ISEL

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

**Área Departamental de Engenharia de Electrónica
e Telecomunicações e de Computadores**

Suporte a Testes Automáticos em Aplicações Web Geradas com a OutSystems Platform

RICARDO NUNO COIMBRA NETO
(Licenciado)

Dissertação de natureza científica para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientadores:
Mestre Fernando M. Carvalho
Doutor Tiago L. Alves

Júri:
Presidente:
Mestre Fernando Manuel Gomes de Sousa
Vogais:
Doutor João Pascoal Faria
Mestre Fernando M. Carvalho

Novembro de 2013

ISEL

INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

**Área Departamental de Engenharia de Electrónica
e Telecomunicações e de Computadores**

Suporte a Testes Automáticos em Aplicações Web Geradas com a OutSystems Platform

RICARDO NUNO COIMBRA NETO
(Licenciado)

Dissertação de natureza científica para obtenção do Grau de Mestre
em Engenharia Informática e de Computadores

Orientadores:
Mestre Fernando M. Carvalho
Doutor Tiago L. Alves

Júri:
Presidente:
Mestre Fernando Manuel Gomes de Sousa
Vogais:
Doutor João Pascoal Faria
Mestre Fernando M. Carvalho

Novembro de 2013

Tese realizada sob orientação do

Professor Fernando M. Carvalho
Instituto Superior de Engenharia de Lisboa
e
Doutor Tiago L. Alves
OutSystems

Resumo

As exigências de um mercado competitivo, no qual as aplicações web são uma peça intrínseca e fundamental, requer que estas se possam adaptar rapidamente a novos requisitos. É fulcral que as empresas consigam evoluir as suas aplicações web para dar resposta a novos requisitos dos seus clientes, ou a reflectir mudanças internas, conseguindo assim manter a sua competitividade.

Neste contexto, a OutSystems desenvolveu a OutSystems Platform, como uma ferramenta para suportar o ciclo de vida de aplicações web. A OutSystems Platform não só permite gerar aplicações web em tecnologias standard (ASP .NET ou JEEE) como também agiliza todo o processo de desenvolvimento e *deployment*.

A necessidade de mudança rápida impõe um acompanhamento imediato na validação destas mudanças algo que nem tecnologias tradicionais, nem a OutSystems Platform conseguem ainda dar resposta. Tecnologias actuais para teste de aplicações web (e.g. HttpUnit, WebDriver) são ainda muito baixo nível oferecendo apenas uma forma de integarir com elementos HTML num browser. Isto torna custoso o desenvolvimento e manutenção destes testes numa situação de rápida evolução das aplicações.

Esta tese de mestrado foca-se em permitir um rápido desenvolvimento e adaptação de testes a aplicações web. Tirando partido dos modelos visuais utilizados para a geração de aplicações web desenvolvidas na OutSystems Platform e da *framework* de testes Selenium WebDriver, apresenta-se uma solução para gerar uma *framework* que permite o desenvolvimento de testes próximo do domínio da aplicação, validando-a através da análise de casos de estudo.

Palavras-Chave: teste de software, aplicações web, qualidade de software, geração de código

Abstract

In a competitive and demanding market where web applications are fundamental it is expectable that they quickly adapt to new requirements. It is paramount that companies can evolve their web applications either to fulfill customers' requirements or adapt to internal changes allowing them to stay competitive.

With this goal in mind, OutSystems created the OutSystems Platform as a tool to support web applications lifecycle. The OutSystems Platform not only allows the creation of web applications in standard technologies (ASP.NET or J2EE) but also streamlines the development process.

The need to change demands rapid validation of the performed changes, something that traditional technologies can not keep up. Web applications testing technologies (e.g. HttpUnit, WebDriver) are still operating at the page structure level and offering no other options to interact with HTML elements in a browser. This impacts the cost of developing and maintaining tests for applications that evolve rapidly.

This thesis focuses on allowing rapid development and adaptation of web application tests. By taking advantage of the visual models used by the OutSystems Platform to generate web applications, and taking advantage of the Selenium WebDriver framework, we present a solution that generates a test framework that allows tests to be developed closer to the application domain.

Keywords: software testing, web applications, software quality, code generation

Agradecimentos

Ao longo do Mestrado em Engenharia Informática e de Computadores, várias foram as pessoas que me ajudaram, directa ou indirectamente, a cumprir os meus objectivos. O facto de o ter feito em regime pós-laboral aumentou a dificuldade como o percurso foi feito, da mesma forma que aumentou a intensidade com que todas as ajudas recebidas foram sentidas. Certo de que o espaço reservado nesta secção não me permitirá agradecer a todas pessoas que me ajudaram, deixo aqui palavras que expressam o meu profundo agradecimento.

Ao *Professor Fernando Miguel Carvalho*, pela orientação deste trabalho, por todo o conhecimento transmitido e por me acompanhar em mais uma jornada da minha vida académica.

Ao *Tiago Alves*, pela orientação deste trabalho, por todas as correcções e sugestões e pelos conselhos de pesquisa.

À *OutSystems* por me ter proporcionado todas as condições necessárias à realização deste trabalho, em especial ao *David Nunes*, por me ter proporcionado o tempo necessário ao desenvolvimento deste trabalho.

Ao *Lúcio Ferrão* pelas palavras de incentivo e pela confiança demonstrada.

Ao *Luis Lopes* e *João Portela*, pelas sugestões e ideias partilhadas, que permitiram enriquecer este trabalho.

A todos os meus colegas do *R&D* da *OutSystems*, pela amizade, momentos de boa disposição e toda a disponibilidade mostrada para ajudar.

Aos meus colegas do *ISEL*, em especial ao Paulo Pires, Nuno Sousa, Mário Vieira, Marco Lopes, Nuno Cancelo e Júlio Olivares, por toda a força e por todos os momentos passados neste longo caminho.

Aos meus amigos de sempre, por toda a amizade, todos os momentos que passamos e todos os momentos que iremos passar.

À minha familia, Bruno, Carlos, Gonçalo, Pedro, Carla, Maria Emilia e Joquinha, por todo o apoio e confiança, pelas palavras de incentivo e por estarem sempre presentes.

Aos meus pais, Paula e Manuel, por todo o carinho, por me terem ensinado a não desisitir e pelos ensinamentos de uma vida.

À Margarida quero agradecer a compreensão, toda a alegria e carinho, o seu sorriso, o seu olhar e o seu pensar. Espero que o meu esforço te sirva de estímulo para perseguires os teus sonhos.

À Patricia quero agradecer pela paciência, pela compreensão, pelo carinho, pelo apoio, pela ajuda.... por tudo. Sem ti nada seria possível. Sem ti nada teria significado. O meu sincero Obrigado.

"We are what we repeatedly do.
Excellence, therefore, is not an act but a habit."
— Aristotle

Índice

Lista de Figuras

Lista de Listagens

Capítulo 1

Introdução

Tem-se registado um aumento significativo na utilização de aplicações web, em parte devido à crescente variedade de dispositivos que se tornaram acessíveis ao utilizador. Do ponto de vista das empresas, é uma forma fácil de oferecer os seus serviços a um vasto número de clientes aparecendo como um elo de ligação com parceiros, fornecedores, clientes ou funcionários.

No contexto do desenvolvimento tecnológico, factores como o aumento da capacidade de processamento e do armazenamento em equipamentos clientes e servidores, o aumento da largura de banda ou o aparecimento de novas linguagens fazem com que a exigência sobre as aplicações web seja grande. Com a necessidade de alterações frequentes às aplicações, no sentido de manter as organizações a par dos seus competidores, o risco de introdução de falhas é aumentado [15], realçando a importância de testes capazes de detectá-las o mais cedo possível.

A detecção de falhas em fases iniciais do ciclo de desenvolvimento evita a sua propagação para outras fases diminuindo consideravelmente o custo associado à sua correcção [24] e, por consequência, reduzindo o custo de desenvolvimento das aplicações. A existência de um processo de teste e automação da execução de testes permite verificar o comportamento da aplicação de forma rápida. No entanto, é preciso que os processos e ferramentas de teste acompanhem a rápida evolução das aplicações web de modo a evitar a degradação da sua qualidade.

O foco deste trabalho é melhorar o processo de criação e manutenção de testes que têm como objectivo verificar uma funcionalidade, derivando normalmente de um requisito ou caso de utilização. Este tipo de testes, designados de *testes funcionais*, verificam o comportamento de um sistema na perspectiva do utilizador final [18]. O objectivo deste trabalho é pemitir que seja possível definir ou modificar de forma ágil testes funcionais para acompanhar a rápida evolução das aplicações web.

1.1 Teste de Aplicações Web

Os objectivos primários do teste de aplicações são: (i) verificar a conformidade com a especificação no que respeita a requisitos funcionais e não funcionais; (ii) identificar erros de uma forma rápida permitindo avaliar o impacto de alterações ou novos desenvolvimentos; e (iii) validar a usabilidade do produto de acordo com as expectativas do utilizador final.

Contudo, apesar destes objectivos serem partilhados por aplicações web e aplicações tradicionais, a forma de os alcançar muda consideravelmente pelas diferenças existentes entre as duas.

As aplicações web baseiam-se numa arquitectura cliente-servidor onde o cliente solicita ou envia informação e o servidor guarda e devolve informação, conforme Figura 1.1. Contudo, do ponto de vista arquitectural, existem particularidades que colocam desafios à forma como os elementos comunicam, destacando-se o facto do protocolo utilizado ser *stateless*. Esta característica obriga o cliente a introduzir informação relevante quando comunica com o servidor, que permita a reposição do estado, criando assim o contexto necessário ao processamento do pedido. Esta responsabilidade reflecte-se directamente no teste de aplicações web uma vez que é da sua responsabilidade garantir a definição do estado actual que, por consequência, provocará transição para estados que vão de encontro ao seu objectivo.

O facto de nos clientes o acesso poder ser feito com browsers de diferentes características faz com que estas diferenças se acentuem uma vez que a apresentação e comportamento das aplicações varia de browser para browser [6]. A geração dinâmica de componentes, baseada em dados transmitidos pelo utilizador, constitui também um desafio pela diversidade de casos de teste que podem ser desenhados devido ao número de combinações possíveis dos valores de input [6]. A evolução de linguagens de script que se executam no cliente (e.g. JavaScript) aumentou a sua capacidade de computação permitindo retirar parte da lógica do servidor, nomeadamente validação de inputs [7]. Contudo, a utilização frequente de mecanismos de invocação assíncrona, vem adicionar complexidade à tarefa de detecção de falhas [7]. Estes motivos, fazem com que os testes a aplicações web sejam uma área de estudo por si mesmo.

No contexto dos testes funcionais é possível testar uma grande parte da lógica aplicacional presente quer no cliente quer no servidor sem recorrer à interface gráfica, operando directamente sobre os compoentes que a implementam (e.g. métodos de classes). Todavia, a combinação de factores como as características dos pro-

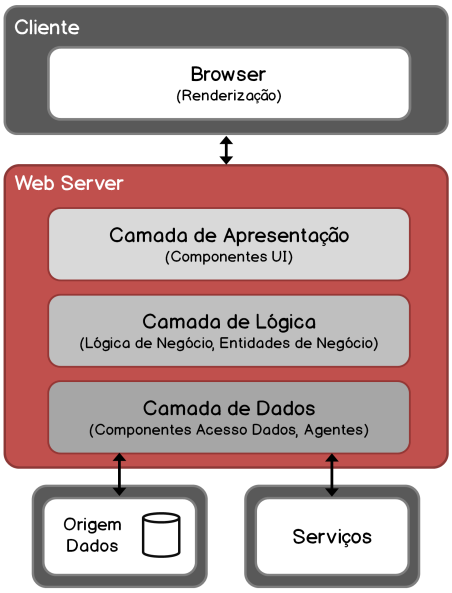

Figura 1.1: Representação da arquitectura genérica de uma aplicação web representando os diferentes intervenientes: (i) Client, onde o browser é o responsável pela renderização das páginas, bem como, pelo envio de informação para o servidor, (ii) Web Server que, neste caso, inclui o servidor aplicacional, onde estão identificadas as camadas de apresentação, lógica de negócio e acesso a dados, (iii) e o servidor de base de dados, bem como, serviços externos, caso se aplique.

tocolos utilizados no modelo cliente-servidor ou as técnicas utilizadas para gestão de estado pode facilmente causar situações de falha, realçando a necessidade de a verificar. Assim, no sentido de exercitar o funcionamento das aplicações em conjunto com a infra-estrutura do ponto de vista do utilizador, é comum serem desenhados testes que incluem a maioria destas variáveis, simulando interacções com a interface gráfica e obervando os resultados dessa interacção. O desenho deste tipo de testes é feito com base no comportamento esperado do componente a testar, cujo processo de criação e manutenção será o foco desta tese.

1.2 A OutSystems Platform

A necessidade de adaptação e inovação por parte das organizações, que obriga a alterações frequentes às suas aplicações web, foi a motivação do produto OutSystems Platform.

A OutSystems Platform é uma plataforma de desenvolvimento rápido composta por um ambiente de desenvolvimento (ServiceStudio), uma ferramenta de suporte à integração (IntegrationStudio) e um servidor (PlatformServer) responsável por, entre outros, compilar e disponibilizar aplicações no servidor aplicacional. A Figura 1.2 mostra algumas das aplicações disponibilizadas pela OutSystems Platform, nome-

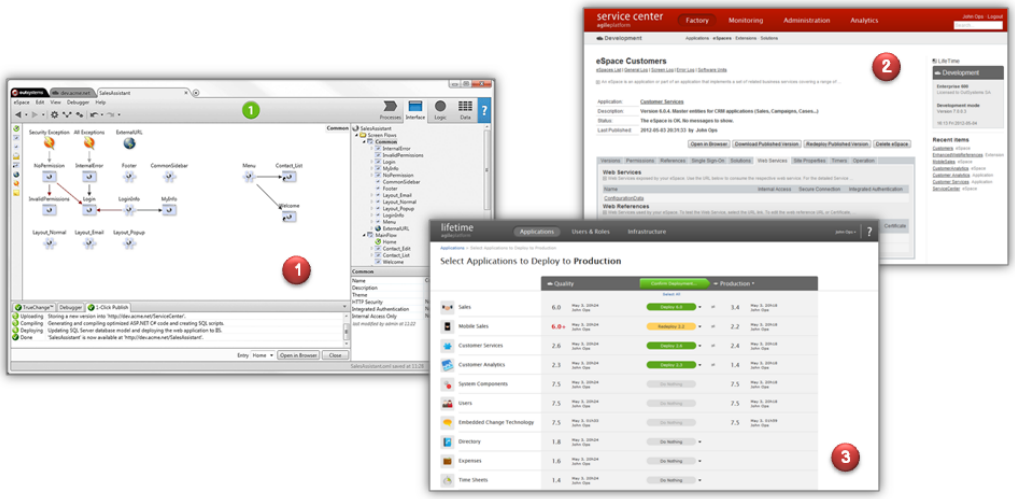

Figura 1.2: *Screenshot* das aplicações disponibilizadas pela OutSystems Platform, nomeadamente: (1) ServiceStudio - ambiente de desenvolvimento; (2) ServiceCenter - aplicação de gestão da plataforma e (3) LifeTime - aplicação de gestão de versões entre ambientes.

adamente, Serice Studio, Service Center e LifeTime. A maioria das funcionalidades do PlatformServer são disponibilizadas através da aplicação Service Center. Além do desenvolvimento ser facilitado pela utilização de uma linguagem de modelação visual, um dos principais benefícios da plataforma é facilitar a adopção de metodologias ágeis no processo de desenvolvimento. A existência de mecanismos que garantem a coerência entre as principais camadas da aplicação (i.e., a user interface, a lógica de negócio e o acesso a dados) permitem uma reacção rápida à mudança. A Figura 1.3 ilustra o processo alto nível pelo qual passa uma aplicação web que é criada e posta em execução usando a OutSystems Platform. Este processo inicia-se com o envio do modelo da aplicação definido no ServiceStudio para o Service Center, sobre o qual será feita a geração automática do código que será *deployed* no servidor aplicacional.

As aplicações OutSystems são compostas por módulos podendo estes ser: (i) eSpaces, que são construidos no ServiceStudio e que permitem a implementação das várias camadas da aplicação e (ii) extensões, que são construidas no IntegrationStudio e que têm por objectivo facilitar a integração com sistemas legados ou fontes de dados externas.

Os eSpaces são módulos desenvolvidos no ServiceStudio sendo neles definidos de uma forma visual todos os aspectos de uma aplicação empresarial. No sentido de auxiliar a alteração de aplicações de forma segura, é disponibilizado o mecanismo TrueChange que auxilia o programador em tarefas de refactorização e de verificação de consistência do módulo. Após o seu desenvolvimento, as aplicações são enviadas para o PlatformServer onde existem serviços responsáveis por iniciar o processo de

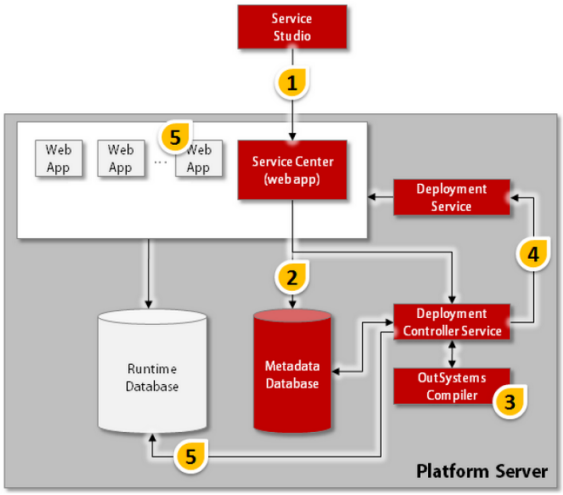

Figura 1.3: Processo de publicação de uma aplicação a partir do ServiceStudio: (1 e 2) envio da aplicação para o Service Center onde será armazenada em base de dados e iniciado o controlo de versões; (3) geração de código Java ou .NET e compilação incluindo as dependências especificadas; (4) preparação dos ficheiros da aplicação e notificação dos servidores aplicacionais para que estes iniciem o processo de obtenção dos ficheiros necessários e respectiva disponiblização; (5) actualização do modelo físico e activação da nova versão da aplicação.

gestão de versões, geração e compilação de código, disponibilização da aplicação no servidor aplicacional e actualização ou criação do modelo de dados.

No que respeita à arquitectura alvo das aplicações, a OutSystems Platform permite geração automática de código para servidores aplicacionais Java ou .NET podendo ser utilizado como base o mesmo módulo para ambos os casos. A persistência de dados pode ser feita com recurso a bases de dados Oracle ou SQL Server.

A OutSystems Platform não é só uma plataforma de desenvolvimento aplicacional, oferecendo também mecanismos de suporte a outras fases na gestão do ciclo de vida das aplicações, entre as quais, gestão centralizada e monitorização das aplicações geradas e obtenção de feedback dos utilizadores directamente das aplicações. Todavia, a OutSystems Platform não oferece suporte para a definção de testes das aplicações web que gera. Assim, devido à importância de garantir a qualidade das aplicações, bem como, da necessidade de verificar automaticamente que novos desenvolvimentos não provocaram falhas no seu comportamento, o suporte a testes é crucial.

O foco desta tese, será tirar partido do modelo de uma aplicação web definida na OutSystems Platform, mais especificamente o modelo de navegação que é utilizado para geração de código, para dar suporte à fácil definição e modificação de testes funcionais.

1.3 Definição do Problema

A execução de testes funcionais em aplicações web passa pela interacção com várias páginas web de forma a verificar que uma determinada funcionalidade está de acordo com a sua especificação.

Tecnologias como o HttpUnit[1] e Selenium WebDriver[2] são usadas para automatizar e abstrair todo o processo de comunicação com um servidor web. O Selenium WebDriver oferece adicionalmente primitivas básicas de procura e accesso aos elementos HTML de uma página web recebida. A criação de testes funcionais usando estas tecnologias requer assim o conhecimento do HTML recebido e dos seus elementos específicos onde os elementos de negócio da aplicação web estão mapeados.

A necessidade de conhecer a estrutura HTML e seus elementos específicos para a definição de testes funcionais introduz diversos problemas. É custoso definir, depurar ou adaptar estes testes pelo uso de conceitos baixo nível (i.e. elementos HTML) distante dos conceitos usados nas aplicações web (e.g. clientes, produtos). Pequenas mudanças na estrutura da página levam a que estes testes falhem durante a sua execução por não terem correspondência com certos elementos HTML específicos.

Tecnologias como Cucumber[3] tentam resolver a dificuldade na definição de testes através de uma linguagem natural mais próxima dos conceitos usados nas aplicações web. No entanto, o mapeamento dos conceitos da linguagem de teste com os elementos específicos de uma página HTML continua a ter que ser feito manualmente sofrendo dos restantes problemas atrás enunciados.

Num contexto em que há uma rápida evolução das aplicações web torna-se importante que os testes que validam a funcionalidade destas aplicações possa evoluir ao mesmo ritmo. Assim sendo, é imperativo encontrar uma solução para a fácil definição, depuração e adaptação de testes funcionais e que estes testes sejam robustos a mudanças na estrutura do HTML.

O problema que esta tese se propõe a resolver é dar suporte automático à definição de testes funcionais para aplicações web permitindo que estes estejam num domínio mais próximo do contexto das aplicações e que sejam robustos à estrutura HTML usada. A abordagem a este problema inclui tirar partido da OutSystems Platform, em particular dos modelos que definem e de onde são geradas aplicações web, e

[1] `http://httpunit.sourceforge.net`
[2] `http://docs.seleniumhq.org/projects/webdriver/`
[3] `http://cukes.info`

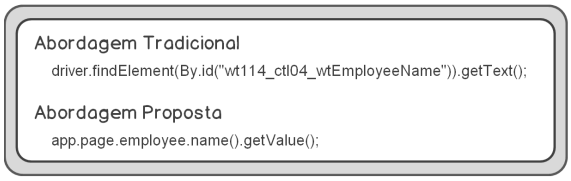

Figura 1.4: Abordagem tradicional e abordagem proposta

das tecnologias actuais para teste de aplicações web.

1.4 Abordagem

Com o objectivo de endereçar o problema da dependências dos testes com o HTML das páginas, a abordagem seguida consiste na criação de uma camada aplicacional com objectos capazes de representar a aplicação, as suas páginas e os elementos que as constituem, com os quais o utilizador pode interagir. Com isto, pretende-se que esta *framework* mapeie conceitos alto-nível de uma aplicação web nos elementos HTML necessários e assim facilitar a criação de testes, abstraindo a estrutura HTML das páginas e dos seus elementos, facilitando a criação e manutenção de testes funcionais. Como exemplo das diferenças que se pretendem introduzir com a *framework*, a Figura 1.4 apresenta um exemplo de código de teste que têm por objectivo obter o valor de um campo de *input* numa página, ilustrando as diferenças entre a abordagem tradicional e a abordagem proposta.

1.5 Estrutura da Tese

A presente tese está estruturada nos seguintes capítulos.

O Capítulo 2 faz um resumo do estado da arte na área de testes a aplicações web. Uma breve explicação dos diferentes níveis, estratégias e modelos de teste é apresentada para contextualizar o trabalho desta tese. Testes funcionais são definidos em maior detalhe, ferramentas de teste usadas são discutidas e comparadas, e, por fim, trabalho relacionado é analizado.

O Capítulo 3 apresenta uma aplicação web definida na OutSystems Platform que será usada como caso de estudo. A aplicação web será explicada juntamente com a definição de vários cenários de teste. Os cenários de teste serão implementados usando várias tecnologias de teste para exemplificar o problema que se pretende solucionar. Esta aplicação web será também usada em capítulos posteriores para exemplificar o resultado da solução apresentada.

O Capítulo 4 propõe uma solução para gerar uma *framework* de suporte a testes a partir de um modelo de aplicação web definido pela OutSystems Platform. Os pressupostos e requisitos desta solução serão definidos. A estratégia de geração da framework a partir de um modelo de aplicação será explicada e demonstrada revisitando o caso de estudo introduzido anteriormente.

O Capítulo 5 conclui a tese incluindo uma análise crítica do trabalho desenvolvido e da solução proposta, assim como direcções futuras de investigação.

1.6 Convenções

O presente documento não foi escrito ao abrigo do novo Acordo Ortográfico. Todo o código apresentado foi escrito em Java, salvo indicação contrária.

Capítulo 2

Estado da Arte

Esta secção faz uma introdução à área de testes de software começando por apresentar estratégias utilizadas no seu desenho, bem como, os tipos de teste mais significativos. Por serem o foco desta tese, os testes funcionais serão apresentados com maior detalhe descrevendo-se uma metodologia adequada ao seu desenho e técnicas para restrição do número total de testes. O suporte para a automação de testes será introduzido comparando as duas abordagens mais comuns e detalhando quais as preocupações a ter na construção de *frameworks* de teste, seguindo-se a apresentação das *frameworks* HttpUnit, Selenium WebDriver e Cucumber. Adicionalmente será feita uma breve apresentação de trabalhos relevantes na área de teste de aplicações web.

2.1 Teste de Aplicações Web

A execução de testes oferece garantias de que as aplicações e a infra-estrutura que as suporta (e.g. servidor de base de dados) funcionam correctamente e estão em conformidade com os requisitos não funcionais e funcionais da aplicação. Requisitos não funcionais incluem restrições e qualidades do sistema [14] entendendo-se restrições como factores que condicionam a solução a apresentar (e.g. capacidade de armazenamento) e qualidades, como propriedades ou características que podem afectar a satisfação dos utilizadores desse sistema (e.g. tempo de resposta). Requisitos funcionais definem o que o sistema deve fazer, i.e., qual o seu comportamento por forma a suportar os objectivos, tarefas ou actividades dos utilizadores [14].

Sendo o suporte à criação de testes o foco do presente trabalho, apresentam-se de seguida termos relativos a esta área, cuja definição importa formalizar. Segundo [18], um *caso de teste* refere-se ao conjunto de valores de entrada, condições

de execução e resultados esperados, sendo esta informação utilizada no contexto de um teste, sujeitando um componente a várias combinações de valores, verificando assim diferentes comportamentos especificados. Uma *test suite* é uma colecção de casos de teste. A sua execução periódica, dá origem a *testes de regressão* que permitem verificar que alterações à aplicação não introduziram comportamentos inválidos. Uma vez que, à correcção de falhas está também associada a criação de testes que as identifiquem [8], os testes de regressão são também responsáveis por garantir que falhas já corrigidas e testadas não voltam a ser introduzidas. As vantagens deste tipo de testes estão relacionadas com a rapidez com que se conseguem verificar alterações ao comportamento, diminuindo o tempo de reacção a falhas [4].

Aquando da execução do teste de aplicações com a intenção de encontrar erros, é impraticável ou impossível encontrar todos os erros existentes [15]. Este problema tem implicações ao nível do desenho de testes uma vez que, não conseguindo garantir a ausência de erros, estes devem ser desenhados de maneira a serem o mais completos possível, devendo ser adoptadas estratégias adequadas. As estratégias de desenho podem basear-se em: (i) comportamentos do programa, designadas de *black-box*; (ii) na sua implementação ou estrutura interna, designadas de *white-box*; ou (iii) em ambos, designadas de *gray-box* [6].

A adopção de uma estratégia *black-box* permite encontrar comportamentos do programa não conformes com a sua especificação. Neste caso, levando o desenho de testes ao limite, a forma de encontrar todos os erros existentes implica o teste exaustivo de todos os valores de entrada, dando origem à criação de tantos casos de teste quantas as combinações possíveis. Estratégias *white-box* diferem quanto à origem dos erros sendo os valores de entrada combinados com o objectivo de encontrar erros, no limite, em todo o código da aplicação. Todavia, a criação de testes exaustivos em ambas as abordagens revela-se impraticável, obrigando à redução do número de casos de testes, sendo muitas vezes a combinação de estratégias uma solução para este problema [15].

2.2 Tipos de Testes

A criação de testes é uma actividade que, em alguns casos, antecede a própria implementação, podendo aparecer em diferentes níveis do desenvolvimento de uma aplicação [18]. Os sistemas são assim testados em diferentes fases do seu desenvolvimento, desde pequenas unidades de código, passando por sub-sistemas integrados até ao sistema final. Os principais níveis de teste durante o processo de desenvolvimento são [17]: unitário, integração e sistema. No início do desenvolvi-

mento de uma aplicação os primeiros testes que é possível criar são testes unitários que verificam no limite cada método codificado. Com o aparecimento de novos componentes e integração de outros já existentes torna-se possível criar testes de integração que verificam o funcionamento conjunto dos componentes. Numa fase mais avançada do desenvolvimento do produto já é possível criar testes de sistema que verificam o comportamento na perspectiva do utilizador.

A existência de diferentes níveis de teste permite garantir o comportamento esperado durante todo o desenvolvimento do produto. A percepão do produto é feita assim a vários níveis de abstracção, viabilizando a identificação de diferentes tipos de defeitos [5]. Do ponto de vista do desenho de testes, o foco num componente específico permite cobrir mais eficazmente os seus comportamentos, reduzindo o número de casos de teste em níveis superiores, permitindo mesmo a sua reutilização [5]. Detalham-se de seguida os níveis de teste identificados aplicando-os ao caso das aplicações web no contexto da criação de testes funcionais.

2.2.1 Teste Unitário

Os testes unitários verificam cada unidade de código produzida, normalmente métodos ou procedimentos, que executam uma função específica e indivisivel, adequando-se a estratégias *black-box* ou *white-box* [5]. No contexto das aplicações web, a unidade mínima de teste é a página [6] havendo uma distinção entre páginas servidor e páginas cliente.

O teste de páginas cliente permite verificar, entre outros, a apresentação dos conteúdos no browser, links com destinos inválidos e comportamento não conforme com a especificação oferecendo desafios no que respeita ao teste de páginas geradas dinamicamente. O teste de páginas servidor tem o objectivo de garantir o correcto funcionamento da parte da lógica de negócio e de acesso a dados que implementam e de permitir identificar falhas na geração das páginas para o cliente.

2.2.2 Teste de Integração

Os testes de integração visam identificar erros na ligação entre componentes, bem como, o seu funcionamento na forma de sub-sistemas [5]. No contexto das aplicações web o objectivo é verificar o comportamento resultante da comunicação entre páginas passando o foco a ser a sua colaboração [6].

Ainda que a existência de testes unitários ofereça garantias ao nível do comportamento individual das unidades, a sua combinação é também a origem de falhas [5].

A definição de casos de teste passa por identificar páginas que se relacionem no sentido de testar a sua funcionalidade combinada. O conhecimento da estrutura da aplicação permite determinar ligações entre páginas, viabilizando a selecção de pares candidatos a testes de integração com o objectivo de testar o seu comportamento de acordo com a especificação. Neste sentido, uma estratégia *gray-box* adequa-se a este nível de teste [6].

2.2.3 Teste de Sistema

Os testes de sistema verificam a conformidade do sistema com os seus requisitos do ponto de vista do utilizador final. Este nível de testes tem como alvo a ligação de todos os sub-sistemas, tendo estes sido sujeitos a testes de integração [5]. O objectivo é o de testar funcionalidades completas, como é o caso de implementações de casos de utilização, adequando-se uma estratégia *black-box* no desenho de testes.

Os testes de sistema antecedem o momento de disponibilização da aplicação ao utilizador final, sendo importante garantir não só a sua funcionalidade, mas também que a sua qualidade foi avaliada, como é o caso da usabilidade, performance ou segurança. Este tipo de testes permite assim verificar não só que o software funciona mas também a comunicação com todos os seus componentes, entre os quais, bases de dados, servidores web e servidores aplicacionais.

2.2.4 Teste Funcional

Os testes funcionais apresentam-se como outro tipo de teste, contudo, existem na literatura diferentes formas de os posicionar quanto ao nível que ocupam, i.e., a que fase do desenvolvimento se adequam. Por um lado, considera-se que a utilização de testes funcionais é adequada a todos os níveis de desenvolvimento, começando mesmo no processo de especificação de requisitos, continuando para cada nível de desenho e especificação de interface [18]. Outra abordagem a este tipo de testes é de que estes se adequam ao nível de testes de sistema [5], com o objectivo de demonstrar a funcionalidade do sistema pela verificação da sua especificação. No contexto do presente trabalho, os testes funcionais serão tratados ao nível do sistema, tendo como objectivo principal verificar a especificação de funcionalidade do ponto de vista do utilizador final.

2.3 Testes Funcionais

No contexto do presente trabalho, considera-se que a utilização de testes funcionais tem como objectivo verificar a conformidade do sistema com a especificação de funcionalidade do ponto de vista do utilizador final [5]. Assim, a criação de testes funcionais de sistema será exemplificada com base em histórias de utilização de aplicações já construídas.

A especificação funcional é a descrição do comportamento esperado de um componente, sendo uma das mais importantes fontes de informação para a definição de testes [18]. No contexto das aplicações web, o componente em teste pode ser uma página, um conjunto de páginas ou a própria aplicação, dependendo do nível a que o teste se aplica. O desenho de testes funcionais caracteriza-se por fazer uso da especificação do componente, focando-se em *inputs* e *outputs*,adequando-se a uma estratégia *black-box* [5].

O desenho de testes com recurso a este tipo de abordagem tem a vantagem de identificar erros não detectados por testes desenhados com conhecimento da implementação do componente [18]. O conhecimento da especificação da funcionalidade de um componente permite estabelecer o conjunto de comportamentos a verificar, idependentemente de existir implementação específica para cada um deles.

2.3.1 Abordagem Sistemática no Desenho de Casos de Teste

A derivação de casos de teste da especificação da aplicação permite em alguns casos formaliza-la, evidenciando aspectos não contemplados na mesma. Na base do desenho de testes funcionais está a divisão do conjunto possível de valores de entrada em classes ou partições, determinadas apenas com recurso à especificação, correspondendo tipicamente a um tratamento diferente por parte da implementação. Em [18] identificaram-se etapas de desenho de casos de teste funcionais que permitem a adopção de uma abordagem sistemática, apresentadas na Figura 2.1. De seguida faz-se uma descrição de cada uma destas etapas.

1 - Identificar Funcionalidades Testadas Isoladamente

Divisão da especificação em funcionalidades isoladas, com base na percepção do utilizador do sistema, tratando a complexidade de forma incremental. As funcionalidades consideram-se descritas quando é identificado o conjunto de valores de entrada que correspondem à sua utilização.

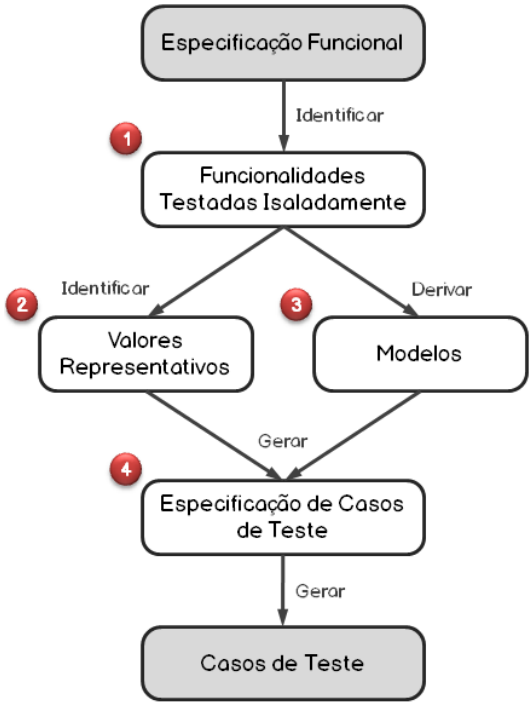

Figura 2.1: Etapas envolvidas numa abordagem sistemática ao desenho de testes funcionais

2, 3 - Identificar Valores Representativos ou Derivar Modelo

Enumeração dos valores de cada input de forma isolada não seleccionando combinações de valores de inputs. A identificação pode ser feita: (i) explicitamente, devendo ser considerados valores de diferentes categorias, valores limite ou susceptiveis de causar erro ou (ii) implicitamente, através da construção de modelos, que podem ou não já existir na especificação, sendo uteis no processo de combinação de valores.

4 - Definir Especificação de Casos de Teste

Combinação de valores para todos os inputs do componente em teste. No contexto do teste de uma página de uma aplicação, a especificação de casos de teste consiste na combinação dos valores possíveis para os inputs da página. A selecção de combinações deverá ser suportada por um conjunto de restrições que eliminem combinações inválidas ou impossíveis

2.4 Suporte Para Automação

A automação de testes funcionais refere-se à implementação e/ou utilização de ferramentas ou frameworks capazes de executar testes sem intervenção humana [2]. No contexto das aplicações web essas ferramentas são capazes de controlar ou

emular um web browser, verificando que o comportamento da aplicação é o correcto.

Existem dois tipos de abordagens para a automação de testes funcionais: (i) com recurso a ferramentas suportadas no mecanismo *Record & Playback*; e (ii) suportada em bibliotecas que visam a criação de *frameworks* de teste.

A utilização de ferramentas suportadas em mecanismos *Record & Playback* (e.g. SeleniumIDE[1], TestComplete[2]) baseiam-se na sua maioria na criação de scripts que são gravados enquanto o utilizador interage com a aplicação, podendo os mesmos ser executados posteriormente sem intervenção humana. Cada script corresponde a uma sequência de acções, descritas numa linguagem específica à ferramenta, contendo os dados utilizados no teste. Este tipo de abordagens apresenta diversas desvantagens, entre as quais, o facto dos scripts estarem comprometidos com a sequência de eventos que ocorreu durante a sua gravação ou promoverem a repetição de acções.

A criação de *frameworks* de testes, suportadas em bibiotecas, surge assim como uma alternativa escalável [13]. A sua implementação é feita com recurso a linguagens de programação (e.g. C#, Java), tirando partido das suas vantagens e desvantagens. Factores como a reutilização e a manutenção da *test suite* devem ser tidos em consideração na criação de uma *framework* de automação, estando algumas técnicas para o conseguir descritas em baixo.

Reutilização

As páginas podem ser testadas várias vezes havendo apenas diferença nos valores de entrada, provocando a existência de testes muito semelhantes. A *framework* de automação deverá utilizar apenas um teste que passará a receber valores de entrada executando assim os vários casos de teste.

Existem operações comuns a vários testes. A decomposição do teste em acções permite isolar este tipo de operações, permitindo a sua reutilização em outros testes. A reutilização de acções repetidas na *test suite* deve ser assim considerada em alternativa a codifica-las múltiplas vezes. No limite existirão testes compostos por associações de acções.

Manutenção

Um cenário comum na criação de testes é a criação de dependências dos scripts com a estrutura das páginas. A falta de abstracção na referência a elementos na página, quer seja através da sua localização quer seja através

[1] `http://docs.seleniumhq.org/projects/ide`

[2] `http://smartbear.com/products/qa-tools/automated-testing-tools`

de um atributo, pode obrigar a alteração não desejável dos scripts de teste que os referenciam quando a estrutura da página muda. Tal alteração pode assumir-se como um esforço muito significativo quando se trate de uma *test suite* de dimensões consideráveis. Outra desvantagem prende-se com o aumento do período em que não há visibilidade relativa às alterações introduzidas. Considera-se uma qualidade da *framework* de automação a capacidade de definir uma camada capaz de abstrair estes aspectos [13]. Desta forma, o acesso aos elementos das páginas passa a ser feito através desta camada aplicacional, sendo aqui reflectidas as alterações à estrutura e conteúdo das páginas e não nos testes.

2.4.1 HttpUnit

O HttpUnit possibilita o teste de aplicações web emulando comportamentos relevantes dos browsers para simular a utilização de uma aplicação. A utilização desta framework permite fazer pedidos ao servidor, tratando as respostas como texto ou XML DOM, ou como páginas. Neste último caso, é disponibilizada uma API que permite identificar elementos presentes na estrutura da página, especificamente para o caso de formulários ou tabelas e genericamente, através de valores de atributos.

2.4.2 Selenium WebDriver

O Selenium WebDriver é uma biblioteca de automação de testes que permite ao programador ter controlo do browser através da API WebDriver. A utilização da biblioteca não necessita de instalação de componentes adicionais, podendo ser referenciada como uma bilioteca convencional. A existência de implementações da API WebDriver para a maioria dos browsers (desktop ou mobile) permite verificar o comportamento das aplicações em diferentes plataformas, sendo este um dos principais pontos fortes desta abordagem. No que respeita ao suporte à biblioteca, foi publicada recentemente a especificação incial da API pelo W3C [27].

O controlo dos browsers é possível em vários aspectos, destacando-se os seguintes: abrir e fechar janelas; inspecção e manipulação do DOM, execução de JavaScript, definir input automático do utilizador ou simular eventos do teclado e do rato.

2.4.3 Cucumber

O Cucumber é uma ferramenta que suporta a adopção do processo de desenvolvimento *Behavior Driven Development*. Este processo caracteriza-se por promover a criação de testes descritos em linguagem natural, tornando-os mais acessíveis a intervenientes mais ligados ao negócio e sem conhecimentos técnicos [11]. Assim, a especificação dos comportamentos do sistema numa linguagem alto nível facilita a comunicação e colaboração na equipa, contribuindo para um melhor entendimento do que o produto deve fazer pelos utilizadores.

A especificação da funcionalidade é feita através da definição de qual o comportamento esperado do sistema, exemplificando-o recorrendo a cenários de teste compostos por passos que promovem a sua execução. Apesar de ser utilizada linguagem natural, a especificação dos passos a seguir deve respeitar regras de sintaxe que permitam ao Cucumber a sua interpretação. Em conjunto com a especificação da funcionalidade, o Cucumber utiliza a definição dos passos para fazer o mapeamento da linguagem relativa ao negócio com uma implementação que será responsável por concretizar a automação.

2.4.4 Outras ferramentas analisadas

Da utilização das ferramentas mencionadas identificaram-se outras que, pela as suas semelhanças, optou-se por não detalhar. Contudo, considera-se importante menciona-las, por terem utilização significativa na indústria. *Breve desrição relativa a Watir, Specflow*

2.5 Trabalho Relacionado

Neste secção será feita uma apresentação superficial de trabalhos analisados que, não tendo directamente a ver com o foco da tese, consideram-se relevantes nesta área de estudo.

Capítulo 3

Caso de Estudo

Esta secção descreve a aplicação que será usada como caso de estudo para ilustrar o desafio de dar suporte automático à criação de testes. A funcionalidade geral da aplicação é descrita e as suas *user stories* definidas. Será explicado o modelo da aplicação em OutSystems Platform, fazendo assim uma breve introdução às suas capacidades de geração de aplicações web. Serão definidos testes funcionais a partir de *user stories* escolhidas. Para cada *framework* de teste HttpUnit, Selenium WebDriver e Cucumber, será discutida a implementação de testes funcionais seleccionados de forma a evidenciar os problemas que pretendemos solucionar com este trabalho.

3.1 Aplicação Directory

A *Directory* é uma aplicação que tem como principal objectivo centralizar a informação de contacto dos colaboradores da organização fictícia *ObserveEverything*. A *ObserveEverything* está dividida em departamentos, estando estes por sua vez repartidos em unidades organizacionais. A Figura 3.1 mostra as janelas principais da aplicação, entre as quais: (1) Página de Login - onde é feita a autenticação do colaborador na aplicação especificando o nome de utilizador e palavra passe; (2) Página Principal - onde são listados os colaboradores da empresa, sendo possível filtrá-los por nome ou departamento e; (3) Página de Edição de Colaborador - onde é possível proceder à alteração dos dados de um colaborador.

A metodologia adoptada para a especificação da aplicação foi baseada na definição de requisitos na forma de histórias de utilização. A história de utilização especifica, em linguagem natural, uma actividade que o utilizador precisa de fazer para desempenhar uma tarefa recorrendo ao sistema, especificando o tipo de utilizador,

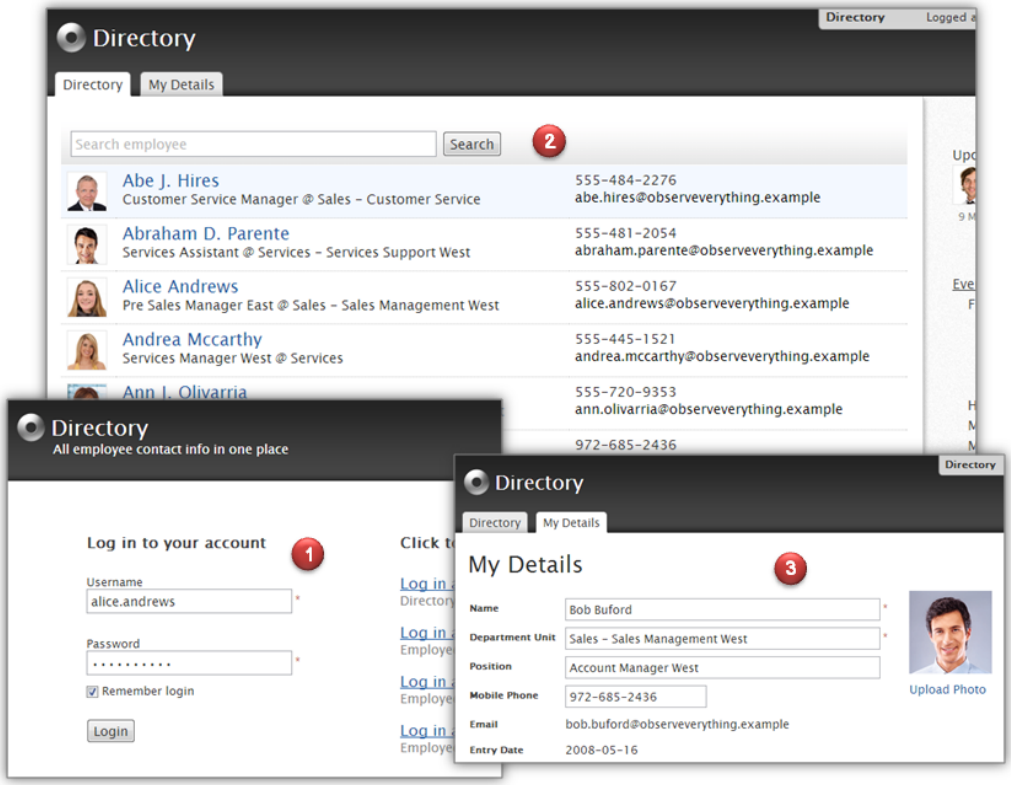

Figura 3.1: Interface gráfica da aplicação *Directory* mostrando: (1) a página de login; (2) a página principal e; (3) a página de edição de colaborador.

a funcionalidade e a sua motivação [26]. Com o objectivo de demonstrar a criação de testes com recurso a diferentes ferramentas utilizadas na indústria, foram seleccionadas duas histórias de utilização presentes na especificação da aplicação *Directory*:

1. Como colaboradora, a Alice precisa de saber quem é o colega responsável pelo website, para que consiga dar a conhecer uma anomalia;
2. Como colaborador, o Bob precisa de editar os seus dados, para que a sua informação de contacto se mantenha actualizada.

3.2 Modelo da Aplicação

A modelação da aplicação *Directory* foi feita com recurso à linguagem da OutSystems usando o ServiceStudio—o ambiente de desenvolvimento da OutSystems Platform. O ServiceStudio permite construir de forma visual um modelo da aplicação, definindo a interface de utilização e os elementos que a compõem, a lógica responsável por concretizar os comportamentos expostos e os objectos do

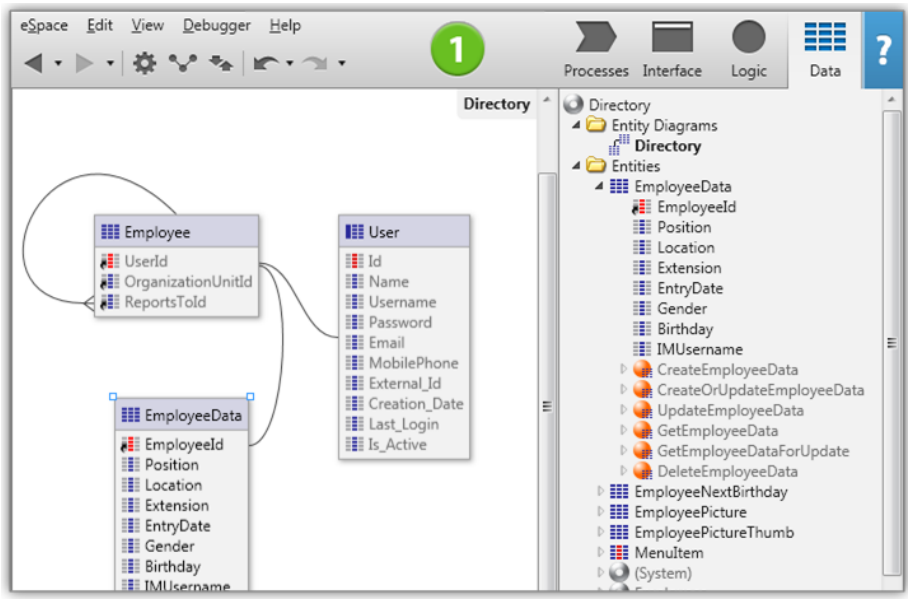

Figura 3.2: Modelação da camada de acesso a dados no ServiceStudio. Representação de entidades, relações, atributos e acções de entidade que permitem operações de leitura, inserção, edição e remoção.

domínio da aplicação. O modelo da aplicação é utilizado pelo PlatformServer no processo de compilação e geração de código, dando origem a diferentes artefactos como código Java ou .NET, páginas HTML e/ou scripts SQL. Assim, o processo de geração de código cria um mapeamento entre o modelo da aplicação definido no ServiceStudio e a sua implementação final, quer seja em código servidor, páginas web ou tabelas do modelo de dados.

Os elementos disponíveis para a criação da aplicação podem referir-se às camadas de acesso a dados (entidades), apresentação (*WebScreens* e *WebBlocks*) e lógica de negócio (acções).

3.2.1 Entidades

A Figura 3.2 mostra do lado esquerdo um diagrama de entidades que é gerado automaticamente pelo ServiceStudio a partir das entidades criadas no contexto da aplicação *Directory*, nomeadamente: (i) *User* - representa um utilizador do sistema; (ii) *EmployeeData* - representa um colaborador e; (iii) *Employee* - relaciona um utilizador do sistema com os seus dados de colaborador e de departamento. Estas entidades representam objectos do dominio da aplicação, constituindo a camada de acesso a dados. A sua criação implica a definição de atributos e, opcionalmente, relações com outras entidades. Como elementos do modelo da aplicação, as entidades são utilizadas no processo de geração de código para: (i) criar tabelas na base de dados, responsáveis pela persistência dos dados; e (ii)

criar acções de entidade, que permitem operações de consulta, edição, inserção e remoção de registos.

Na Figura 3.2 é apresentado no lado direito o detalhe da entidade *EmployeeData*, onde são mostrados os seus atributos e as acções de entidade, geradas automaticamente, que permitem a manipulação de registos na base de dados. A manipulação dos dados é feita através do uso de acções de entidades, utilizadas na camada lógica da aplicação.

3.2.2 WebScreens e WebBlocks

Os *WebScreens* e *WebBlocks* constituem a camada de apresentação da aplicação, representando as suas páginas, sendo a sua criação feita com recurso às *widgets*. As *widgets* viabilizam a apresentação, aquisição e estruturação de dados na interface de utilização. Os *WebScreens* representam páginas da aplicação que podem ser acedidas directamente — através do seu URL — ou indirectamente —como destino de uma navegação. Os *WebBlocks* representam fragmentos da interface de utilização que se repetem em várias páginas, não estando acessíveis de forma directa ou indirecta, tendo como objectivo permitir a reutilização destes fragmentos, centralizando a sua definição. A apresentação de um *WebBlock* na interface de utilização só é possível quando incluído na definição de um *WebScreens*, em conjunto com outras *widgets*.

A Figura 3.3 mostra a criação de um *WebScreen* no ServiceStudio, que corresponde ao ecrã principal da aplicação representado na Figura 3.1. A sua criação é feita com recurso a diferentes tipos de elementos visuais, nomeadamente *widgets*, destacando-se: (1) *Expression* — Apresenta um valor avaliado em tempo de execução, sendo neste exemplo o departamento a que o colaborador pertence; (2) *Input* — Permite a introdução de texto de um tipo específico, suportando neste exemplo a pesquisa de colaboradores; e (3) *Hyperlink* — Utilizado para submissão de dados, navegação ou execução de acção, permitindo neste exemplo seleccionar um colaborador apresentando informação com maior detalhe.

No processo de geração de código os *WebScreens* e *WebBlocks* são utilizados para dar origem às páginas da aplicação e as *widgets* para a criação de elementos HTML que as constituem. A criação de testes funcionais a aplicações web baseia-se na interacção com os elementos HTML da página que, no contexto da uma aplicação web gerada com a OutSystems Platform, representam as *widgets* definidas no ServiceStudio. Com este trabalho pretende-se permitir a criação de testes funcionais que, em vez de fazerem referência directa ao HTML das páginas, façam referência

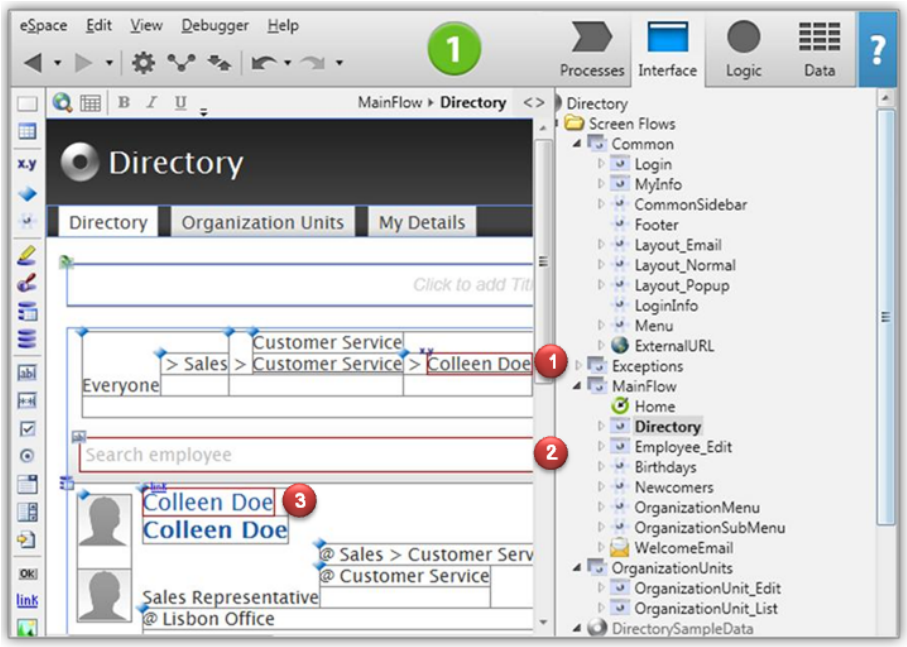

Figura 3.3: Modelação do ecrã principal da aplicação no ServiceStudio. Exemplo de utilização de *widgets* entre as quais: (1) Expressão avaliada em tempo de execução; (2) Introdução de texto; (3) *Hyperlink* para submissão de dados, execução de acção ou navegação.

à *framework* de testes, gerada com base na informação do modelo da aplicação definido no ServiceStudio, sendo esta a principal contribuição.

3.2.3 Acções

As acções representam a camada de lógica de negócio existindo diferentes tipos de acções, entre as quais, acções de preparação, de ecrã, de aplicação e de entidade. As acções de preparação existem no contexto de uma página e permitem a execução de lógica responsável por, entre outros, reunir os dados a apresentar na página que representam. As acções de ecrã apenas estão disponíveis no contexto do ecrã onde foram criadas, podendo estar associadas a eventos de *widgets*, nomeadamente ao click de botões ou de *hyperlinks*. As acções de aplicação podem ser invocadas em toda a aplicação, sendo tipicamente utilizadas para promover a reutilização de lógica. As acções de entidade permitem a manipulação dos dados que as entidades representam.

A Figura 3.4 apresenta a acção de preparação que contém a lógica responsável por preparar os dados a serem apresentados pela página *Employee_Edit*. A sua implementação é feita com recurso a um grafo em que as arestas representam os fluxos de execução possíveis e os vértices as operações a serem realizadas, destacando-se: (1) *Query* - Permite obter registos de colaboradores; (2) *If* - Criação de novo

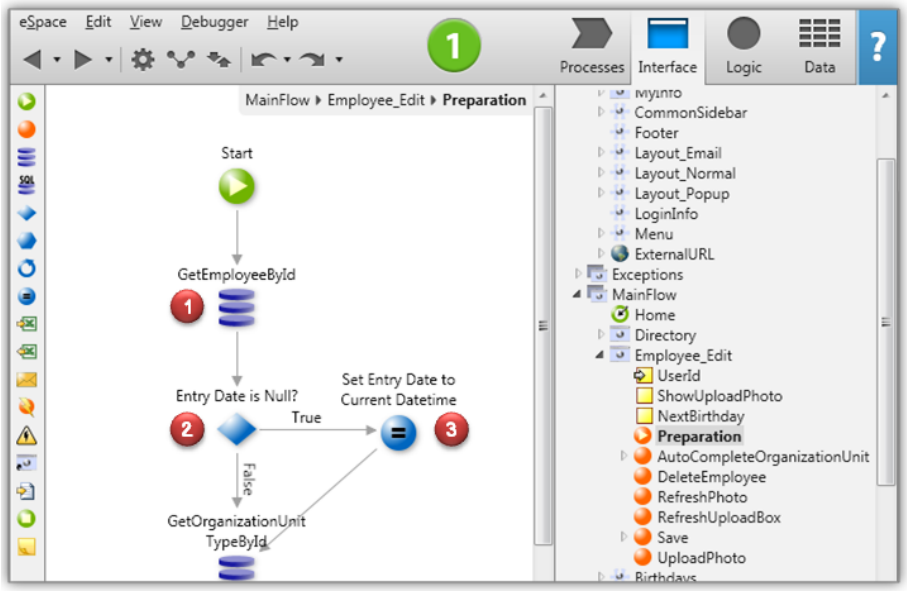

Figura 3.4: Modelação da camada de lógica de negócio no ServiceStudio com o exemplo de uma acção de preparação de dados a apresentar na página *Employee_Edit*. Da implementação da acção destacam-se as seguintes operações: (1) Obtenção de registos de colaboradores; (2) Execução condicional e; (3) Atribuição de valores a variáveis.

fluxo de execução baseado numa condição e; (3) *Assign* - Atribuição de valores a variáveis.

3.3 Modelo de Navegação

O modelo da aplicação definido no ServiceStudio inclui diversos elementos, entre os quais: entidades, *WebScreens* e *WebBlocks* e *Actions*, representando respectivamente, as camadas de acesso a dados, apresentação e lógica de negócio. A OutSystems Platform faz uso destes elementos no processo de geração automática de código para dar origem a artefactos como C# ou Java, scripts SQL e páginas HTML. O ServiceStudio utiliza o modelo da aplicação para, com base nas páginas e elementos que promovem transições entre páginas, criar automaticamnente um modelo de navegação da aplicação.

A Figura 3.5 mostra o modelo de navegação da aplicação *Directory* onde estão representados elementos que pertencem à aplicação, bem como relações entre eles, destacando-se os seguintes: (1) *Entry Point*, que aponta para a página principal da aplicação; (2) *WebScreen*, página da aplicação que, para além das *widgets* definidas no seu modelo, inclui fragmentos reutilizáveis da interface de utilização representados pelos *WebBlocks* por ela apontados, permitindo a navegação para a página *Employee_Edit*; (3) *MailScreen*, página que representa um e-mail a ser

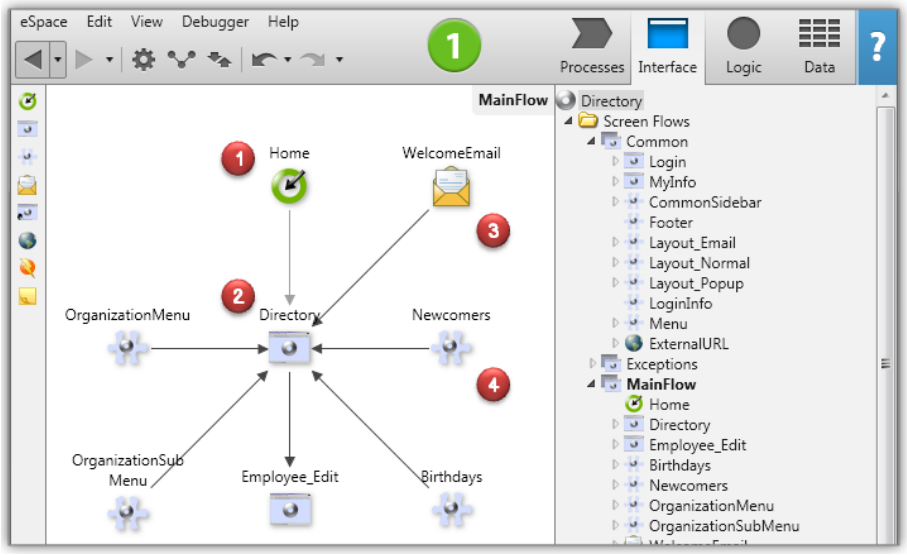

Figura 3.5: Modelo de navegação da aplicação *Directory* no ServiceStudio representando relações entre páginas e elementos visuais, destancado-se os seguintes elementos da linguagem: (1) *Entry Point*; (2) *WebScreen*; (3) *MailScreen*; e (4) *WebBlock*.

enviado; (4) *WebBlock*, elemento reutilizável instanciado na página *Directory*.

Na Figura 3.5 é possível identificar que na página 'Directory', o ponto de entrada na aplicação, sendo apontada pelo *entry point* '*Home*', é possível navegar para a página '*Employee_Edit*'. No contexto do teste de aplicações web, o modelo de navegação permite identificar as páginas que pertencem à aplicação, facilitando a identificação de sequências de páginas a serem visitadas num caso de teste.

3.4 Especificação e Implementação de Testes Funcionais

A aplicação *Directory* faz parte de um conjunto de aplicações de intranet [1] que têm como objectivo facilitar a demonstração não só das capacidades da plataforma, mas também das aplicações nela criadas. Para estas aplicações foi assegurada a existência de dados de exemplo (*seed data*) que simulassem um cenário real, facilitando a criação de novos dados e, neste caso particular, de testes.

Com o objectivo de especificar os testes funcionais necessários a verificar as histórias de utilização enumeradas em 3.1, seguiu-se a abordagem sistemática proposta na secção 2.3.1 para cada uma has histórias.

De seguida, serão apresentadas as história de utilização seleccionadas que servirão

[1] `http://www.outsystems.com/apps`

Figura 3.6: Páginas envolvidas na história de utilização de consulta de dados e respectivos *inputs*

para demonstrar a implementação do teste que as representa recorrendo a três *frameworks* de teste utilizadas na indústria.

User Story #1 - 'Como colaboradora, a Alice precisa de saber quem é o colega com o cargo Online Operations Manager, para que consiga dar a conhecer uma anomalia'

Para esta história de utilização, cujo objectivo é a leitura de dados, a Figura 3.6 ilustra o caminho percorrido pelo utilizador na aplicação. Este terá sido obtido através da análise do modelo de navegação, estando representadas na figura as páginas envolvidas. Os valores de entrada necessários foram identificados no modelo da aplicação tendo o seu valor, apresentado em baixo, sido obtido da observação dos dados de exemplo da aplicação. A Figura 3.6 representa a especificação do caso de teste da seguinte forma: (1) Login - Autenticação do utilizador tendo como entradas o nome de utilizador '`alice.andrews`' e a password '`outsystems`'; (2) Lista de Áreas - Selecção da área '`Online Operations`' no ecrã principal; (3) Dados do Colaborador - Verificar que na apresentação de dados de colaborador '`Larry R. Logan`' a sua posição é '`Online Operations Manager`'.

User Story #2 - 'Como colaborador, o Bob precisa de editar os seus dados, para que a sua informação de contacto se mantenha actualizada'

No que respeita à concretização desta história de utilização, relativa à escrita de dados, é representado na Figura 3.7 o caminho percorrido pelo utilizador, tendo sido obtido da mesma forma que na história anterior, bem como os valores de entrada necessários. A Figura 3.7 representa a especificação do caso de teste da seguinte forma: (1) Login - Autenticação do utilizador tendo como entradas o nome de utilizador '`bob.bufford`' e a password '`outsystems`'; (2) Menu - Acesso

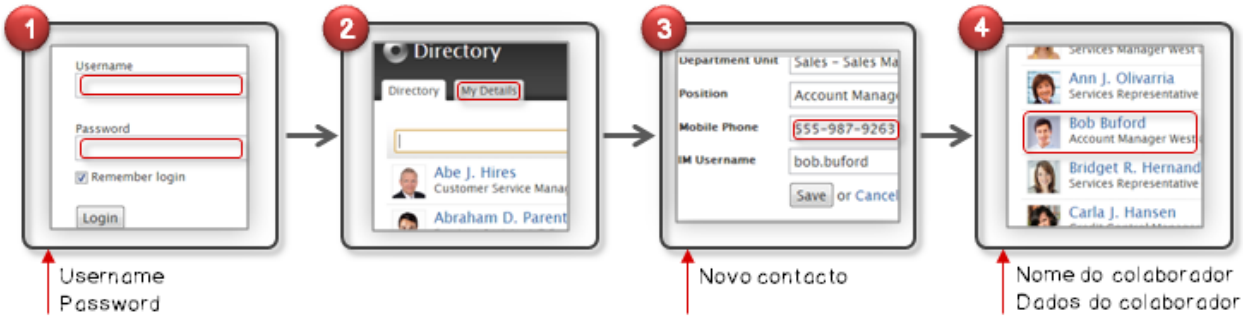

Figura 3.7: Páginas envolvidas na história de utilização de edição de dados

ao menu '`My Details`' no ecrã principal; (3) Edição de dados - Alterar contacto do colaborador para '223–584–3990'; (4) Dados do Colaborador - verificar que o novo contacto é o apresentado nos dados do colaborador.

3.4.1 HttpUnit

O HttpUnit é uma *framework* que simula um browser, vocacionada para operar num ambiente web. Viabiliza o teste de aplicações pela troca de pedidos HTTP com o servidor, manuseando as respostas directamente, não concretizando assim a renderização das páginas. O HTML retornado é disponibilizado nos formatos texto ou de documento XML, sendo adicionalmente oferecida uma API destinada à identificação e interacção com os elementos que o constituem.

A listagem 3.1 mostra a implementação da história de utilização 1, conforme Secção 3.1, utilizando a *framework* HttpUnit. Na base da sua utilização está uma instância de `WebConversation` (linha 4) que representa o objecto responsável por emular o browser e os seus comportamentos. O teste inicia-se com o pedido de acesso à página principal da aplicação (linha 7) sendo retornada a página de Login. O Login na aplicação consiste no preenchimento dos campos relativos ao nome de utilizador e palavra passe e envio para validação através da invocação do evento *click* do botão rotulado com 'Login'. A localização do *input* relativo ao *username* é feita com recurso ao seu identificador (linha 8), sendo o seu conteúdo alterado pela modificação do atributo '*value*' (linha 9). O preenchimento da palavra passe é feito da mesma forma nas linhas 10 e 11. Note-se que, estas alterações implicam o conhecimento prévio do localizador dos *inputs* na página, neste caso o seu identificador, bem como o atributo que reflecte o seu estado, neste caso o atributo *value*. A submissão dos dados de Login é feita nas linhas 12 e 13 pela localização do botão de Login e invocação do evento de '*click*'. Com a validação dos dados de Login é retornada a página principal da aplicação, obtida através da instância de `WebConversation` (linha 16). Na página principal é localizado o *hyperlink*

com o texto *'Online Operations'* (linha 17), que representa a área organizacional da qual queremos identificar o colaborador responsável, sendo sobre ele invocado o evento de *click*, que dá origem à página que mostra colaboradores dessa área, obtida na linha 18. O teste termina com a verificação nas linhas 21 e 22 de que o colaborador *'Larry R. Logan'* tem a posição *'Online Operations Manager'*, comparando o conteúdo dos elementos específicos da página com os valores de entrada do teste.

Listagem 3.1: Implementação Java de teste funcional para história de utilização de consulta de dados relativos a *Online Operations Manager* com recurso HttpUnit

```
1   @Test
2   public void searchForOnlineOperationsManager() throws Exception{
3
4     WebConversation testSession = new WebConversation();
5
6     // Login
7     WebResponse loginPage = testSession.getResponse("http://localhost/
          Directory");
8     HTMLElement inputUsername = loginPage.getElementWithID("
          wt12_wtMainContent_wtUserNameInput");
9     inputUsername.setAttribute("value", "alice.andrews");
10    HTMLElement inputPassword = loginPage.getElementWithID("
          wt12_wtMainContent_wtPasswordInput");
11    inputPassword.setAttribute("value", "outsystems");
12    Button buttonLogin = loginPage.getFormWithID("WebForm1").
          getButtonWithID("wt12_wtMainContent_wtLoginButton");
13    buttonLogin.click();
14
15    // Selecção de área da organização
16    WebResponse mainPage = testSession.getCurrentPage();
17    mainPage.getLinkWith("Online Operations").click();
18    WebResponse orgPage = testSession.getCurrentPage();
19
20    // Verificação de dados do colaborador
21    assertEquals("Larry R. Logan", orgPage.getElementWithID("
          wt114_wtMainContent_wtEmployeeTable_ctl04_wtEmployeeName").
          getText());
22    assertEquals("Online Operations Manager", orgPage.getElementWithID("
          wt114_wtMainContent_wtEmployeeTable_ctl04_wtEmployeePosition").
          getText());
23  }
```

A criação do teste com recurso à *framework* HttpUnit identificaram-se fortes dependências com a estrutura da página, nomeadamente com os identificadores dos elementos que viabilizam a sua localização. Nesse sentido, modificações sobre a aplicação que provoquem alterações que tenham impacto sobre a localização de elementos (e.g. valor dos identificadores) poderão fazer com que os testes que os utilizam deixem de passar. Além desta situação obrigar a esforço de manutenção dos testes, aumenta o tempo sem visibilidade sobre alterações à aplicação.

3.4.2 Selenium WebDriver

O Selenium *WebDriver* é uma *framework* que permite o controlo directo de um browser, permitindo criar testes que simulam interacções de utilizadores com a aplicação. O controlo do browser é possível através da manipulação programática de um objecto que implementa a interface `WebDriver`. A uma implementação dessa interface dá-se o nome de *driver* e, existem diversas implementações para os browsers mais utilizados, entre os quais, *Internet Explorer*, *Firefox*, *Google Chrome* ou *Android*. A execução de testes em diferentes browsers torna-se assim bastante simples, bastando para isso a instanciação de um novo *driver*, i.e., uma nova implementação da interface.

A interface `WebDriver` especifica comportamentos capazes de, entre outros: i) solicitar uma página; ii) localizar elementos na página actual utilizando diferentes tipos de localizadores (e.g. nome, selector CSS ou XPath, identificador); iii) controlar a janela do browser e, iv) definir mecanismos de *timeout* relativos à pesquisa de elemento, carregamento de páginas ou execução assíncrona de *scripts*.

A listagem 3.2 mostra a implementação da história de utilização 1, conforme Secção 3.1, utilizando a *framework* Selenium *WebDriver*. O controlo do browser é feito com recurso à instância criada na linha 4, que implementa a *interface* `WebDriver`, responsável controlar o browser *FireFox*. Na linha 5 ordena-se ao browser que faça o pedido da página principal da aplicação sendo que, uma vez que ainda não existe uma sessão de utilizador estabelecida, a página de Login é retornada por lógica aplicacional específica para este efeito. O Login é feito pela localização do elemento capaz de aceitar o *username* (linha 8), recorrendo ao selector `By.id` que verifica o valor do atributo '*id*' do elemento. A alteração do conteúdo do elemento é feita através da invocação do método `sendKeys` (linha 9), introduzindo o nome de utilizador do colaborador, valor de entrada do teste, da mesma forma que um utilizador o faria. A submissão de valores é feita nas linhas 12 e 13 localizando o botão de Login e invocando o evento de '*click*' sobre este elemento. A localização do botão de Login recorre ao selector `By.cssSelector`, identificando o texto apresentado no botão. Após selecção da área organizacional correcta, o teste termina com a verificação nas linhas 19 a 21 dos valores apresentados, comparando o conteúdo dos elementos com os valores de entrada do teste.

Listagem 3.2: Implementação Java de teste funcional para consulta de dados relativos a *Online Operations Manager* com recurso a Selenium WebDriver

```
1   @Test
2   public void searchForOnlineOperationsManager() {
3
4     WebDriver driver = new FirefoxDriver();
5     driver.get("http://localhost/Directory");
6
7     // Login
8     WebElement inputUsername = driver.findElement(By.id("
          wt11_wtMainContent_wtUserNameInput"));
9     inputUsername.sendKeys("alice.andrews");
10    WebElement inputPassword = driver.findElement(By.id("
          wt11_wtMainContent_wtPasswordInput"));
11    inputPassword.sendKeys("outsystems");
12    WebElement buttonLogin = driver.findElement(By.cssSelector("input[
          value='Login']"));
13    buttonLogin.click();
14
15    // Selecção de área da organização
16    driver.findElement(By.linkText("Online Operations")).click();
17
18    // Apresentação de dados do colaborador
19    assertEquals("Online Operations", driver.findElement(By.cssSelector(
          "div.BreadcrumbText a span")).getText());
20    assertEquals("Larry R. Logan", driver.findElement(By.id("
          wt104_wtMainContent_wtEmployeeTable_ctl04_wtEmployeeName")).
          getText());
21    assertEquals("Online Operations Manager", driver.findElement(By.id("
          wt104_wtMainContent_wtEmployeeTable_ctl04_wtEmployeePosition")).
          getText());
22  }
```

Do ponto de vista do teste funcional de aplicações web, a *framework* Selenium *WebDriver* permite o controlo de diferentes tipos de browsers, sendo esta a diferença mais significativa quando comparada com o HttpUnit, simulando a interacção de um utilizador. Contudo, a utilização de identificadores de elementos (e.g. listagem 3.2, linha 8), valores de atributos ou classes de estilo (e.g. listagem 3.2, linha 12) cria dependência com o HTML das páginas. Esta dependência cria problemas de manutenção dos testes e aumento do tempo sem visibilidade sobre alterações à aplicação. Os problemas de manutenção podem, de certa forma ser diminuídos, devido à estruturação do código de teste, diminuindo o número de alterações necessárias causadas por mudança da aplicação, todavia, a dependência existe e, pequenas alterações à aplicação podem significar esforço adicional na adaptação dos testes.

3.4.3 Cucumber

Um dos objectivos principais no desenho da *framework* Cucumber foi o de permitir a compreensão e criação de testes por pessoas com diferentes níveis de conhecimento da aplicação [11]. Nesse sentido, a execução de testes tem em consideração duas camadas distintas: (i) uma alto-nível, destinada a facilitar a comunicação entre áreas técnicas e áreas ligadas ao negócio; e outra (ii) baixo-nivel, que permite a criação de código capaz de invocar código da aplicação. A camada alto-nível permite a descrição dos passos envolvidos num cenário de teste em linguagem natural, agrupando os cenários em *features*. Na camada baixo-nivel é feita a correspondência entre acções descritas em linguagem natural e operações sobre a aplicação através de código de teste.

Listagem 3.3: Descrição da história de utilização de consulta de informação relativa a gestor de área organizacional com recurso a Cucumber

```
1 Feature: Organizational Units
2
3   Scenario: Search for Online Operations responsible
4     Given I am logged in as "Alice"
5     When I select the "Online Operations" organizational unit
6     Then employee "Larry R. Logan" is listed as "Online Operations Manager"
```

A listagem 3.3 representa a camada alto-nível, com a descrição dos passos do cenário de teste. A linguagem utilizada na descrição, chamada *Gherkin*, especifica o contexto -*Given*- a acção a ser realizada -*When*- e o resultado esperado -*Then*. O teste especifica que o utilizador autenticado é a 'Alice' (linha 3) e que quando é seleccionada a área organizacional '*Online Operations*' (linha 5) então o colaborador '*Larry R. Logan*' aparece listado com o titulo '*Online Operations Manager*' (linha 6).

A listagem 3.4 contém a definição dos passos do cenário apresentado na listagem 3.3. A definição de cada passo começa com a respectiva palavra chave: *Given*, *When* ou *Then*) e com uma expressão regular que permite identificar os passos capazes de ser controlados. A implementação utiliza a *framework* de automação *Capybara* [2] para simular a interacção de um utilizador com a aplicação, utilizando *drivers* que implementam a especificação *WebDriver*. O Login na aplicação é feito nas linhas 5 a 10, identificando os *inputs* responsáveis por receber o username e a password através dos seus identificadores, preenchendo-os com o valor correcto e invocando o evento de *click* do botão com o texto 'Login' e verificando que a página principal foi devolvida. Após selecção da área organizacional correcta, definida na linha 12, o teste termina com a verificação nas linhas 17 a 19 de que

[2] `https://github.com/jnicklas/capybara`

Listagem 3.4: Definição em Ruby dos passos envolvidos na realização da história de utilização de consulta de informação relativa ao *Online Operations Manager* com recurso a Cucumber

```
1  Before do
2    visit 'http://localhost/Directory'
3  end
4
5  Given /^I am logged in as "Alice"$/ do |name|
6    fill_in 'wt12_wtMainContent_wtUserNameInput', :with => 'alice.andrews'
7    fill_in 'wt12_wtMainContent_wtPasswordInput', :with => 'outsystems'
8    click_button "Login"
9    current_path.should == '/Directory/Directory.aspx'
10 end
11
12 When /^I select the "(.*)" organizational unit$/ do |areaName|
13   click_link areaName
14 end
15
16 Then /^employee "(.*)" is listed as "(.*)"$/ do |empName, position|
17   find(:css, 'div .BreadcrumbText a span').text.should == areaName
18   find_by_id('wt114_wtMainContent_wtEmployeeTable_ctl04_wtEmployeeName').text.
       should == empName
19   find_by_id('wt114_wtMainContent_wtEmployeeTable_ctl04_wtEmployeePosition').text.
       should == position
20 end
```

os valores apresentados correspondem aos valores de entrada do teste.

No Cucumber a descrição dos testes é facilmente compreendida pela utilização de uma linguagem alto-nível, permitindo não só que as mesmas sejam utilizadas por pessoas ligada ao negócio, mas também, incorporar termos do dominio da aplicação. Todavia, a localização de elementos baseada em identificadores (e.g. listagem 3.4, linha 6 e 7), valores de atributos (e.g. listagem 3.4, linha 8) ou posição na página (e.g. listagem 3.4, linha 17) compromete o teste com o HTML das páginas. Alterações à aplicação que impliquem modificações nos localizadores utilizados irão traduzir-se num esforço na adaptação dos testes, tal como já acontecia para o HttpUnit e Selenium Webdriver, bem como, em falta de visibilidade sobre o impacto dessas alterações.

3.5 Limitações e melhorias das soluções analisadas

No contexto dos testes funcionais, as *frameworks* analisadas permitem o teste de interacção do utilizador com a aplicação, quer pela emulação ou manipulação directa de um browser, através de uma API. Todavia, identificou-se em todas as *frameworks* uma forte dependência da implementação dos testes com o HTML das páginas manipuladas, sendo este o problema que se propõe resolver com este

trabalho. A localização de elementos nas páginas recorre frequentemente a selectores XPath ou CSS que fazem uso de identificadores, valores de atributos, classes de estilo ou posição relativa dos elementos na página.

Alterações à aplicação que provoquem a modificação dos atributos utilizados na localização dos elementos na página, ou da sua posição relativa, provocarão erros na execução dos testes por não serem encontrados os elementos com o qual deve interagir. A resolução destes erros passa por actualizar todas as referências ao HTML alterado exigindo esforço na manutenção de testes que, até estarem consistentes com as páginas da aplicação, não serão executados com sucesso, aumentando o tempo sem visibilidade sobre o efeito das alterações à aplicação.

No contexto do presente trabalho optou-se por tirar partido da *framework* Selenium *WebDriver* uma vez que expõe uma API abrangente no que respeita a manipulação e localização de elementos no DOM. Adicionalmente, a possibilidade de permitir de forma simples a manipulação de diferentes browsers, *desktop* ou *mobile*, foi considerada uma vantagem importante face às outras *frameworks* apresentadas.

Capítulo 4

Solução Proposta

Esta secção introduz a *OutSystems Web-Application Test Framework*, a solução proposta para o problema da dependência dos testes com o HTML das páginas. Será apresentada a sua arquitectura, explicam-se os elementos que nela participam e a forma como comunicam. Os princípios envolvidos na criação da *framework* serão descritos identificando-se padrões de desenvolvimento em uso na indústria, e a forma como se aplicam à solução proposta, bem como, padrões identificados durante o desenvolvimento deste trabalho. Da análise das *widgets* utilizadas no ServiceStudio para a construção de páginas serão identificadas as propriedades relevantes no contexto de um teste, detalhando-se o mapeamento com a sua implementação na *framework* de testes. No fim será revisitada a aplicação Directory, apresentada em 3.1, com o objectivo de demonstrar a utilização da *OutSystems Web-Application Test Framework* aplicada a uma aplicação real.

4.1 Arquitectura

A solução proposta visa retirar dos testes referências directas à estrutura das páginas da aplicação em teste, fazendo com que alterações à aplicação não tenham impacto nos testes mas sim na *framework* de testes, que pode ser gerada automaticamente a partir do modelo da aplicação definido no ServiceStudio. Direccionada à criação de testes funcionais, a *OutSystems Web-Application Test Framework* testes suportará a criação de testes funcionais ao nível do sistema. A Figura 4.1 apresenta a arquitectura da solução proposta, da qual fazem parte os seguintes elementos: (1) o modelo da aplicação definido no ServiceStudio; (2) o processo de geração da *framework*, a ser definido no presente trabalho; (3) a *framework* de testes gerada a partir do modelo da aplicação; (4) o *core* da *framework*

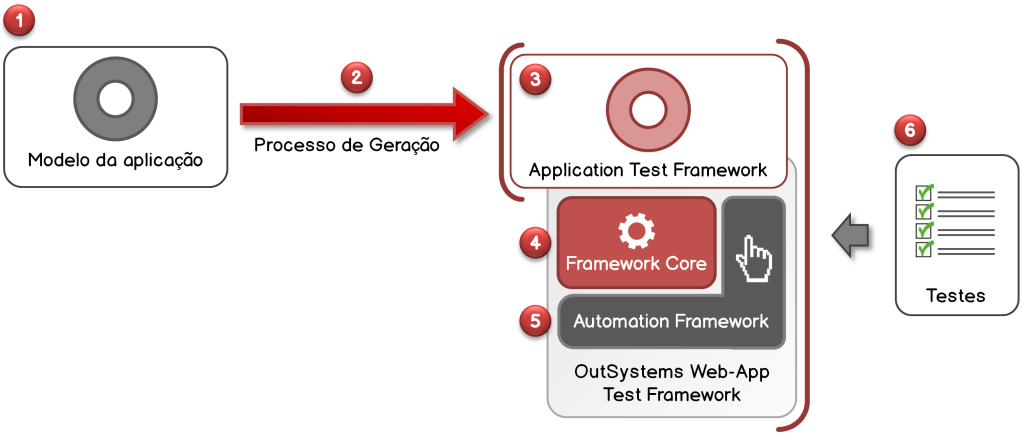

Figura 4.1: Arquitectura da solução onde se identificam: (1) o modelo da aplicação em ServiceStudio; (2) o processo de geração da *framework*; (3) a *framework* de testes obtida do modelo da aplicação; (4) o *core* da *framework* de testes; (5) a *framework* de automação e; (6) os testes.

de testes; (5) a *framework* de automação; e (6) os testes.

De seguida detalham-se cada um dos elementos referenciados quanto ao seu papel na arquitectura da solução proposta.

1 - Modelo da Aplicação

O modelo da aplicação, definido no ServiceStudio com recurso à linguagem visual da OutSystems Platform, contém a definição das camadas de apresentação, lógica aplicacional e acesso a dados. É utilizado pelo PlatformServer no processo automático de compilação e geração de código que dá origem à aplicação web, podendo ser alterado, sendo re-utilizado nesse processo para a geração automática de uma nova versão da aplicação. Na solução proposta o seu contributo é fundamental por descrever, com todo o detalhe, elementos como as páginas na aplicação, os comportamentos que expõem e as *wigets* que as constituem, que dão origem ao HTML das páginas com as quais o utilizador da aplicação pode interagir.

2 - Processo de geração da framework

O conjunto de regras que especificam a forma como o modelo da aplicação se mapeia nos objectos a serem utilizados na criação de testes constitui o processo de geração da *framework* de testes. Este processo é, no contexto deste trabalho, uma tarefa manual que permite fazer a prova do conceito. A implementação de um gerador automático tiraria partido dos mapeamentos especificados na presente

solução, estando nestes o contributo essencial para a abordagem seguida. O processo de geração tem como *input* o modelo da aplicação de onde obtém dados como as suas páginas, os elementos que as compõem ou valores de atributos que viabilizam localização de elementos na página, utilizados para dar origem à camada aplicacional que os representa na criação de testes. O seu objectivo principal é reflectir na *framework* de testes, e por consequência nos testes, alterações feitas à aplicação.

3 - Application Test Framework

A *Application Test Framework* é a camada da *OutSystems Web-Application Test Framework* que é específica da aplicação sendo implementada manualmente, representando em API própria o modelo da aplicação definido no ServiceStudio. A sua função é a de disponibilizar objectos que representam a aplicação, as suas páginas e as *widgets* que as compõem, sendo estes os elementos com os quais os utilizadores da aplicação podem interagir.

Por se basear no modelo da aplicação, é na *Application Test Framework* que são guardadas as referências para o HTML das páginas da aplicação (e.g. localizadores de *widgets*), sendo actualizadas sempre que é gerada uma nova versão da aplicação. Assim, o problema da dependência dos testes com o HTML das páginas da aplicação é diminuído, uma vez que referências ao mesmo serão aqui centralizadas.

O tipo de dados aceite/representado pelas *widgets* das páginas é também uma informação do modelo utilizada no processo de geração da *Application Test Framework*, dando origem a uma representação fortemente tipificada dos elementos manipulados pelos testes. Esta representação permite verificar estaticamente a conformidade dos tipos de dados utilizados nos testes com os especificados no modelo da aplicação no momento da compilação da *test suite*. A API disponibilizada pelos objectos facilita não só a criação de testes, mas também a sua legibilidade, e consequente manutenção, disponibilizando estruturas que representam entidades do domínio da aplicação.

4 - Framework Core

A *Framework Core* é a camada na solução proposta criada para dar suporte à *Application Test Framework*. Nela estão contidas implementações genéricas dos

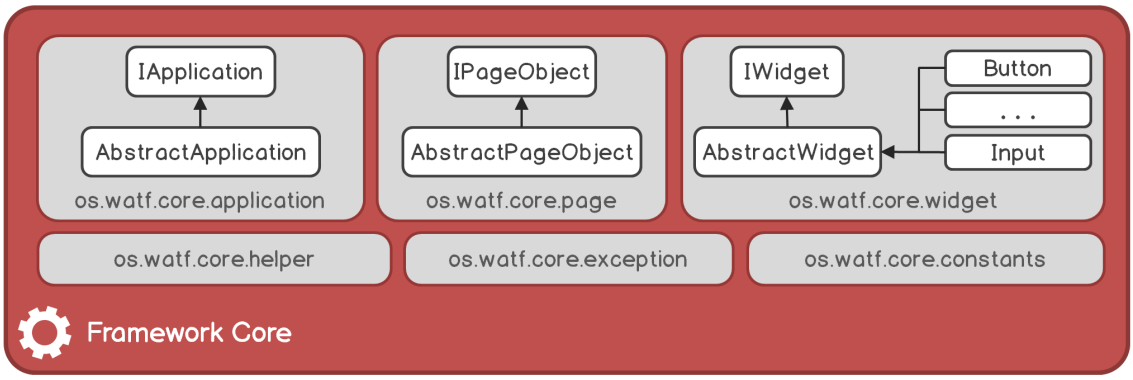

Figura 4.2: Estrutura interna de *Framework Core* onde são mostrados os *packages* que a compõem

objectos que representam a aplicação, as páginas e as *widgets*. A implementação diz respeito às características e comportamentos objectos que não dependem da aplicação. Assim, o processo de geração da *Application Test Framework* dá origem a tipos que extendem os objectos presentes na *Framework Core*. A Figura 4.2 dá uma visão simplificada do conteúdo desta camada mostrando os *packages* que a compõem, onde estão definidas as *interface* e implementações genéricas de objectos que serão utilizados e cujo comportamento pode ser estendido.

5 - Automation Framework

A *Automation Framework* é a camada responsável por viabilizar a interacção dos testes com as páginas da aplicação pela manipulação de uma instância de um browser, tendo-se optado por utilizar a *framework* Selenium *WebDriver*, opção justificada em 3.5. Esta camada é utilizada pela *Framework Core* para reflectir chamadas à API dos objectos por si definidos nos elementos a que correpondem no browser, sendo o caso mais evidente o das *widgets*. A utilização que é feita da *Automation Framework* por parte da *Application Test Framework* foca-se especialmente nas APIs de localização e nos tipos da API *WebDriver* que representam elementos das páginas uma vez que, é na *Application Test Framework* que estão especificadas as dependências com o HTML das páginas (e.g. identificador).

6 - Testes

Os testes são criados manualmente e fazem uso da *OutSystems Web-Application Test Framework* e da *Application Test Framework*, referenciando-as como bibliotecas externas. Na sua implementação serão utilizadas instâncias de objectos que representam o modelo da aplicação, utilizando uma API que viabiliza a obtenção

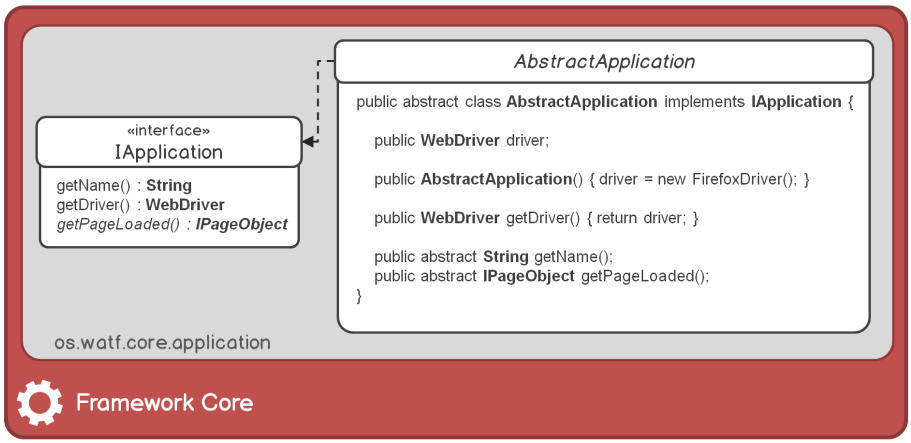

Figura 4.3: Diagrama de classes do objecto aplicação em *Framework Core* e implementação de *AbstractApplication*

e alteração de estado de elementos que compõem as páginas da aplicação sem ser necessário conhecimento acerca da sua representação.

4.2 Framework Core e Application Test Framework

Nesta secção será apresentada com detalhe a forma como a aplicação, as suas páginas e as *widgets* que as compõem são disponibilizadas pela camada *Framework Core* da *OutSystems Web-Application Test Framework*. Será também apresentada a aplicação de exemplo *DemoApp* cujo modelo, definido no ServiceStudio, permitirá demonstrar o processo de geração manual da *Application Test Framework*.

4.2.1 Modelação da aplicação e padrão ApplicationObject

O objecto aplicação consiste no encapsulamento da aplicação definida no ServiceStudio num objecto que a representa, a partir do qual é possível ter acesso a todas as páginas que a compõem. Das responsabilidades deste objecto destacam-se a criação da instância do *driver* capaz de manipular o browser utilizado no teste e a capacidade de retornar objectos que representam todas as páginas da aplicação. O retorno de uma página pelo objecto aplicação pode ser feito de duas formas: i) directamente, expondo um método por página que retorna um objecto que implementa `IPageObject`, referente a uma página específica, ou; ii) indirectamente, recorrendo ao método `getPageLoaded` que, na sua implementação, identifica a página apresentada localizando um elemento DOM criado especificamente para ser usado pela *framework*, retornando o *IPageObject* respectivo.

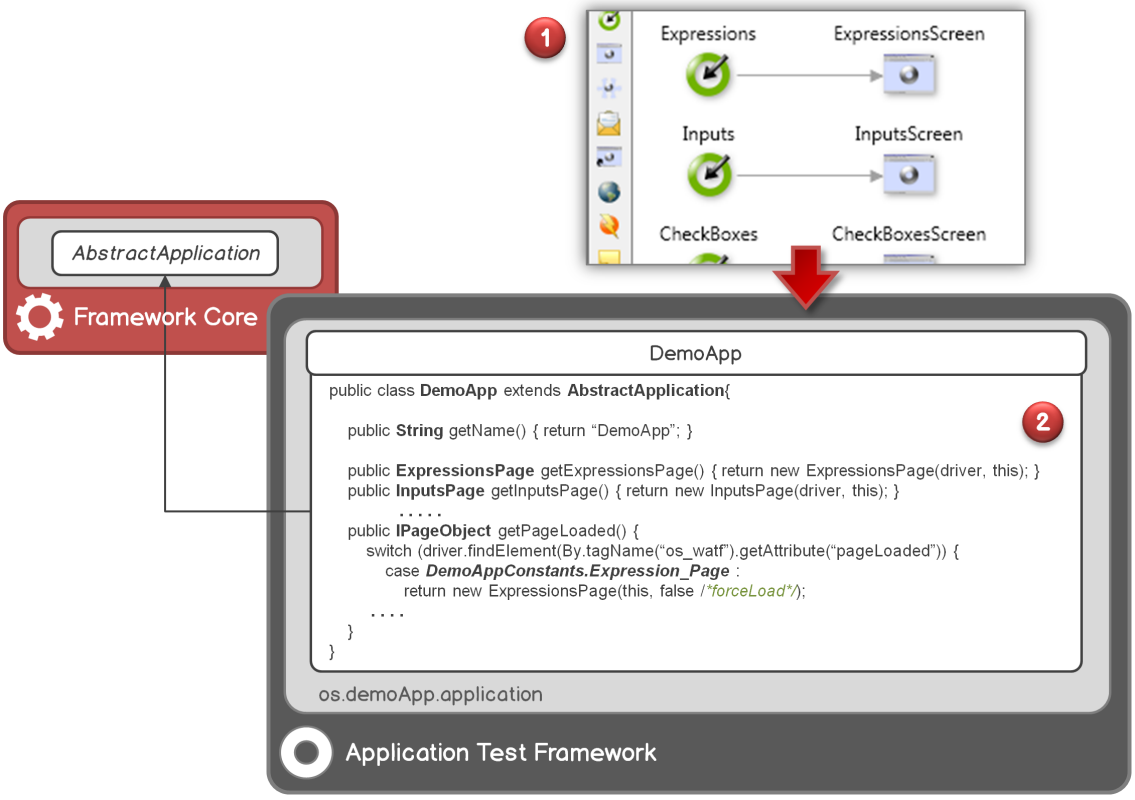

Figura 4.4: Modelo de navegação de *DemoApp* e implementação do objecto aplicação resultante na *Application Test Framework*

Padrão ApplicationObject

Este padrão é uma contribuição do presente trabalho, tendo sido identificado da necessidade de aceder a qualquer página da aplicação sem ser necessário percorrer uma navegação específica para a alcançar. Na *Framework Core* o objecto aplicação é representado pelo tipo *AbstractApplication* que implementa a interface *IApplication*, estando esta hierarquia e respectiva implementação representadas na Figura 4.3. Nela podem-se visualizar as implementações dos comportamentos comuns deste objecto a qualquer aplicação, relacionados com a construção e retorno da instância do *driver*.

O comportamento deste objecto no que respeita a uma aplicação específica é implementado no processo de geração da *Application Test Framework*, sendo disponibilizado por essa camada. A Figura 4.4 apresenta: (1) o modelo de navegação da aplicação *DemoApp* no ServiceStudio; e (2) o objecto aplicação resultante do processo de geração da *Application Test Framework*. Nela pode-se ver o mapeamento entre os vários *entry points* da aplicação e os métodos que retornam explicitamente as páginas correspondentes.

4.2.2 Modelação de páginas e padrão PageObject

O objecto página, ou *PageObject* encapsula uma página da aplicação num objecto que oferece uma API que permite o acesso aos seus elementos e aos comportamentos que expõe. A utilização deste objecto, decorrente da aplicação do padrão *PageObject*, permite remover dos testes referências à estrutura da página, depositando no *PageObject* referências directas ao HTML da página que representa.

Padrão PageObject

Este padrão, identificado em [22], consiste na criação de objectos que representam as páginas da aplicação, encapsulando os comportamentos por elas expostos e os elementos apresentados na interface de utilização. O objectivo é depositar nestes objectos código dependente do HTML das páginas, nomeadamente localizadores de elementos, que viabilizam a interacção com a interface de utilização. Desta forma, os testes passam a invocar métodos expostos pelo `IPageObject` como forma de interagir com a interface de utilização ou executar acções disponiveis na pagina, deixando de ter referência explícitas ao HTML das páginas. Como vantagens da utilização deste padrão destacam-se a separação entre código de teste e código específico da interface de utilização (e.g. localizadores de elementos) e um maior foco nos serviços expostos pela página e não na sua implementação, facilitando o desenho e compreensão dos testes.

Na *Framework Core* este objecto é representado pelo tipo `AbstractPageObject` que implementa a interface `IPageObject`, representados na Figura 4.5. A implementação proposta apenas inclui comportamentos comuns a qualquer página, como é o caso da obtenção do seu título ou caminho, através dos métodos `getTitle()` e `getUrl()`, ou obtenção do objecto aplicação a que pertence.

Uma vez que os elementos que compõem as páginas são específicos da aplicação, é apenas durante a implementação da *Application Test Framework* que estes são gerados com base na informação presente no modelo da aplicação. Elementos da linguagem, como as *widgets*, contribuirão para geração do código interno de objectos que extendem `AbstractPageObject`, cujo conteúdo irá incluir métodos responsáveis por, entre outros, permitir o acesso aos seus componenentes. Adicionalmente, a especificação de campos de entrada nas páginas é também utilizada na geração destes objectos dando origem a contrutores que os incluem. Na secção 4.2.3 são apresentados exemplos de objectos que implementam `IPageObject`, permitindo visualizar a forma como os elementos das páginas e os seus comportamentos são expostos.

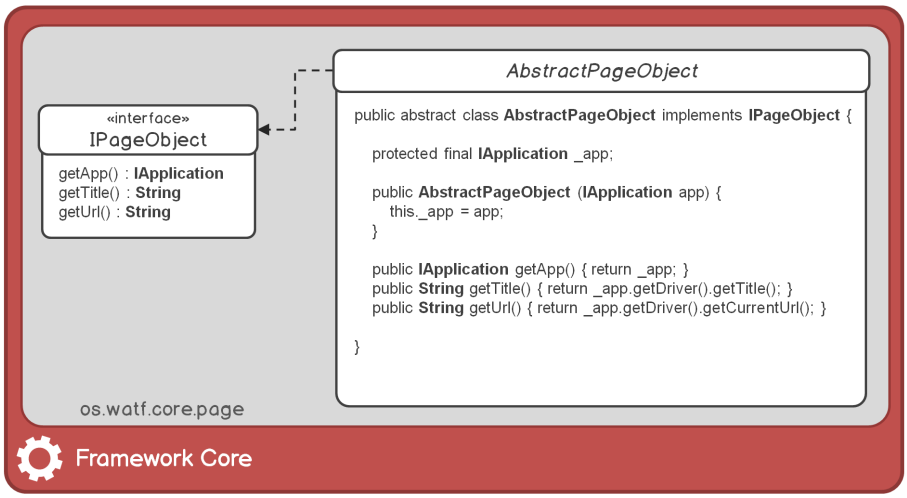

Figura 4.5: Diagrama de classes do objecto página em *Framework Core* e implementação de *AbstractPageObject*

4.2.3 Modelação de widgets

As *widgets* são elementos gráficos disponibilizados no ServiceStudio para a modelação da interface de utilização das páginas, podendo ser utilizadas no contexto de um *WebScreen* ou *WebBlock*. No processo de geração automática de código, levado a cabo pelo PlatformServer, dão origem aos elementos HTML que compõem as páginas da aplicação. Em tempo de desenvolvimento têm um conjunto de propriedades (e.g. visibilidade, nome), que condicionam a sua apresentação, e a uma origem de dados, cujo conteúdo irão representar na interface de utilização.

Na *Framework Core* este tipo de objecto é representado por tipos que estendem `AbstractWidget`, que implementa a interface `IWidget`. Esta hierarquia, e respectiva implementação, estão representadas na Figura 4.6, sendo também apresentadas as classes que correspondem às implementações das *widgets* especificadas ou concretizadas na *Application Test Framework*. Uma vez que a *widget* só existe no contexto de uma página, todas as *widgets* guardam uma referência para o objecto `IPageObject` que as instancia.

A existência de uma propriedade que associa a *widget* a uma origem de dados é característica comum a todos os tipos de *widgets*, permitindo a apresentação de um valor calculado no servidor, apresentado na interface de utilização no elemento HTML que representa a *widget*. Este valor pode ter várias origens, nomeadamente, consultas a bases de dados, variáveis locais ou de entrada da página ou o resultado do processamento de acções. Dos tipos de valores possíveis a que uma *widget* pode estar associada destacam-se os tipos primitivos da linguagem da OutSystems Platform, entre os quais: *Text*, *Integer*, *Decimal*, *Boolean*, *Date*, que correspondem

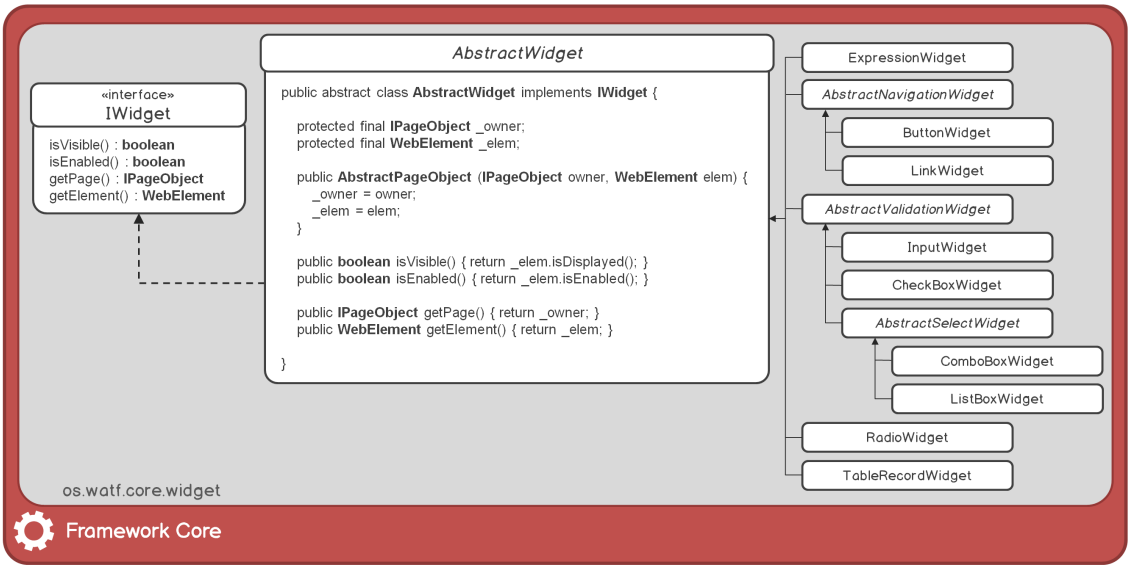

Figura 4.6: Diagrama de classes da hierarquia da *widgets* em *Framework Core* e implementação de `AbstractWidget`

de forma directa aos tipos primitivos suportados pelas máquinas virtuais .NET ou Java.

Além do tipo de valor associado à *widget*, é também possível definir no ServiceStudio propriedades que podem modificar a sua representação concreta na interface de utilização. Algumas propriedades são específicas da *widget*, sendo apresentadas mais à frente em secção própria. Propriedades como nome e visibilidade são comuns à maioria, estando a implementação representada na Figura 4.6 no tipo `AbstractWidget`. A definição de um valor para propriedade `Name` da *widget* é o que permite ao processo de geração da *Application Test Framework* a criação de uma instância que a representa no `PageObject` a que pertence, definindo o nome da variável que a representa. Além de viabilizar a utilização da *widget* em testes, a propriedade *Name* assume uma grande importância na geração da *Application Test Framework*, uma vez que está na base do cálculo do identificador do elemento HTML que representa a *widget*, permitindo a sua localização no DOM da página da aplicação.

De seguida serão apresentados os diferentes tipos de *widgets* suportados pela *OutSystems Web-Application Test Framework* na criação de testes, sendo a sua apresentação feita da seguinte forma: i) caracterização e utilização no ServiceStudio; ii) apresentação da API exposta e da implementação genérica, disponibilizada pela camada de *Framework Core*, bem como da implementação específica à aplicação, gerada para a *Application Test Framework*; e iii) utilização em testes. Para a apresentação das *widgets* será utilizada a aplicação *DemoApp*, especificamente desenvolvida para o efeito, composta por páginas destinadas a demonstrar

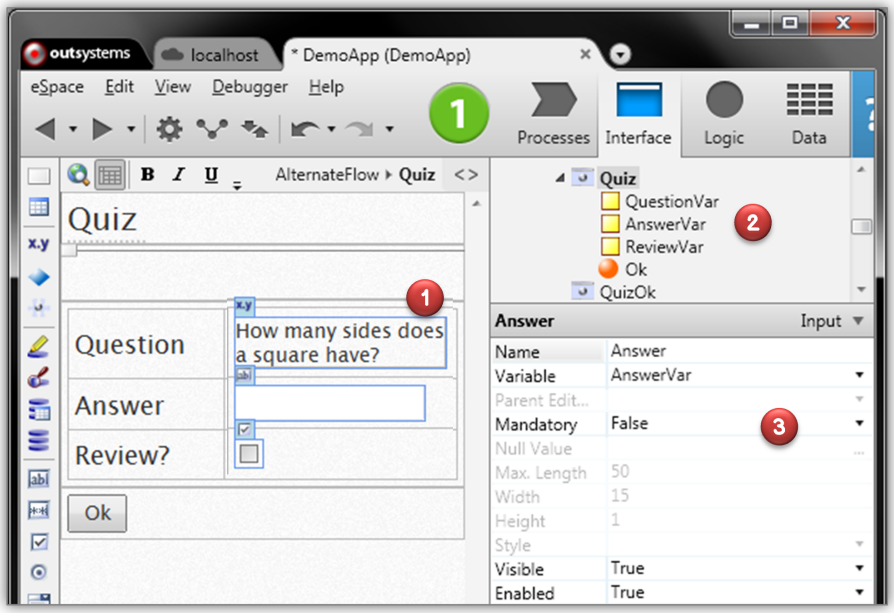

Figura 4.7: Utilização das *widgets* no ServiceStudio destacando-se: (1) utilização no *WebScreen Quiz* onde é apresentada uma pergunta, aceite uma resposta e é dada informação relativa à sua revisão; (2) variáveis da página utilizadas individualmente pelas *widgets*; e (3) propriedades Name, Variable, Mandatory, Visible e Enabled.

a utilização de cada *widget*.

Widgets Expression, Input e CheckBox

As *widgets* Expression, Input e CheckBox apresentam-se como as *widgets* mais elementares no que respeita à construção de interfaces de utilização. A função comum a estas *widgets* é a de apresentar informação sendo que, no caso específico da *widget* CheckBox, apenas é possível representar dados do tipo booleano. As *widgets* Input e CheckBox, além de apresentarem informação, permitem também que o utilizador proceda à sua alteração. Os valores apresentados pelas *widgets* são calculados em *server-side*, obrigando a que, durante o desenvolvimento de aplicações no ServiceStudio, seja definida a propriedade que associa a *widget* a uma origem de dados A origem de dados, tipicamente representada por uma variável, define assim o tipo dos dados apresentados pela *widget*, condicionando o tipo de dados por elas recebidos.

A Figura 4.7 apresenta a utilização das *widgets* no ServiceStudio destancando-se os seguintes pontos: (1) utilização no *WebScreen Quiz* onde é apresentada uma pergunta que para a qual é possível indicar uma resposta e se a mesma será revista; (2) variáveis locais utilizadas individualmente por cada *widget*; e (3) propriedades

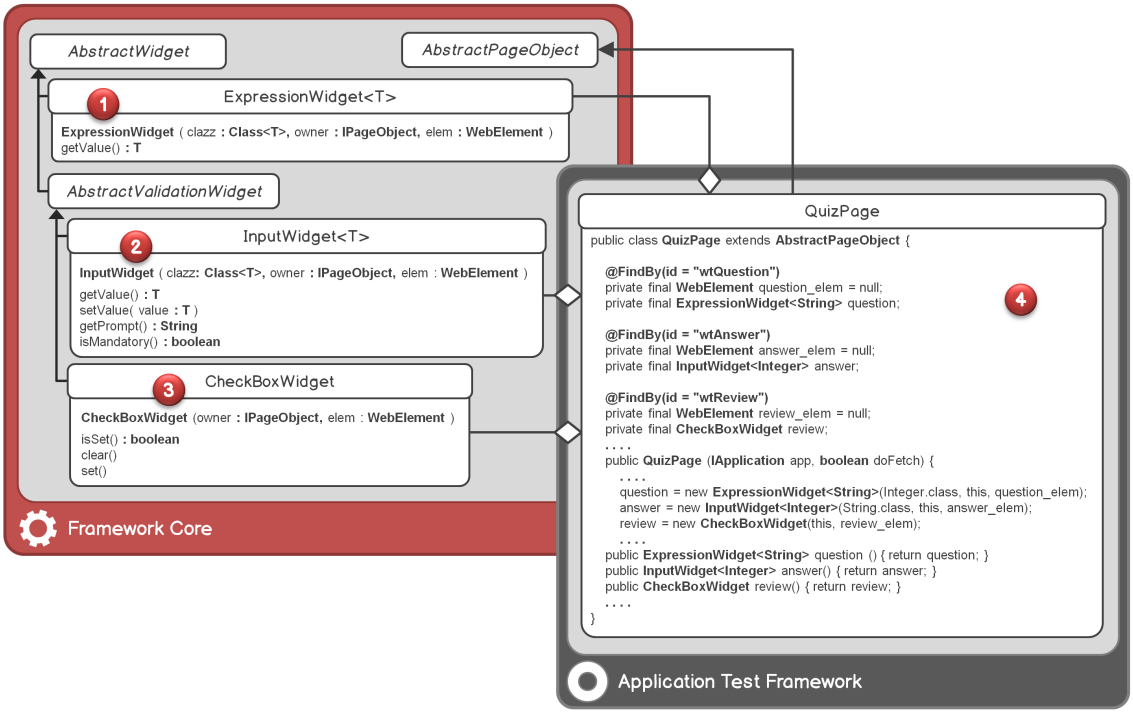

Figura 4.8: API exposta pelos objectos `ExpressionWidget`, `InputWidget` e `CheckBoxWidget` na *framework* de testes

da *widget* Input que representa a resposta à pergunta, das quais se destacam o nome (*Name*) com o valor `Answer`, a variável local a utilizar como origem de dados (*Variable*) com o valor `AnswerVar`, o facto de a sua introdução não ser mandatória (*Mandatory*), de estar visível (*Visible*) e activa (*Enabled*).

No contexto da *framework* de testes, as *widgets* serão acedidas pelos testes através de instâncias de objectos que extendem `AbstractWidget`, sendo estes objectos capazes de encaminhar os pedidos feitos à sua API para o elemento HTML que encapsulam, mostrado no browser, recorrendo para isso à *Automation Framework*. Na camada específica da aplicação, estes objectos são gerados tirando partido da propriedade *Name* para obter o identificador do elemento HTML que representam, sabendo assim localiza-lo na página da aplicação.

Na Figura 4.8 é possível visualizar: (1) a API cujos objectos do tipo `ExpressionWidget`; (2) `InputWidget` e; (3) `CheckBoxWidget` expõem através da implementação genérica na camada *Framework Core* e; (4) a sua utilização na *Application Test Framework* da *DemoApp*, resultante do processo de geração do objecto aplicação `QuizPage`. Uma vez que o tipo de dados a que as *widgets* Expression e Input podem estar associadas varia, optou-se por recorrer ao uso de genéricos na sua implementação na *Framework Core* com o objectivo de diminuir o número de implementações existentes em detrimento de uma implementação específica para

cada tipo de dados. O tipo de dados que as *widgets* representam ou aceitam é definido no processo de geração da *Application Test Framework*, tirando-se partido do tipo de variável associada à origem de dados da *widget*. Uma vez que uma *widget* só existe no contexto de uma página, as suas instâncias são sempre devolvidas por objectos do tipo `IPageObject`.

Listagem 4.1: Implementação de teste com recurso aos objectos `ExpressionWidget`, `InputWidget` e `CheckBoxWidget` da *framework* de testes

```
1   @Test
2   public void checkQuiz() {
3     DemoApp app = new DemoApp();
4     QuizPage page = app.getQuizPage();
5     page.answer().setValue(4);
6     page.review().set();
7     page.ok().click();
8     assertTrue(app.getPageLoaded() instanceof QuizOkPage);
9   }
```

A Listagem 4.1 exemplifica a criação do teste à página *Quiz* onde o objectivo é verificar que ao carregar no botão *Ok* depois de responder `'4'` à pergunta *'How many sides does a square have?'* e solicitar a revisão da mesma, o utilizador é direccionado para a página *QuizOk*. O teste inicia com a criação de uma instância do `ApplicationObject` que representa a aplicação (linha 3) e com a obtenção do objecto `QuizPage` que representa a página a testar (linha 4). Na linha 5 é alterado o valor da resposta para `'4'`, onde é obtida a instância de `InputWidget` que representa na página a *widget* com o nome `Answer`. Na linha 6, activa-se a checkbox que indica que a resposta será revista, através da invocação do método `set()` da instância de `CheckBoxWidget`. O evento de *click* sobre a instância de `ButtonWidget` que representa o botão `'Ok'` é invocado na linha 7, teminando o teste na linha 8 com a verificação de que a página carregada no *driver* é do tipo `QuizOkPage`, comparando-o com a instância devolvida pela chamada ao método `getPageLoaded()` do objecto aplicação.

Da utilização da *framework* de testes observa-se a inexistência de dependências directas do teste com o HTML da página, facilitando não só a sua escrita, mas também a sua legibilidade, que a longo prazo terá consequências positivas na sua manutenção. Uma vez que os objectos que representam as *widget* na página se encontram tipificados quanto ao seu valor, a visibilidade sobre alterações ao tipo na aplicação é detectada estaticamente após geração de nova versão da *Application Test Framework*, conforme 4.1 na apresentação da *Application Test Framework*.

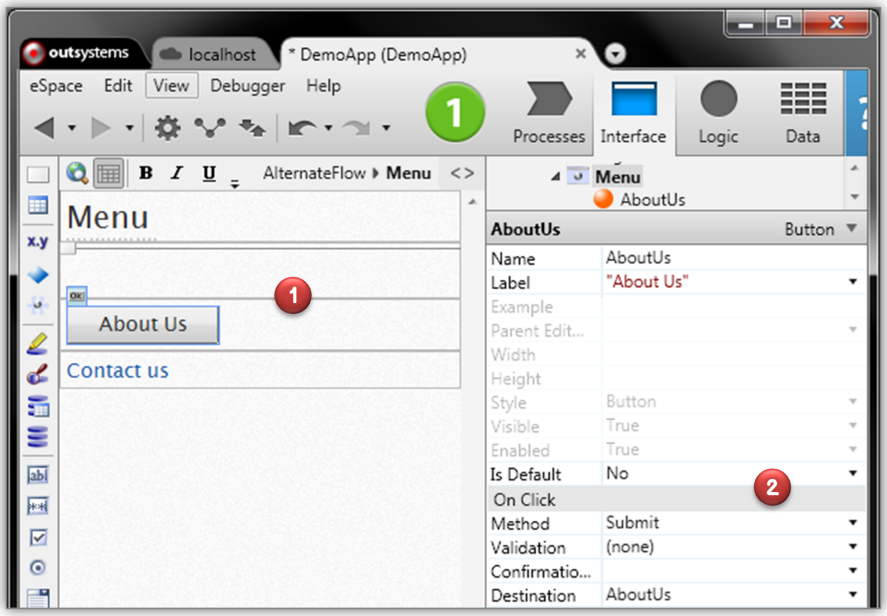

Figura 4.9: Utilização das *widgets* Button e Hyperlink no ServiceStudio destacando-se: (1) utilização no *WebScreen Menu* onde é possível navegar para duas páginas diferentes; (2) propriedades *Method*, *Destination*, *Validation* e *Default*.

Widgets Button e Hyperlink

As *widgets* Button e Hyperlink têm o objectivo de permitir o envio de informação introduzida pelo utilizador nas *widgets* da página para o servidor, viabilizando assim a transição entre páginas ou actualização do seu conteúdo. A principal diferença entre as duas *widgets* diz respeito à forma como aparecem na interface de utilização, dando origem a elementos diferentes no HTML das páginas geradas.

A utilização das *widgets* Button e Hyperlink no ServiceStudio não apresenta diferenças significativas, fazendo uso das mesmas propriedades que definem a forma como a navegação é feita, o conjunto de dados enviados para o servidor e a forma como a resposta é apresentada. A Figura 4.9 apresenta a utilização destas *widgets* no ServiceStudio destacando-se os seguintes pontos: (1) utilização no *WebScreen Menu* onde é possível navegar para duas páginas diferentes; e (2) propriedades da *widget* Button que definem o método de envio (*Method*) com o valor `Submit`, ecrã ou acção de destino (*Destination*) com o valor `AboutUs`, o tipo de validação de *inputs* (*Validation*) com o valor `(none)` e se é a *widget* cujo evento de *click* é invocado por omissão ao ser pressionado a tecla *Enter* no ecrã (*Default*).

A propriedade *Method*, ponto 2 da Figura 4.9, permite configurar a comunicação entre o cliente e o servidor quanto ao tipo de pedido, os dados a serem enviados e de que modo a resposta devolvida é apresentada. Para a definir são disponibilizadas

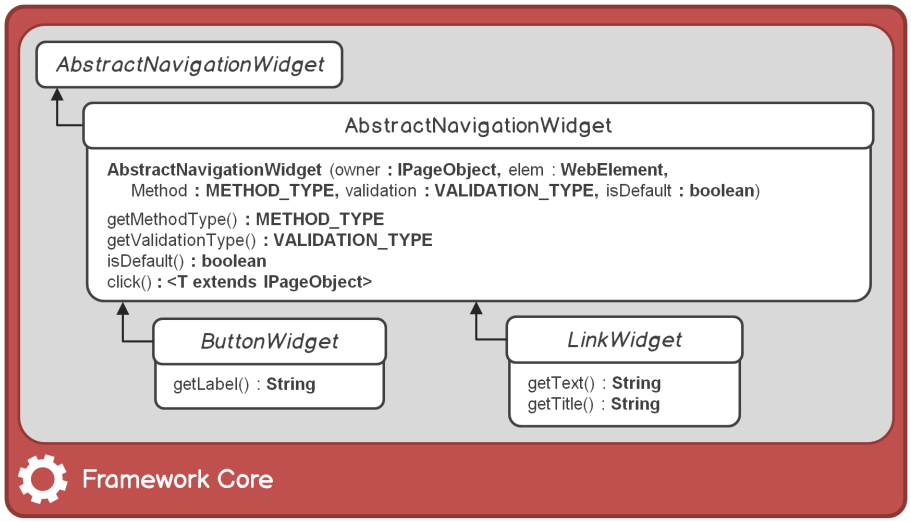

Figura 4.10: API exposta pelos objectos `ButtonWidget` e `LinkWidget` na *framework* de testes

as opções: i) *Submit*, que envia o valor de todos os campos da página através de um pedido do tipo POST e que provoca sempre o carregamento da página obtida na resposta; ii) *Navigate*, que envia apenas os campos utilizados como *input* da página de destino num pedido do tipo GET e que também provoca sempre o carregamento da página obtida na resposta; e iii) *Ajax*, que envia o valor de todos os campos da página através de um pedido do tipo POST e, além de permitir o carregamento da página obtida na resposta, permite manter a página actual carregada, viabilizando a actualização de elementos nela existentes.

Uma vez que a introdução de dados pelos utilizadores nas *widgets* da página carece em muitos casos de validação, é no momento do seu envio para o servidor que esta pode ser feita. A OutSystems Platform oferece mecanismos de verificação da conformidade dos dados introduzidos com o seu tipo ou com o facto de a uma *widget* mandatória não ter sido associado nenhum valor. Estes mecanismos de verificação estão disponíveis tanto em *client-side* como em *server-side*, resultando na associação a cada *widget* inválida de uma mensagem de erro e de um estilo diferente que a destaca das outras na interface de utilização.

A Figura 4.10 mostra a API e hierarquia de tipos na *Framework Core*, que suporta a utilização das *widgets* Button e Hyperlink na *Application Test Framework*. Uma vez que a página obtida do evento de *click* pode ser condicionada pela acção a este associada, a implementação proposta invoca este evento sobre a *widget*, delegando no objecto `IApplication` a que pertence o retorno de uma instância do objecto apropriado, através da invocação do método `getPageLoaded()`. Com base em informação específica de cada página, o objecto `IApplication` sabe instanciar o `IPageObject` que a representa, permitindo a interacção com os seus elementos. O facto de um `IPageObject` não guardar estado acerca da página,

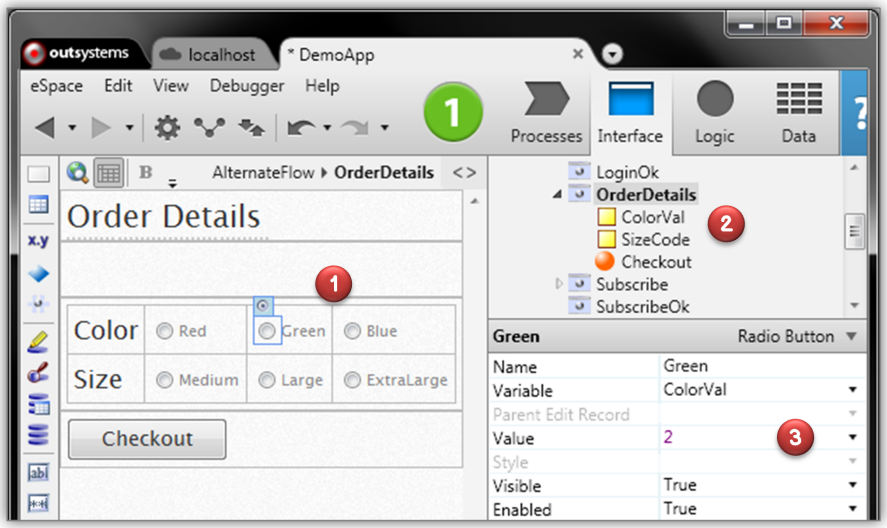

Figura 4.11: Utilização da *widget* Radio Button no ServiceStudio destacando-se: (1) utilização em grupo no *WebScreen OrderDetails*; (2) campos de entrada da página utilizados por grupo de *widgets*; e (3) propriedades Name, Variable, Value, Visible e Enabled.

obtendo-o directamente da interface de utilização, não trás problemas para o caso dos pedidos Ajax que não promovém o carregamento de uma página.

No que respeita à criação de testes com recurso às *widgets* Button e Link, é possível observar na Listagem 4.1 um exemplo de manipulação de um botão na página (linha 8).

Widget Radio Button

A *widget* Radio Button tem na sua origem um padrão de utilização que se baseia na possibilidade de seleccionar um de vários valores num grupo. A interacção de um utilizador com uma *widget* do grupo passa pela consulta de estado (seleccionada ou não) e pela consulta do seu valor. O grupo permite assim a alternância entre os valores representados por cada Radio Button que contem. No ServiceStudio a associação da mesma origem de dados a várias *widgets* Radio Button, além de definir o tipo de dados que representam, dá origem a um grupo. Neste contexto, apenas uma das *widgets* do grupo pode estar seleccionada, sendo o valor que ela representa o valor do grupo.

A Figura 4.11 apresenta a utilização de *widgets* Radio Button no ServiceStudio destancando-se os seguintes pontos: (1) utilização das *widgets* no *WebScreen OrderDetails* permitindo a selecção de uma cor, representada internamente por um número, e um tamanho, representado por um código textual; (2) campos da

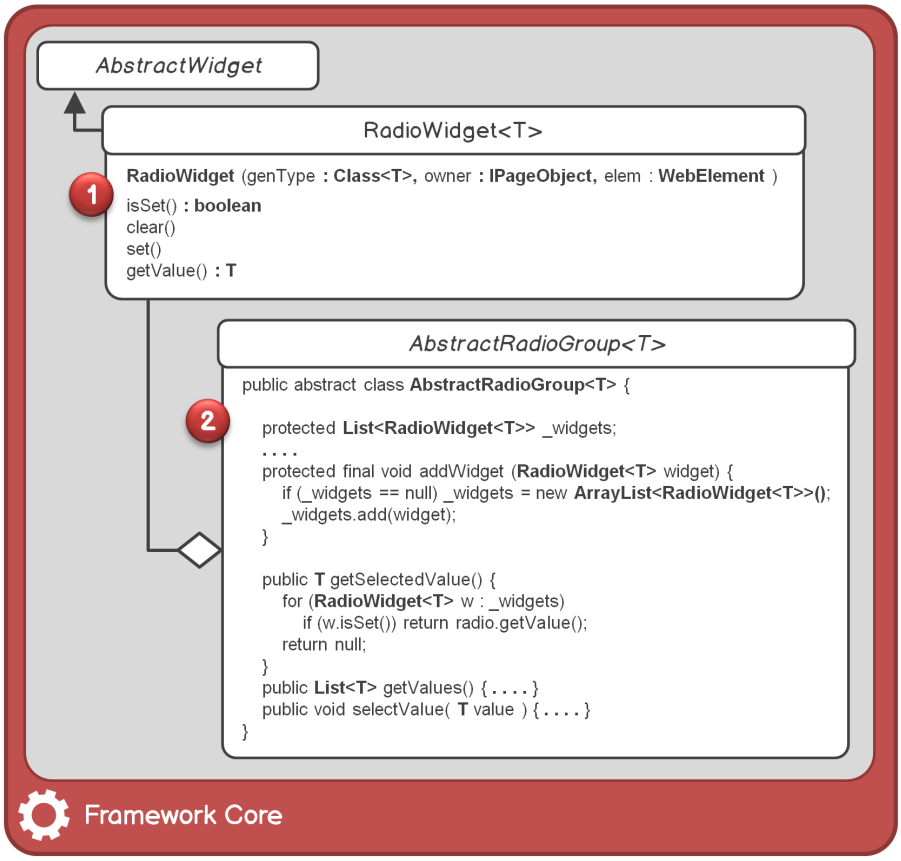

Figura 4.12: API exposta pelo objecto `RadioWidget` na *framework* de testes

página utilizados pelas *widgets*, promovendo a criação de dois grupos de Radio Buttons; e (3) propriedades de uma *widget* Radio Button, das quais se destacam o nome (*Name*) com o valor `Green`, a origem de dados (*Variable*) com o valor `ColorVal`, o valor representado (*Value*) igual a 2 e o facto de estar visível e activa.

Na Figura 4.12 está representada: (1) a API exposta na *framework* por objectos do tipo `RadioWidget`; e (2) parte da implementação genérica de um grupo de *widgets* RadioButton, representada pelo tipo `AbstractRadioGroup`. Apesar do tipo de dados que a *widget* representa variar, os métodos `set()` e `clear()` de `RadioWidget`, que promovem a alternância de estado, não recebem qualquer argumento uma vez que a *widget* tem um comportamento binário (seleccionado e não seleccionado). Assim, o método responsável pela obtenção do valor que representa, utiliza um argumento genérico, diminuindo o número de implementações de `RadioWidget` existentes. A implementação genérica de um grupo de *widgets* RadioButton, representada no tipo `AbstractRadioGroup`, inclui uma colecção que referência as *widgets* do mesmo grupo. Dela fazem parte métodos capazes de: i) seleccionar um valor no grupo de *widgets* (`selectValue`), sendo lançada uma excepção ao tentar selecionar um valor inexistente; ii) obter o valor da *widget* seleccionada (`getValue`) ou todos os valores do grupo (`getValues`); e iii) adi-

cionar *widgets* ao grupo (`addWidget`), sendo este método chamado por um tipo derivado.

Listagem 4.2: Implementação de teste com recurso ao objecto `CheckBoxWidget` da *framework* de testes

```
1   @Test
2   public void checkOrderDetails() {
3     DemoApp app = new DemoApp();
4     OrderDetailsPage page = app.getOrderDetailsPage();
5     page.colorVal().selectValue(3/*Blue*/);
6     page.sizeCode().selectValue("L"/*Large*/);
7     assertTrue(page.checkout().click() instanceof BlueLargeOrderedPage);
8   }
```

Na Listagem 4.2 é exemplificada a utilização de objectos do tipo `RadioWidget` pela criação do teste à página *OrderDetails*. O objectivo do teste é confirmar que quando as opções `'Blue'`, do grupo `Color`, e `'Large'`, do grupo `'Size'` estão seleccionadas, o utilizador é direccionado para a página *BlueLargeOrdered* após carregar no botão *Checkout*. No inicio do teste é criada uma instância de `ApplicationObject` (linha 3) de onde é obtido o objecto `OrderDetailsPage`, que representa a página a testar (linha 4). Na linha 5 é obtida a instância `colorVal`, que representa o *RadioGroup* relativo à cor, sendo seleccionado o valor `'3'`, que representa a cor `'Blue'`. Na linha 6 é obtida a instância `sizeCode()`, que representa o *RadioGroup* relativo ao tamanho, onde é seleccionado o valor `'L'`, que representa o tamanho`'Large'`. O teste termina na linha 7 com a verificação de que a página obtida, após invocação do evento click do botão `'Checkout'`, é do tipo `BlueLargeOrderedPage`.

A não existência de dependências com o HTML da página e a possibilidade de referir directamente um grupo de *RadioButtons*, através de API própria, além de permitir um maior foco no objectivo do testes diminui consideravelmente a quantidade de código necessária. Desta forma, é expectável que a probabilidade de serem introduzidos erros diminua ao mesmo tempo que se aumenta a facilidade com que estes podem ser detectados.

Widget Combo Box

A *widget* ComboBox tem o objectivo de permitir a selecção de um valor numa lista, viabilizando interacções ao nível da consulta de valores e respectiva selecção. A cada linha ou opção da lista de valores apresentada por uma ComboBox está associado: i) um *valor interno*, cujo conteúdo não é apresentado na interface de

utilização mas que representa o valor seleccionado pela *widget*, e; ii) um *valor etiqueta*, que é o que apresenta a opção na interface de utilização aos utilizadores. A *widget* ComboBox permite a definição de dois tipos de listas de valores: i) uma lista dinâmica, cujos elementos podem ter origem numa consulta à base de dados ou no processamento de uma acção; e ii) uma lista especial, cujos elementos têm origem na definição das propriedades *Value* e *Option* no ServiceStudio.

As listas especial e dinâmica podem representar tipos de dados diferentes, sendo no entanto apresentadas como uma lista única da *widget* ComboBox a que se referem. A existência de valores com tipos diferentes na mesma lista só é possível devido à interface de utilização ser representada por HTML, sendo todos os valores da *widget*, internos ou de etiqueta, tratados como texto.

A Figura 4.13 apresenta a utilização da *widget* ComboBox no ServiceStudio, tendo sido utilizada a estrutura de dados `Continent`, destacando-se os seguintes pontos: (1) utilização da *widget* ComboBox no *WebScreen ContinentSelection*, exemplificando a utilização de listas dinâmicas com tipos de valores interno e de etiqueta de texto; (2) estrutura de dados `Continent` utilizada no exemplo; (3) campos da página utilizados pela ComboBox como origem de dados e armazenamento do valor seleccionado; e (4) propriedades relacionadas com a apresentação da *widget* e dados apresentados e seleccionados.

No ponto (4) da Figura 4.13 são mostradas as propriedades da *widget* com o nome `Continent`, relacionadas com a origem dos dados da lista dinâmica, a sua estrutura, tipos de valores e valor seleccionado, interpretando-se da seguinte forma:

- Associação da variável *ContinentList*, do tipo lista de *Continent*, preenchida na acção de preparação da página, à origem dos dados da lista dinâmica através da propriedade *Source Record List*;
- Definição de *Continent* como a estrutura de dados representada pela lista dinâmica através da propriedade *Source Entity Structure*;
- Associação do atributo do tipo texto *Name* da estrutura *Continent*, ao valor etiqueta da lista através da propriedade *Source Attribute*;
- Associação do atributo do tipo texto *Code* da estrutura *Continent*, ao valor interno da lista através da propriedade *Source Identifier Attribute*;
- Definição da variável *ContinentVar*, do tipo texto, como responsável pelo armazenamento do valor interno da opção seleccionada na lista dinâmica através da propriedade *Variable*;

As propriedades relacionadas com a lista especial de valores da *widget* que representa a estrutura *Continent* são também apresentadas no ponto (4) da Figura 4.13

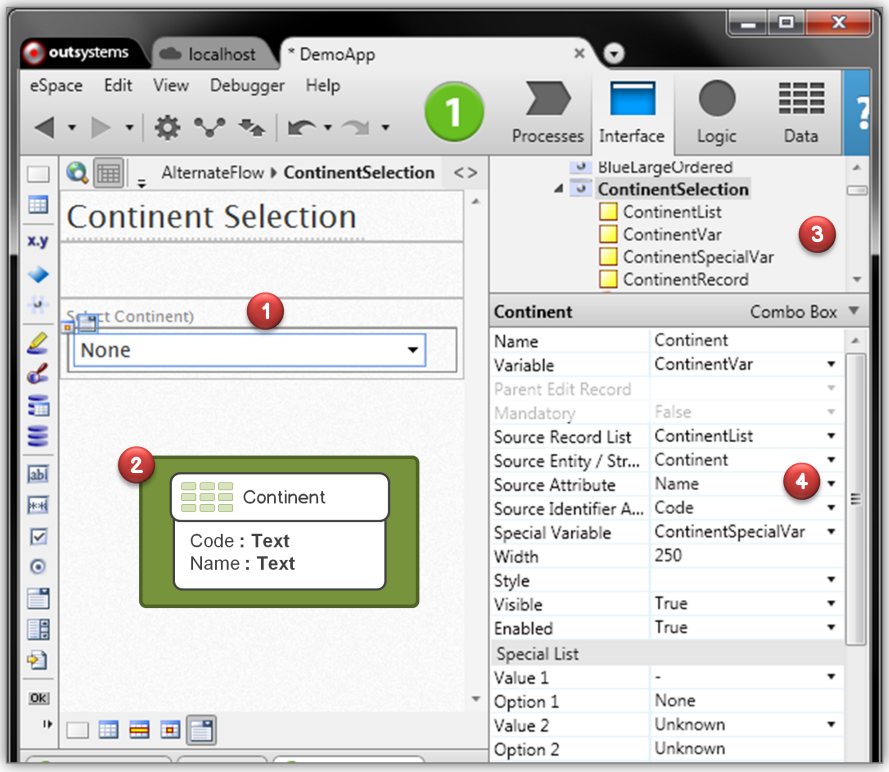

Figura 4.13: Utilização da ComboBox no ServiceStudio: (1) utilização da *widget* ComboBox no *WebScreen ContinentSelection*; (2) estrutura *Continent* utilizada no exemplo; (3) campos da página utilizados pela ComboBox para origem de dados e armazenamento de valor seleccionado; e (4) propriedades da *widget* relacionadas com dados apresentados e seleccionados

interpretando-se da seguinte forma:

- Criação dos elementos *('-', 'None')* e *('Unknown', 'Unknown')* na lista especial com os valores interno e de etiqueta do tipo texto na secção *Special List*;
- Definição da variável *ContinentSpecialVar*, do tipo texto, como responsável pelo armazenamento do valor interno da opção seleccionada na lista especial através da propriedade *Special Variable*;

A Figura 4.14 mostra a API, hierarquia de tipos e um excerto da implementação na *Framework Core*, que suporta a utilização da *widget* ComboBox na *Application Test Framework*. Para a implementação da ComboBox optou-se por criar o tipo `ComboBoxWidget`, que estende `AbstractSelectWidget`, sendo este último um agregador de comportamento e características comuns a *widgets* que viabilizam a selecção de opções numa lista. Com o objectivo de gerar instâncias de ComboBoxes tipificadas quanto aos valores apresentados, de acordo com o modelo definido no ServiceStudio, optou-se por modelar uma opção da lista no tipo

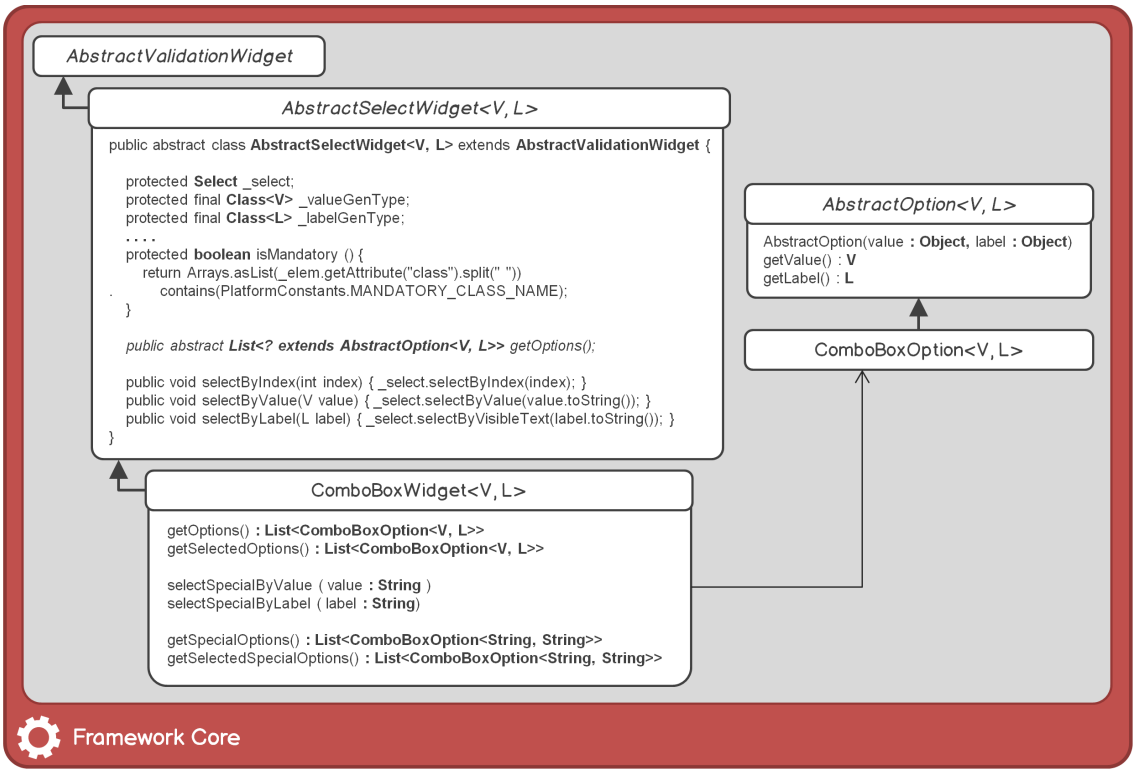

Figura 4.14: API exposta pela hierarquia de suporte a ComboBoxWidget em *Framework Core*

`ComboBoxOption`, que estende `AbstractOption`. Este, recebe dois parâmetros genéricos relativos ao tipo do valor interno e ao tipo do valor etiqueta, permitindo a criação de instâncias de `ComboBoxWidget` específicas para o tipo de dados a representar. No que respeita à lista especial de valores, uma vez que tanto o tipo do seu valor interno como do valor de etiqueta é texto, são utilizadas instâncias de `ComboBoxOption` devidamente parametrizadas, conforme linha X da Figura 4.14.

No que respeita à utilização da *widget* ComboBox na *Application Test Framework*, a Figura 4.15 mostra a implementação da página que corresponde ao *WebScreen* da Figura 4.13. No contexto deste trabalho a implementação apresentada foi feita de forma manual devido à inexistência de um gerador automático. Contudo, a definição do mapeamento entre a página e seus elementos e o objecto que a representa, permite observar a quantidade de código se de forma automática deixaria de ser escrito.

A Listagem 4.3 exemplifica a verificação do valor seleccionado nas listas especial e dinâmica de valores, interagindo com a ComboBox disponibilizada na página *ContinentSelection*, tendo sido criado apenas um teste para ambos os casos por motivos de simplificação. O teste inicia com a criação de uma instância do `ApplicationObject` que representa a aplicação (linha 3) e, da obtenção do objecto `ContinentSelectionPage` que representa a página a testar (linha 4). Na

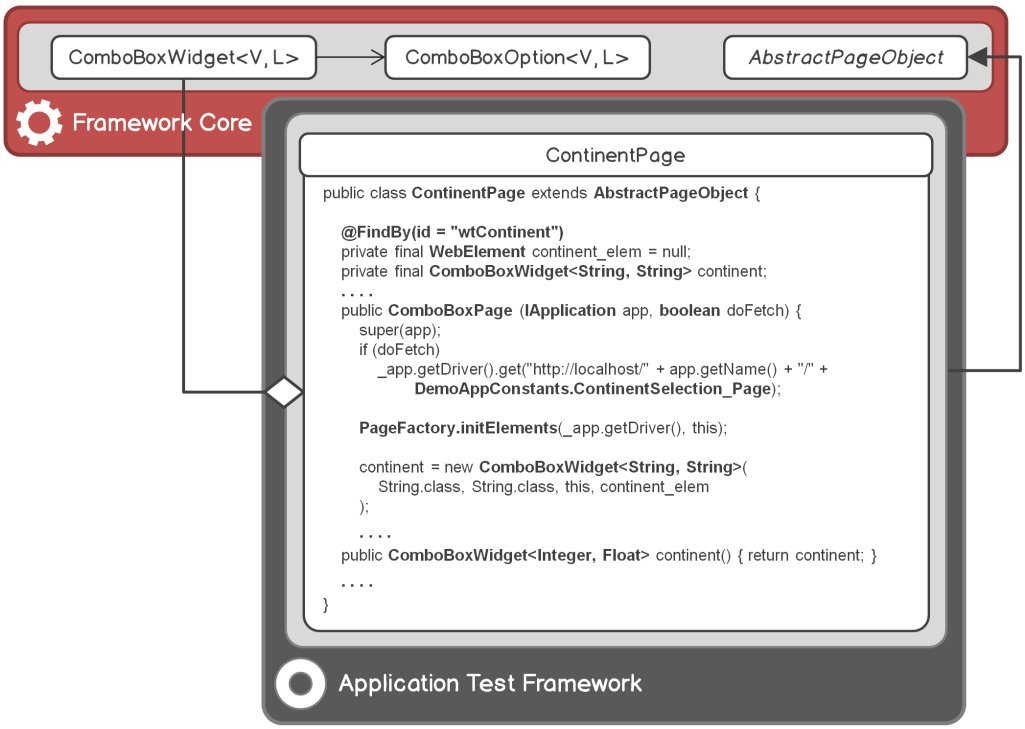

Figura 4.15: Utilização de ComboBoxWidget em *Application Test Framework*

```
1   @Test
2   public void checkContinentSelection() {
3     DemoApp app = new DemoApp();
4     ContinentSelectionPage page = app.getContinentSelectionPage();
5     page.continent().selectByLabel("Europe");
6     assertEquals("EU", page.continent().getSelectedOption().getValue());
7     assertEquals(null, page.continent().getSelectedSpecialOption());
8     page.continent().selectSpecialByLabel("None");
9     assertEquals(null, page.continent().getSelectedOption());
10    assertEquals("-", page.continent().getSelectedSpecialOption().getValue());
11  }
```

linha 5 obtém-se a instância `continent`, que representa a *widget* ComboBox na página, seleccionando-se da lista dinâmica o valor de etiqueta `Europe`. Nas linhas 6 e 7 verifica-se que o valor interno da opção seleccionada na lista dinâmica corresponde a `EU` e que na lista especial nenhum valor está seleccionado, correspondendo a `null`. Na linha 8 selecciona-se da lista especial o valor de etiqueta `None`, verificando-se nas linhas 9 e 10 que nenhuma opção está seleccionada na lista dinâmica e que o valor interno da opção seleccionada na lista especial corresponde a `'-'`.

Figura 4.16: Utilização da ListBox no ServiceStudio: (1) utilização no *WebScreen CountrySelection*; (2) estrutura *Country*; (3) campos de página utilizados para origem de dados; (4) propriedades da *widget* relacionadas com os dados apresentados

Widget List Box

A *widget* ListBox tem o objectivo de permitir a selecção de um ou mais valores de uma lista, sendo as suas características e comportamento bastante semelhantes *widget* ComboBox, recomendando-se por isso o conhecimento desta última. As principais diferenças entre estas *widgets* têm a ver com: i) a interacção, uma vez que a ListBox permite a escolha de mais do que um valor na lista apresentada, e; ii) com os dados apresentados, por não existir a lista especial de valores. A noção de opção seleccionada deixa de existir, passando a considerar-se a lista de opções seleccionadas, mesmo que a dimensão desta seja uma unidade. A cada linha ou opção da lista de valores continua a estar associado um *valor interno*, contudo, este passa a guardar o índice que a opção tem na lista. O *valor etiqueta* de uma opção da lista mantém a semântica descrita na *widget* ComboBox, estando visível para os utilizadores. Uma vez que podem ser seleccionadas várias opções, a cada uma foi adicionado um campo que sinaliza o seu *estado na lista*, i.e., se está seleccionada ou não.

A Figura 4.16 mostra a utilização da *widget* ListBox no ServiceStudio onde se destaca: (1) utilização no *WebScreen CountrySelection*; (2) estrutura `Country` utilizada no exemplo; (3) campos de página utilizados pela ListBox para origem de

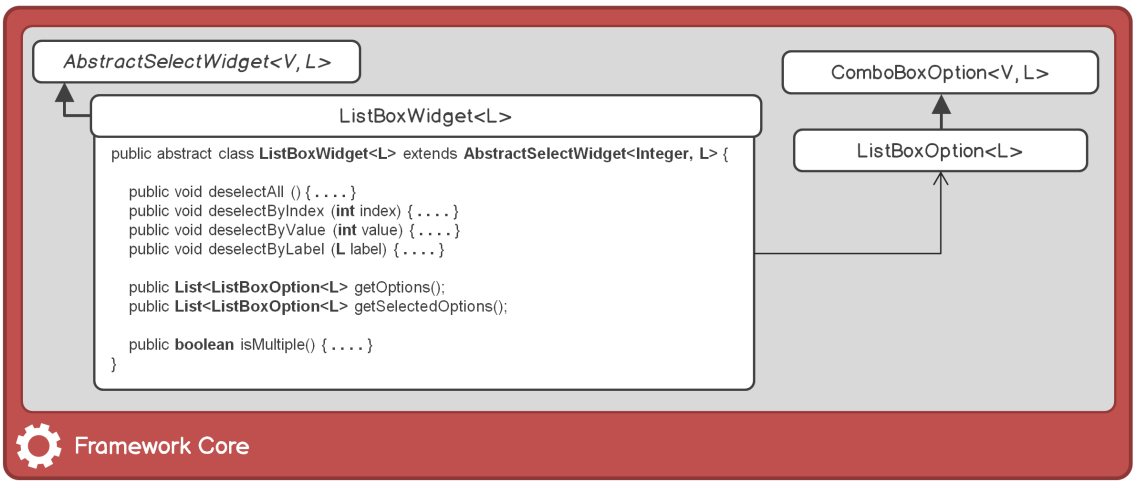

Figura 4.17: API exposta pela hierarquia de suporte a ListBoxWidget em *Framework Core*

dados; (4) propriedades da *widget* relacionadas com os dados apresentados. Das propriedades responsáveis pela aparência e comportamento da *widget* na interface de utilização destaca-se *Selection Mode* que define: i) se podem ser seleccionadas várias linhas, sendo este o comportamento por omissão a que corresponde o valor *Multiple*, ou; ii) se apenas uma linha pode ser seleccionada, a que corresponde o valor *Single*. No que respeita às propriedades relativas aos dados apresentados, a sua definição é bastante mais simples quando comparada com a *widget* ComboBox. A origem de dados apresentados continua a ser representada pela propriedade *Source Record List*, deixando de ser necessário indicar a entidade ou estrutura a ser representada pela *widget*. Para cada opção da lista de valores é necessário definir qual o atributo representado pelo valor etiqueta, associando-o à propriedade *Source Attribute*, e o atributo do tipo booleano que representa o seu estado na lista, associando-o à propriedade *Selection Attribute*.

Na Figura 4.17 mostra-se a API da *widget* ListBox na *Framework Core*, bem como, a hierarquia de tipos e implementação que suporta a sua utilização na *Application Test Framework*. A *widget* ListBox, por viabilizar a selecção de opções numa lista, é representada pelo tipo `ListBoxWidget`, que estende `AbstractSelectWidget`. As opções disponibilizadas pela lista são representadas pelo tipo `ListBoxOption`, que estende `AbstractOption`. Este, define que o tipo inteiro corresponde ao parâmetro genérico do valor interno da opção e adiciona um atributo booleano, que reflecte o estado da opção na lista.

No que respeita à utilização da *widget* ListboxBox na *Application Test Framework*, a Figura 4.18 mostra a implementação da página que corresponde ao *WebScreen* da Figura 4.16.

A Listagem 4.4 exemplifica a criação do teste à página *CountrySelection* onde o

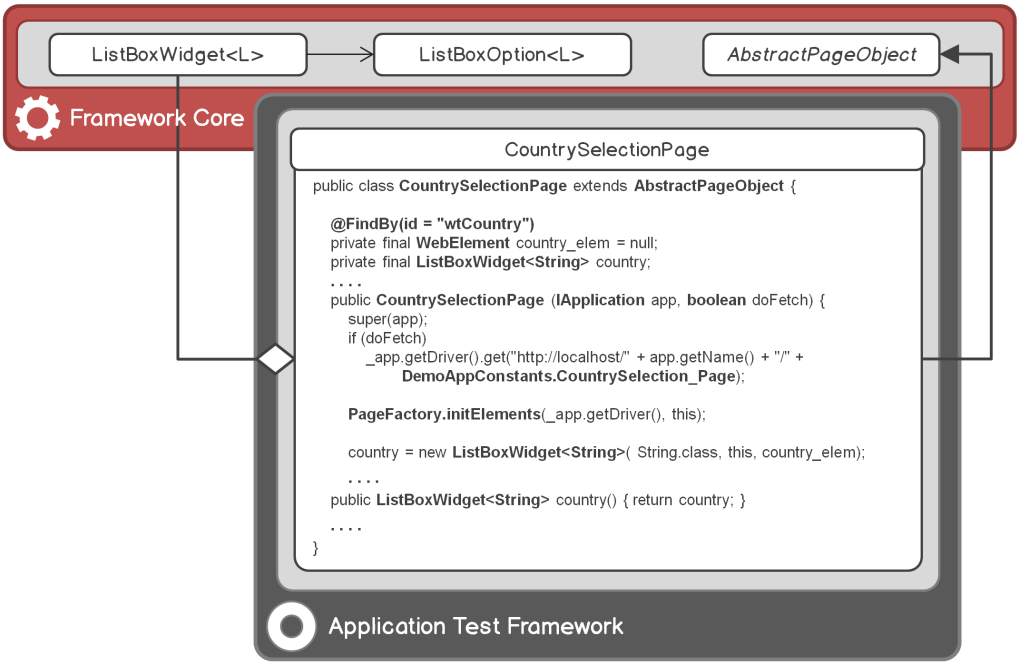

Figura 4.18: Utilização de ListBoxWidget em *Application Test Framework*

objectivo é verificar os valores seleccionados na *widget*. O teste inicia com a criação de uma instância do `ApplicationObject` que representa a aplicação (linha 3) e, da obtenção do objecto `CountrySelectionPage` que representa a página a testar (linha 4). Nas linha 5 a 7, através da instância `country` do tipo `ListBoxWidget`, seleccionam-se opções na lista utilizando o seu valor, índice e etiqueta. Na linha 8 verifica-se que o número de opções seleccionadas é 3. Na linha 9 desmarcam-se todas as opções da lista, verificando-se na linha 10 que não existem opções seleccionadas.

Listagem 4.4: Implementação de teste com recurso ao objecto `ListBoxWidget` da *framework* de testes

```
1   @Test
2   public void checkCountrySelection() {
3     DemoApp app = new DemoApp();
4     CountrySelectionPage page = app.getCountrySelectionPage();
5     page.country().selectByIndex(11); // Denmark
6     page.country().selectByValue(22); // Ireland
7     page.country().selectByLabel("Portugal");
8     assertEquals(3, page.country().getSelectedOptions().size());
9     page.country().deselectAll();
10    assertEquals(0, page.country().getSelectedOptions().size());
11  }
```

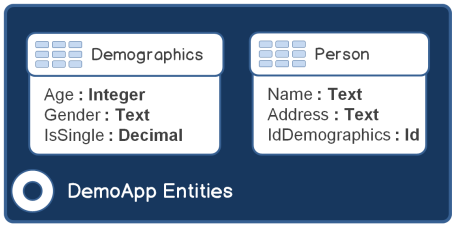

Figura 4.19: Modelo das estruturas *Person* e *Demographics*

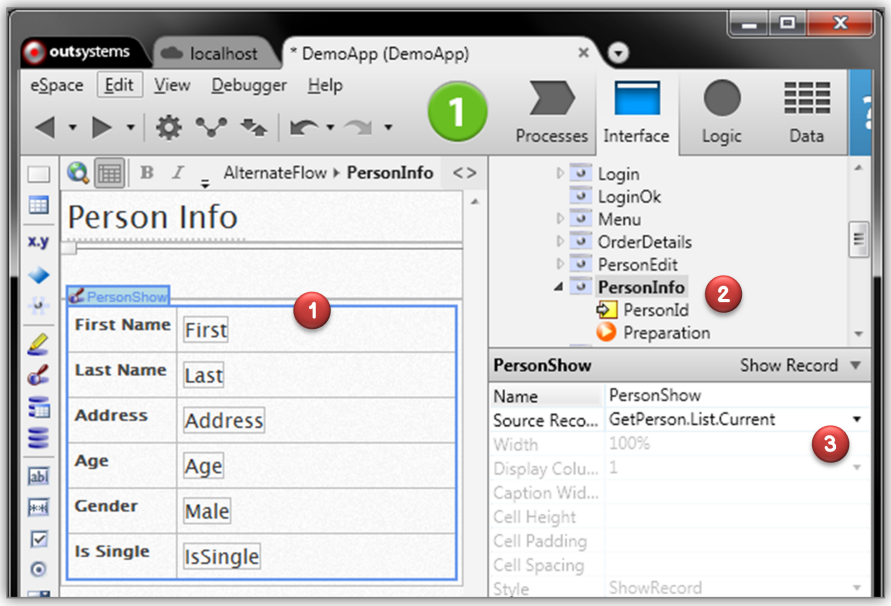

Figura 4.20: Utilização da *widget* ShowRecord no ServiceStudio destacando-se: (1) utilização no *WebScreen PersonInfo* exemplificando a apresentação de um registo de *Person* e o registo *Demographics* associado; (2) campo de entrada *PersonId* que indica o identificador do registo a apresentar; e (3) propriedades relacionadas com a identificação da *widget* e com a origem dos dados a apresentar

Widget Show Record

A *widget* ShowRecord tem como objectivo representar informação relativa a um registo de uma entidade, estrutura ou uma combinação dos dois, sendo utilizada para permitir consultas de dados. A sua representação na interface de utilização é feita com recurso a uma tabela onde o valor de cada campo da entidade que representa é feito por uma *widget* do tipo Expression. A sua utilização no ServiceStudio obriga a que esteja associada a uma variável cujo conteúdo representa uma ocorrência de um ou mais objectos do domínio da aplicação.

Com o objectivo de exemplificar a utilização da *widget* ShowRecord com a aplicação DemoApp, apresentam-se na Figura 4.19 as entidades utilizadas no exemplo: (1) entidade *Demographics* composta por um identificador e pelos campos *Age* do tipo inteiro, *Gender* do tipo texto e *IsSingle* do tipo decimal; e (2) Entidade *Per-*

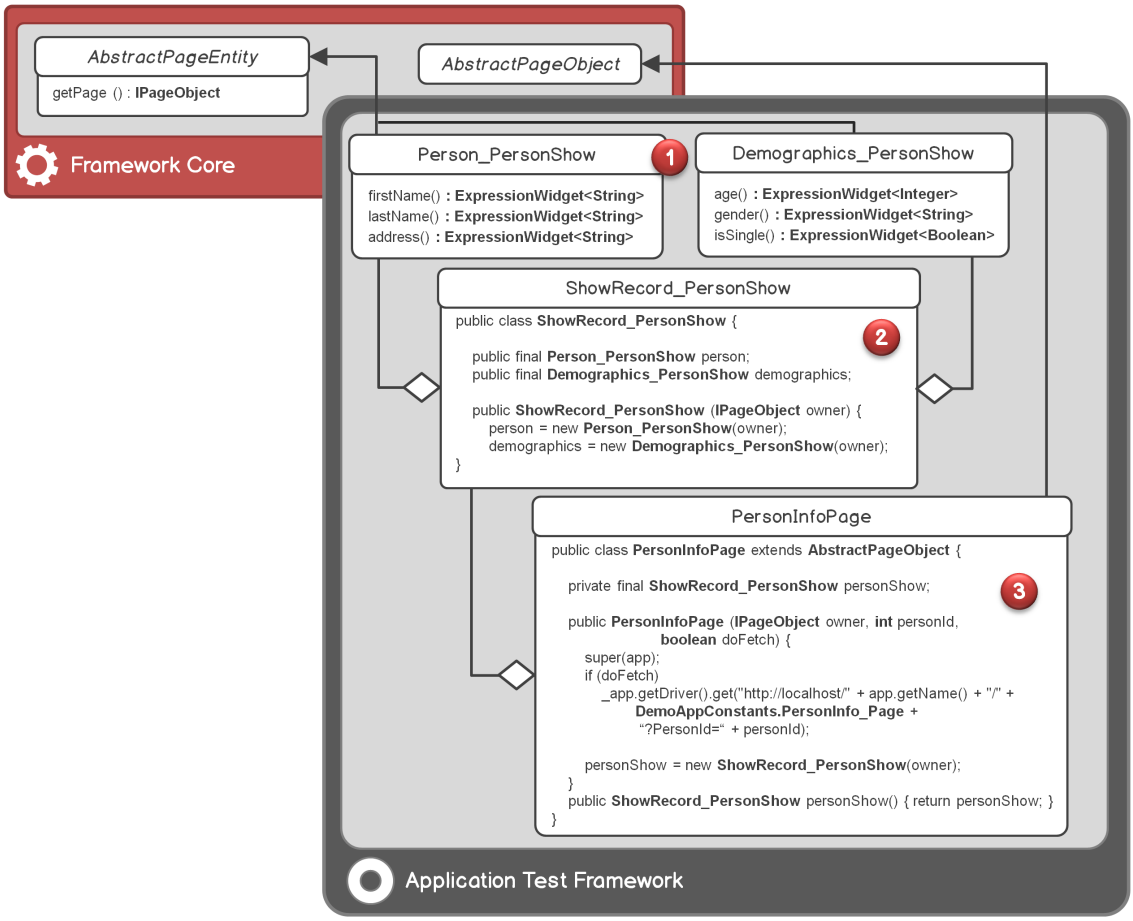

Figura 4.21: Utilização da *widget* ShowRecord na *Application Test Framework* destacando-se: (1) modelação do conjunto de atributos usados por entidade, e das *widgets* que os representam, num objecto do tipo `AbstractPageEntity`; (2) modelação da *widget* ShowRecord num objecto que agrupa as entidades em si representadas; e (3) criação da instância da *widget* no `IPageObject` a que se refere

son composta por um identificador, pelos campos *Name* e *Address* do tipo texto e por uma referência para *Demographics*.

A Figura 4.20 apresenta a utilização da *widget* no ServiceStudio, destacando-se os seguintes pontos: (1) utilização no *WebScreen PersonInfo* exemplificando a apresentação de um registo de *Person* e o registo *Demographics* associado; (2) campo de entrada `PersonId` que indica o identificador do registo a apresentar; e (3) propriedades relacionadas com a identificação da *widget* e com a origem dos dados a apresentar. No ponto (3) da Figura 4.20 destacam-se as propriedades: i) nome (*Name*) com o valor `PersonShow`, que permitirá identificar a instância da *widget* no `IPageObject` a que pertence, e; ii) a origem de dados (*Source Record*), com o valor `GetHarry.List.Current`, que indica o registo que irá alimentar a *widget*, cujo resultado é obtido da execução da *query GetHarry* em *server-side*.

O conteúdo do registo definido na origem de dados é utilizado pelas *widgets* Ex-

pression, ponto (1) da Figura 4.20, sendo referidos os campos significativos de *Person* e *Demographics*. Assim, é possível determinar para cada entidade o conjunto de atributos que são mostrados num ShowRecord. Contudo, uma vez que a utilização dos atributos de uma entidade pode ser feita de diferentes formas, é possível obter uma entidade específica do ShowRecord que representa a original. Considere-se o exemplo da Figura 4.20 onde o campo *Name* de *Person* é apresentado num outro formato, separando *First Name* e *Last Name* em duas linhas distintas. Neste caso específico, a entidade representada pela *widget* ShowRecord passa ser vista pelos utilizadores como tendo atributos *First Name* e *Last Name*.

Uma vez que uma *widget* ShowRecord é representada por um conjunto de *widgets* Expression representativas de um registo, a sua utilização na *Application Test Framework* não necessita de suporte específico na *Framework Core*. A Figura 4.21 mostra a implementação da *widget* ShowRecord na *Application Test Framework* da aplicação DemoApp destacando-se: (1) modelação do conjunto de atributos usados por entidade, e das *widgets* que os representam, num objecto do tipo `AbstractPageEntity`; (2) modelação da *widget* ShowRecord num objecto que agrupa as entidades em si representadas; e (3) criação da instância da *widget* no `IPageObject` a que se refere. A cada atributo da entidade, criada no contexto de um ShowRecord, está associada a *widget* que o representa, cujo nome da instância corresponde ao nome especificado na primeira coluna do ShowRecord (ponto 1, Figura 4.20), fazendo-se o acesso ao seu valor através da API por si exposta.

Listagem 4.5: Implementação de teste com recurso à representação da *widget ShowRecord* na *framework* de testes

```
1   @Test
2   public void checkPersonInfo() {
3     DemoApp app = new DemoApp();
4     PersonInfoPage page = app.getPersonInfoPage(2 /*personId*/);
5     assertEquals("Harry", page.personShow().person.firstName().getValue());
6     assertEquals("Potter", page.personShow().person.lastName().getValue());
7     assertEquals("Hogwarts", page.personShow().person.address().getValue());
8     assertEquals(9, page.personShow().demographics.age().getValue().intValue());
9     assertEquals("Male", page.personShow().demographics.gender().getValue());
10    assertTrue(page.personShow().demographics.isSingle().getValue());
11  }
```

A Listagem 4.5 exemplifica a criação do teste à página *PersonInfo* onde o objectivo é confirmar o valor apresentado pelos atributos da entidade *Person* e da entidade *Demographics* associada. O teste inicia com a criação de uma instância do `ApplicationObject` que representa a aplicação (linha 3). A obtenção do objecto `PersonInfoPage`, que representa a página a testar, é feita na linha 4, sendo passado o valor `'2'` ao *input* da página responsável por representar o identificador

de *Person*. Nas linhas 5 a 7 é verificado o valor das *widgets* que representam os atributos `firstName`, `lastName` e `address` da entidade *Person*. Nas linhas 8 a 10 é verificado o valor das *widgets* que representam os atributos `age`, `gender` e `isSingle` da entidade *Demographics*.

A modelação proposta concretiza um dos objectivos propostos de permitir a criação de testes a um nivel mais próximo do dominio da aplicação, através da geração de objectos do `AbstractPageEntity`. Estes, por serem específicos de um ShowRecord, apenas incluem os atributos utilizados pela *widget*, sendo possível verificar estaticamente utilizações indevidas na *test suite*. A utilização de uma linguagem mais próxima do dominio da aplicação facilita assim a criação e compreensão dos testes, sendo este facto observável na Listagem 4.5.

Widget Edit Record

A *widget* EditRecord partilha o objectivo de representar informação relativa a um registo de uma entidade com a *widget* ShowRecord, acrescentando a possibilidade de fazer modificações aos seus atributos, recomendando-se por isso o conhecimento desta última. A sua representação na interface de utilização é feita com recurso a uma tabela onde cada campo é representado por uma *widget* apropriada à representação do seu valor. A sua utilização no ServiceStudio obriga igualmente à associação a uma variável cujo conteúdo representa uma ocorrência de um ou mais objectos do domínio da aplicação.

A Figura 4.22 apresenta a utilização da *widget* no ServiceStudio, destacando-se: (1) utilização no *WebScreen PersonEdit* exemplificando a modificação de um registo de *Person* e o registo *Demographics* associado; (2) campo de entrada `PersonId` que indica o identificador do registo a editar; e (3) propriedades relacionadas com a identificação da *widget* e com a origem dos dados. No que respeita a propriedades, a *widget* EditRecord não apresenta alterações significativas quando comparada com a *widget* ShowRecord, sendo usada a mesma semântica para definição do registo a editar. Quanto à apresentação, por permitir a edição de dados, a *widget* EditRecord substitui a utilização de *widgets* Expression por *widgets* capazes de receber *input* do utilizador, neste exemplo *widgets* do tipo InputWidget, CheckBoxWidget e, ComboBoxWidget.

A *widget* EditRecord é igualmente representada por um conjunto de *widgets* representativas de um registo, não necessitando por isso de suporte específico na *Framework Core* quando utilizada na *Application Test Framework*. A Figura 4.23 mostra a implementação da *widget* EditRecord na *Application Test Framework*

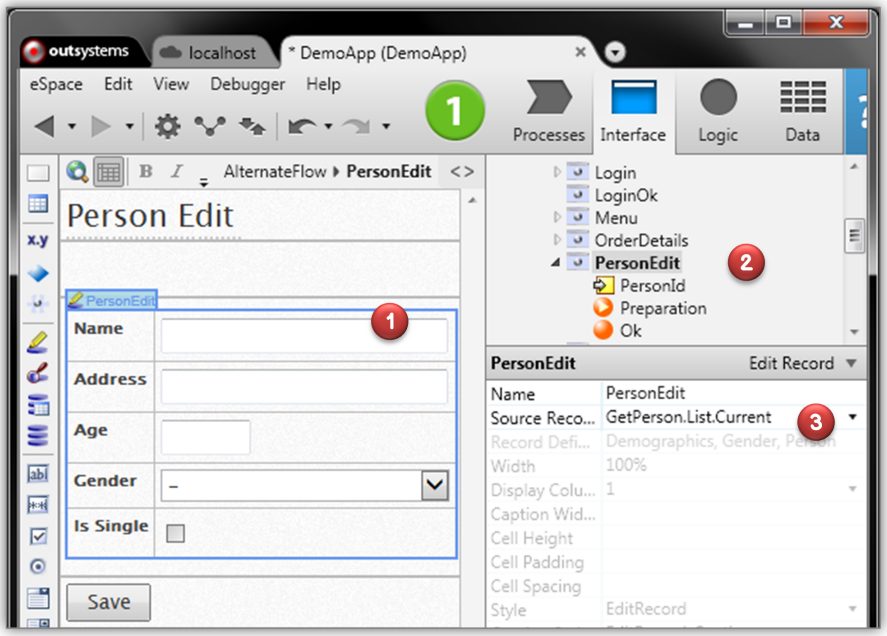

Figura 4.22: Utilização da *widget* EditRecord no ServiceStudio destacando-se: (1) utilização no *WebScreen PersonEdit* exemplificando a apresentação de um registo de *Person* e o registo *Demographics* associado; (2) campo de entrada *PersonId* que indica o identificador do registo a editar; e (3) propriedades relacionadas com a identificação da *widget* e com a origem dos dados a apresentar

da aplicação DemoApp destacando-se: (1) modelação do conjunto de atributos usados por entidade, e das *widgets* que os representam, num objecto do tipo `AbstractPageEntity`; (2) modelação da *widget* EditRecord num objecto que agrupa as entidades em si representadas; e (3) criação da instância da *widget* no `IPageObject` a que se refere.

Listagem 4.6: Implementação de teste com recurso à representação da *widget EditRecord* na *framework* de testes

```
1   @Test
2   public void checkPersonEdit() {
3     DemoApp app = new DemoApp();
4     PersonEditPage page = app.getPersonEditPage(1 /*personId*/);
5     page.personEdit().person.name().setValue("Jedi Luke Skywalker");
6     page.personEdit().demographics.age().setValue(31);
7     page.save().click();
8     page = (PersonEditPage) app.getPageLoaded();
9     assertEquals("Jedi Luke Skywalker", page.personEdit().person.name().
          getValue());
10    assertEquals(31, page.personEdit().demographics.age().getValue().
          intValue());
11  }
```

A Listagem 4.6 exemplifica a criação do teste à página *PersonEdit*, verificando a alteração dos atributos da entidade *Person*. O teste inicia com a criação de uma instância do `ApplicationObject` que representa a aplicação (linha 3). A

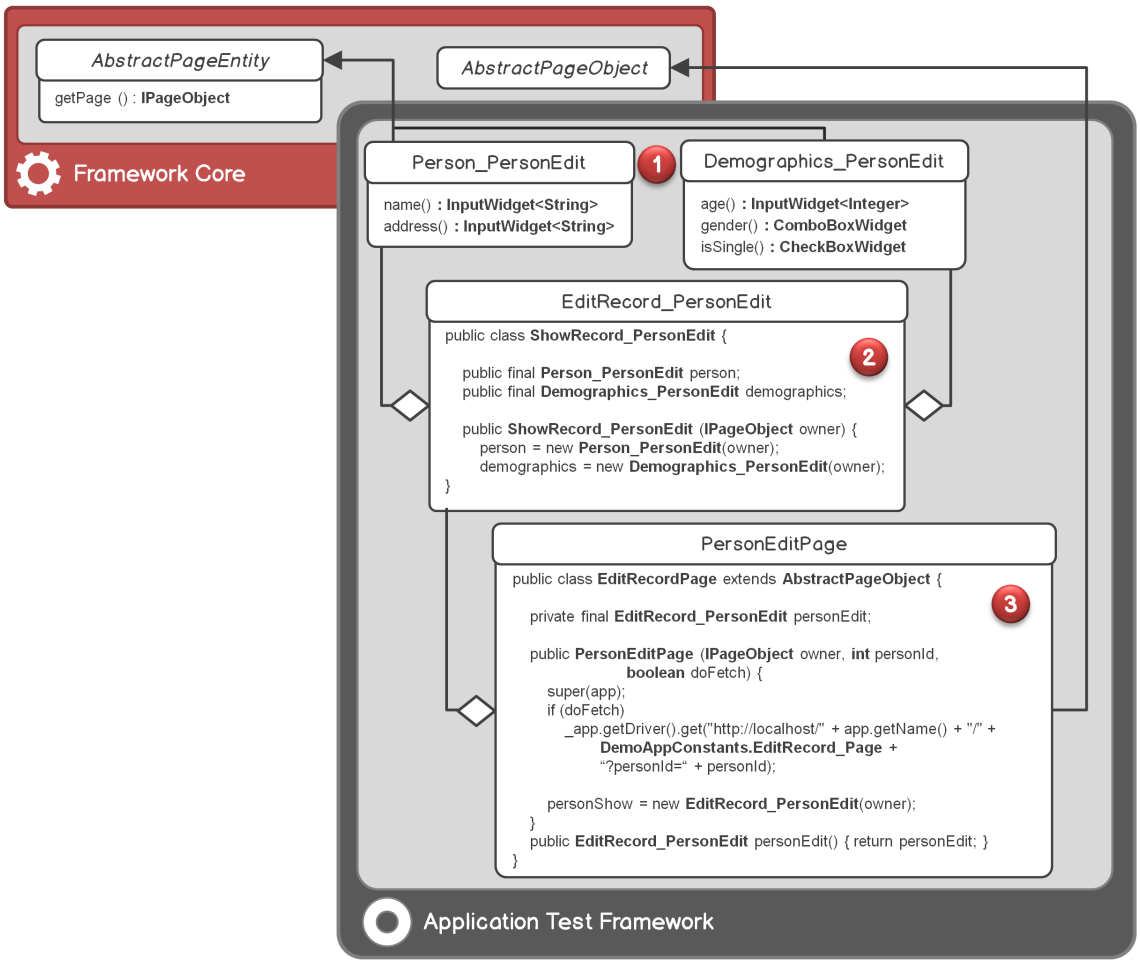

Figura 4.23: Utilização da *widget* EditRecord na *Application Test Framework* destacando-se: (1) modelação do conjunto de atributos usados por entidade, e das *widgets* que os representam, num objecto do tipo *AbstractPageEntity*; (2) modelação da *widget* EditRecord num objecto que agrupa as entidades em si representadas; e (3) criação da instância da *widget* no `IPageObject` a que se refere

obtenção do objecto `PersonEditPage`, que representa a página a testar, é feita na linha 4, sendo passado o valor `'1'` ao *input* da página responsável por representar o identificador de *Person*. Nas linhas 5 e 6 é alterado o valor das *widgets* que representam os atributos `name` e age da entidade *Person*. Nas linhas 7 e 8 é invocado o evento de *click* do botão `Save` que actualiza o registo em edição com os novos valores, e provoca o carregamento da página `PersonEditPage`. O teste termina nas linhas 9 e 10 com a verificação de que os atributos modificados apresentam o novo valor.

Widget Table Record

A *widget* TableRecord permite apresentar o conteúdo de uma lista de registos de

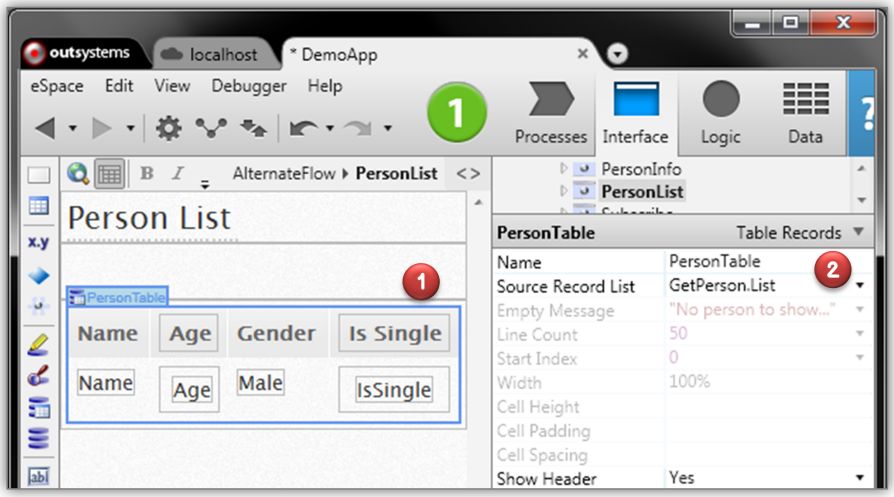

Figura 4.24: Utilização da *widget* TableRecord no ServiceStudio destacando-se: (1) utilização no *WebScreen PersonList*, onde se exemplifica a apresentação de uma lista de registos de *Person* e registos *Demographics* associados; e (2) propriedades *Name* e *Source Record List*

entidades, estruturas ou combinações dos dois. A sua representação na interface de utilização recorre a uma tabela onde as colunas representam os atributos da entidade e as linhas os vários registos. No ServiceStudio a sua utilização obriga à associação de uma variável cujo conteúdo representa uma lista de objectos do domínio da aplicação.

A Figura 4.24 apresenta a utilização da *widget* TableRecord no ServiceStudio, destacando-se: (1) utilização no *WebScreen PersonList*, onde se exemplifica a apresentação de uma lista de registos de *Person* e registos *Demographics* associados; e (2) propriedades relacionadas com a apresentação e identificação da *widget* e com a origem dos dados a apresentar, das quais se destacam o nome (*Name*), com o valor `PersonTable`, a existência de *headers* (*Show Headers*), com o valor *Yes*, e a origem de dados (*Source Record List*), com o valor `GetPerson.List`.

Uma vez que o conteúdo de cada registo da lista definida na origem de dados é representado numa linha da tabela, é possível mapear o seu conteúdo com um objecto que o represente. Recorrendo a *widgets* Expression (ponto 1, Figura 4.24), os atributos da entidade são apresentadas em cada coluna da tabela. Assim, no contexto da modelação da *widget* TableRecord na *Framework Core*, optou-se por modelar uma linha da tabela no objecto *AbstractTableLine* que, com base num índice de linha, permite o acesso às *widgets* que apresentam os valores dos atributos da entidade.

A Figura 4.25 mostra a API, hierarquia de tipos e implementação na *Framework Core* que suporta a utilização da *widget* TableRecord na *Application Test Fra-*

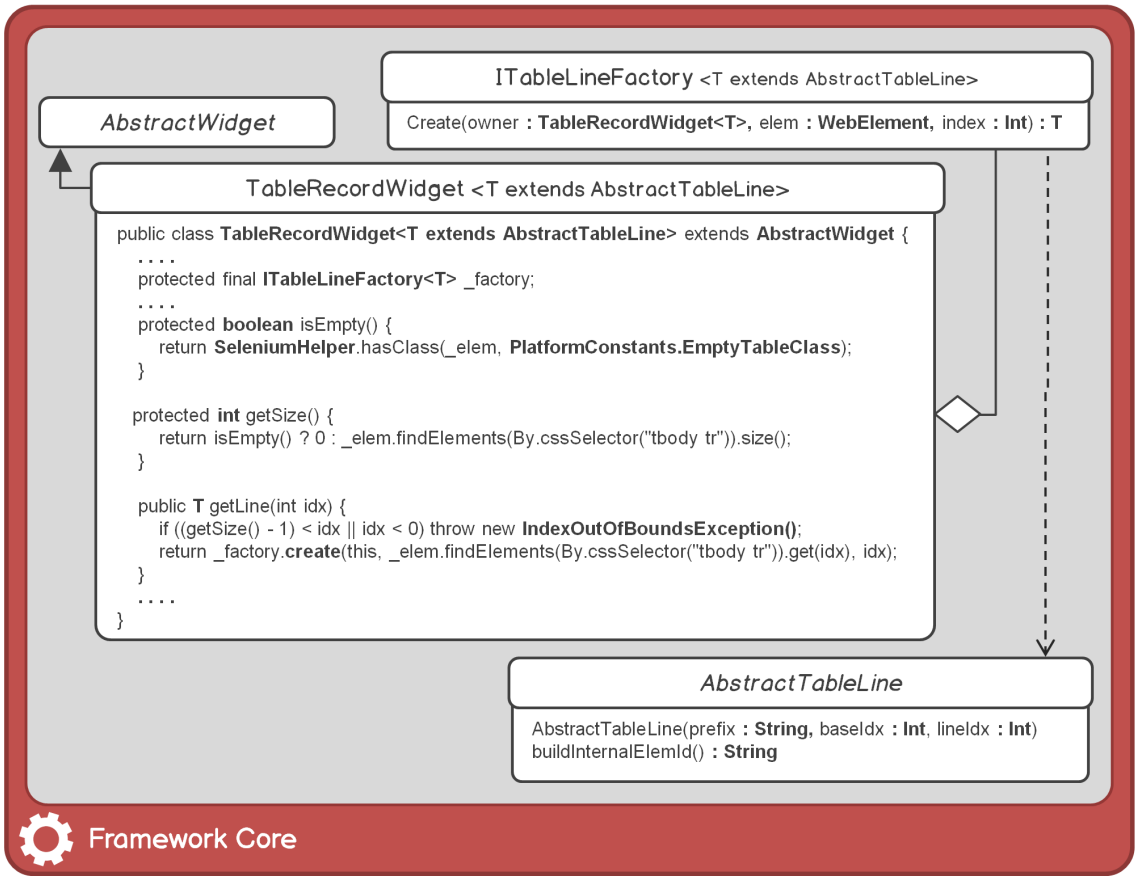

Figura 4.25: API exposta pela hierarquia de suporte a TableRecordWidget em *Framework Core*

mework. Para a implementação de TableRecord criou-se o tipo *TableRecordWidget* que recebe um parâmetro genérico relativo ao tipo da linha da tabela. A construção de uma instância desta *widget* obriga à existência de um objecto *ITableLineFactory* capaz de criar instâncias de linhas da tabela.

No que respeita à utilização da *widget* TableRecord na *Application Test Framework*, a Figura 4.26 mostra a implementação da página resultante do processo de geração, bem como, das concretizações dos tipos que suportam a *widget* na *Framework Core*.

A Listagem 4.7 exemplifica a criação do teste à página *PersonList* onde o objectivo é verificar o conteúdo da segunda linha da tabela. O teste inicia com a criação de uma instância do *ApplicationObject* que representa a aplicação (linha 3) e, da obtenção do objecto *PersonListPage* que representa a página a testar (linha 4). Na linha 5 é obtida uma instância do objecto *TableLine_PeopleTable* relativo à segunda linha da tabela. Por fim, nas linha 6 a 9 são verificados os valores apresentados em cada coluna da linha.

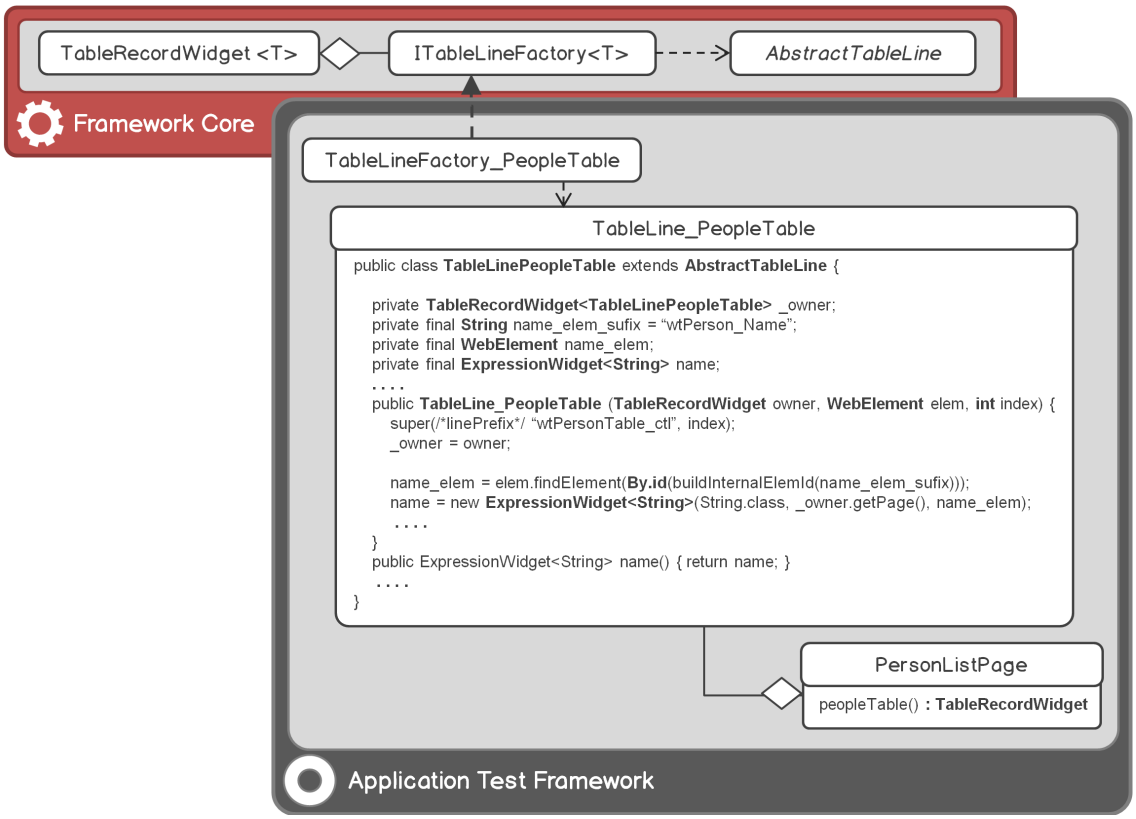

Figura 4.26: API exposta pela hierarquia de suporte a TableRecordWidget em *Application Test Framework*

Listagem 4.7: Implementação de teste com recurso ao objecto *TableRecordWidget* da *framework* de testes

```
1   @Test
2   public void checkPersonList() {
3     DemoApp app = new DemoApp();
4     PersonListPage page = app.getPersonListPage();
5     TableLine_PeopleTable line = page.peopleTable().getLine(1);
6     assertEquals("Lara Croft", line.name().getValue());
7     assertEquals(26, line.age().getValue().intValue());
8     assertEquals("Female", line.gender().getValue());
9     assertTrue(line.isSingle().getValue());
10  }
```

4.3 Caso de Estudo Revisitado

Com o objectivo de exemplificar a utilização da *OutSystems Web-Application Test Framework*, tirando partido dos objectos apresentados, será revisitada a aplicação Directory, apresentada em 3.1. Para esta aplicação será criado um teste funcional, baseado na especificação apresentada em 3.4, referente à história de utilização *'Como colaboradora, a Alice precisa de saber quem é o colega responsável pelo website, para que consiga dar a conhecer uma anomalia'*.

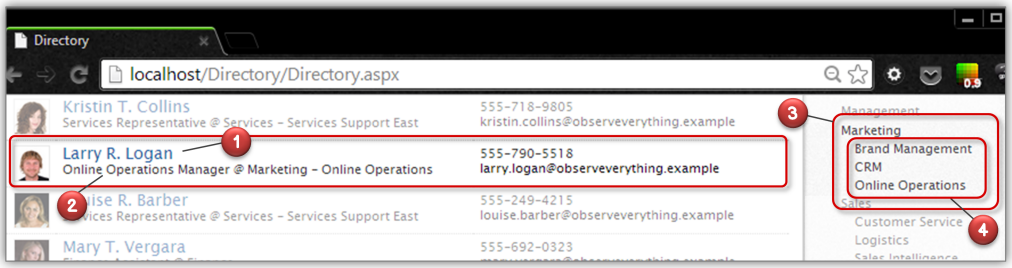

Figura 4.27: Estrutura da página principal de Directory onde se identificam: (1) Nome do colaborador, na lista de colaboradores; (2) Posição do colaborador, na lista de colaboradores; (3) Área Organizacional, na lista de áreas; e (4) Sub-Área Organizacional, na lista de sub-áreas.

A história inicia na página *'Login'*, composta por duas *wigets* de input, referentes ao *username* e *password*, e por um botão com o texto Login, que submete os valores introduzidos. De seguida a página *'Main'* é apresentada, estando a sua estrutura representada na Figura 4.27, onde se destacam os seguintes pontos: (1) Nome do colaborador, apresentado na lista de colaboradores; (2) Posição do colaborador, apresentada na lista de colaboradores; (3) Área Organizacional, apresentada na lista de áreas; e (4) Sub-Área Organizacional, apresentada na lista de sub-areas. A selecção de uma área ou sub-área organizacional provoca novamente o carregamento da página *'Main'*, com os colaboradores filtrdos de acordo com a selecção.

Para ser possível a criação de um teste funcional à aplicação Directory, tirando partido da *OutSystems Web-Application Test Framework*, foi feita uma implementação manual da *Application Test Framework*, camada específica à aplicação, onde apenas se consideraram as páginas envolvidas nesta história de utilização, bem como, as *widgets* manipuladas pelo teste. No futuro, esta implementação manual poderia ser substituida por uma geração da *Application Test Framework*, o que está fora do âmbito do presente trabalho.

A Listagem 4.8 mostra a implementação do teste, que corresponde à história de utilização referida. O teste inicia com a criação de uma instância do `ApplicationObject` que representa a aplicação (linha 3) e, da obtenção do objecto `LoginPage` que representa a página de Login (linha 4). Nas linhas 7 a 9 é feito o login na aplicação, sendo introduzido o username e password nas *widgets* apropriadas, e invocado o evento de *click* sobre o botão Login. Na linha 12 é obtida a instância de `MainPage`, que representa a páginna carregada no *driver*. A linha 13 selecciona a quarta linha da tabela de lista de áreas, guardando a sua referência na instância `'orgUnitTableLine'`. O acesso à tabela de sub-áreas, na linha referida por `'orgUnitTableLine'`, é feito na linha 14, sendo invocado o evento de click sobre a *widget LinkWidget* que representa a sub-área `'Online Operations'`. O teste

Listagem 4.8: Implementação de teste com recurso à *OutSystems Web-Application Test Framework*

```
1   @Test
2   public void searchForOnlineOperationsManager() {
3     Directory app = new Directory();
4     LoginPage loginPage = app.getLoginPage();
5
6     // Login
7     loginPage.userNameInput().setValue("alice.andrews");
8     loginPage.passwordInput().setValue("outsystems");
9     loginPage.loginButton().click();
10
11    // Selecção de área da organização
12    MainPage mainPage = (MainPage) app.getPageLoaded();
13    TableLine_OrgUnitsTable orgUnitTableLine = mainPage.orgUnitsTable().getLine(3);
14    orgUnitTableLine.subOrgUnitsTable().getLine(2).subOrgUnit().click();
15
16    // Verificação de dados do colaborador
17    mainPage = (MainPage) app.getPageLoaded();
18    TableLine_EmployeeTable employee = mainPage.employeeTable().getLine(1);
19    assertEquals("Larry R. Logan", employee.name().getText());
20    assertEquals("Online Operations Manager", employee.position().getValue());
21  }
```

termina nas linhas 19 a 20 com a verificação de que os valores apresentados na tabela de colaboradores correspondem aos esperados.

A implementação da *Application Test Framework* referente à aplicação Directory permitiu avaliar o adequação da *OutSystems Web-Application Test Framework* à interface de utilização de uma aplicação real e cujo modelo apresenta uma maior complexidade, nomeadamante no que respeita à utilização de *TableRecords*. Neste aspecto, a página *MainPage* da aplicação Directory apresenta a particularidade de no seu modelo, definido no ServiceStudio, ter sido utilizada uma *widget TableRecord* dentro de outra *widget TableRecord*. Este padrão colocou o desafio de conseguir modelar com a *framework* a possibilidade de cada linha de uma tabela poder conter outra tabela. A implementação proposta da *Application Test Framework* foi feita com o cuidado da API da página ser de fácil utilização, dando origem a código complexo que, na existência de um processo automático de geração, não seria conhecido por quem cria os testes.

A utilização da *framework* permitiu retirar do teste as dependências directas com o HTML da página, existentes na implementação proposta com Selenium *WebDriver* na Listagem 3.2. Além da escrita do teste ser facilitada pela API exposta, a sua legibilidade foi também melhorada, podendo-se traduzir a longo prazo numa melhor manutenção.

Capítulo 5

Conclusões

A necessidade de serem feitas alterações frequentes às aplicações web exige que os testes que verificam a sua funcionalidade, interagindo com a interface de utilização, sejam robustos a tais alterações. Contudo, devido às dependências que estes têm com a estrutura HTML das páginas, por necessitarem de interagir com os seus elementos, a sua execução falha não por alterações à especificação da aplicação, mas pela estrutura das suas páginas ser diferente. A falta de robustez dos testes leva assim a que a visibilidade sobre alterações à funcionalidade por si verificada não ocorra de imediato, obrigando a uma fase de adaptação à nova estrutura das páginas. O facto da implementação dos testes ser feita a um nível muito baixo, incluíndo diversas referências a localizadores de elementos (e.g. identificadores, classes de estilo), dificulta a sua legibilidade e posterior manutenção, uma vez que não são utilizados termos relativos ao domínio da aplicação.

A identificação dos problemas apontados permitiu estabelecer os objectivos principais da solução a apresentar: i) a possibilidade de serem criados testes agnósticos ao HTML das páginas da aplicação; e ii) a utilização de uma linguagem mais alto nível, que permita o uso de termos do domínio das aplicações (e.g. produto). Para atingir estes objectivos foram utilizadas aplicações web geradas com recurso à OutSystems Platform, cujo modelo é definido visualmente no ServiceStudio. Na implementação da solução proposta foi utilizada informação presente no modelo das aplicações, nomeadamente: i) o modelo de navegação, que permite determinar todas as páginas que pertencem à aplicação e as relações existentes entre elas; ii) o modelo das páginas, onde estão definidos os elementos com os quais os testes irão interagir; e iii) os objectos de domínio, que permitem a utilização de uma linguagem orientada ao domínio da aplicação.

A abordagem seguida para a implementação da solução proposta consistiu na

criação de uma camada aplicacional, a *OutSystems Web-Application Test Framework*, que representa, num modelo de objectos, o modelo das aplicações definidas com recurso à OutSystems Platform. Esta camada aplicacional é constituida por objectos que representam as características e comportamentos da aplicação, das suas páginas e das *widgets* que as compõem. Na solução proposta a *OutSystems Web-Application Test Framework* apresenta-se dividida em duas camadas: i) a *Framework Core*, uma camada de suporte, que contém as implementações dos objectos atrás indicados, aplicáveis a qualquer aplicação, constituindo a parte fixa da solução; e ii) a *Application Test Framework*, uma camada específica, que contém a implementação dos ojectos relativa a uma aplicação concreta, onde estão localizadas as dependências com o HTML das páginas da aplicação, constituindo a parte dinâmica da solução.

A *Framework Core* foi desenhada com o objectivo de permitir a representação da aplicação, páginas e *widgets* da OutSystems Platform num modelo de objectos, disponibilizando as interfaces e implementações dos mesmos. O mapeamento entre o modelo OutSystems e os objectos da *Framework Core* foi feito, tirando partido de propriedades presentes no modelo cuja utilização foi considerada apropriada do ponto de vista de criação de testes. Assim, a cada tipo de objecto irá corresponder uma API que expõe as suas propriedades e comporamentos específicos.

A *Application Test Framework* tem a função de, implementando e extendendo objectos expostos pela *Framework Core*, representar uma aplicação. Nesta camada é feito o mapeamento de uma aplicação específica, das suas páginas e dos elementos que as compõem num modelo de objectos, estando aqui codificadas as dependências com o HTML (e.g. identificadores). Objectos da *framework* de automação, que representam um elemento específico na interface de utilização, foram utilizados para permitir a comunicação directa com a página, sendo anotados com um localizador que viabiliza esse mapeamento.

No contexto do presente trabalho a implementação da *Application Test Framework* foi feita de forma manual, permitindo a prova de conceito relativa ao mapeamento entre o modelo de uma aplicação OutSystems e o modelo de objectos numa *framework* de testes. A implementação de um gerador automático da *Application Test Framework* não foi feita por exigir um esforço adicional considerável e por apresentar um valor menos significativo que a definição dos mapeamentos.

O resultado final da solução proposta é a interacção dos testes directamente com a *OutSystems Web-Application Test Framework* para testar a aplicação, aceder às suas páginas e referir os elementos que as compõem. O objectivo de tornar os testes agnósticos à estrutura das páginas foi alcançado com a estratégia de centra-

lizar na *Application Test Framework* as dependências com o HTML das aplicações, possibilitando a manipulação de instâncias de *widgets* sem conhecer a sua localização na página. Por tirar partido da informação relativa a objectos de domínio, presente no modelo da aplicação, foi possível modelar objectos que os representam na *Application Test Framework*, expondo a sua estrutura em API própria, alcançando-se também o objectivo de permitir a criação de testes utilizando uma linguagem mais alto nível.

5.1 Desenvolvimentos Futuros

A solução proposta permitiu caracterizar a forma como o modelo de uma aplicação OutSystems se mapeia numa camada aplicacional a ser utilizada na criação de testes funcionais à aplicação. Contudo, o tempo disponível para a realização deste trabalho não permitiu abordar questões que eram conhecidas à partida, bem como, questões que agora fazem sentido no âmbito da problemática do teste funcional a aplicações web geradas com a OutSystems Platform.

Das questões conhecidas à partida destaca-se a especificação do comportamento da *framework* quando utilizada em conjunto com padrões Ajax e com Javascript. Apesar da utilização de Ajax ser possível e suportada, as implementações apresentadas não tomam em consideração a característica dinâmica a que os elementos das páginas são sujeitos, obrigando a um trabalho mais cuidado na criação de uma especificação mais detalhada. A execução de Javascript por parte da *framework* é também um tópico a desenvolver existindo aqui um grande desafio ao nível da detecção de alterações ao DOM ou cenários em que é provocada uma redirecção para uma outra página.

A implementação de um gerador automático da *Application Test Framework* é também uma tarefa de grande importância no sentido em que permite a validação da solução apresentada com uma aplicação real. Das validações possíveis destacam-se: i) a avaliação da robustez com que a *Application Test Framework* reage a alterações à aplicação; e ii) a obtenção de métricas relativas aos ganhos de produtividade conseguidos com esta abordagem.

Com o objectivo de tirar partido da facilidade de utilização oferecida pela linguagem visual OutSystems, a possibilidade de criar testes directamente no ServiceStudio é uma abordagem a investigar. Nela deverá ser considerada a possibilidade de estender a DSL, incorporando elementos capazes de definir os testes de forma visual, colocando-os ao nível do modelo da aplicação, podendo-se tirar partido desta proximidade para os testes influenciarem o código da aplicação. Um exemplo

possível seria a geração de código específico na aplicação para obtenção de métricas de ocupação de memória e tempo gasto no processamento das páginas envolvidas num teste de performance.

A geração automática de casos de teste é também uma área a desenvolver no futuro. O objectivo principal é o de, automaticamente, criar vários tipos de teste a uma aplicação. Numa primeira fase seria considerado um tipo simples de testes que permite verificar se todas as páginas da aplicação são devolvidas (*smoke-tests*). Numa fase mais avançada poderia-se considerar a implementação de um mecanismo de regressão que permitiria, entre outros, a criação de um *test oracle* [5], capaz de guardar o resultado esperado da execução de testes, dando visibilidade sobre alterações ao comportamento ou aparência da aplicação.

Em relação ao Selenium *WebDriver*, a *framework* de automação utilizada neste trabalho, algumas melhorias podem ser feitas no sentido de melhorar a estabilidade na execução de testes. Factores como a gestão do *timeout* no carregamento de páginas podem influenciar a execução de testes em cenários em que o servidor aplicacional está em carga, demorando a devolver o recurso, ou a aplicação está a ser carregada pela primeira vez. A falta de visibilidade sobre a fim da execução de Javascript é também um factor a melhorar, permitindo um maior controlo sobre, por exemplo, manipulações ao DOM.

Apêndice A

Framework Core

Código fonte da *Framework Core* implementado no contexto deste trabalho.

A.1 Package `outsystems.watf.application`

Listagem A.1: outsystems.watf.application.IApplication

```
1  package outsystems.watf.application;
2
3  import org.openqa.selenium.WebDriver;
4  import outsystems.watf.page.IPageObject;
5
6  public interface IApplication {
7
8    public WebDriver getDriver();
9    public String getName();
10   public IPageObject getPageLoaded();
11
12 }
```

Listagem A.2: outsystems.watf.application.AbstractApplication

```
1  package outsystems.watf.application;
2
3  import java.util.concurrent.TimeUnit;
4  import org.openqa.selenium.WebDriver;
5  import org.openqa.selenium.firefox.FirefoxDriver;
6
7  public abstract class AbstractApplication implements IApplication {
8
9    public WebDriver driver;
10
11   public AbstractApplication() {
12     driver = new FirefoxDriver();
13     driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
14     driver.manage().timeouts().pageLoadTimeout(30, TimeUnit.SECONDS);
```

```
15      driver.manage().timeouts().setScriptTimeout(30, TimeUnit.SECONDS);
16    }
17
18    public abstract String getName();
19    public WebDriver getDriver() { return driver; }
20
21  }
```

A.2 Package `outsystems.watf.entity`

Listagem A.3: outsystems.watf.entity.IPageEntity

```
1  package outsystems.watf.entity;
2
3  import outsystems.watf.page.IPageObject;
4
5  public interface IPageEntity {
6
7    IPageObject getPage();
8
9  }
```

Listagem A.4: outsystems.watf.entity.AbstractPageEntity

```
1  package outsystems.watf.entity;
2
3  import outsystems.watf.page.IPageObject;
4
5  public abstract class AbstractPageEntity implements IPageEntity {
6
7    protected final IPageObject _owner;;
8
9    public AbstractPageEntity(IPageObject owner) { _owner = owner; }
10   public IPageObject getPage() { return _owner; }
11
12 }
```

A.3 Package `outsystems.watf.helper`

Listagem A.5: outsystems.watf.helper.RuntimeHelper

```
1  package outsystems.watf.helper;
2
3  import java.io.UnsupportedEncodingException;
4  import java.lang.reflect.InvocationTargetException;
5  import java.net.URLEncoder;
6  import java.util.Hashtable;
7  import java.util.List;
8
9  import outsystems.watf.exception.ParameterEncodingException;
10 import outsystems.watf.exception.UnallowedParameterException;
11 import outsystems.watf.exception.UnexpectedException;
12
13 public class RuntimeHelper {
```

```
14
15    @SuppressWarnings("unchecked")
16    public static <T> T getValue(Class<?> genType, String value) {
17      if (genType == String.class) {
18        return (T) genType.cast(value);
19      }
20      try {
21        return (T) genType.cast(
22          genType.getMethod("valueOf", String.class)
23            .invoke(null, value)
24        );
25      catch (Exception e) { throw new UnexpectedException(); }
26    }
27
28    public static String buildUrlParams(String host, String appName,
          String page,
29        Hashtable<String, Object> params, List<String> allowedInputs) {
30
31      StringBuilder url = new StringBuilder(host);
32      url.append("/").append(appName).append("/").append(page);
33
34      StringBuilder pageParams = new StringBuilder();
35      for(String param : params.keySet()) {
36        if (!allowedInputs.contains(param)) {
37          throw new UnallowedParameterException();
38        }
39        pageParams.append((pageParams.length() == 0 ? "?" : "&"));
40        pageParams.append(param).append("=");
41        try {
42          pageParams.append(
43            URLEncoder.encode(params.get(param).toString(), "UTF-8")
44          );
45        } catch (UnsupportedEncodingException e) {
46          throw new ParameterEncodingException();
47        }
48      }
49      url.append(pageParams.toString());
50      return url.toString();
51    }
52
53  }
```

Listagem A.6: outsystems.watf.helper.SeleniumHelper

```
1   package outsystems.watf.helper;
2
3   import java.util.Arrays;
4   import org.openqa.selenium.WebElement;
5
6   public class SeleniumHelper {
7
8     public static void inputClearAndSet(WebElement input, String value) {
9       input.clear();
10      input.sendKeys(value);
11    }
12
13    public static String getValue(WebElement elem) {
```

```
14      return elem.getAttribute("value");
15    }
16
17    public static String getText(WebElement elem) {
18      return elem.getText();
19    }
20
21    public static boolean isSelected(WebElement elem) {
22      return elem.isSelected();
23    }
24
25    public static boolean isVisible(WebElement elem) {
26      return elem.isDisplayed();
27    }
28
29    public static boolean isEnabled(WebElement elem) {
30      return elem.isEnabled();
31    }
32
33    public static void checkboxToggle(WebElement checkbox) {
34      checkbox.click();
35    }
36
37    public static void checkboxSet(WebElement checkbox) {
38      if (! isSelected(checkbox)) { checkboxToggle(checkbox); }
39    }
40
41    public static void checkboxClear(WebElement checkbox) {
42      if (isSelected(checkbox)) { checkboxToggle(checkbox); }
43    }
44
45    public static void radioToggle(WebElement radio) {
46      radio.click();
47    }
48
49    public static void radioSet(WebElement radio) {
50      if (! isSelected(radio)) { radioToggle(radio); }
51    }
52
53    public static void radioClear(WebElement radio) {
54      if (isSelected(radio)) { radioToggle(radio); }
55    }
56
57    public static boolean hasClass(WebElement elem, String className) {
58      return Arrays.asList(elem.getAttribute("class").split(" ")).contains
          (className);
59    }
60  }
```

A.4 Package `outsystems.watf.page`

Listagem A.7: outsystems.watf.page.IPageObject

```
1  package outsystems.watf.page;
2
3  import outsystems.watf.application.IApplication;
```

```
4
5  public interface IPageObject {
6
7    public IApplication getApp();;
8    public String getTitle();
9    public String getUrl();
10
11 }
```

Listagem A.8: outsystems.watf.page.AbstractPageObject

```
1  package outsystems.watf.page;
2
3  import outsystems.watf.application.IApplication;
4
5  public abstract class AbstractPageObject implements IPageObject{
6
7    protected final IApplication _app;
8
9    public AbstractPageObject(IApplication app) { this._app = app; }
10
11   public IApplication getApp() { return _app; }
12   public String getTitle() { return _app.getDriver().getTitle(); }
13   public String getUrl() { return _app.getDriver().getCurrentUrl(); }
14
15 }
```

A.5 Package `outsystems.watf.widget`

Listagem A.9: outsystems.watf.widget.IWidget

```
1  package outsystems.watf.widget;
2
3  import org.openqa.selenium.WebElement;
4  import outsystems.watf.page.IPageObject;
5
6  public interface IWidget {
7
8    public boolean isVisible();
9    public boolean isEnabled();
10   public WebElement getWebElement();
11   public IPageObject getPage();
12
13 }
```

Listagem A.10: outsystems.watf.widget.AbstractWidget

```
1  package outsystems.watf.widget;
2
3  import org.openqa.selenium.WebElement;
4  import outsystems.watf.page.IPageObject;
5
6  public abstract class AbstractWidget implements IWidget{
7
8    protected final IPageObject _owner;
```

```
9     protected final WebElement _elem;
10
11    public AbstractWidget(IPageObject owner, WebElement elem) {
12      _owner = owner;
13      _elem = elem;
14    }
15
16    public boolean isVisible() { return _elem.isDisplayed(); }
17    public boolean isEnabled() { return _elem.isEnabled(); }
18    public WebElement getWebElement() { return _elem; }
19    public IPageObject getPage() { return _owner; }
20
21  }
```

Listagem A.11: outsystems.watf.widget.ExpressionWidget

```
1   package outsystems.watf.widget;
2
3   import org.openqa.selenium.WebElement;
4   import outsystems.watf.helper.RuntimeHelper;
5   import outsystems.watf.page.IPageObject;
6
7   public class ExpressionWidget<T> extends AbstractWidget {
8
9     private final Class<T> _genType;
10
11    public ExpressionWidget(Class<T> genType, IPageObject owner,
          WebElement elem) {
12      super(owner, elem);
13      _genType = genType;
14    }
15
16    public T getValue() {
17      return RuntimeHelper.getValue(_genType,  _elem.getText());
18    }
19
20  }
```

Listagem A.12: outsystems.watf.widget.AbstractValidationWidget

```
1   package outsystems.watf.widget;
2
3   import java.util.Arrays;
4   import org.openqa.selenium.By;
5   import org.openqa.selenium.NoSuchElementException;
6   import org.openqa.selenium.WebElement;
7   import outsystems.watf.constants.PlatformConstants;
8   import outsystems.watf.page.IPageObject;
9
10  public abstract class AbstractValidationWidget extends AbstractWidget{
11
12    public AbstractValidationWidget(IPageObject owner, WebElement elem) {
13      super(owner, elem);
14    }
15
16    public boolean isInvalid() {
17      return Arrays.asList(
```

```
18        _elem.getAttribute("class").split(" ")).contains(
19          PlatformConstants.NOT_VALID_CLASS_NAME
20      );
21    }
22
23    public String getValidationMessage() {
24      try {
25        return _owner.getApp().getDriver().findElement(By.id(
26          "ValidationMessage_" + _elem.getAttribute("id")
27        )).getText();
28      } catch (NoSuchElementException e) { return null; }
29    }
30
31  }
```

Listagem A.13: outsystems.watf.widget.InputWidget

```
1   package outsystems.watf.widget;
2
3   import java.util.Arrays;
4   import org.openqa.selenium.WebElement;
5   import outsystems.watf.constants.PlatformConstants;
6   import outsystems.watf.helper.RuntimeHelper;
7   import outsystems.watf.page.IPageObject;
8
9   public class InputWidget<T> extends AbstractValidationWidget{
10
11    private final Class<T> _genType;
12
13    public InputWidget(Class<T> genType, IPageObject owner, WebElement
          elem) {
14      super(owner, elem);
15      _genType = genType;
16    }
17
18    public T getValue() {
19      return RuntimeHelper.getValue(_genType, _elem.getAttribute("value"))
            ;
20    }
21
22    public void setValue(T value) {
23      _elem.clear();
24      _elem.sendKeys(value.toString());
25    }
26
27    public String getPrompt() {
28      return _elem.getAttribute("title");
29    }
30
31    public boolean isMandatory() {
32      return Arrays.asList(
33        _elem.getAttribute("class").split(" ")).contains(
34          PlatformConstants.MANDATORY_CLASS_NAME
35      );
36    }
37  }
```

Listagem A.14: outsystems.watf.widget.CheckBoxWidget

```
1  package outsystems.watf.widget;
2
3  import org.openqa.selenium.WebElement;
4  import outsystems.watf.helper.SeleniumHelper;
5  import outsystems.watf.page.IPageObject;
6
7  public class CheckBoxWidget extends AbstractValidationWidget {
8
9    public CheckBoxWidget(IPageObject owner, WebElement elem) {
10     super(owner, elem);
11   }
12
13   public boolean isSet() {
14     return SeleniumHelper.isSelected(_elem);
15   }
16
17   public void clear() {
18     SeleniumHelper.checkboxClear(_elem);
19   }
20
21   public void set() {
22     SeleniumHelper.checkboxSet(_elem);
23   }
24
25 }
```

Listagem A.15: outsystems.watf.widget.AbstractNavigationWidget

```
1  package outsystems.watf.widget;
2
3  import org.openqa.selenium.WebElement;
4  import outsystems.watf.constants.PlatformConstants.METHOD_TYPE;
5  import outsystems.watf.constants.PlatformConstants.VALIDATION_TYPE;
6  import outsystems.watf.page.IPageObject;
7
8  public abstract class AbstractNavigationWidget extends AbstractWidget {
9
10   protected final IPageObject _owner;
11   protected final METHOD_TYPE _method;
12   protected final VALIDATION_TYPE _validation;
13   protected final boolean _isDefault;
14
15   public AbstractNavigationWidget(IPageObject owner, WebElement elem,
16       METHOD_TYPE method, VALIDATION_TYPE validation, boolean isDefault)
            {
17     super(owner, elem);
18     _owner = owner;
19     _method = method;
20     _validation = validation;
21     _isDefault = isDefault;
22   }
23
24   public METHOD_TYPE getMethodType() { return _method; }
25   public VALIDATION_TYPE getValidationType() { return _validation; }
26   public boolean isDefault() { return _isDefault; }
27
```

```
28    @SuppressWarnings("unchecked")
29    public <T extends IPageObject> T click() {
30      _elem.click();
31      return (T) _owner.getApp().getPageLoaded();
32    }
33
34  }
```

Listagem A.16: outsystems.watf.widget.ButtonWidget

```
1   package outsystems.watf.widget;
2
3   import org.openqa.selenium.WebElement;
4   import outsystems.watf.constants.PlatformConstants.METHOD_TYPE;
5   import outsystems.watf.constants.PlatformConstants.VALIDATION_TYPE;
6   import outsystems.watf.page.IPageObject;
7
8   public class ButtonWidget extends AbstractNavigationWidget{
9
10    public ButtonWidget(IPageObject owner, WebElement elem,
11        METHOD_TYPE method, VALIDATION_TYPE validation, boolean isDefault)
              {
12      super(owner, elem, method, validation, isDefault);
13    }
14
15    public String getLabel() {
16      return _elem.getAttribute("value");
17    }
18
19  }
```

Listagem A.17: outsystems.watf.widget.LinkWidget

```
1   package outsystems.watf.widget;
2
3   import org.openqa.selenium.WebElement;
4   import outsystems.watf.constants.PlatformConstants.METHOD_TYPE;
5   import outsystems.watf.constants.PlatformConstants.VALIDATION_TYPE;
6   import outsystems.watf.page.IPageObject;
7
8   public class LinkWidget extends AbstractNavigationWidget{
9
10    public LinkWidget(IPageObject owner, WebElement elem, METHOD_TYPE
          method,
11        VALIDATION_TYPE validation, boolean isDefault) {
12      super(owner, elem, method, validation, isDefault);
13    }
14
15    public String getText() {
16      return _elem.getText();
17    }
18
19    public String getTitle() {
20      return _elem.getAttribute("title");
21    }
22
23  }
```

Listagem A.18: outsystems.watf.widget.RadioWidget

```
1  package outsystems.watf.widget;
2
3  import org.openqa.selenium.WebElement;
4  import outsystems.watf.helper.RuntimeHelper;
5  import outsystems.watf.helper.SeleniumHelper;
6  import outsystems.watf.page.IPageObject;
7
8
9  public class RadioWidget<T> extends AbstractWidget {
10
11   private final Class<T> _genType;
12
13   public RadioWidget(Class<T> genType, IPageObject owner, WebElement
         elem) {
14     super(owner, elem);
15     _genType = genType;
16   }
17
18   public boolean isSet() {
19     return SeleniumHelper.isSelected(_elem);
20   }
21
22   public void set() {
23     SeleniumHelper.radioSet(_elem);
24   }
25
26   public void clear() {
27     SeleniumHelper.radioClear(_elem);
28   }
29
30   public T getValue() {
31     return RuntimeHelper.getValue(_genType, SeleniumHelper.getValue(
           _elem));
32   }
33
34 }
```

Listagem A.19: outsystems.watf.widget.AbstractRadioGroup

```
1  package outsystems.watf.widget;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import outsystems.watf.exception.UnavailableValueException;
6  import outsystems.watf.exception.UnselectableValueException;
7
8  public abstract class AbstractRadioGroup<T> {
9
10   protected List<RadioWidget<T>> _widgets;
11
12   protected final void addWidget(RadioWidget<T> widget) {
13     if (_widgets == null)
14       _widgets = new ArrayList<RadioWidget<T>>();
15     _widgets.add(widget);
16   }
17
```

```
18    public T getSelectedValue() {
19      for (RadioWidget<T> radio : _widgets) {
20        if (radio.isSet())
21          return radio.getValue();
22      }
23      return null;
24    }
25
26    public List<T> getValues() {
27      List<T> ret = new ArrayList<T>();
28      for (RadioWidget<T> radio : _widgets) {
29        ret.add(radio.getValue());
30      }
31      return ret;
32    }
33
34    public void selectValue(T val) {
35      for (RadioWidget<T> radio : _widgets) {
36        if (radio.getValue().equals(val)) {
37          if (!radio.isEnabled())
38            throw new UnselectableValueException();
39          radio.set();
40          return;
41        }
42      }
43      throw new UnavailableValueException();
44    }
45
46  }
```

Listagem A.20: outsystems.watf.widget.AbstractSelectWidget

```
1   package outsystems.watf.widget;
2
3   import java.util.Arrays;
4   import java.util.List;
5   import org.openqa.selenium.WebElement;
6   import org.openqa.selenium.support.ui.Select;
7   import outsystems.watf.constants.PlatformConstants;
8   import outsystems.watf.page.IPageObject;
9
10  public abstract class AbstractSelectWidget<V, L> extends
        AbstractValidationWidget {
11
12    protected Select _select;
13    protected final Class<V> _valueGenType;
14    protected final Class<L> _labelGenType;
15
16    public AbstractSelectWidget(Class<V> valueGenType, Class<L>
          labelGenType, IPageObject owner, WebElement elem) {
17      super(owner, elem);
18      _select = new Select(_elem);
19      _valueGenType = valueGenType;
20      _labelGenType = labelGenType;
21    }
22
23    public boolean isMandatory() {
```

```
24      return Arrays.asList(_elem.getAttribute("class").split(" "))
25        .contains(PlatformConstants.MANDATORY_CLASS_NAME);
26    }
27
28    public abstract List<? extends AbstractOption<V, L>> getOptions();
29
30    public void selectByIndex(int index) { _select.selectByIndex(index); }
31    public void selectByValue(V value) { _select.selectByValue(value.
          toString()); }
32    public void selectByLabel(L label) { _select.selectByVisibleText(label
          .toString()); }
33
34  }
```

Listagem A.21: outsystems.watf.widget.ComboBoxWidget

```
1   package outsystems.watf.widget;
2
3   import java.util.ArrayList;
4   import java.util.List;
5   import org.openqa.selenium.By;
6   import org.openqa.selenium.WebElement;
7   import outsystems.watf.constants.PlatformConstants;
8   import outsystems.watf.helper.RuntimeHelper;
9   import outsystems.watf.page.IPageObject;
10
11  public class ComboBoxWidget<V, L> extends AbstractSelectWidget<V, L> {
12
13    public ComboBoxWidget(Class<V> valueGenType, Class<L> labelGenType,
          IPageObject owner, WebElement elem) {
14      super(valueGenType, labelGenType, owner, elem);
15    }
16
17    // Dynamic List Region
18    @Override
19    public void selectByValue(V value) { selectOptionByValue(value.
          toString(), false /*isSpecial*/); }
20
21    @Override
22    public void selectByLabel(L label) { selectOptionByLabel(label.
          toString(), false /*isSpecial*/); }
23
24    public List<ComboBoxOption<V, L>> getOptions() {
25      List<ComboBoxOption<V, L>> ret = new ArrayList<ComboBoxOption<V, L
            >>();
26      for (WebElement elem : _elem.findElements(By.tagName("option"))) {
27        if (!isSpecialValue(elem) ) {
28          ret.add(optionFromElement(elem));
29        }
30      }
31      return ret;
32    }
33
34    public ComboBoxOption<V, L> getSelectedOption() {
35      for (WebElement elem : _elem.findElements(By.tagName("option"))) {
36        if (!isSpecialValue(elem) && elem.isSelected()) {
37          return optionFromElement(elem);
```

```
38        }
39      }
40      return null;
41    }
42
43    private ComboBoxOption<V, L> optionFromElement(WebElement elem) {
44      return new ComboBoxOption<V, L>(
45        RuntimeHelper.getValue(_valueGenType, elem.getAttribute("value").
              replace(PlatformConstants.SPECIAL_VALUE_PREFIX, "")),
46        RuntimeHelper.getValue(_labelGenType, elem.getText())
47      );
48    }
49
50    // Special List Region
51    private boolean isSpecialValue(WebElement elem) {
52      return elem.getAttribute("value").startsWith(PlatformConstants.
          SPECIAL_VALUE_PREFIX);
53    }
54
55    public void selectSpecialByValue(String value) { selectOptionByValue(
        value, true /*isSpecial*/); }
56    public void selectSpecialByLabel(String label) { selectOptionByLabel(
        label, true /*isSpecial*/); }
57
58    public List<ComboBoxOption<String, String>> getSpecialOptions() {
59      List<ComboBoxOption<String, String>> ret = new ArrayList<
            ComboBoxOption<String, String>>();
60      for (WebElement elem : _elem.findElements(By.tagName("option"))) {
61        if (isSpecialValue(elem)) {
62          ret.add(specialOptionFromElement(elem));
63        } else {
64          return ret;
65        }
66      }
67      return ret;
68    }
69
70    public ComboBoxOption<String, String> getSelectedSpecialOption() {
71      for (WebElement elem : _elem.findElements(By.tagName("option"))) {
72        if (!isSpecialValue(elem)) return null;
73        if (elem.isSelected()) {
74          return specialOptionFromElement(elem);
75        }
76      }
77      return null;
78    }
79
80    private ComboBoxOption<String, String> specialOptionFromElement(
        WebElement elem) {
81      return new ComboBoxOption<String, String>(
82        elem.getAttribute("value").replace(PlatformConstants.
              SPECIAL_VALUE_PREFIX, ""),
83        elem.getText()
84      );
85    }
86
87    // Utility Region
```

```
88
89    private void selectOptionByValue(String value, boolean isSpecial) {
90      for (WebElement elem : _elem.findElements(By.tagName("option"))) {
91        if (!elem.isSelected() && !(isSpecial ^ isSpecialValue(elem)) &&
              elem.getAttribute("value").equals(value)) {
92          elem.click();
93          return;
94        }
95      }
96    }
97
98    private void selectOptionByLabel(String label, boolean isSpecial) {
99      for (WebElement elem : _elem.findElements(By.tagName("option"))) {
100       if (!elem.isSelected() && !(isSpecial ^ isSpecialValue(elem)) &&
              elem.getText().equals(label)) {
101         elem.click();
102         return;
103       }
104     }
105   }
106 }
```

Listagem A.22: outsystems.watf.widget.ComboBoxOption

```
1  package outsystems.watf.widget;
2
3  public class ComboBoxOption<V, L> extends AbstractOption<V, L> {
4
5    public ComboBoxOption(Object value, Object label) {
6      super(value, label);
7    }
8
9  }
```

Listagem A.23: outsystems.watf.widget.ListBoxWidget

```
1  package outsystems.watf.widget;
2
3  import java.util.ArrayList;
4  import java.util.List;
5  import org.openqa.selenium.By;
6  import org.openqa.selenium.WebElement;
7  import outsystems.watf.helper.RuntimeHelper;
8  import outsystems.watf.page.IPageObject;
9
10 public class ListBoxWidget<L> extends AbstractSelectWidget<Integer, L> {
11
12   public ListBoxWidget(Class<L> labelGenType, IPageObject owner,
         WebElement elem) {
13     super(Integer.class, labelGenType, owner, elem);
14   }
15
16   public void deselectAll() { _select.deselectAll(); }
17   public void deselectByIndex(int index) { _select.deselectByIndex(index
         ); }
18   public void deselectByValue(int value) { _select.deselectByValue(
         Integer.toString(value)); }
```

```
19    public void deselectByLabel(L label) { _select.deselectByVisibleText(
          label.toString()); }
20
21    public List<ListBoxOption<L>> getOptions() {
22      List<ListBoxOption<L>> ret = new ArrayList<ListBoxOption<L>>();
23
24      for (WebElement elem : _elem.findElements(By.tagName("option"))) {
25        ret.add(new ListBoxOption<L>(
26          Integer.parseInt(elem.getAttribute("value")),
27          RuntimeHelper.getValue(_labelGenType, elem.getText()),
28          elem.isSelected()
29        ));
30      }
31      return ret;
32    }
33
34    public List<ListBoxOption<L>> getSelectedOptions() {
35      List<ListBoxOption<L>> ret = new ArrayList<ListBoxOption<L>>();
36
37      for(WebElement elem : _select.getAllSelectedOptions()) {
38        ret.add(new ListBoxOption<L>(
39          Integer.parseInt(elem.getAttribute("value")),
40          RuntimeHelper.getValue(_labelGenType, elem.getText()),
41          true /*isSelected*/
42        ));
43      }
44
45      return ret;
46    }
47
48    public boolean isMultiple() { return _select.isMultiple(); }
49
50  }
```

Listagem A.24: outsystems.watf.widget.ListBoxOption

```
1   package outsystems.watf.widget;
2
3   public class ListBoxOption<L> extends ComboBoxOption<Integer, L>{
4
5     private boolean _isSelected;
6
7     public ListBoxOption(int value, Object label) {
8       this(value, label, false /*isSelected*/);
9     }
10
11    public ListBoxOption(int value, Object label, boolean isSelected) {
12      super(value, label);
13      _isSelected = isSelected;
14    }
15
16    public boolean isSelected() { return _isSelected; }
17
18    @Override
19    public int hashCode() {
20      return super.hashCode() ^ (_isSelected ? 1 : 0);
21    }
```

```
22
23    @SuppressWarnings("unchecked")
24    @Override
25    public boolean equals(Object obj) {
26      if (this == obj)
27        return true;
28      if (obj == null)
29        return false;
30      ListBoxOption<L> other = (ListBoxOption<L>) obj;
31      return (super.equals(other) && (_isSelected == other.isSelected()));
32    }
33
34  }
```

Listagem A.25: outsystems.watf.widget.TableRecordWidget

```
1   package outsystems.watf.widget;
2
3   import org.openqa.selenium.By;
4   import org.openqa.selenium.WebElement;
5   import outsystems.watf.constants.PlatformConstants;
6   import outsystems.watf.exception.IndexOutOfBoundsException;
7   import outsystems.watf.helper.SeleniumHelper;
8   import outsystems.watf.page.IPageObject;
9
10  public class TableRecordWidget<T extends AbstractTableLine> extends
        AbstractWidget {
11
12    private final boolean _hasHeader;
13    private final int _cols;
14    private final ITableLineFactory<T> _factory;
15
16    public TableRecordWidget(IPageObject owner, WebElement elem,
          ITableLineFactory<T> factory, int cols, boolean hasHeader) {
17      super(owner, elem);
18      _factory = factory;
19      _cols = cols;
20      _hasHeader = hasHeader;
21    }
22
23    public int getSize() {
24      return isEmpty() ? 0 : _elem.findElements(By.cssSelector("tbody tr")
            ).size();
25    }
26
27    public int getCols() { return _cols; }
28    public boolean hasHeader() { return _hasHeader; }
29
30    private boolean isEmpty() {
31      return SeleniumHelper.hasClass(_elem, PlatformConstants.
            EMPTY_TABLE_CLASS_NAME);
32    }
33
34    public T getLine(int idx) {
35      if ((getSize() - 1) < idx || idx < 0) throw new
            IndexOutOfBoundsException();
```

```
36      return _factory.create(this, _elem.findElements(By.cssSelector("
            tbody tr")).get(idx), idx);
37    }
38
39    @Override
40    public boolean isEnabled() { return true; }
41    @Override
42    public boolean isVisible() { return true; }
43
44  }
```

Listagem A.26: outsystems.watf.widget.AbstractTableLine

```
1  package outsystems.watf.widget;
2
3  import outsystems.watf.constants.PlatformConstants;
4
5  public abstract class AbstractTableLine {
6
7    protected final String _linePrefix;
8    protected final int _lineIndex;
9
10   public AbstractTableLine(String linePrefix, int lineIndex) {
11     _linePrefix = linePrefix;
12     _lineIndex = lineIndex;
13   }
14
15   protected String buildInternalElemId(String elemId) {
16     return
17       new StringBuilder(_linePrefix)
18         .append(String.format("%02d", (PlatformConstants.
               TABLE_LINE_BASE_IDX + _lineIndex)))
19         .append("_")
20         .append(elemId)
21         .toString();
22   }
23
24 }
```

Listagem A.27: outsystems.watf.widget.ITableLineFactory

```
1  package outsystems.watf.widget;
2
3  import org.openqa.selenium.WebElement;
4
5  public interface ITableLineFactory<T extends AbstractTableLine> {
6    public T create(TableRecordWidget<T> owner, WebElement elem, int index
       );
7  }
```

Apêndice B

Application Test Framework para a aplicação DemoApp

Código fonte da *Application Test Framework*, implementado no contexto deste trabalho, específico para a aplicação DemoApp utilizada nos exemplos de apresentação das *widgets* na *framework* de testes.

B.1 Package `demoApp.application`

Listagem B.1: demoApp.application.DemoApp

```
1  package outsystems.demoApp.application;
2
3  public class DemoApp extends AbstractApplication{
4
5    protected final String appName = "DemoApp";
6
7    public String getName() { return appName; }
8
9    public AboutUsPage getAboutUsPage() { return new AboutUsPage(this); }
10   public ContactUsPage getContactUsPage() { return new ContactUsPage(
         this); }
11   public QuizPage getQuizPage() { return new QuizPage(this); }
12   public QuizOkPage getQuizOkPage() { return new QuizOkPage(this); }
13   public OrderDetailsPage getOrderDetailsPage() { return new
         OrderDetailsPage(this); }
14   public BlueLargeOrderedPage getBlueLargeOrderedPage() { return new
         BlueLargeOrderedPage(this); }
15   public ContinentSelectionPage getContinentSelectionPage() { return new
          ContinentSelectionPage(this); }
16   public CountrySelectionPage getCountrySelectionPage() { return new
         CountrySelectionPage(this); }
17   public PersonInfoPage getPersonInfoPage() { return new PersonInfoPage(
         this); }
```

```
18    public PersonEditPage getPersonEditPage() { return new PersonEditPage(
          this); }
19    public PersonListPage getPersonListPage() { return new PersonListPage(
          this); }
20
21    public IPageObject getPageLoaded() {
22      switch (driver.findElement(By.tagName("os_watf")).getAttribute("
            pageLoaded")) {
23        case DemoAppConstants.ABOUTUS_PAGE :
24          return new AboutUsPage(this, false);
25        case DemoAppConstants.CONTACTUS_PAGE :
26          return new ContactUsPage(this, false);
27        case DemoAppConstants.QUIZ_PAGE :
28          return new QuizPage(this, false);
29        case DemoAppConstants.QUIZOK_PAGE :
30          return new QuizOkPage(this, false);
31        case DemoAppConstants.ORDERDETAILS_PAGE :
32          return new OrderDetailsPage(this, false);
33        case DemoAppConstants.BLUELARGEORDERED_PAGE :
34          return new BlueLargeOrderedPage(this, false);
35        case DemoAppConstants.CONTINENTSELECTION_PAGE :
36          return new ContinentSelectionPage(this, false);
37        case DemoAppConstants.COUNTRYSELECTION_PAGE :
38          return new CountrySelectionPage(this, false);
39        case DemoAppConstants.PERSONINFO_PAGE :
40          return new PersonInfoPage(this, false);
41        case DemoAppConstants.PERSONEDIT_PAGE :
42          return new PersonEditPage(this, false);
43        case DemoAppConstants.PERSONLIST_PAGE :
44          return new PersonListPage(this, false);
45        default:
46            throw new UnidentifiedPageException();
47      }
48    }
49  }
```

B.2 Package `demoApp.constants`

Listagem B.2: demoApp.constants.DemoAppConstants

```
1  package outsystems.demoApp.constants;
2
3  public class DemoAppConstants {
4
5    public static final String MENU_PAGE = "Menu.aspx";
6    public static final String ABOUTUS_PAGE = "AboutUs.aspx";
7    public static final String CONTACTUS_PAGE = "ContactUs.aspx";
8    public static final String ATHLETEINFO_PAGE = "AthleteInfo.aspx";
9    public static final String LOGIN_PAGE = "Login.aspx";
10   public static final String LOGINOK_PAGE = "LoginOk.aspx";
11   public static final String QUIZ_PAGE = "Quiz.aspx";
12   public static final String QUIZOK_PAGE = "QuizOk.aspx";
13   public static final String SUBSCRIBE_PAGE = "Subscribe.aspx";
14   public static final String SUBSCRIBEOK_PAGE = "SubscribeOk.aspx";
15   public static final String ORDERDETAILS_PAGE = "OrderDetails.aspx";
```

```
16    public static final String BLUELARGEORDERED_PAGE = "BlueLargeOrdered.
          aspx";
17    public static final String CONTINENTSELECTION_PAGE = "
          ContinentSelection.aspx";
18    public static final String COUNTRYSELECTION_PAGE = "CountrySelection.
          aspx";
19    public static final String PERSONINFO_PAGE = "PersonInfo.aspx";
20    public static final String PERSONEDIT_PAGE = "PersonEdit.aspx";
21    public static final String PERSONLIST_PAGE = "PersonList.aspx";
22
23  }
```

B.3 Package demoApp.page

Listagem B.3: demoApp.page.AboutUs

```
1  package outsystems.demoApp.page;
2
3  import org.openqa.selenium.WebElement;
4  import org.openqa.selenium.support.FindBy;
5
6  import outsystems.watf.application.IApplication;
7  import outsystems.watf.constants.PlatformConstants.METHOD_TYPE;
8  import outsystems.watf.constants.PlatformConstants.VALIDATION_TYPE;
9  import outsystems.watf.page.AbstractPageObject;
10 import outsystems.watf.widget.ButtonWidget;
11 import outsystems.watf.widget.LinkWidget;
12 import demoApp.constants.DemoAppConstants;
13
14 public class AboutUsPage extends AbstractPageObject {
15
16   @FindBy(id = "wtBack")
17   private final WebElement back_elem = null;
18   private final ButtonWidget back;
19
20   public AboutUsPage(IApplication app) { this(app, true); }
21   public AboutUsPage(IApplication app, boolean doFetch) {
22     super(app);
23     if (doFetch)
24       _app.getDriver().get("http://localhost/" + app.getName() + "/" +
             DemoAppConstants.ABOUTUS_PAGE);
25
26     back = new ButtonWidget(this, back_elem, METHOD_TYPE.Submit,
           VALIDATION_TYPE.None, true);
27   }
28
29   public ButtonWidget back() { return back; }
30 }
```

Listagem B.4: demoApp.page.ContactUs

```
1  package outsystems.demoApp.page;
2
3  import org.openqa.selenium.WebElement;
4  import org.openqa.selenium.support.FindBy;
5
```

```
6  import outsystems.watf.application.IApplication;
7  import outsystems.watf.constants.PlatformConstants.METHOD_TYPE;
8  import outsystems.watf.constants.PlatformConstants.VALIDATION_TYPE;
9  import outsystems.watf.page.AbstractPageObject;
10 import outsystems.watf.widget.ButtonWidget;
11 import outsystems.watf.widget.LinkWidget;
12 import demoApp.constants.DemoAppConstants;
13
14 public class ContactUsPage extends AbstractPageObject {
15
16   @FindBy(id = "wtBack")
17   private final WebElement back_elem = null;
18   private final LinkWidget back;
19
20   public ContactUsPage(IApplication app) { this(app, true); }
21   public ContactUsPage(IApplication app, boolean doFetch) {
22     super(app);
23     if (doFetch)
24       _app.getDriver().get("http://localhost/" + app.getName() + "/" +
           DemoAppConstants.CONTACTUS_PAGE);
25
26     back = new LinkWidget(this, back_elem, METHOD_TYPE.Submit,
         VALIDATION_TYPE.None, false);
27   }
28
29   public LinkWidget back() { return back; }
30 }
```

Listagem B.5: demoApp.page.ContinentSelectionPage

```
1  package outsystems.demoApp.page;
2
3  import org.openqa.selenium.WebElement;
4  import org.openqa.selenium.support.FindBy;
5  import org.openqa.selenium.support.PageFactory;
6
7  import outsystems.watf.application.IApplication;
8  import outsystems.watf.page.AbstractPageObject;
9  import outsystems.watf.widget.ComboBoxWidget;
10 import demoApp.constants.DemoAppConstants;
11
12 public class ContinentSelectionPage extends AbstractPageObject {
13
14   @FindBy(id = "wtContinent")
15   private final WebElement continent_elem = null;
16   private final ComboBoxWidget<String, String> continent;
17
18   public ContinentSelectionPage(IApplication app) { this(app, true); }
19   public ContinentSelectionPage(IApplication app, boolean doFetch) {
20     super(app);
21     if (doFetch)
22       _app.getDriver().get("http://localhost/" + app.getName() + "/" +
           DemoAppConstants.CONTINENTSELECTION_PAGE);
23
24     PageFactory.initElements(_app.getDriver(), this);
25
```

```
26      continent = new ComboBoxWidget<String, String>(String.class, String.
            class, this, continent_elem);
27    }
28
29    public ComboBoxWidget<String, String> continent() { return continent;
          }
30  }
```

Listagem B.6: demoApp.page.CountrySelectionPage

```
1   package outsystems.demoApp.page;
2
3   import org.openqa.selenium.WebElement;
4   import org.openqa.selenium.support.FindBy;
5   import org.openqa.selenium.support.PageFactory;
6
7   import outsystems.watf.application.IApplication;
8   import outsystems.watf.page.AbstractPageObject;
9   import outsystems.watf.widget.ListBoxWidget;
10  import demoApp.constants.DemoAppConstants;
11
12  public class CountrySelectionPage extends AbstractPageObject {
13
14    @FindBy(id = "wtCountry")
15    private final WebElement country_elem = null;
16    private final ListBoxWidget<String> country;
17
18    public CountrySelectionPage(IApplication app) { this(app, true); }
19    public CountrySelectionPage(IApplication app, boolean doFetch) {
20      super(app);
21      if (doFetch)
22        _app.getDriver().get("http://localhost/" + app.getName() + "/" +
              DemoAppConstants.COUNTRYSELECTION_PAGE);
23
24      PageFactory.initElements(_app.getDriver(), this);
25
26      country = new ListBoxWidget<String>(String.class, this, country_elem
            );
27    }
28
29    public ListBoxWidget<String> country() {
30      return country;
31    }
32  }
```

Listagem B.7: demoApp.page.OrderDetailsPage

```
1   package outsystems.demoApp.page;
2
3   import org.openqa.selenium.WebElement;
4   import org.openqa.selenium.support.FindBy;
5   import org.openqa.selenium.support.PageFactory;
6
7   import outsystems.watf.application.IApplication;
8   import outsystems.watf.constants.PlatformConstants.METHOD_TYPE;
9   import outsystems.watf.constants.PlatformConstants.VALIDATION_TYPE;
10  import outsystems.watf.page.AbstractPageObject;
```

```
11 import outsystems.watf.widget.AbstractRadioGroup;
12 import outsystems.watf.widget.ButtonWidget;
13 import demoApp.alternateFlow.widget.orderDetailsPage.RadioGroup_ColorVal
       ;
14 import demoApp.alternateFlow.widget.orderDetailsPage.RadioGroup_SizeCode
       ;
15 import demoApp.constants.DemoAppConstants;
16
17 public class OrderDetailsPage extends AbstractPageObject {
18
19   private final RadioGroup_ColorVal colorVal;
20   private final RadioGroup_SizeCode sizeCode;
21
22   @FindBy(id = "wtCheckout")
23   private final WebElement checkout_elem = null;
24   private final ButtonWidget checkout;
25
26   public OrderDetailsPage(IApplication app) { this(app, true); }
27   public OrderDetailsPage(IApplication app, boolean doFetch) {
28     super(app);
29     if (doFetch)
30       _app.getDriver().get("http://localhost/" + app.getName() + "/" +
             DemoAppConstants.ORDERDETAILS_PAGE);
31
32     PageFactory.initElements(_app.getDriver(), this);
33
34     colorVal = new RadioGroup_ColorVal(this);
35     sizeCode = new RadioGroup_SizeCode(this);
36     checkout =  new ButtonWidget(this, checkout_elem, METHOD_TYPE.Submit
           , VALIDATION_TYPE.None, true);
37   }
38
39   public AbstractRadioGroup<Integer> colorVal() { return colorVal; }
40   public AbstractRadioGroup<String> sizeCode() { return sizeCode; }
41   public ButtonWidget checkout() { return checkout; }
42 }
```

Listagem B.8: demoApp.page.PersonEditPage

```
1  package outsystems.demoApp.page;
2
3  import outsystems.watf.application.IApplication;
4  import outsystems.watf.page.AbstractPageObject;
5  import demoApp.alternateFlow.widget.personEditPage.EditRecord_PersonEdit
       ;
6  import demoApp.constants.DemoAppConstants;
7
8  public class PersonEditPage extends AbstractPageObject {
9
10   private final EditRecord_PersonEdit personEdit;
11
12   public PersonEditPage(IApplication app) { this(app, true); }
13   public PersonEditPage(IApplication app, boolean doFetch) {
14     super(app);
15     if (doFetch)
16       _app.getDriver().get("http://localhost/" + app.getName() + "/" +
             DemoAppConstants.PERSONEDIT_PAGE);
```

```
17
18      personEdit = new EditRecord_PersonEdit(this);
19    }
20
21    public EditRecord_PersonEdit personEdit() { return personEdit; }
22  }
```

Listagem B.9: demoApp.page.PersonInfoPage

```
1   package outsystems.demoApp.page;
2
3   import outsystems.watf.application.IApplication;
4   import outsystems.watf.page.AbstractPageObject;
5   import demoApp.alternateFlow.widget.personInfoPage.ShowRecord_PersonShow
        ;
6   import demoApp.constants.DemoAppConstants;
7
8   public class PersonInfoPage extends AbstractPageObject {
9
10    private final ShowRecord_PersonShow personShow;
11
12    public PersonInfoPage(IApplication app) { this(app, true); }
13    public PersonInfoPage(IApplication app, boolean doFetch) {
14      super(app);
15      if (doFetch)
16        _app.getDriver().get("http://localhost/" + app.getName() + "/" +
              DemoAppConstants.PERSONINFO_PAGE);
17
18      personShow = new ShowRecord_PersonShow(this);
19    }
20
21    public ShowRecord_PersonShow personShow() { return personShow; }
22  }
```

Listagem B.10: demoApp.page.PersonListPage

```
1   package outsystems.demoApp.page;
2
3   import org.openqa.selenium.By;
4
5   import outsystems.watf.application.IApplication;
6   import outsystems.watf.page.AbstractPageObject;
7   import outsystems.watf.widget.TableRecordWidget;
8   import demoApp.alternateFlow.widget.personListPage.
        TableLineFactory_PeopleTable;
9   import demoApp.alternateFlow.widget.personListPage.TableLine_PeopleTable
        ;
10  import demoApp.constants.DemoAppConstants;
11
12  public class PersonListPage extends AbstractPageObject {
13
14    private final int _numCols = 4;
15
16    private final TableLineFactory_PeopleTable _lineFactory;
17    private final TableRecordWidget<TableLine_PeopleTable> _peopleTable;
18
19    public PersonListPage(IApplication app) { this(app, true); }
```

```
20      public PersonListPage(IApplication app, boolean doFetch) {
21      super(app);
22      if (doFetch)
23        _app.getDriver().get("http://localhost/" + app.getName() + "/" +
              DemoAppConstants.PERSONLIST_PAGE);
24
25      _lineFactory = new TableLineFactory_PeopleTable();
26      _peopleTable = new TableRecordWidget<TableLine_PeopleTable>(this,
27        _app.getDriver().findElement(By.id("wtPersonTable")), _lineFactory
              , _numCols, true
28      );
29    }
30
31    public TableRecordWidget<TableLine_PeopleTable> peopleTable() { return
          _peopleTable; }
32  }
```

Listagem B.11: demoApp.page.QuizPage

```
1   package outsystems.demoApp.page;
2
3   import org.openqa.selenium.WebElement;
4   import org.openqa.selenium.support.FindBy;
5   import org.openqa.selenium.support.PageFactory;
6
7   import outsystems.watf.application.IApplication;
8   import outsystems.watf.constants.PlatformConstants.METHOD_TYPE;
9   import outsystems.watf.constants.PlatformConstants.VALIDATION_TYPE;
10  import outsystems.watf.page.AbstractPageObject;
11  import outsystems.watf.widget.ButtonWidget;
12  import outsystems.watf.widget.CheckBoxWidget;
13  import outsystems.watf.widget.ExpressionWidget;
14  import outsystems.watf.widget.InputWidget;
15  import demoApp.constants.DemoAppConstants;
16
17  public class QuizPage extends AbstractPageObject {
18
19    @FindBy(id = "wtQuestion")
20    private final WebElement question_elem = null;
21    private final ExpressionWidget<String> question;
22
23    @FindBy(id = "wtAnswer")
24    private final WebElement answer_elem = null;
25    private final InputWidget<Integer> answer;
26
27    @FindBy(id = "wtReview")
28    private final WebElement review_elem = null;
29    private final CheckBoxWidget review;
30
31    @FindBy(id = "wtLogin")
32    private final WebElement ok_elem = null;
33    private final ButtonWidget ok;
34
35    public QuizPage(IApplication app) { this(app, true); }
36    public QuizPage(IApplication app, boolean doFetch) {
37      super(app);
38      if (doFetch)
```

```
39        _app.getDriver().get("http://localhost/" + app.getName() + "/" +
              DemoAppConstants.QUIZ_PAGE);
40
41      PageFactory.initElements(_app.getDriver(), this);
42
43      question = new ExpressionWidget<String>(String.class, this,
            question_elem);
44      answer = new InputWidget<Integer>(Integer.class, this, answer_elem);
45      review = new CheckBoxWidget(this, review_elem);
46      ok = new ButtonWidget(this, ok_elem, METHOD_TYPE.Submit,
            VALIDATION_TYPE.None, true);
47    }
48
49    public ExpressionWidget<String> question() { return question; }
50    public InputWidget<Integer> answer() { return answer; }
51    public CheckBoxWidget review() { return review; }
52    public ButtonWidget ok() { return ok; }
53  }
```

Listagem B.12: demoApp.page.SubscribePage

```
1   package outsystems.demoApp.page;
2
3   import org.openqa.selenium.WebElement;
4   import org.openqa.selenium.support.FindBy;
5   import org.openqa.selenium.support.PageFactory;
6
7   import outsystems.watf.application.IApplication;
8   import outsystems.watf.constants.PlatformConstants.METHOD_TYPE;
9   import outsystems.watf.constants.PlatformConstants.VALIDATION_TYPE;
10  import outsystems.watf.page.AbstractPageObject;
11  import outsystems.watf.widget.ButtonWidget;
12  import outsystems.watf.widget.CheckBoxWidget;
13  import demoApp.constants.DemoAppConstants;
14
15  public class SubscribePage extends AbstractPageObject {
16
17    @FindBy(id = "wtSubscribe")
18    private final WebElement subscribe_elem = null;
19    private final CheckBoxWidget subscribe;
20
21    @FindBy(id = "wtContinueBtn")
22    private final WebElement continue_elem = null;
23    private final ButtonWidget continueBtn;
24
25    public SubscribePage(IApplication app) { this(app, true); }
26    public SubscribePage(IApplication app, boolean doFetch) {
27      super(app);
28      if (doFetch)
29        _app.getDriver().get("http://localhost/" + app.getName() + "/" +
              DemoAppConstants.SUBSCRIBE_PAGE);
30
31      PageFactory.initElements(_app.getDriver(), this);
32
33      subscribe = new CheckBoxWidget(this, subscribe_elem);
34      continueBtn = new ButtonWidget(this, continue_elem, METHOD_TYPE.
            Submit, VALIDATION_TYPE.None, true);
```

```
35    }
36
37    public CheckBoxWidget subscribe() { return subscribe; }
38    public ButtonWidget continueBtn() { return continueBtn; }
39  }
```

B.4 Package demoApp.entity

Listagem B.13: demoApp.entity.personEditPage.Demographics_PersonEdit

```
1  package outsystems.demoApp.entity.personEditPage;
2
3  import org.openqa.selenium.WebElement;
4  import org.openqa.selenium.support.FindBy;
5  import org.openqa.selenium.support.PageFactory;
6
7  import outsystems.watf.entity.AbstractPageEntity;
8  import outsystems.watf.page.IPageObject;
9  import outsystems.watf.widget.CheckBoxWidget;
10 import outsystems.watf.widget.ComboBoxWidget;
11 import outsystems.watf.widget.InputWidget;
12
13 public class Demographics_PersonEdit extends AbstractPageEntity {
14
15   @FindBy(id = "wtDemographic_Age")
16   private final WebElement age_elem = null;
17   private final InputWidget<Integer> age;
18
19   @FindBy(id = "wtDemographic_Gender")
20   private final WebElement gender_elem = null;
21   private final ComboBoxWidget<Integer, String> gender;
22
23   @FindBy(id = "wtDemographic_IsSingle")
24   private final WebElement isSingle_elem = null;
25   private final CheckBoxWidget isSingle;
26
27   public Demographics_PersonEdit(IPageObject owner) {
28     super(owner);
29
30     PageFactory.initElements(_owner.getApp().getDriver(), this);
31
32     age = new InputWidget<Integer>(Integer.class, _owner, age_elem);
33     gender = new ComboBoxWidget<Integer, String>(Integer.class, String.
           class, _owner, gender_elem);
34     isSingle = new CheckBoxWidget(_owner, isSingle_elem);
35   }
36
37   public InputWidget<Integer> age() { return age; }
38   public ComboBoxWidget<Integer, String> gender() { return gender; }
39   public CheckBoxWidget isSingle() { return isSingle; }
40 }
```

Listagem B.14: demoApp.entity.personEditPage.Person_PersonEdit

```
1  package outsystems.demoApp.entity.personEditPage;
2
```

```
3  import org.openqa.selenium.WebElement;
4  import org.openqa.selenium.support.FindBy;
5  import org.openqa.selenium.support.PageFactory;
6
7  import outsystems.watf.entity.AbstractPageEntity;
8  import outsystems.watf.page.IPageObject;
9  import outsystems.watf.widget.InputWidget;
10
11 public class Person_PersonEdit extends AbstractPageEntity {
12
13   @FindBy(id = "wtPerson_Name")
14   private final WebElement name_elem = null;
15   private final InputWidget<String> name;
16
17
18   @FindBy(id = "wtPerson_Address")
19   private final WebElement address_elem = null;
20   private final InputWidget<String> address;
21
22   public Person_PersonEdit(IPageObject owner) {
23     super(owner);
24
25     PageFactory.initElements(_owner.getApp().getDriver(), this);
26
27     name = new InputWidget<String>(String.class, _owner, name_elem);
28     address = new InputWidget<String>(String.class, _owner, address_elem
           );
29   }
30
31   public InputWidget<String> name() { return name; }
32   public InputWidget<String> address() { return address; }
33
34 }
```

Listagem B.15: demoApp.entity.personInfoPage.Demographics_PersonShow

```
1  package outsystems.demoApp.entity.personInfoPage;
2
3  import org.openqa.selenium.WebElement;
4  import org.openqa.selenium.support.FindBy;
5  import org.openqa.selenium.support.PageFactory;
6
7  import outsystems.watf.entity.AbstractPageEntity;
8  import outsystems.watf.page.IPageObject;
9  import outsystems.watf.widget.ExpressionWidget;
10
11 public class Demographics_PersonShow extends AbstractPageEntity {
12
13   @FindBy(id = "wtDemographic_Age")
14   private final WebElement age_elem = null;
15   private final ExpressionWidget<Integer> age;
16
17   @FindBy(id = "wtDemographic_Gender")
18   private final WebElement gender_elem = null;
19   private final ExpressionWidget<String> gender;
20
21   @FindBy(id = "wtDemographic_IsSingle")
```

```
22    private final WebElement isSingle_elem = null;
23    private final ExpressionWidget<Boolean> isSingle;
24
25    public Demographics_PersonShow(IPageObject owner) {
26      super(owner);
27
28      PageFactory.initElements(_owner.getApp().getDriver(), this);
29
30      age = new ExpressionWidget<Integer>(Integer.class, _owner, age_elem)
            ;
31      gender = new ExpressionWidget<String>(String.class, _owner,
            gender_elem);
32      isSingle = new ExpressionWidget<Boolean>(Boolean.class, _owner,
            isSingle_elem);
33    }
34
35    public ExpressionWidget<Integer> age() { return age; }
36    public ExpressionWidget<String> gender() { return gender; }
37    public ExpressionWidget<Boolean> isSingle() { return isSingle; }
38  }
```

Listagem B.16: demoApp.entity.personInfoPage.Person_PersonShow

```
1   package outsystems.demoApp.entity.personInfoPage;
2
3   import org.openqa.selenium.WebElement;
4   import org.openqa.selenium.support.FindBy;
5   import org.openqa.selenium.support.PageFactory;
6
7   import outsystems.watf.entity.AbstractPageEntity;
8   import outsystems.watf.page.IPageObject;
9   import outsystems.watf.widget.ExpressionWidget;
10
11  public class Person_PersonShow extends AbstractPageEntity {
12
13    @FindBy(id = "wtPerson_FirstName")
14    private final WebElement firstName_elem = null;
15    private final ExpressionWidget<String> firstName;
16
17    @FindBy(id = "wtPerson_LastName")
18    private final WebElement lastName_elem = null;
19    private final ExpressionWidget<String> lastName;
20
21
22    @FindBy(id = "wtPerson_Address")
23    private final WebElement address_elem = null;
24    private final ExpressionWidget<String> address;
25
26    public Person_PersonShow(IPageObject owner) {
27      super(owner);
28
29      PageFactory.initElements(_owner.getApp().getDriver(), this);
30
31      firstName = new ExpressionWidget<String>(String.class, _owner,
            firstName_elem);
32      lastName = new ExpressionWidget<String>(String.class, _owner,
            lastName_elem);
```

```
33      address = new ExpressionWidget<String>(String.class, _owner,
            address_elem);
34    }
35
36    public ExpressionWidget<String> firstName() { return firstName; }
37    public ExpressionWidget<String> lastName() { return lastName; }
38    public ExpressionWidget<String> address() { return address; }
39
40  }
```

B.5 Package demoApp.widget

Listagem B.17: demoApp.widget.orderDetailsPage.RadioGroup_ColorVal

```
1  package outsystems.demoApp.widget.orderDetailsPage;
2
3  import org.openqa.selenium.WebElement;
4  import org.openqa.selenium.support.FindBy;
5  import org.openqa.selenium.support.PageFactory;
6
7  import outsystems.watf.page.IPageObject;
8  import outsystems.watf.widget.AbstractRadioGroup;
9  import outsystems.watf.widget.RadioWidget;
10
11 public class RadioGroup_ColorVal extends AbstractRadioGroup<Integer>{
12
13   @FindBy(id = "wtRed")
14   private final WebElement red_elem = null;
15   private final RadioWidget<Integer> red;
16
17   @FindBy(id = "wtGreen")
18   private final WebElement green_elem = null;
19   private final RadioWidget<Integer> green;
20
21   @FindBy(id = "wtBlue")
22   private final WebElement blue_elem = null;
23   private final RadioWidget<Integer> blue;
24
25   public RadioGroup_ColorVal(IPageObject owner) {
26
27     PageFactory.initElements(owner.getApp().getDriver(),  this);
28
29     red = new RadioWidget<Integer>(Integer.class, owner, red_elem);
30     addWidget(red);
31     green = new RadioWidget<Integer>(Integer.class, owner, green_elem);
32     addWidget(green);
33     blue = new RadioWidget<Integer>(Integer.class, owner, blue_elem);
34     addWidget(blue);
35   }
36 }
```

Listagem B.18: demoApp.widget.orderDetailsPage.RadioGroup_SizeCode

```
1  package outsystems.demoApp.widget.orderDetailsPage;
2
3  import org.openqa.selenium.WebElement;
```

```
4  import org.openqa.selenium.support.FindBy;
5  import org.openqa.selenium.support.PageFactory;
6
7  import outsystems.watf.page.IPageObject;
8  import outsystems.watf.widget.AbstractRadioGroup;
9  import outsystems.watf.widget.RadioWidget;
10
11 public class RadioGroup_SizeCode extends AbstractRadioGroup<String>{
12
13   @FindBy(id = "wtMedium")
14   private final WebElement medium_elem = null;
15   private final RadioWidget<String> medium;
16
17   @FindBy(id = "wtLarge")
18   private final WebElement large_elem = null;
19   private final RadioWidget<String> large;
20
21   @FindBy(id = "wtExtraLarge")
22   private final WebElement extraLarge_elem = null;
23   private final RadioWidget<String> extraLarge;
24
25   public RadioGroup_SizeCode(IPageObject owner) {
26
27     PageFactory.initElements(owner.getApp().getDriver(),  this);
28
29     medium = new RadioWidget<String>(String.class, owner, medium_elem);
30     addWidget(medium);
31     large = new RadioWidget<String>(String.class, owner, large_elem);
32     addWidget(large);
33     extraLarge = new RadioWidget<String>(String.class, owner,
           extraLarge_elem);
34     addWidget(extraLarge);
35   }
36 }
```

Listagem B.19: demoApp.widget.personEditPage.EditRecord_PersonEdit

```
1  package outsystems.demoApp.widget.personEditPage;
2
3  import outsystems.watf.page.IPageObject;
4  import demoApp.mainFlow.entity.editRecordPage.Demographics_PersonEdit;
5  import demoApp.mainFlow.entity.editRecordPage.Person_PersonEdit;
6
7
8  public class EditRecord_PersonEdit {
9
10   public final Person_PersonEdit person;
11   public final Demographics_PersonEdit demographics;
12
13   public EditRecord_PersonEdit(IPageObject owner) {
14     person = new Person_PersonEdit(owner);
15     demographics = new Demographics_PersonEdit(owner);
16   }
17 }
```

Listagem B.20: demoApp.widget.personInfoPage.EditRecord_PersonShow

```
1  package outsystems.demoApp.widget.personInfoPage;
```

```
2
3  import outsystems.watf.page.IPageObject;
4  import demoApp.mainFlow.entity.showRecordPage.Demographics_PersonShow;
5  import demoApp.mainFlow.entity.showRecordPage.Person_PersonShow;
6
7
8  public class ShowRecord_PersonShow {
9
10   public final Person_PersonShow person;
11   public final Demographics_PersonShow demographics;
12
13   public ShowRecord_PersonShow(IPageObject owner) {
14     person = new Person_PersonShow(owner);
15     demographics = new Demographics_PersonShow(owner);
16   }
17 }
```

Listagem B.21: demoApp.widget.personListPage.TableLine_PeopleTable

```
1  package outsystems.demoApp.widget.personListPage;
2
3  import org.openqa.selenium.By;
4  import org.openqa.selenium.WebElement;
5
6  import outsystems.watf.widget.AbstractTableLine;
7  import outsystems.watf.widget.ExpressionWidget;
8  import outsystems.watf.widget.TableRecordWidget;
9
10 public class TableLine_PeopleTable extends AbstractTableLine {
11
12   private final TableRecordWidget<TableLine_PeopleTable> _owner;
13
14   private final String name_elem_sufix = "wtPerson_Name";
15   private final WebElement name_elem;
16   private final ExpressionWidget<String> name;
17
18   private final String age_elem_sufix = "wtDemographic_Age";
19   private final WebElement age_elem;
20   private final ExpressionWidget<Integer> age;
21
22   private final String gender_elem_sufix = "wtDemographic_Gender";
23   private final WebElement gender_elem;
24   private final ExpressionWidget<String> gender;
25
26   private final String isSingle_elem_sufix = "wtDemographic_IsSingle";
27   private final WebElement isSingle_elem;
28   private final ExpressionWidget<Boolean> isSingle;
29
30   public TableLine_PeopleTable(TableRecordWidget<TableLine_PeopleTable>
       owner, WebElement elem, int index) {
31     super(/*linePrefix*/owner.getWebElement().getAttribute("id") + "_ctl
         ", index);
32     _owner = owner;
33
34     name_elem = elem.findElement(By.id(buildInternalElemId(
         name_elem_sufix)));
```

```
35      age_elem = elem.findElement(By.id(buildInternalElemId(age_elem_sufix
            )));
36      gender_elem = elem.findElement(By.id(buildInternalElemId(
            gender_elem_sufix)));
37      isSingle_elem = elem.findElement(By.id(buildInternalElemId(
            isSingle_elem_sufix)));
38
39      name = new ExpressionWidget<String>(String.class, _owner.getPage(),
            name_elem);
40      age = new ExpressionWidget<Integer>(Integer.class, _owner.getPage(),
             age_elem);
41      gender = new ExpressionWidget<String>(String.class, _owner.getPage()
            , gender_elem);
42      isSingle = new ExpressionWidget<Boolean>(Boolean.class, _owner.
            getPage(), isSingle_elem);
43    }
44
45    public ExpressionWidget<String> name() { return name; }
46    public ExpressionWidget<Integer> age() { return age; }
47    public ExpressionWidget<String> gender() { return gender; }
48    public ExpressionWidget<Boolean> isSingle() { return isSingle; }
49
50  }
```

Listagem B.22: demoApp.widget.personListPage.TableLineFactory_PeopleTable

```
1   package outsystems.demoApp.widget.personListPage;
2
3   import org.openqa.selenium.WebElement;
4
5   import outsystems.watf.widget.ITableLineFactory;
6   import outsystems.watf.widget.TableRecordWidget;
7
8   public class TableLineFactory_PeopleTable implements ITableLineFactory<
        TableLine_PeopleTable> {
9
10    @Override
11    public TableLine_PeopleTable create(TableRecordWidget<
          TableLine_PeopleTable> owner,
12        WebElement elem, int index) {
13      return new TableLine_PeopleTable(owner, elem, index);
14    }
15
16  }
```

Apêndice C

Application Test Framework para a aplicação Directory

Código fonte da *Application Test Framework*, implementado no contexto deste trabalho, específico para a aplicação Directory utilizada como caso de estudo.

C.1 Package `directory.application`

Listagem C.1: directory.application.Directory

```
1  public class Directory extends AbstractApplication{
2
3    public final String _appName = "Directory";
4
5    public Directory() { super(); }
6
7    public String getName() { return _appName; }
8
9    public LoginPage getLoginPage() { return new LoginPage(this); }
10   public MainPage getMainPage() { return new MainPage(this); }
11
12   public IPageObject getPageLoaded() {
13     switch (driver.findElement(By.tagName("os_watf")).getAttribute("
           pageLoaded")) {
14       case DirectoryConstants.LOGIN_PAGE :
15           return new LoginPage(this, false);
16       case DirectoryConstants.MAIN_PAGE:
17           return new MainPage(this, false);
18       default:
19           throw new UnidentifiedPageException();
20     }
21   }
22 }
```

C.2 Package `directory.constants`

Listagem C.2: directory.constants.DirectoryConstants

```
1  public class DirectoryConstants {
2
3    public static final String LOGIN_PAGE = "Login.aspx";
4    public static final String MAIN_PAGE = "Main.aspx";
5
6  }
```

C.3 Package `directory.page`

Listagem C.3: directory.page.LoginPage

```
1  public class LoginPage extends AbstractPageObject {
2
3    @FindBy(id = "wt11_wtMainContent_wtUserNameInput")
4    private final WebElement userNameInput_elem = null;
5    private final InputWidget<String> userNameInput;
6
7    @FindBy(id = "wt11_wtMainContent_wtPasswordInput")
8    private final WebElement passwordInput_elem = null;
9    private final InputWidget<String> passwordInput;
10
11   @FindBy(id = "wt11_wtMainContent_wtRememberLoginChk")
12   private WebElement rememberLoginChk_elem;
13   private final CheckBoxWidget rememberLoginChk;
14
15   @FindBy(id = "wt11_wtMainContent_wtLoginButton")
16   private final WebElement loginButton_elem = null;
17   private final ButtonWidget loginButton;
18
19   public LoginPage(IApplication app) { this(app, true); }
20   public LoginPage(IApplication app, boolean doFetch) {
21     super(app);
22     if (doFetch)
23       _app.getDriver().get("http://localhost/" + app.getName() + "/" +
             DirectoryConstants.LOGIN_PAGE);
24
25     PageFactory.initElements(_app.getDriver(), this);
26
27     userNameInput = new InputWidget<String>(String.class, this,
           userNameInput_elem);
28     passwordInput = new InputWidget<String>(String.class, this,
           passwordInput_elem);
29     rememberLoginChk = new CheckBoxWidget(this, rememberLoginChk_elem);
30     loginButton = new ButtonWidget(this, loginButton_elem, METHOD_TYPE.
           Submit, VALIDATION_TYPE.None, true);
31   }
32
33   public InputWidget<String> userNameInput() { return userNameInput; }
34   public InputWidget<String> passwordInput() { return passwordInput; }
35   public CheckBoxWidget rememberLoginChk() { return rememberLoginChk; }
36   public ButtonWidget loginButton() { return loginButton; }
```

```
37 }
```

Listagem C.4: directory.page.MainPage

```
1  public class MainPage extends AbstractPageObject {
2
3    private final int _empTable_numCols = 1;
4    private final TableLineFactory_EmployeeTable _empTablelineFactory;
5    private final TableRecordWidget<TableLine_EmployeeTable>
         _employeeTable;
6
7    private final int _orgUnitsTable_numCols = 1;
8    private final TableLineFactory_OrgUnitsTable _orgUnitsTablelineFactory
         ;
9    private final TableRecordWidget<TableLine_OrgUnitsTable>
         _orgUnitsTable;
10
11   @FindBy(id = "wt104_wt27_wt9_wtEveryone")
12   private final WebElement everyone_elem = null;
13   private final LinkWidget everyone;
14
15   @FindBy(id = "wt104_wt27_wt9_wtEveryoneSelected")
16   private final WebElement everyoneSelected_elem = null;
17   private final LinkWidget everyoneSelected;
18
19
20   public MainPage(IApplication app) { this(app, true); }
21   public MainPage(IApplication app, boolean doFetch) {
22     super(app);
23     if (doFetch)
24       _app.getDriver().get("http://localhost/" + app.getName() + "/" +
             DirectoryConstants.MAIN_PAGE);
25
26     everyone = new LinkWidget(this, everyone_elem, METHOD_TYPE.Navigate,
           VALIDATION_TYPE.None, true);
27     everyoneSelected = new LinkWidget(this, everyoneSelected_elem,
         METHOD_TYPE.Navigate, VALIDATION_TYPE.None, true);
28
29     _empTablelineFactory = new TableLineFactory_EmployeeTable();
30     _employeeTable = new TableRecordWidget<TableLine_EmployeeTable>(this
           ,
31       _app.getDriver().findElement(By.id("
             wt104_wtMainContent_wtEmployeeTable")),
32       _empTablelineFactory, _empTable_numCols, false /*hasHeader*/
33     );
34
35     _orgUnitsTablelineFactory = new TableLineFactory_OrgUnitsTable();
36     _orgUnitsTable = new TableRecordWidget<TableLine_OrgUnitsTable>(this
           ,
37       _app.getDriver().findElement(By.id("wt104_wt27_wt9_wtOrgUnitsTable
             ")),
38       _orgUnitsTablelineFactory, _orgUnitsTable_numCols, false /*
             hasHeader*/
39     );
40   }
41
42   public LinkWidget everyone() { return everyone; }
```

```
43    public LinkWidget everyoneSelected() { return everyoneSelected; }
44    public TableRecordWidget<TableLine_EmployeeTable> employeeTable() {
          return _employeeTable; }
45    public TableRecordWidget<TableLine_OrgUnitsTable> orgUnitsTable() {
          return _orgUnitsTable; }
46
47  }
```

C.4 Package demoApp.widget

Listagem C.5: directory.widget.mainPage.TableLine_EmployeeTable

```
1  public class TableLine_EmployeeTable extends AbstractTableLine {
2
3    private final TableRecordWidget<TableLine_EmployeeTable> _owner;
4
5    private final String name_elem_sufix = "wtEmployeeName";
6    private final WebElement name_elem;
7    private final LinkWidget name;
8
9    private final String position_elem_sufix = "wtEmployeePosition";
10   private final WebElement position_elem;
11   private final ExpressionWidget<String> position;
12
13   private final String mobilePhone_elem_sufix = "wtMobilePhone";
14   private final WebElement mobilePhone_elem;
15   private final ExpressionWidget<String> mobilePhone;
16
17   private final String eMail_elem_sufix = "wtEMail";
18   private final WebElement eMail_elem;
19   private final LinkWidget eMail;
20
21   public TableLine_EmployeeTable(TableRecordWidget<
         TableLine_EmployeeTable> owner, WebElement elem, int index) {
22     super(/*linePrefix*/owner.getWebElement().getAttribute("id") + "_ctl
           ", index);
23     _owner = owner;
24
25     name_elem = elem.findElement(By.id(buildInternalElemId(
           name_elem_sufix)));
26     position_elem = elem.findElement(By.id(buildInternalElemId(
           position_elem_sufix)));
27     mobilePhone_elem = elem.findElement(By.id(buildInternalElemId(
           mobilePhone_elem_sufix)));
28     eMail_elem = elem.findElement(By.id(buildInternalElemId(
           eMail_elem_sufix)));
29
30     name = new LinkWidget(_owner.getPage(), name_elem, METHOD_TYPE.Ajax,
            VALIDATION_TYPE.None, false);
31     position = new ExpressionWidget<String>(String.class, _owner.getPage
           (), position_elem);
32     mobilePhone = new ExpressionWidget<String>(String.class, _owner.
           getPage(), mobilePhone_elem);
33     eMail = new LinkWidget(_owner.getPage(), eMail_elem, METHOD_TYPE.
           Navigate, VALIDATION_TYPE.None, false);
34   }
```

```
35
36    public LinkWidget name() { return name; }
37    public ExpressionWidget<String> position() { return position; }
38    public ExpressionWidget<String> mobilePhone() { return mobilePhone; }
39    public LinkWidget eMail() { return eMail; }
40  }
```

Listagem C.6: directory.widget.mainPage.TableLine_OrgUnitsTable

```
1  public class TableLine_OrgUnitsTable extends AbstractTableLine {
2
3    private final TableRecordWidget<TableLine_OrgUnitsTable> _owner;
4
5    private final int _subOrgUnitsTable_numCols = 1;
6    private final TableLineFactory_SubOrgUnitsTable _lineFactory;
7    private final TableRecordWidget<TableLine_SubOrgUnitsTable>
         _subOrgUnitsTable;
8
9    private final String orgUnitSelected_elem_sufix = "wtOrgUnitSelected";
10   private final WebElement orgUnitSelected_elem;
11   private final ExpressionWidget<String> orgUnitSelected;
12
13   private final String orgUnit_elem_sufix = "wtOrgUnit";
14   private final WebElement orgUnit_elem;
15   private final LinkWidget orgUnit;
16
17   public TableLine_OrgUnitsTable(TableRecordWidget<
         TableLine_OrgUnitsTable> owner, WebElement elem, int index) {
18     super(/*linePrefix*/owner.getWebElement().getAttribute("id") + "_ctl
           ", index);
19     _owner = owner;
20
21     List<WebElement> elems;
22
23     elems = _owner.getPage().getApp().getDriver().findElements(By.id(
           buildInternalElemId(orgUnitSelected_elem_sufix)));
24     orgUnitSelected_elem = elems.size() != 0 ? elems.get(0) : null;
25
26     elems = _owner.getPage().getApp().getDriver().findElements(By.id(
           buildInternalElemId(orgUnit_elem_sufix)));
27     orgUnit_elem = elems.size() != 0 ? elems.get(0) : null;
28
29     orgUnitSelected = orgUnitSelected_elem != null ? new
           ExpressionWidget<String>(String.class, _owner.getPage(),
           orgUnitSelected_elem) : null;
30     orgUnit = orgUnit_elem != null ? new LinkWidget(_owner.getPage(),
           orgUnit_elem, METHOD_TYPE.Ajax, VALIDATION_TYPE.None, false) :
           null;
31
32     _lineFactory = new TableLineFactory_SubOrgUnitsTable();
33     _subOrgUnitsTable = new TableRecordWidget<TableLine_SubOrgUnitsTable
           >(_owner.getPage(),
34       _owner.getPage().getApp().getDriver().findElement(
35         By.id(buildInternalElemId("wt12_wtSubOrgUnitsTable"))
36       ), _lineFactory, _subOrgUnitsTable_numCols, false /*hasHeaders*/
37     );
38
```

```
39    }
40
41    public ExpressionWidget<String> orgUnitSelected() { return
          orgUnitSelected; }
42    public LinkWidget orgUnit() { return orgUnit; }
43
44    public TableRecordWidget<TableLine_SubOrgUnitsTable> subOrgUnitsTable
          () { return _subOrgUnitsTable; }
45
46  }
```

Listagem C.7: directory.widget.mainPage.TableLine_SubOrgUnitsTable

```
1   public class TableLine_SubOrgUnitsTable extends AbstractTableLine {
2
3     private final TableRecordWidget<TableLine_SubOrgUnitsTable> _owner;
4
5     private final String subOrgUnit_elem_sufix = "wtSubOrgUnit";
6     private final WebElement subOrgUnit_elem;
7     private final LinkWidget subOrgUnit;
8
9     public TableLine_SubOrgUnitsTable(TableRecordWidget<
          TableLine_SubOrgUnitsTable> owner, WebElement elem, int index) {
10      super(/*linePrefix*/owner.getWebElement().getAttribute("id") + "_ctl
            ", index);
11      _owner = owner;
12
13      subOrgUnit_elem = _owner.getPage().getApp().getDriver().findElement(
14        By.id(buildInternalElemId(subOrgUnit_elem_sufix))
15      );
16
17      subOrgUnit = new LinkWidget(_owner.getPage(), subOrgUnit_elem,
            METHOD_TYPE.Ajax, VALIDATION_TYPE.None, false);
18    }
19
20    public LinkWidget subOrgUnit() { return subOrgUnit; }
21
22  }
```

Listagem C.8: directory.widget.mainPage.TableLineFactory_EmployeeTable

```
1   public class TableLineFactory_EmployeeTable implements ITableLineFactory
      <TableLine_EmployeeTable> {
2
3     @Override
4     public TableLine_EmployeeTable create(TableRecordWidget<
          TableLine_EmployeeTable> owner,
5       WebElement elem, int index) {
6       return new TableLine_EmployeeTable(owner, elem, index);
7     }
8
9   }
```

Listagem C.9: directory.widget.mainPage.TableLineFactory_OrgUnitsTable

```
1   public class TableLineFactory_OrgUnitsTable implements ITableLineFactory
      <TableLine_OrgUnitsTable> {
```

```
2
3     @Override
4     public TableLine_OrgUnitsTable create(TableRecordWidget<
          TableLine_OrgUnitsTable> owner,
5         WebElement elem, int index) {
6       return new TableLine_OrgUnitsTable(owner, elem, index);
7     }
8
9   }
```

Listagem C.10: directory.widget.mainPage.TableLineFactory_SubOrgUnitsTable

```
1   public class TableLineFactory_SubOrgUnitsTable implements
        ITableLineFactory<TableLine_SubOrgUnitsTable> {
2
3     @Override
4     public TableLine_SubOrgUnitsTable create(TableRecordWidget<
          TableLine_SubOrgUnitsTable> owner,
5         WebElement elem, int index) {
6       return new TableLine_SubOrgUnitsTable(owner, elem, index);
7     }
8
9   }
```

Referências

[1] Alister Scott, *WatirMelon Testing Blog*, http://watirmelon.com/.

[2] Prashant Anaskure, *Functional Test Automation White Paper*, Geometric Global, 2009.

[3] Anneliese A. Andrews, Jeff Offutt, and Roger T. Alexander, *Testing web applications by modeling with FSMs*, Software and Systems Modeling (2005).

[4] Borland, *How to Successfully Automate The Functional Testing Process*, (2013).

[5] Jean-Francois Collard and Ilene Burnstein, *Practical Software Testing*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[6] Giuseppe A. Di Lucca and Anna Rita Fasolino, *Testing Web-based applications: The state of the art and future trends*, Inf. Softw. Technol. (2006).

[7] Blaine Donley and Jeff Offutt, *Web Applications Testing Challenges*, (2009).

[8] Kagan Erdil, Emily Finn, Kevin Keating, Jay Meattle, Sunyoung Park, and Deborah Yoon, *Software Maintenance As Part Of The Software Life Cycle*, December 2003.

[9] Russell Gold, *HttpUnit*, http://www.httpunit.org/.

[10] Aslak Hellesoy, *Cucumber*, http://cukes.info/.

[11] Hellesoy, A. and Wynne, M., *The Cucumber Book: Behaviour-Driven Development for Testers and Developers*, Pragmatic Programmers, Pragmatic Bookshelf, 2012.

[12] JUnit Testing Framework, *junit*, http://junit.org.

[13] John Kent, *Test Automation: From Record/Playback to Frameworks*, (2007).

[14] Ruth Malan and Dana Bredemeyer, *Defining Non-functional Requirements*, Bredemeyer Consulting, White Paper. http://www.bredemeyer.com/papers.htm, 2001.

[15] Glenford J. Myers and Corey Sandler, *The Art of Software Testing*, John Wiley & Sons, 2004.

[16] OutSystems, *Web Application Development Tool - OutSystems OutSystems Platform*, http://www.outsystems.com/agile-platform/.

[17] Hossein Saiedian Mary Shaw Robert Dupuis J. Barrie Thompson Peter Freeman, Donald J. Bagert, *Software Engineering Body of Knowledge (SWEBOK)*, IEEE Computer Society, 2001.

[18] Mauro Pezze and Michal Young, *Software Testing and Analysis: Process, Principles and Techniques*, Wiley, 2007.

[19] Roger Pressman, *Software Engineering: A Practitioner's Approach*, McGraw-Hill Series in Computer Science.

[20] Ewald Roodenrijs, *Test Design Technique - Use Case Test*.

[21] Selenium, *Selenium WebDriver*, http://docs.seleniumhq.org/projects/webdriver/.

[22] Selenium Browser Automation Framework, *Selenium browser automation framework*, https://code.google.com/p/selenium/.

[23] SmartBear Software, *TestComplete*, http://smartbear.com/products.

[24] Gregory Tassey, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, Diane Pub Co, 2002.

[25] Watir Community, *Watir - Web Application Testing in Ruby*, http://watir.com/.

[26] Wikipedia, *User Stories*, http://en.wikipedia.org/wiki/User_stories.

[27] World Wide Web Consortium (W3C), *WebDriver API Specification*, http://www.w3.org/TR/2013/WD-webdriver-20130117/.

[28] Li Ye, *Model-Based Testing Approach for Web Applications*, June 2007.