. Segue-se uma lista dos principais *Patterns* necessários para a abordagem referida no capítulo 3.

4.1.3.1 *InvokePattern*

O *InvokePattern* permite invocar um elemento do *Desktop*. Esse elemento não necessita obrigatoriamente de ser um botão dado que outros elementos (por exemplo, uma *Label*) também podem permitir esta acção.

4.1.3.2 *ValuePattern*

O *ValuePattern* permite obter e definir o valor associado a um elemento do *Desktop*. Pode ser usado, por exemplo, para definir o conteúdo de uma caixa de texto.

4.1.3.3 *ExpandCollapsePattern*

As *combobox* e *listbox* podem ser "*drop-down*". Nesses casos, a lista de valores possíveis está escondida (Figura 4.3, lado esquerdo) e para ser vista (Figura 4.3, lado direito) é necessário carregar num botão. O *ExpandCollapsePattern* permite realizar as acções de apresentar e esconder as listas de valores.

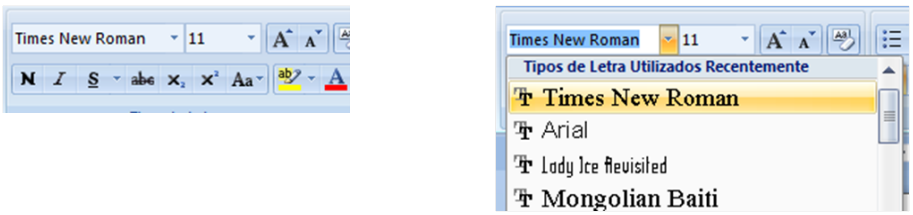


Figura 4.3: Exemplo de uma *combobox* no Microsoft Word 2007

4.1.3.4 *SelectionItemPattern*

O *SelectionItemPattern* selecciona um elemento numa lista de valores, quer esteja numa *listbox* ou numa *combobox*. A selecção é feita quer o elemento esteja visível, quer seja necessário navegar na lista de valores de forma a seleccionar o elemento.

4.1.3.5 *ScrollPattern*

O *ScrollPattern* permite navegar numa lista de valores. Quando uma lista contém mais valores do que aqueles que estão visíveis, este *Pattern* permite descer na lista elemento a elemento ou vários elementos de cada vez.

4.2 *MyAutomationElement*

De forma a tentar minimizar as desvantagens do uso da *framework* UIAutomation foi criada uma classe chamada *MyAutomationElement*. Esta classe visa tentar minimizar os problemas relacionados com as perdas de referências para os *AutomationElements* e permitir que determinadas acções possam ser executadas de forma transparente, quer os *Patterns* estejam disponíveis ou não.

Nesta secção serão apresentados os detalhes mais relevantes relativamente à implementação desta classe.

4.2.1 Caminho para um *AutomationElement*

Anteriormente foi referido que, quando uma acção era realizada, muitas vezes os *AutomationElements* deixavam de estar acessíveis. Quando a aplicação muda de página isto faz sentido mas, quando a aplicação permanece na mesma página, estando exactamente os mesmos elementos no mesmo sítio (ou muito próximos) parece estranho que as referências se percam. Um exemplo de uma acção que faz com que isto aconteça pode ser vista na figura 4.4, onde do lado esquerdo pode ser vista a página de autenticação do GMail e do lado direito o resultado de uma autenticação sem terem sido inseridos dados.

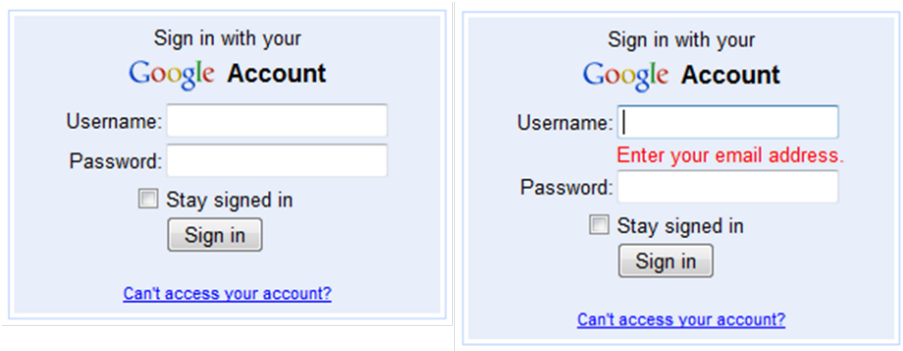


Figura 4.4: GMail antes e depois de uma tentativa de Autenticação com dados inválidos

O facto de os vários elementos da página permanecerem aproximadamente no mesmo local antes e depois da acção leva a crer que estes, na estrutura da árvore de elementos do *Desktop*, se mantém igualmente próximos. Nesse sentido, para cada *AutomationElement* é extraído o caminho dentro da árvore de elementos que lhes permite aceder. Para isso é obtido em cada nó da árvore um conjunto de dados:

- *ControlType* - O tipo de controlo. Este tipo de controlo é um dos principais critérios de identificação do controlo.
- *Name* - O nome do controlo. Permite, quando disponível, ajudar a identificar o controlo desejado.
- Índice - O índice do controlo na lista de filhos do nó pai. Caso o nó pai possua N filhos do mesmo tipo e com o mesmo nome, indica qual deles é mais provável que seja o nó desejado. Este índice é relativo apenas aos filhos com o tipo especificado.

Na secção seguinte será apresentado como poderá ser obtido o caminho para um determinado controlo.

4.2.1.1 Obtenção do Caminho

Na figura 4.5 é possível ver uma estrutura possível da árvore de controlos do *Desktop*.

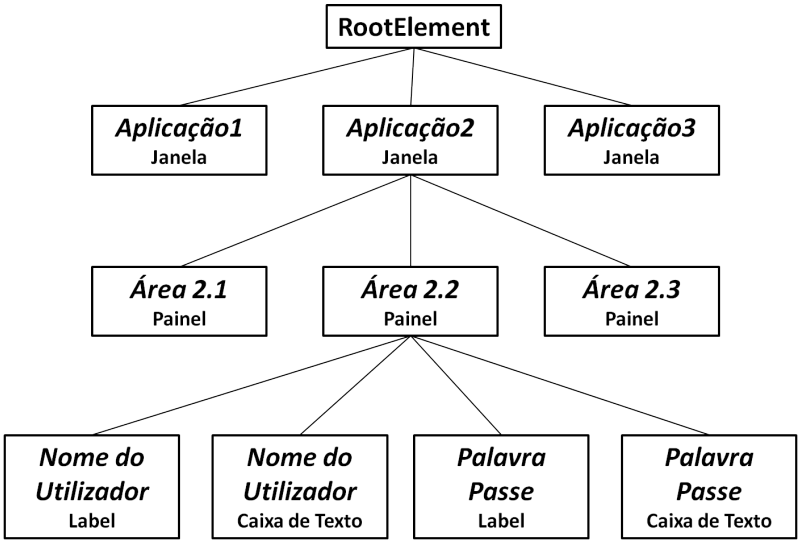


Figura 4.5: Exemplo de uma árvore de controlos no *Desktop*

A tabela 4.1 mostra os vários passos que permitem extrair o caminho para a caixa de texto "Nome do Utilizador" da árvore de elementos da Figura 4.5. O primeiro passo consiste em extrair a informação do nó folha. Dado que o nó folha não possui descendentes este não terá índice do filho. No segundo nó (Painel chamado Área 2.2) o índice registado é 0 uma vez que o "Nome do Utilizador" corresponde à primeira caixa de texto e o índice é avaliado em função do tipo de elemento do nó filho. Nos restantes passos o índice é 1 uma vez que em ambos os casos o nó filho corresponde ao segundo nó do respectivo tipo na lista de descendentes.

Tabela 4.1: Passos para a obtenção do caminho de um elemento da árvore da Figura 4.5

Nó	**Tipo de Controlo**	**Nome**	**Índice**
1	Caixa de Texto	Nome do Utilizador	-
2	Painel	Área 2.2	0
3	Janela	Aplicação 2	1
4	*RootElement*	*RootElement*	1

Com esta sequência de passos é possível obter o caminho até um dado elemento na árvore do *Desktop*. Na secção seguinte será apresentado o algoritmo que permite, a partir do caminho avaliado, obter o *AutomationElement* associado.

4.2.1.2 Obtenção do *AutomationElement* em função do Caminho

De forma a utilizar o caminho avaliado para obter um *AutomationElement*, existe um conjunto de factores que devem ser tidos em conta e que dificultam este processo. Em primeiro lugar existe a possibilidade de um controlo mudar de nome. Isto acontece em

Labels visto que o nome destas é o seu próprio texto. No entanto, isto acontece igualmente, por exemplo, em aplicações como o Bloco de Notas do Windows. O nome do *AutomationElement* que corresponde à janela do Bloco de Notas é tipicamente "Sem título - Bloco de Notas". No entanto, ao abrir um ficheiro, o nome da janela passa a tomar o nome do ficheiro aberto. Este facto torna necessário que o algoritmo de obtenção do elemento permita alguma variação nos nomes dos *AutomationElements*.

No entanto, em alguns casos, esta flexibilidade é um inconveniente. Assumindo que se pretende verificar que um determinado elemento desapareceu da página, esta característica permite que um elemento do mesmo tipo mas com nome diferente possa ser detectado como se fosse o próprio elemento, o que leva a erros.

A possibilidade de detecção de um elemento de forma incorrecta pode igualmente ocorrer se existirem vários elementos do mesmo tipo sem nome associado. Apesar de existir o índice, caso surja um controlo novo (do mesmo tipo dos existentes), o índice indica onde o elemento se encontrava, não sendo ajustado em função dos novos controlos que surjam.

De forma a implementar a funcionalidade de obtenção de um *AutomationElement* em função do seu caminho foi utilizado o algoritmo da Figura 4.6.

```
1  No_Corrente = Primeiro elemento de Caminho
2  Retirar Primeiro elemento de Caminho
3  Proximo_No = Primeiro elemento de Caminho
4  Repetir:
5      Filhos = Filhos de No_Corrente com tipo igual a Proximo_No.Tipo
6      Existe_Nome = Algum elemento de Filhos tem Nome igual a Proximo_No.Nome?
7      Para cada AutomationElement em Filhos, a começar por No_Corrente.Indice
8          Se Proximo_No for o último elemento de Caminho
9              Se AutomationElement.Nome = Proximo_No.Nome ou Existe_Nome for falso
10                 Retornar o AutomationElement
11         Senão, se AutomationElement.Nome = Proximo_No.Nome ou Existe_Nome for falso
12             Retira um elemento do caminho
13             No_Corrente = Proximo_No
14             Proximo_No = Primeiro elemento de Caminho
15             Retornar a (5)
16 Assinalar Erro pois foi impossível encontrar o elemento
```

Figura 4.6: Algoritmo para a obtenção de um elemento através do caminho

Este algoritmo percorre o Caminho ao mesmo tempo que navega nos elementos do *Desktop*. O *No_Corrente*, no inicio, toma o valor do *RootElement*. *Proximo_No* regista as informações do nó para o qual se deve avançar. Na linha 5 são obtidos todos os filhos do nó actual que correspondam ao tipo de *Proximo_No*. Isto permite reduzir o número de nós que se tem que explorar por iteração. A linha 6 permite saber se, entre os nós identificados na linha 5, algum possui um nome igual ao esperado. Esta linha ajuda a avaliar se o elemento mudou de nome. Na linha 7 começa a exploração de todos os nós identificados. No entanto, a exploração começa pelo índice guardado de forma a avaliar

se o nó pretendido manteve a sua posição na árvore. As linhas 8 a 10 dizem respeito ao fim das iterações. Caso tenha sido atingido o fim do caminho e o nome é igual ao esperado ou não exista nenhum nó com esse nome, o algoritmo termina, devolvendo o *AutomationElement* identificado. Caso contrário, o algoritmo avança na árvore, retirando um elemento do Caminho e repete o processo. Caso nenhum *AutomationElement* tenha sido identificado é reportado um erro pelo algoritmo.

4.2.2 *Patterns*

Tal como foi referido na secção 4.1.2, os *Patterns* nem sempre estão disponíveis. Este facto leva a que se tenha que encontrar uma abordagem alternativa para realizar as acções.

A classe *MyAutomationElement* possui um conjunto de funcionalidades que correspondem a acções típicas dos *Patterns*. Estas funcionalidades tentam, sempre que possível, utilizar os *Patterns* mas caso estes não estejam disponíveis possuem uma estratégia alternativa.

Nesta secção será descrita de que forma estas estratégias foram implementadas.

4.2.2.1 Realizar uma invocação

Na secção 4.1.3.1 foi referida a utilização do *InvokePattern* para realizar invocações.

Quando o *InvokePattern* não se encontra disponível é necessário em primeiro lugar saber onde se encontra o elemento que se pretende invocar. A classe *AutomationElement* possui um método chamado *TryGetClickablePoint*. Este método, em caso de sucesso, permite obter um ponto no ecrã onde o se pode realizar um clique de modo a realizar a acção de invocar o controlo. No entanto, este método possui uma limitação relativamente ao uso do *InvokePattern*: caso uma janela se encontre à frente do elemento, a função *TryGetClickablePoint* não retorna nenhum ponto, o que impossibilita a realização da acção.

Nos casos em que a função *TryGetClickablePoint* retorna um ponto, é necessário mover o rato e efectuar um clique. Estas acções têm que ser realizadas recorrendo a bibliotecas do sistema operativo.

4.2.2.2 Introduzir valor em Caixas de Texto

O *ValuePattern* é o *Pattern* geralmente utilizado para consultar e alterar o texto em caixas de texto.

Quando este *Pattern* não se encontra disponível torna-se necessário simular eventos do teclado para a introdução do texto. No entanto, de forma a estes eventos produzirem o resultado esperado, é necessário verificar se o controlo possui o foco. A classe *AutomationElement* possui três métodos que podem ser usados para este fim. O primeiro método chama-se *IsKeyboardFocusable* e permite saber se é possível colocar o foco no controlo

pretendido. Caso seja possível, pode-se invocar o método *SetFocus* que fornece o foco ao controlo. De seguida, chama-se o método *HasKeyboardFocus* para verificar se o elemento de facto possui o foco. O último passo consiste em apagar o conteúdo da caixa de texto e inserir o valor pretendido recorrendo, para isso, a lançamento de eventos do teclado.

4.2.2.3 Seleccionar um elemento de uma *Combobox/Listbox*

Para seleccionar um elemento de uma *combobox* ou *listbox*, existe uma sequência de passos que devem ser seguidos.

Em primeiro lugar é necessário obter a lista de elementos sendo que esta lista pode estar escondida. Em alguns casos, a lista pode ser obtida directamente navegando na estrutura da árvore de elementos do *Desktop*. No entanto, por vezes, é necessário simular *inputs* do utilizador. Nesses casos, de forma a aceder à lista, é necessário clicar no botão que a permite exibir como foi apresentado na Figura 4.3 na secção 4.1.3.3.

O segundo passo consiste em seleccionar o elemento pretendido. Obtendo a lista é simples navegar pelo seu conteúdo e identificar o *AutomationElement* que se pretende seleccionar. Caso a *combobox* ou *listbox* permita usar o *ExpandCollapsePattern* e o elemento permita utilizar o *SelectionItemPattern*, a selecção pode ser feita directamente realizando uma acção de *Expand* (*ExpandCollapsePattern*) seguida de uma acção de *Select* (*SelectionItemPattern*). Caso o *SelectionItemPattern* não esteja disponível, é necessário recorrer ao método *TryGetClickablePoint* do elemento, mover o rato e simular o clique. No entanto, caso o elemento não esteja visível, a função *TryGetClickablePoint* falha. Nesse caso, é necessário usar o *ScrollPattern* para subir/descer a lista até ao elemento pretendido estar visível. Caso o *ScrollPattern* não esteja disponível, é necessário, mais uma vez, simular eventos do teclado para subir/descer a lista até que o elemento fique visível.

4.3 PETTool

De forma a verificar a validade da abordagem mencionada no capítulo 3 foi desenvolvida a ferramenta PETTool. Esta ferramenta permite identificar alguns Padrões de comportamento de GUIs, nomeadamente o Formulário, Campo de Entrada de Dados, Campo Calculado, Mestre/Detalhe e Autenticação e para eles gera e executa testes de forma automática. A arquitectura geral da ferramenta pode ser vista na Figura 4.7.

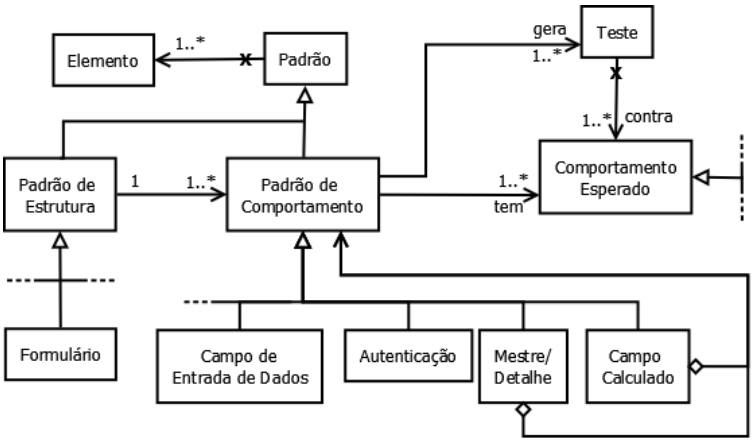


Figura 4.7: Arquitectura da ferramenta PETTool

Os Padrões podem ser de dois tipos: Estrutura e Comportamento. Todos os Padrões possuem elementos associados (por exemplo caixas de texto). Os Padrões de Estrutura possuem ainda a si associados um conjunto de Padrões de Comportamento uma vez que os Padrões de Comportamento se encontram contidos em Padrões de Estrutura, definindo o comportamento destes últimos. Os Padrões de Comportamento podem ser simples ou composição de várias instâncias. Cada Padrão de Comportamento gera um conjunto de testes. Este conjunto é gerado em função das estratégias definidas no capítulo 3 e é testado numa GUI. O resultado da execução dos testes é comparado com um Comportamento Esperado. Este comportamento pode ser uma mensagem dentro ou fora dá página, saída ou permanência na página bem como a inactivação de um botão.

4.3.1 Funcionamento da Ferramenta

Para configurar a ferramenta PETTool é necessário em primeiro lugar identificar o formulário a ser testado e qual o botão que valida os dados. Para isso, o utilizador deve carregar em "*Select*" na PETTool (Figura 4.8) e de seguida no Formulário e depois fazer o mesmo para o botão ("*Submit Button*").

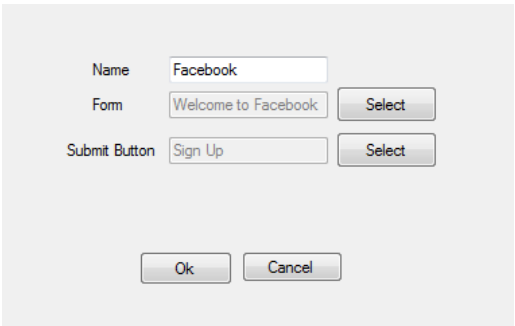


Figura 4.8: Página que permite identificar o Formulário a ser testado

Após identificar o formulário é necessário identificar os Campos de Entrada de Dados nele contidos. Estes Campos serão usados posteriormente na definição do comportamento dos restantes Padrões. Para cada Campo de Entrada de Dados é necessário definir, recorrendo a um conjunto de opções pré-definidas, qual o comportamento que o campo deverá ter. Estas opções são comuns a vários Padrões. É ainda necessário indicar alguns dados de entrada válidos e inválidos associados a este Padrão.

Após definir os Campos de Entrada de Dados, os restantes Padrões podem ser definidos. Para a Autenticação é necessário definir os Campos associados ao nome do utilizador e palavra-passe, bem como fornecer dados de autenticação válidos e inválidos. Para o Mestre/Detalhe, é necessário definir, além de quais os Campos associados ao Mestre e ao Detalhe, quais as restrições a que estes devem obedecer. Para o Campo Calculado, além de definir qual o campo onde deverá ser apresentado o resultado, é necessário definir qual a fórmula de cálculo. Para isso a interface dispõe de um conjunto de botões que permite definir a fórmula.

Após este processo manual de configuração dos Padrões, basta carregar no botão *Run Light Tests* ou *Run Exhaustive Tests* e a PETTool encarrega-se de gerar e executar os testes. Durante esta fase, a ferramenta verifica a concordância entre os resultados obtidos e o comportamento esperado definido e apresenta um relatório dos testes efectuados.

A interface que permite realizar estas acções pode ser vista na figura 4.9

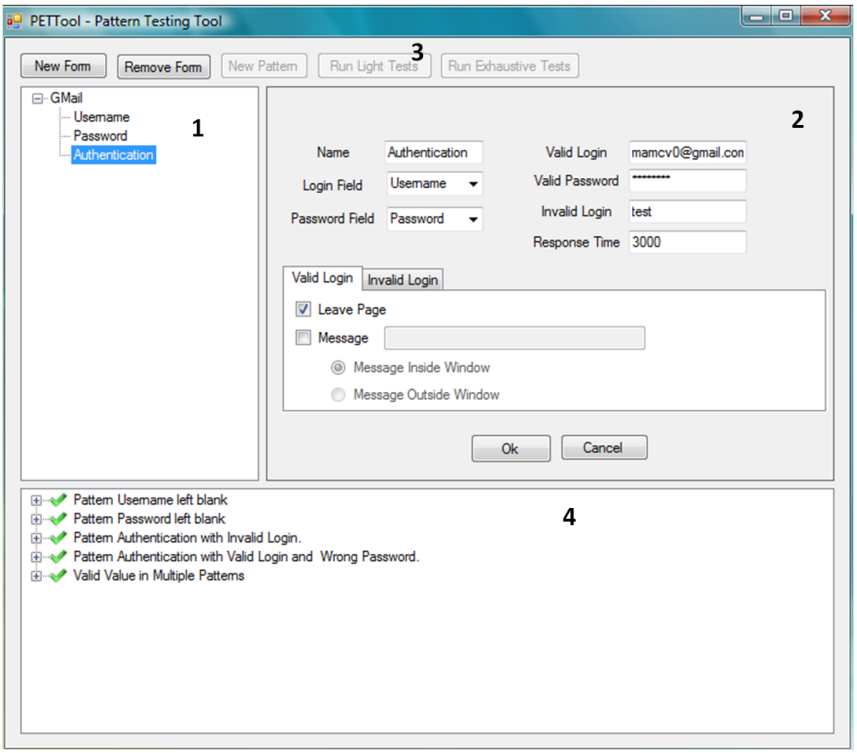


Figura 4.9: Interface da PETTool

Do lado esquerdo da interface (1) encontram-se os Formulários configurados e quais os Padrões associados a cada um. No painel assinalado com (2) encontra-se a configuração do Padrão seleccionado, neste caso, uma autenticação. Para gerar e executar os testes, basta efectuar um clique no botão assinalado por (3) sendo que os resultados surgem no painel (4).

4.4 Conclusões

Neste capítulo foram apresentados os principais detalhes de implementação relativos à abordagem do capítulo 3.

De forma a criar uma ferramenta que pudesse ser igualmente utilizada em qualquer tipo de aplicação foi utilizada a *framework* UIAutomation. Esta *framework* possui várias vantagens mas, no entanto, possui igualmente algumas limitações. Estas limitações tornaram necessária a criação de uma abstracção que permite esconder alguns detalhes de implementação relativos, por exemplo, à simulação da acção de carregar num botão.

Recorrendo à *framework* UIAutomation foi então possível criar uma ferramenta designada PETTool que gera e executa testes para vários Padrões recorrendo à estratégia do capítulo 3.

No capítulo seguinte será apresentado um conjunto de casos de estudo que permitem avaliar a validade da abordagem definida.

Capítulo 5

Casos de Estudo

Neste capítulo será analisado um conjunto de Casos de Estudo. Estes Casos de Estudo têm como objectivo analisar, recorrendo à ferramenta PETTool, a validade da abordagem proposta no capítulo 3.

Para levar a cabo o objectivo proposto, os Casos de Estudo serão separados em dois grupos. No primeiro grupo, será verificado se a abordagem permite especificar o comportamento esperado de um conjunto de sítios *Web*, se consegue gerar e executar testes e verificar o seu correcto funcionamento. No segundo grupo serão introduzidos alguns erros típicos em várias aplicações e sítios *Web* para avaliar se a abordagem os consegue detectar.

Todos os sítios *Web* ilustrados neste capítulo foram acedidos em Junho de 2010.

5.1 Verificação de Comportamento

De forma a avaliar se a ferramenta PETTool permite verificar o comportamento de sítios *Web* foram escolhidos cinco exemplos. Cada um destes exemplos possui um conjunto de Padrões e Comportamentos específicos. Nesse sentido, para cada exemplo será apresentado o contexto em que se insere, o seu objectivo, quais os Padrões nele existentes bem como o comportamento esperado em vários cenários de execução.

5.1.1 Formulário de Registo no Facebook

O Facebook[1] é uma das mais famosas redes sociais actualmente existente. Tal como muitas outras redes sociais, possui um Formulário de registo que tem como objectivo recolher alguma informação pessoal do utilizador. Este Formulário possui um conjunto de campos sendo todos eles obrigatórios. Alguns dos campos presentes consistem em caixas de texto (por exemplo, a *Password*) enquanto outros consistem em *combobox* (por exemplo, o Sexo do utilizador).

[1] www.facebook.com

5.1.1.1 Padrões Existentes

Na figura 5.1 é possível ver o aspecto do Formulário de Registo no Facebook. Este Formulário, para além do Padrão Formulário, possui oito instancias do Padrão Campo de Entrada de Dados, nomeadamente:

1. *First Name* (Nome)
2. *Last Name* (Sobrenome)
3. *Your Email* (Endereço de e-mail)
4. *New Password* (Palavra-passe)
5. *I am* (Sexo)
6. *Day* (Dia da Data de Nascimento)
7. *Month* (Mês da Data de Nascimento)
8. *Year* (Ano da Data de Nascimento)

Figura 5.1: Formulário de Registo no Facebook

5.1.1.2 Testes e Comportamento Esperado

Todos os campos presentes no Formulário são obrigatórios. Este facto leva a que para cada um deles exista um caso de insucesso que consiste em deixar o campo em branco (no caso das caixas de texto), ou sem nenhum valor seleccionado (nas *combobox*). Relativamente aos Campos *First Name*, *Last Name* e *Password*, uma vez que são Campos que aceitam qualquer cadeia de caracteres, não existe nenhum teste adicional em caso de insucesso. No caso do Campo *Your Email*, existe um caso de insucesso que corresponde à introdução de um endereço de e-mail inválido. Os restantes Campos, por terem os seus

valores limitados aos presentes nas respectivas *combobox*, não possuem nenhum caso de insucesso adicional.

A resposta em caso de insucesso é semelhante em todos os casos e consiste numa mensagem dentro da página. No entanto, a mensagem varia consoante o erro verificado. Caso todos os campos possuam um valor válido, é feita uma transição para uma nova página. Na tabela 5.1 pode ser vista a lista de Campos existentes, exemplos de valores válidos e inválidos e o resultado esperado em cada caso.

Tabela 5.1: Campos e Comportamento esperado do Formulário de Registo do Facebook

Campo	**Valor**	**Comportamento Esperado**
First Name	"Maria"	Sair da Página
	Campo em Branco	Mensagem "*You must provide your full name.*" dentro da página
Last Name	"Silva"	Sair da Página
	Campo em Branco	Mensagem "*You must provide your full name.*" dentro da página
Your Email	E-mail Válido	Sair da Página
	"Teste"	Mensagem "*Please enter a valid email address.*" dentro da página
	Campo em Branco	Mensagem "*You must fill in all of the fields.*" dentro da página
New Password	"qwerty"	Sair da Página
	Campo em Branco	Mensagem "*You must fill in all of the fields.*" dentro da página
I am	"*Female*"	Sair da Página
	Nenhum valor escolhido	Mensagem "*Please select either Male or Female.*" dentro da página
Day	"15"	Sair da Página
	Nenhum valor escolhido	Mensagem "*You must indicate your full date of birth to register.*" dentro da página
Month	"*September*"	Sair da Página
	Nenhum valor escolhido	Mensagem "*You must indicate your full date of birth to register.*" dentro da página
Year	"1985"	Sair da Página
	Nenhum valor escolhido	Mensagem "*You must indicate your full date of birth to register.*" dentro da página

5.1.1.3 Configuração dos Testes e Resultados Obtidos

Após se ter identificado o Formulário a testar e o botão que valida os dados, tal como foi visto na secção 4.3.1, é necessário indicar, para cada Padrão, qual o seu comportamento. Nas Figuras 5.2 e 5.3 é possível ver o modo de configuração do comportamento mencionado na secção anterior para o Campo de Entrada de Dados *First Name*.

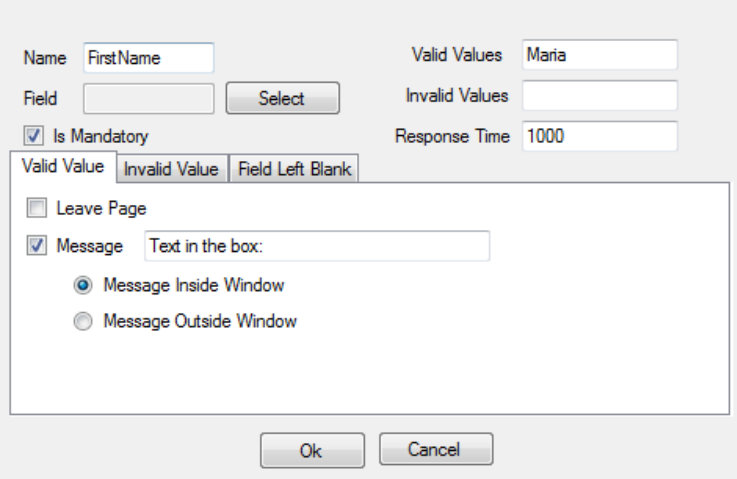


Figura 5.2: Comportamento esperado com um valor válido no campo *First Name*

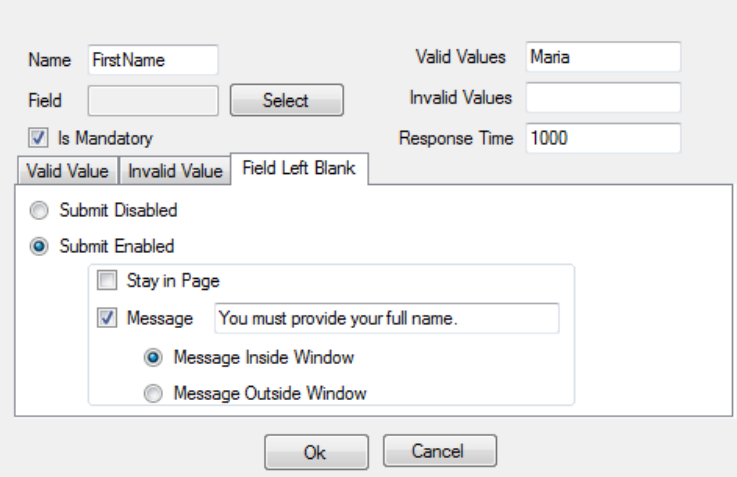


Figura 5.3: Comportamento esperado com o Campo *First Name* em branco

Cada Padrão possui um campo chamado "*Response Time*"(tempo de resposta). Este campo permite definir, em milissegundos, quanto tempo se espera que a aplicação demore a responder à acção. Desta forma, para aplicações mais lentas pode-se definir um tempo de resposta maior, procurando deste modo evitar a indicação de erros por a aplicação demorar mais tempo a responder do que previsto

No caso dos Campos que têm por base uma *combobox*, a configuração é ligeiramente diferente. Nestes casos, apesar de o Campo ser obrigatório, é impossível colocar o seu valor a branco. Assim sendo, deve ser identificado um valor inválido correspondente ao valor seleccionado por defeito. Esse valor é utilizado para determinar se o utilizador colocou no Campo algum valor. Na Figura 5.4 é possível ver um exemplo da configuração descrita.

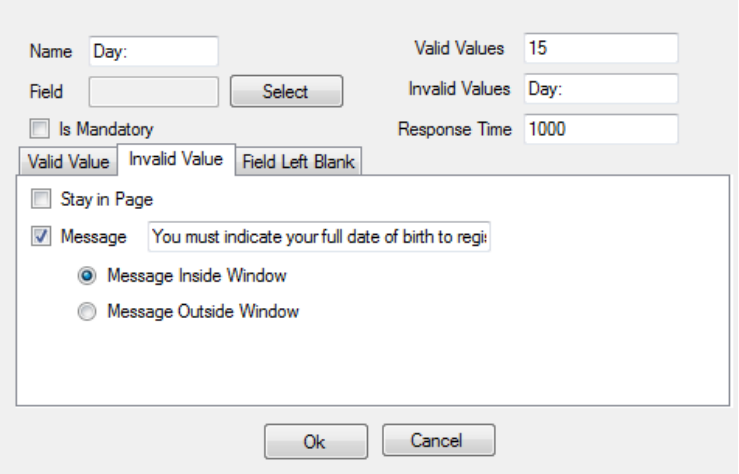


Figura 5.4: Comportamento esperado com um valor inválido no Campo *Day*

Após terem sido configurados todos os Padrões, os testes podem ser executados. O relatório obtido pode ser visto na Figura 5.5. Como era esperado foram executados dez testes. Cada campo foi testado com um valor inválido, excepto o *Your Email* que teve um teste adicional. O último teste diz respeito à colocação de valores válidos em todos os campos que corresponde ao sucesso da operação de registo.

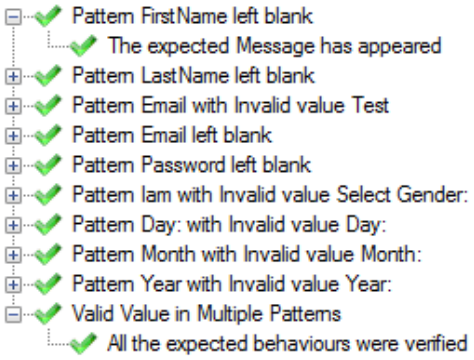


Figura 5.5: Resultados dos testes ao Formulário de registo do Facebook

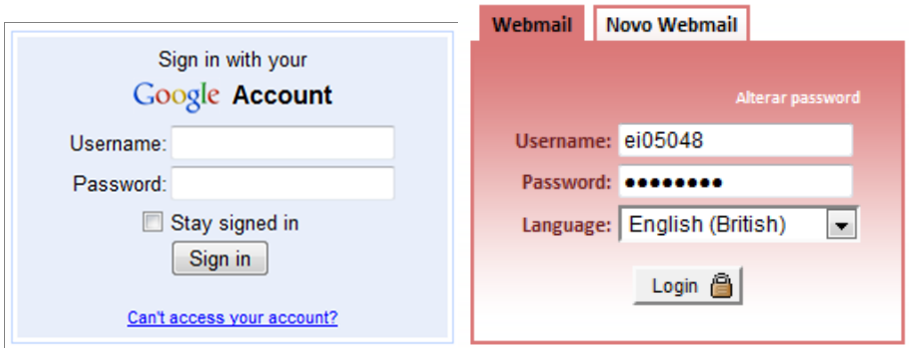
5.1.2 Clientes de e-mail

O presente caso de estudo incide sobre a Autenticação de dois clientes de e-mail: GMail[2] e o Webmail[3] usado na FEUP[4]. A Autenticação tem por objectivo impedir que terceiros acedam a uma conta que não lhes pertence. Em ambos os casos, a Autenticação é constituída por dois campos: Nome do Utilizador e Palavra-passe.

5.1.2.1 Padrões Existentes

Na figura 5.6 é possível ver o Formulário dos clientes de e-mail referidos. Em ambos os Formulários existem os dois campos mencionados acima. Deste modo, cada um destes exemplos possui os seguintes Padrões:

1. Campo de Entrada de Dados *Username* (Nome do Utilizador)
2. Campo de Entrada de Dados *Password* (Palavra-passe)
3. Autenticação



(a) GMail (b) Webmail da FEUP

Figura 5.6: Formulários de Autenticação de dois clientes de e-mail

5.1.2.2 Testes e Comportamento Esperado

Os Campos de Entrada de Dados presentes em ambos os Formulários de Autenticação são obrigatórios. Nesse sentido, é necessário testar estes campos com valores em branco. Uma vez que estes campos aceitam qualquer cadeia de caracteres, este é o único caso de insucesso a fazer em cada Campo de forma individual.

Os restantes casos de insucesso dizem respeito ao Padrão Autenticação. Neste Padrão devem ser testados os cenários em que é colocado um nome do utilizador válido com uma palavra-passe inválida, e em que é colocado o nome do utilizador inválido.

[2] http://www.gmail.com
[3] http://webmail.fe.up.pt
[4] www.fe.up.pt

Cada um dos clientes de e-mail apresentados possui um formato de resposta semelhante, no entanto, existem pequenas variações. Nas tabelas 5.2 e 5.3 serão apresentados os comportamentos esperados pelo GMail e pelo Webmail da FEUP respectivamente. Em ambos os casos, a introdução de dados correctos leva a que a aplicação mude de página.

Tabela 5.2: Comportamento esperado do Formulário de Autenticação do GMail

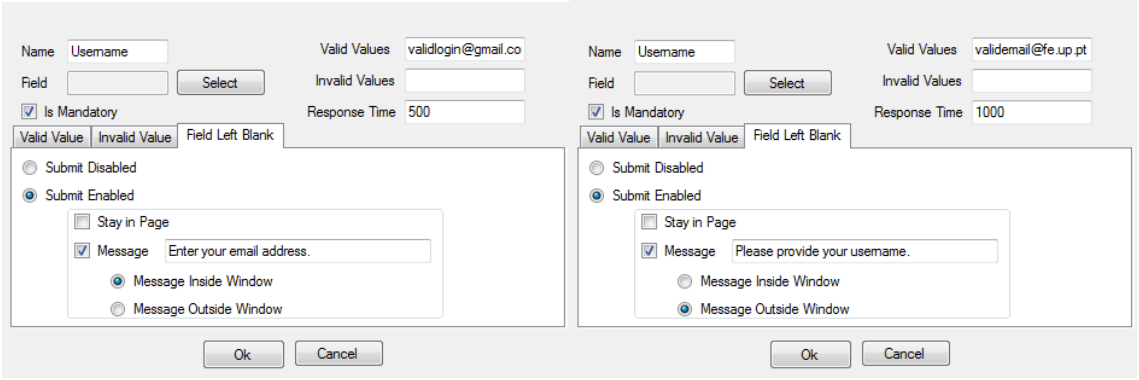
Erro	**Comportamento Esperado**
Nome do Utilizador em Branco	Mensagem "*Enter your email address.*" *dentro da página*
Palavra-passe em Branco	Mensagem "*Enter your password.*" *dentro da página*
Nome do Utilizador inválido ou Palavra-passe Inválida	Mensagem "*The username or password you entered is incorrect.*" dentro da página

Tabela 5.3: Comportamento esperado do Formulário de Autenticação do Webmail da FEUP

Erro	**Comportamento Esperado**
Nome do Utilizador em Branco	Mensagem "*Please provide your username.*" fora da página
Palavra-passe em Branco	Mensagem "*Please provide your password*" fora da página
Nome do Utilizador inválido ou Palavra-passe Inválida	Mensagem "*Login failed because your username or password was entered incorrectly.*" dentro da página

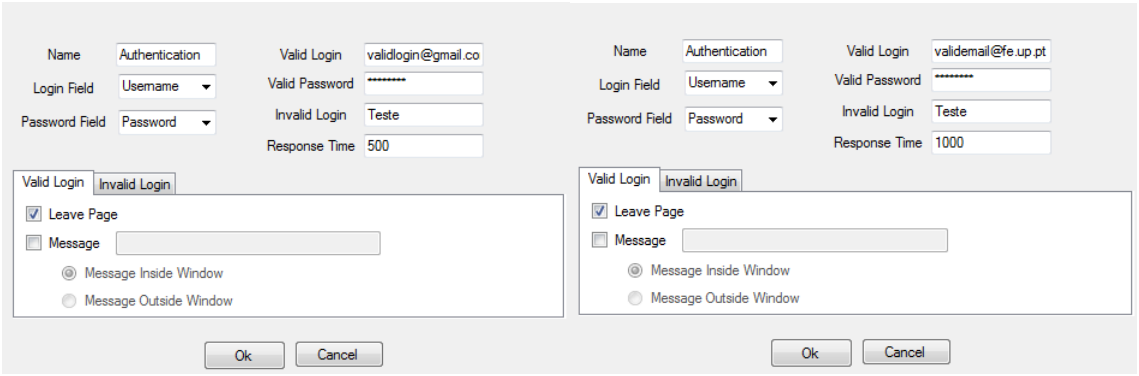
5.1.2.3 Configuração dos Testes e Resultados Obtidos

Após ter sido escolhido o Formulário e o botão que valida os dados, o utilizador tem que configurar os Padrões. Nas Figuras 5.7, 5.8 e 5.9 é possível ver como, apesar de existirem algumas diferenças entre os comportamentos esperados dos dois clientes de e-mail, a sua configuração é muito semelhante. Deste modo, torna-se simples adaptar uma configuração existente para poder ser usada num novo cenário de teste. O caso de sucesso relativo aos Campos de Entrada de Dados não é configurado uma vez que o seu sucesso depende do sucesso do Padrão Autenticação. Neste Padrão é definida, recorrendo a uma *combobox*, qual é o Padrão Campo de Entrada de Dados que corresponde ao Nome do Utilizador e à Palavra-passe. Deste modo a relação entre os vários Padrões fica estabelecida.



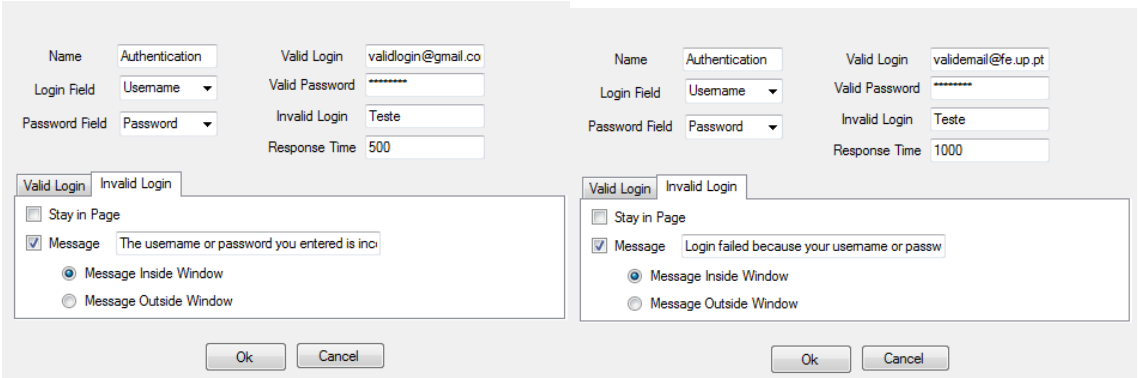
(a) GMail (b) Webmail da FEUP

Figura 5.7: Configuração do comportamento dos clientes de e-mail caso o Campo *Username* não seja preenchido.



(a) GMail (b) Webmail da FEUP

Figura 5.8: Configuração do comportamento dos clientes de e-mail relativo a uma tentativa de autenticação válida



(a) GMail (b) Webmail da FEUP

Figura 5.9: Configuração do comportamento dos clientes de e-mail relativo a uma tentativa de autenticação inválida.

As configurações apresentadas anteriormente conduzem à execução de cinco testes: dois relativos aos Campos de Entrada de Dados em que estes ficam vazios, dois relativos à Autenticação em que são introduzidos dados inválidos e um último em que dados válidos são inseridos. Na Figura 5.10 é possível ver o resultado dos testes executados que é comum a ambos os clientes de e-mail.

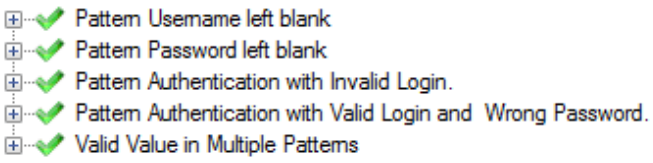


Figura 5.10: Resultados dos testes à Autenticação dos clientes de e-mail

5.1.3 ERA Imobiliária

A agência ERA[5] é uma agência imobiliária portuguesa. Esta agência vende imóveis por todo o país o que implica que necessite no seu sítio *Web* de possuir um mecanismo que permita aos seus clientes filtrar apenas os imóveis da região que pretendem. Efectuar uma selecção ao nível da freguesia numa mesma lista poderia ser complicado dada a quantidade de freguesias existentes em Portugal. Nesse sentido, o sitio *Web* está dotado de três *combobox*:

1. Distrito
2. Concelho
3. Freguesia

Cada *combobox* limita o conteúdo da seguinte. Assim sendo, se em Distrito for escolhido "Porto", na *combobox* Concelho apenas serão listados os concelhos do Porto. É este mecanismo que será testado neste caso de estudo.

5.1.3.1 Padrões Existentes

Na figura 5.11 é possível ver o formulário de pesquisa de imóveis do sítio *Web* da agência ERA. Nele existem, vários campos mas, uma vez que apenas se pretende testar a relação entre as várias *combobox*, apenas serão usados os Campos de Entrada de Dados Distrito, Concelho e Freguesia.

[5] http://era.pt/pg_home.aspx?lang=pt

Figura 5.11: Página de pesquisa de imóveis no sítio *Web* da agência ERA

Para além dos três Padrões mencionados existem ainda duas instâncias do Padrão Mestre/Detalhe. Estas instancias dizem respeito à relação entre Distrito e Concelho e entre Concelho e Freguesia.

5.1.3.2 Testes e Comportamento Esperado

Os Campos de Entrada de Dados por si só não possuem nenhum comportamento esperado no âmbito deste caso de estudo. Os testes incidem portanto sobre os Padrões Mestre/Detalhe.

Foi identificado, na tabela 5.4, um conjunto de restrições a que o conteúdo das *combobox* deve obedecer. No Detalhe da restrição relativa à Ilha do Corvo surge como um dos valores "- Seleccione -". Este valor encontra-se presente uma vez que é indistinguível do ponto de vista dos elementos da *combobox* dos restantes elementos. Assim sendo, para utilizar uma restrição "Contém Apenas" este tem que ser indicado como se fosse um elemento.

Tabela 5.4: Restrições dos Mestre/Detalhe no sitio *Web* da agência ERA

Campos	Valor do Mestre	Restrição	Valor do Detalhe
Distrito-Concelho	Porto	Contém	Matosinhos, Maia
	Lisboa	Não Contém	Porto
	Ilha do Corvo	Contém Apenas	"- Seleccione -" ; Corvo
Concelho-Freguesia	Matosinhos	Contém	Lavra
	Corvo	Não Contém	Lavra

5.1.3.3 Configuração dos Testes e Resultados Obtidos

Dado que este caso de estudo apenas incide sobre o Padrão Mestre/Detalhe, o botão de validação de dados não é utilizado neste contexto. Nesse sentido, na página de escolha de formulário pode ser seleccionado um botão qualquer. A identificação dos Campos de Entrada de Dados também não necessita de definição de nenhum comportamento específico uma vez que o comportamento é definido pelo Padrão Mestre/Detalhe. Deste modo, os únicos Padrões que necessitam de uma definição do seu comportamento são os Mestre/Detalhe Distrito-Concelho e Concelho-Freguesia. Nas Figuras 5.12 e 5.13 pode ser vista parte da configuração dos Padrões identificados mediante os testes definidos na secção anterior.

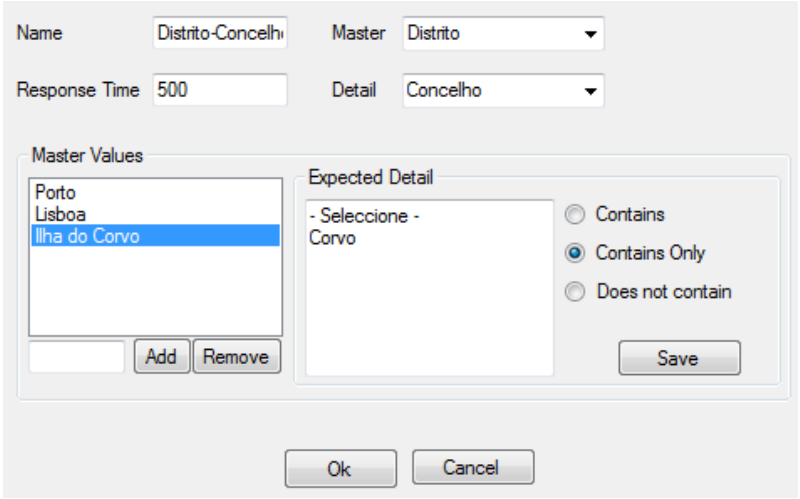


Figura 5.12: Configuração do Padrão Mestre/Detalhe Distrito-Concelho

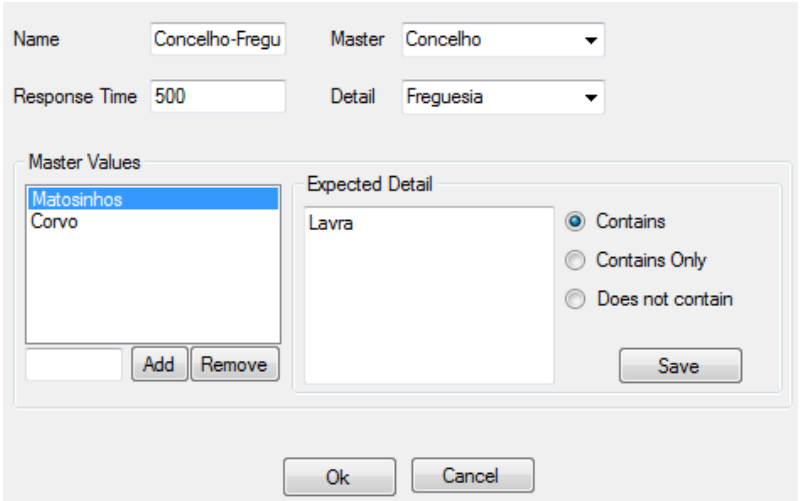


Figura 5.13: Configuração do Padrão Mestre/Detalhe Concelho-Freguesia

As configurações apresentadas levam à execução dos testes descritos anteriormente e cujo resultado está presente na Figura 5.14.

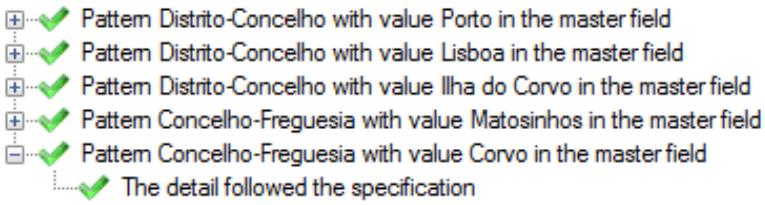


Figura 5.14: Resultados dos testes à Pesquisa de Imóveis do sitio *Web* da Agência ERA

5.1.4 Calculador de Rendimentos

Existem vários sítios *Web* que permitem efectuar cálculos sobre os rendimentos obtidos. Estes sítios tem por objectivo ajudar o utilizador a avaliar quais as suas fontes de rendimento e quais as suas despesas mensais. Isto permite indicar qual será a margem de manobra que o utilizador terá a nível de disponibilidade monetária ao longo do mês. Um destes sítios é `http://www.bygpub.com/finance/CashFlowCalc.htm`. Neste sítio é possível indicar duas fontes de rendimentos (regular e outro) e vinte tipos de despesa (gás, electricidade, gasolina, etc.). Com os valores inseridos a aplicação calcula o total de ganhos, o total de despesas e a diferença entre estes dois últimos valores.

5.1.4.1 Padrões Existentes

Neste sítio *Web*, para a operação descrita acima existem vários Campos de Entrada de Dados, mais concretamente vinte e dois. Ao fundo da página existem três Campos Calculados que correspondem ao cálculo efectuado nos restantes campos. De forma a simplificar o caso de estudo vão ser utilizados apenas dois campos relativos a fontes de rendimentos e dois campos relativos a despesas. Estes campos podem ser vistos na Figura 5.15 que representa parte dos Campos existentes no sítio *Web*.

Income	
Regular Income - Enter you normal monthly income here. If you are married add your income and your spouses.	0
Other Income - Enter any extra monthly income you might make from a night job, royalties, etc.	0
Expenses	
Rent or Mortgage - Enter your monthly rent or mortgage bill here.	0
Car Payment - If you have a car loan(s), enter the total monthly payment(s) here.	0
Click this button to calculate the total.	Calculate
Total Income - This is your total income per month	0
Total Expenses - This is the total amount of money you spend per month on expenses.	0
Loss or gain - This number is total income minus total expenses, and indicates your loss or gain each month. A positive number indicates that you make more than you spend and therefore are able to save money each month. A negative number indicates that you spend more than you make and are therefore borrowing money each month.	0

Figura 5.15: Parte da interface do sítio *Web* do calculador de rendimentos

A relação entre os campos pode ser vista na Figura 5.16.

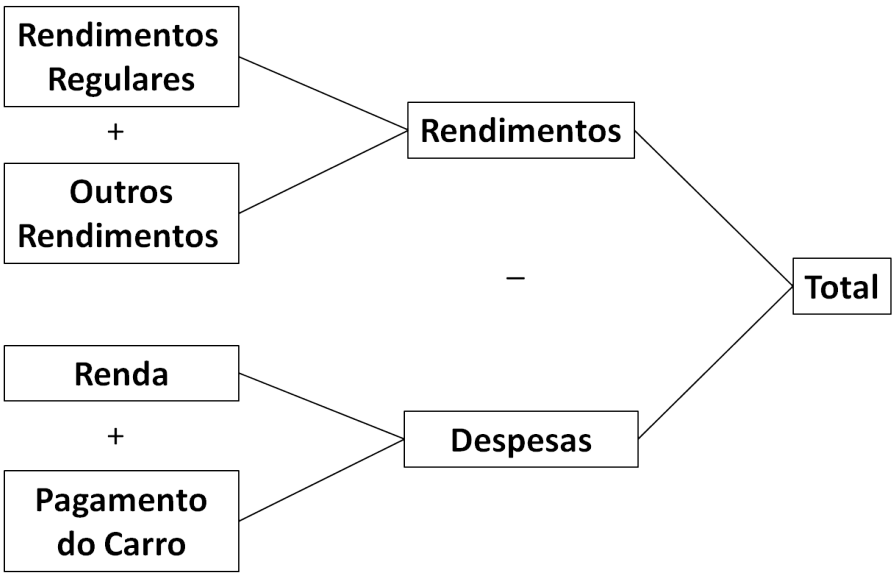


Figura 5.16: Relações entre os campos existentes no formulário do calculador de rendimentos

5.1.4.2 Testes e Comportamento Esperado

Todos os Campos de Entrada de Dados são obrigatórios. Uma vez que apenas será utilizado um subconjunto dos Campos existentes é importante verificar se os restantes possuem por defeito um valor válido. Dado que todos os Campos têm por defeito o valor zero, não é necessário tomar medidas neste aspecto. Os Campos de Entrada de Dados a serem testados requerem valores numéricos. Deste modo existem três valores que devem ser testados em cada campo:

- Valor numérico válido (caso de sucesso)
- Campo em branco (caso de insucesso)
- Campo com caracteres não numéricos (caso de insucesso)

Cada um dos casos de insucesso gera uma mensagem dentro da página com o texto "*NaN*". Assim sendo serão executados oito testes, dois por cada Campo de Entrada de Dados

Para testar os Campos Calculados só irá ser executado um teste. Isso acontece porque os Campos calculados estão dependentes entre si. Assim sendo, o Campo Total irá ser calculado em função de Rendimentos e Despesas. Cada um destes Campos só tem um cenário de teste que corresponde ao valor numérico válido em cada um dos Campos de Entrada de dados associados. Caso esses Campos possuíssem múltiplos valores válidos, seria executada uma combinação desses valores.

5.1.4.3 Configuração dos Testes e Resultados Obtidos

De forma a configurar a PETTool para testar os Padrões mencionados acima é necessário identificar em primeiro lugar o Formulário e o botão de cálculo seguindo o mesmo método identificado nos casos de estudo anteriores. De seguida é necessário identificar os vários Campos de Entrada de Dados. Na Figura 5.17 é possível ver uma configuração do Campo de Rendimentos Regulares relativamente a dados inválidos. Para o Campo em Branco o comportamento é idêntico. Para o caso de sucesso, não deverá ser identificado nenhum comportamento uma vez que este campo faz parte de um campo calculado.

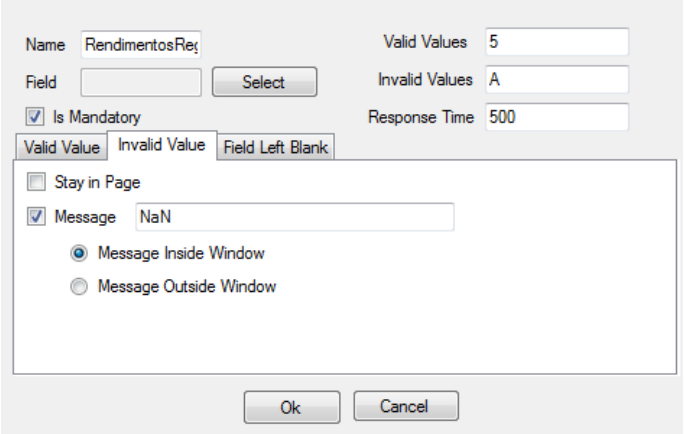


Figura 5.17: Definição do comportamento do Campo Calculado relativo aos Rendimentos Regulares

Para configurar os Campos Calculados devem ser utilizados os Campos de Entrada de Dados definidos. Isso pode ser feito como se encontra ilustrado nas Figuras 5.18 e 5.19. Em cada uma das Figuras é possível ver que na expressão são usados os Campos de Entrada de Dados previamente definidos.

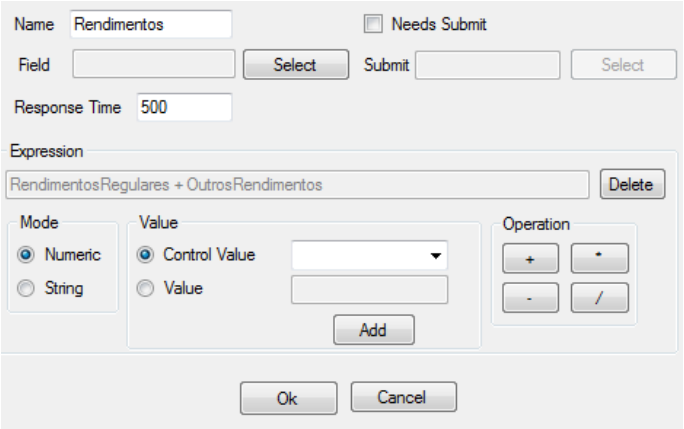


Figura 5.18: Definição do comportamento do Campo Calculado relativo às Receitas

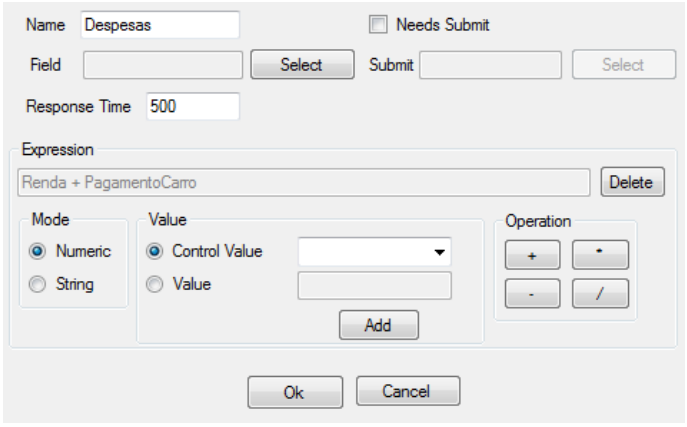


Figura 5.19: Definição do comportamento do Campo Calculado relativo às Despesas

Os dois Campos Calculados definidos acima serão usados na definição do último Campo Calculado (Figura 5.20). Este campo, ao contrário dos outros possui um botão associado. Dado que os outros campos são cálculos intermédios, apenas este necessita de invocar o botão que realiza a acção.

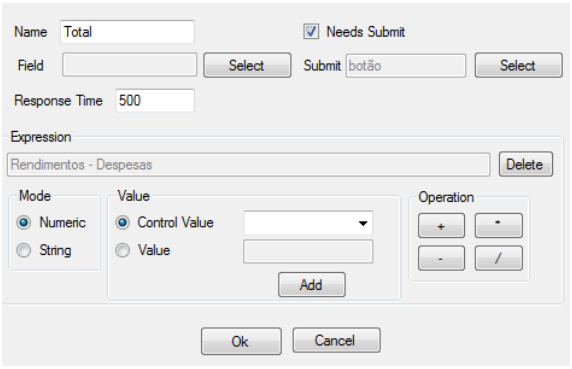


Figura 5.20: Definição do comportamento do Campo Calculado relativo ao Total

O resultado da execução dos testes está representado na Figura 5.21.

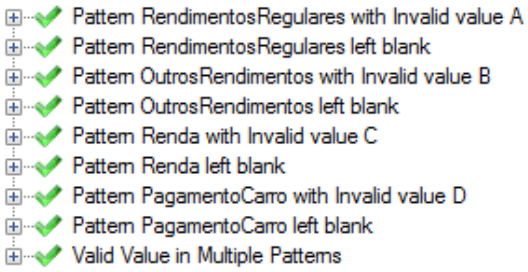


Figura 5.21: Resultados dos testes ao Calculador de Rendimentos

5.2 Detecção de Erros

Uma vez que a ferramenta PETTool permite definir o comportamento esperado de uma GUI, gerar e executar testes para essa mesma GUI torna-se necessário avaliar se este mecanismo permite detectar erros. Assim sendo, nos casos de estudo desta secção serão introduzidos erros em aplicações e será avaliado se a PETTool é capaz de os detectar. Tendo em conta que a configuração da PETTool para cada caso de estudo é semelhante à apresentada na secção anterior, optou-se por omitir as imagens relativas à configuração da ferramenta.

5.2.1 Formulário de Registo e Autenticação

Em `http://www.codeproject.com/KB/asp/MultiUserLoginAuth.aspx` encontra-se um exemplo de um sistema de Registo e Autenticação desenvolvido em ASP e VBScript. O Formulário de Registo possui um conjunto de Campos como o nome de utilizador e palavra-passe. De forma a simplificar o exemplo foram retirados todos os campos excepto o nome do utilizador e palavra-passe (Figura 5.22). A interface da Autenticação (Figura 5.23) é semelhante aos exemplos apresentados na secção anterior.

Figura 5.22: Formulário de Registo

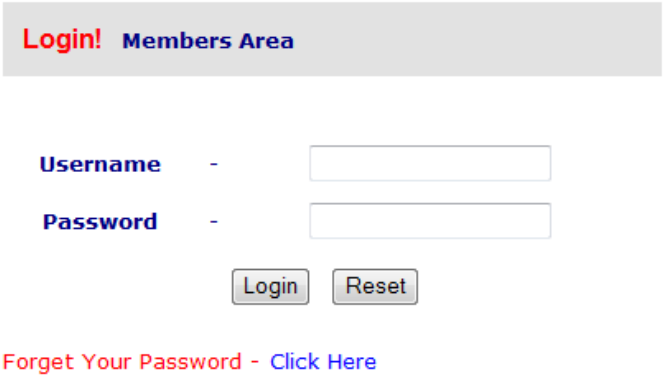


Figura 5.23: Formulário de Autenticação

5.2.1.1 Comportamento Esperado

No caso do Registo, caso algum dos campos não tenha sido preenchido, a aplicação permanece na mesma página não indicando nenhuma mensagem de erro. Caso os dados estejam correctos, a aplicação sai do Formulário de Registo para outra página.

No caso da Autenticação, o comportamento é semelhante mas, em caso de dados inválidos ou em falta, apresenta uma mensagem de erro na própria página.

5.2.1.2 Erros

Do ponto de vista do Registo o único erro que pode ser avaliado consiste em permitir que algum dos campos esteja em branco. Nesse sentido, e após retirar a validação do campo da palavra-passe, foi obtido o relatório de testes da Figura 5.24.

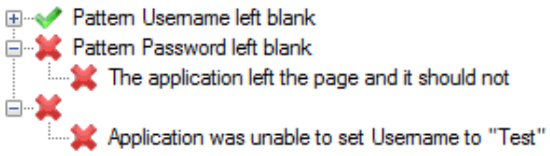


Figura 5.24: Resultado dos testes removendo a validação do Campo Palavra-passe

Segundo o relatório apresentado existem dois erros. O primeiro erro consiste na palavra-passe. No teste em que a palavra-passe foi colocada em branco, a aplicação saiu da página e não devia. Este foi o erro introduzido. O segundo erro foi causado pelo primeiro erro. A PETTool tentou efectuar o teste mas, como houve uma mudança de página inesperada, foi incapaz de o realizar e reportou o erro.

Relativamente à Autenticação existem dois erros típicos que podem ocorrer. O primeiro erro consiste em permitir a Autenticação com dados errados ou em falta. Para verificar se a PETTool consegue identificar estes erros foi modificada a condição de sucesso da autenticação. Assim sendo, a condição que normalmente verificaria se ambos os Campos são válidos, passa a verificar apenas se um deles é válido. O resultado dos testes pode ser visto na Figura 5.25.

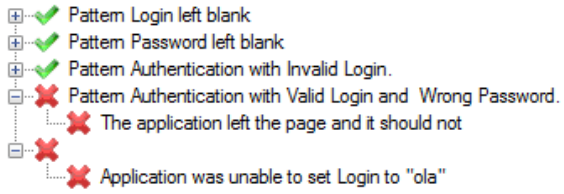


Figura 5.25: Resultado dos testes introduzindo um erro na Autenticação (palavra-passe não validada)

Neste caso, a aplicação transitou para uma nova página com um nome do utilizador válido, apesar de a palavra-passe ser inválida. O erro seguinte mais uma vez é causado pela transição para uma página inesperada.

O segundo erro relativo à autenticação consiste em haver falha na Autenticação mesmo que os dados sejam válidos. Este erro pode ser causado por uma falha nos critérios de igualdade da autenticação ou por uma simples transição para a página errada (que foi o erro inserido neste caso). O resultado dos testes pode ser visto na Figura 5.26.

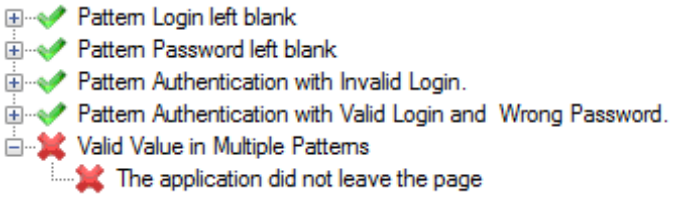


Figura 5.26: Resultado dos testes introduzindo um erro na Autenticação (erro na transição de página com dados válidos)

5.2.1.3 Observações

Os testes apresentados indicam que a PETTool consegue de facto detectar erros relativos a transições incorrectas de página e campos não preenchidos. Devido ao facto de as transições incorrectas serem por vezes difíceis de prever torna-se difícil definir um mecanismo que permita contornar esta situação e retomar os testes. Do ponto de vista dos *browsers* existe a possibilidade de efectuar uma operação de retroceder. No entanto, essa solução seria pouco genérica dado que estaria limitada aos *browsers*, e mesmo nestes por vezes não seria aplicável. Por esses motivos essa abordagem não foi contemplada.

5.2.2 Mestre/Detalhe

No sítio *Web* `http://www.java2s.com/Code/CSharp/Database-ADO.net/ADONETBindingMasterDetail.htm` encontra-se disponível um conjunto de aplicações que funcionam de acordo com as características do Padrão Mestre/Detalhe. Este caso de estudo consiste num par de listas em que a primeira define uma categoria. A segunda lista contém os produtos relativos à categoria da primeira lista. Na Figura 5.27 pode ser vista a interface gráfica da aplicação.

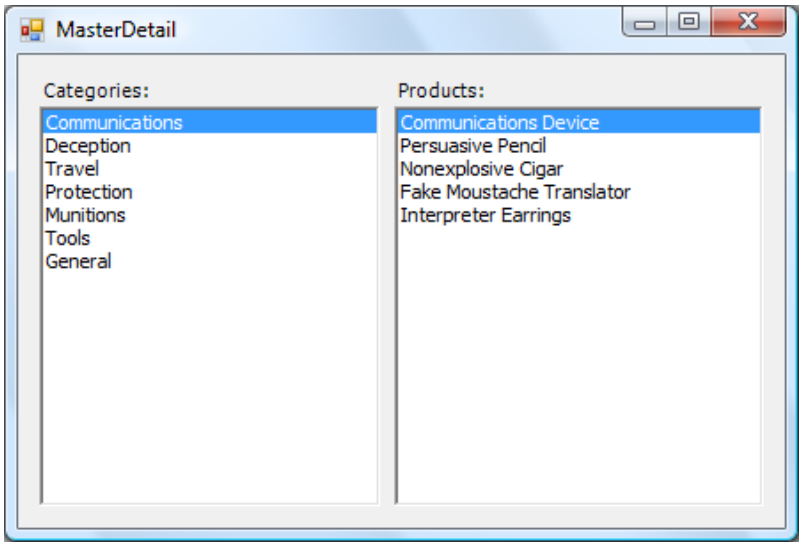


Figura 5.27: Aplicação constituída por um Mestre/Detalhe

5.2.2.1 Comportamento Esperado

O comportamento esperado consiste simplesmente em variar a informação contida na segunda lista após a selecção de um elemento da primeira lista.

5.2.2.2 Erros

De forma a avaliar a detecção de erros foram identificadas duas restrições para os dados da aplicação:

- A categoria "*Communications*" possui os produtos "*Persuasive Pencil*" e "*Interpreter Earrings*"
- A categoria "*General*" não possui os produtos "*Persuasive Pencil*" e "*Interpreter Earrings*"

Um dos erros que pode ocorrer numa aplicação deste tipo consiste em não ser feita a actualização do Detalhe quando o Mestre muda de valor. Uma das causas deste erro consiste em não atribuir um evento ao elemento do Mestre. Após a introdução deste erro foram executados os testes descritos acima e foi obtido o seguinte relatório:

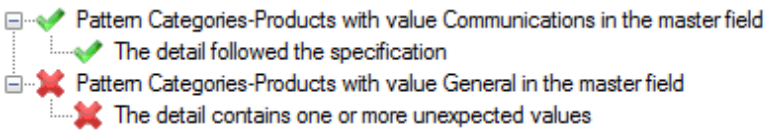


Figura 5.28: Resultado dos testes ao Mestre/Detalhe sem actualização do Detalhe

Apesar da gravidade do erro introduzido, este só afectou um teste. Isto acontece porque as listas encontram-se preenchidas por defeito com os valores associados ao primeiro Mestre ("*Communications*"). Uma vez que o primeiro teste incide no Mestre "*Communications*" apesar de o valor não ter sido mudado, os dados estão certos.

O outro erro que pode ocorrer nestas aplicações, principalmente quando estas acedem a bases de dados, consiste em colocar numa das listas o campo errado da base de dados. Nesse sentido, introduziu-se um segundo erro que consiste em colocar o número do produto em vez do seu nome, na lista do Detalhe. Os resultados são visíveis na Figura 5.29.

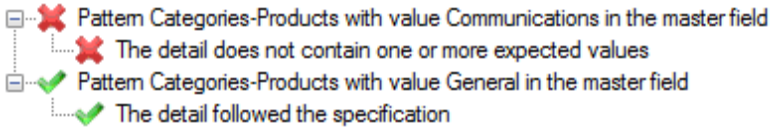


Figura 5.29: Resultado dos testes ao Mestre/Detalhe com troca do Campo apresentado

Este erro passou, à semelhança do anterior, sem ser detectado em um dos testes. Isto mostra a limitação da restrição "Não contém". Neste caso, o facto de ter sido usado o campo errado da base de dados passou despercebido, uma vez que a restrição não verificava que valores deveriam estar presentes mas sim os que não deviam existir.

O último erro identificado consiste em existir informação em falta ou em excesso no detalhe. Para isso foi acedido o ficheiro que contém a informação e foi removido um elemento da categoria "*Communications*". O resultados dos testes podem ser vistos na Figura 5.30.

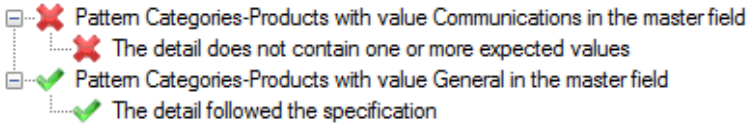


Figura 5.30: Resultado dos testes ao Mestre/Detalhe com remoção de um item de informação

Uma vez que o elemento fazia parte de uma categoria que estava a ser testada, o erro foi detectado. No entanto, se o elemento removido estivesse noutra categoria o erro poderia não ser detectado.

5.2.2.3 Observações

Conforme foi mencionado no Capítulo 3, a detecção de erros no Padrão Mestre/Detalhe está muito dependente das restrições definidas. Tal como pôde ser visto neste caso de estudo, as restrições "Contém" e Não Contém", dado que não analisam a totalidade dos

valores podem deixar escapar valores inválidos ou valores em falta. Por sua vez a restrição "Contém Apenas" leva a um trabalho por parte do responsável de testes que pode ser excessivo. Cabe portanto ao responsável de testes avaliar quais as restrições mais adequadas para cada caso.

5.2.3 Calculadora

O último caso de estudo consiste numa calculadora simples (Figura 5.31). Esta calculadora possui campos para dois operandos e quatro botões, um para cada uma das operações elementares.

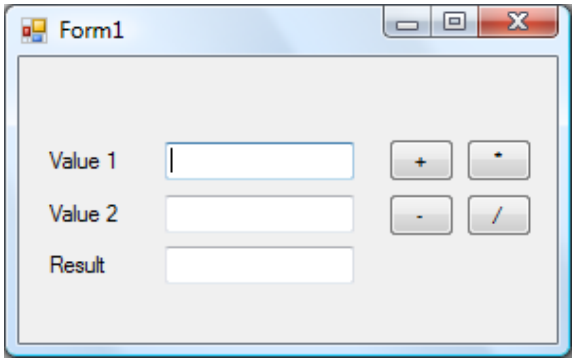


Figura 5.31: Calculadora simples

5.2.3.1 Comportamento Esperado

O comportamento esperado desta calculadora consiste em realizar a operação seleccionada nos botões sobre os valores das caixas de texto dos operandos, apresentando o resultado na caixa de texto *"Result"*. Caso exista um operando em falta ou de tipo de dados inválido, é apresentada uma mensagem fora da aplicação.

5.2.3.2 Erros

Tendo em conta que a operação mais susceptível a erros é a divisão, esta será utilizada para avaliar a detecção de erros por parte da PETTool.

O primeiro erro está relacionado com a precisão da operação. Caso os operandos sejam inteiros, facilmente é feita uma divisão inteira o que conduz a uma perda de precisão da operação. O resultado dos testes obtido foi o seguinte:

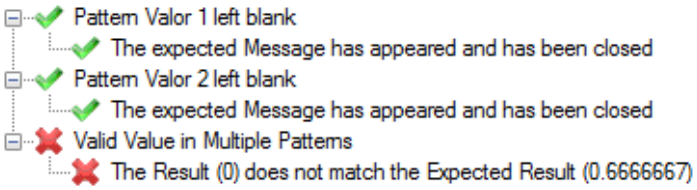


Figura 5.32: Resultado dos testes da Calculadora com erro de precisão

Os dois primeiros testes dizem respeito aos valores inválidos nos operandos, neste caso em branco. O último caso mostra que a ferramenta foi capaz de detectar o erro introduzido. O resultado da divisão de 4 por 6 deu resultado 0 enquanto o esperado era 0,6666667.

Um outro erro que pode ocorrer no contexto deste caso de estudo consiste em trocar a operação num par de botões, por exemplo, trocar o produto pela divisão. Após ter sido introduzido o erro foram executados novamente os testes tendo-se obtido o resultado ilustrado na Figura 5.33

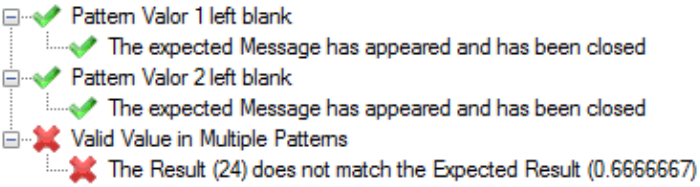


Figura 5.33: Resultado dos testes da Calculadora com troca de operadores

À semelhança do caso de teste anterior, a PETTool identificou o erro de forma apropriada.

O último erro consiste em trocar a ordem dos operandos. Mais uma vez a PETTool detectou o erro como pode ser visto na Figura 5.34.

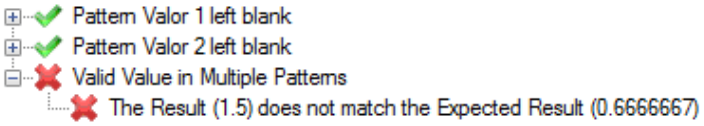


Figura 5.34: Resultado dos testes da Calculadora com troca de operandos

5.2.3.3 Observações

Com os erros introduzidos neste caso de estudo foi possível verificar que a ferramenta PETTool é capaz de detectar vários erros num Padrão Campo Calculado. No entanto, esta

detecção encontra-se muito dependente do responsável de testes. Ao utilizar os valores 4 e 6 numa divisão foi possível detectar todos os erros introduzidos. No entanto, se tivessem sido utilizados outros valores, alguns erros poderiam passar despercebidos.

O facto de ter sido utilizada a divisão também ajudou a detectar melhor os erros. Outras operações como a soma poderiam não detectar o erro de troca de operandos por exemplo.

Conclui-se portanto que apesar da automatização dos testes, o papel do responsável de testes continua a ser muito importante neste cenário de utilização.

5.3 Conclusões

Neste capítulo foi avaliada a utilização da abordagem do capítulo 3. Para isso, foi verificado em primeiro lugar se a PETTool permite definir o comportamento esperado de uma GUI e gerar e executar testes para essa mesma GUI. Os testes realizados sugerem que este processo é possível e efectivamente foram gerados e executados os testes esperados.

Numa segunda fase, foram realizados testes em aplicações com erros. Estes testes permitiram detectar uma limitação na abordagem. A abordagem definida não contempla transições inesperadas sendo incapaz de recuperar delas. Assim sendo, caso um teste falhe devido a uma transição inesperada, todos os testes seguintes falham.

Nesta fase foi ainda possível verificar que funcionalidades semelhantes requerem configurações semelhantes. Deste modo, uma configuração existente pode ser adaptada sem grande esforço para testar outra aplicação. No caso do Padrão Mestre/Detalhe, tal como havia sido sugerido, algumas restrições podem não detectar determinados erros. Torna-se portanto necessário definir um conjunto de restrições adequado para o grau de confiança pretendido para a aplicação a ser testada.

De um modo geral a PETTool permitiu automatizar em muito o processo de teste das GUIs. Apesar de o papel do responsável de testes ser importante, o esforço necessário à geração e execução de testes é reduzido. Deste modo, pode ser dada uma maior atenção à identificação dos valores de teste. No entanto, é necessário melhorar alguns aspectos da abordagem de forma a tornar o processo de teste ainda mais eficiente.

Capítulo 6

Conclusões e Trabalho Futuro

Cada vez mais as pessoas encontram-se dependentes de *software* para realizar as mais variadas tarefas. A necessidade de facilitar o uso do *software* leva cada vez mais à necessidade do uso de uma interface gráfica com o utilizador (GUI). Esta GUI tem como objectivo facilitar a aprendizagem, o uso e a melhorar a experiência de utilização do *software*. Como consequência, o sucesso ou insucesso de um determinado *software* pode depender da correcção da sua GUI.

Esta dependência torna importante garantir que, para além da correcção aos níveis mais baixos da aplicação, existe também correcção ao nível da GUI. No entanto, esta correcção é difícil, e na maior parte das vezes impossível de garantir. Problemas como a enorme combinação de eventos possíveis e elevados custos de criação e execução dos testes podem levar a que, por vezes, o teste ao nível da GUI não seja tão completo quanto deveria ser.

No ramo de testes de *software* existem várias estratégias e ferramentas que podem auxiliar neste processo de teste. Por um lado, os testes de caixa preta podem ser utilizados para testar GUIs com algumas alterações. A sua aplicação pode ser usada para, por exemplo, testar valores limite num Campo de Entrada de Dados. Por outro lado, os testes de caixa branca tendem a ser difíceis de aplicar dado o desconhecimento do código da GUI ou o seu complexo fluxo de execução. A nível de ferramentas, já existem algumas ferramentas que visam resolver os problemas do teste de GUIs, como as de *Record/Playback* e de Teste Baseado em Modelos. No entanto, estas continuam a possuir limitações como, por exemplo, ao nível dos custos associados à sua utilização.

6.1 Satisfação dos Objectivos

Com este trabalho foi abordada uma nova estratégia. Esta estratégia pretende identificar quais os elementos/comportamentos das GUIs mais comuns (Padrões) e, em função deles, definir soluções de teste genéricas. Um Padrão pode ser algo simples como um

Campo de Entrada de Dados ou algo mais complexo como um Campo cujo valor depende de outros Campos. No entanto, estes Padrões não estão isolados, havendo várias relações entre si. O estudo das relações entre Padrões permitiu obter um algoritmo genérico de teste aos seus comportamentos.

De forma a validar a estratégia, foi implementada uma ferramenta. Esta ferramenta foi desenvolvida recorrendo à *framework* UIAutomation e permite identificar vários Padrões, relações entre si e comportamentos esperados. Esta ferramenta foi inicialmente utilizada para avaliar o comportamento de aplicações online tendo os resultados demonstrado que esta permite definir o comportamento esperado, gerar e executar testes. De seguida, foi utilizada para detectar erros introduzidos em aplicações. A aplicação detectou todos os erros introduzidos. No entanto, alguns erros poderiam passar despercebidos dependendo da configuração usada nos testes.

Conclui-se portanto que o conhecimento ao nível dos Padrões pode efectivamente ser usado para testar GUIs de uma forma automatizada. Esta estratégia possui a vantagem de o número de testes executados não estar dependente do número de Padrões, mas sim da configuração de cada Padrão. Assim sendo, um único Padrão pode gerar vários testes. No entanto, esta não pode, em vários casos, ser utilizada como única estratégia de teste. Para os Padrões de Comportamento identificados, a estratégia pode ser usada e, desde que seja bem configurada, é possível testar alguns comportamentos da GUI com menos esforço e de uma forma aceitável. Para os comportamentos menos recorrentes devem continuar a ser utilizadas estratégias alternativas. No entanto, a estratégia possui um conjunto de aspectos a serem melhorados que serão enumerados na secção seguinte.

No âmbito deste trabalho de investigação foi escrito um artigo científico intitulado "PETTool: A Pattern-Based GUI Testing Tool" [CPFA10] tendo este sido submetido e aceite na conferência ICSTE 2010[1].

6.2 Trabalho Futuro

Uma vez que no âmbito deste trabalho se pretende avaliar a possibilidade de utilização de Padrões para testar GUIs, apenas um pequeno conjunto de Padrões foram utilizados. A implementação de muitos outros Padrões poderia aumentar a utilidade desta abordagem e a quantidade de comportamentos que esta pode testar.

Um aspecto importante a ser melhorado mas que não é de todo trivial consiste na detecção de transições inválidas de página. A PETTool não consegue, na sua versão actual, recuperar de uma transição inválida de página durante a execução dos testes. Existem algumas possíveis soluções para este problema como, por exemplo, efectuar uma operação de retroceder num *browser*. Contudo esta operação não é suficientemente genérica.

[1] http://www.icste.org/

Uma outra opção pode partir pela identificação de um conjunto de passos a executar caso a aplicação chegue a um determinado estado.

De momento, a PETTool apenas executa testes de um Formulário de cada vez. Seria interessante relacionar Formulários. Desta forma, seria possível, após efectuar um conjunto de testes, transitar para um Formulário diferente e executar novo conjunto de testes. Isto tornaria o processo de teste mais automatizado e eficiente.

A PETTool, como foi dito anteriormente, permite executar várias vezes o mesmo conjunto de testes. No entanto, quando a aplicação é fechada, a informação relativa à configuração dos testes é perdida. Seria interessante permitir guardar esta informação em ficheiro para mais tarde recuperar e executar novamente os testes. Contudo, este processo pode enfrentar dificuldades na recuperação dos *AutomationElements*. Estes podem trocar de local na estrutura da árvore de controlos e, consequentemente, pode ser necessário proceder a uma nova selecção dos elementos. Porém, a menos que a aplicação mude o seu comportamento esperado, a restante informação de configuração deverá manter-se válida.

Referências

[Bei90] Boris Beizer. *Software testing techniques (2nd ed.)*. Van Nostrand Reinhold Co., New York, NY, USA, 1990.

[Bur03] Ilene Burnstein. *Practical Software Testing: A Process-oriented Approach*. Springer Inc., 2003.

[CHE10] Gui testing checklist. http://www.members.tripod.com/~bazman/checklist.html, acedido em Janeiro de 2010.

[Cor10a] Microsoft Corporation. Model-based testing with specexplorer. http://research.microsoft.com/en-us/projects/specexplorer/, acedido em Janeiro de 2010.

[Cor10b] Microsoft Corporation. Ui automation fundamentals. http://msdn.microsoft.com/en-us/library/ms753107.aspx, acedido em Junho de 2010.

[CPFA10] Marco Cunha, Ana C. R. Paiva, Hugo Sereno Ferreira, and Rui Abreu. Pettool: A pattern-based gui testing tool. In *2nd International Conference on Software Technology and Engineering (ICSTE'10)*, 2010.

[Ger97] Paul Gerrard. Testing gui applications. In *EuroSTAR '97*, November 1997.

[GTWJ03] Jerry Zayu Gao, Jacob Tsao, Ye Wu, and Taso H.-S. Jacob. *Testing and Quality Assurance for Component-Based Software*. Artech House, Inc., Norwood, MA, USA, 2003.

[GUI10] Guitar - a gui testing framework. http://guitar.sourceforge.net/index.shtml, acedido em Janeiro de 2010.

[Laa10] Sari A. Laakso. User interface design patterns. http://www.cs.helsinki.fi/u/salaakso/patterns/, acedido em Janeiro de 2010.

[LHRM07] Ping Li, Toan Huynh, Marek Reformat, and James Miller. A practical approach to testing gui systems. *Empirical Softw. Engg.*, 12(4):331–357, 2007.

[Ltd10] RPM Solutions Pty Ltd. Winrunner - as a gui based load testing tool. http://www.loadtest.com.au/Technology/winrunner.htm, acedido em Janeiro de 2010.

[Mem02] Atif M. Memon. Gui testing: Pitfalls and process. *Computer*, 35(8):87–88, 2002.

[Mem03] Atif M. Memon. Advances in GUI testing. In Marvin V. Zelkowitz, editor, *Highly Dependable Software – Advances in Computers*, volume 58, pages 149–201. Academic Press, 2003.

[MH03] P. J Molina and J. Hernandez. Just-UI: Using patterns as concepts for IU specification and code generation, 2003.

[Nym00] Noel Nyman. Using monkey test tools. *Software Testing and Quality Engineering*, pages 18–21, 2000.

[Oy10] Pattern Factory Oy. Ui pattern factory. http://uipatternfactory.com/, acedido em Janeiro de 2010.

[PFTV05] Ana Paiva, João C. P. Faria, Nikolai Tillmann, and Raul F. A. M. Vidal. A model-to-implementation mapping tool for automated model-based gui testing. In Kung-Kiu Lau and Richard Banach, editors, *ICFEM*, volume 3785 of *Lecture Notes in Computer Science*, pages 450–464. Springer, 2005.

[PFV07] Ana Paiva, João C. P. Faria, and Raul F. A. M. Vidal. Towards the integration of visual and formal models for gui testing. *Electr. Notes Theor. Comput. Sci.*, 190(2):99–111, 2007.

[Sof10] Badboy Software. Badboy software. http://www.badboy.com.au/, acedido em Janeiro de 2010.

[Sys10] Jalian Systems. Marathon. http://www.marathontesting.com/Marathon.html, acedido em Janeiro de 2010.

[Tid05] Jenifer Tidwell. *Designing Interfaces : Patterns for Effective Interaction Design*. O'Reilly Media, Inc., November 2005.

[Tox10] Anders Toxboe. Ui patterns - user interface design patterns library. http://ui-patterns.com/, acedido em Janeiro de 2010.

[UL06] Mark Utting and Bruno Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2006.

[vW10] Martijn van Welie. Welie.com - patterns in interaction design. http://www.welie.com/patterns/index.php, acedido em Janeiro de 2010.

[vWvdVE00] M. van Welie, G. van der Veer, and A. Eliens. Patterns as tools for user interface design. In *1st Int. Workshop on Tools for Working with Guidelines*, pages 313–324, 2000.

[Wal10] Timothy Wall. Abbot - java gui test framework. http://abbot.sourceforge.net/doc/overview.shtml, acedido em Janeiro de 2010.