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

Ricardo Neto[1], Fernando M. Carvalho[2] and Tiago L. Alves[3]

[1] Instituto Superior de Engenharia de Lisboa, 1959-007 Lisboa, PT,
neto.lx@gmail.com, http://www.isel.pt
[2] Instituto Superior de Engenharia de Lisboa, 1959-007 Lisboa, PT,
mcarvalho@cc.isel.ipl.pt, http://www.isel.pt
[3] OutSystems S.A., 2795-242 Linda-a-Velha, PT,
tiago.alves@outsystems.com, http://www.outsystems.com

Resumo A necessidade de adaptação rápida das aplicações web aos requisitos dos seus consumidores exige um acompanhamento imediato na validação das alterações introduzidas. As tecnologias actuais para teste de aplicações web oferecem apenas APIs de baixo nível que se baseiam na interacção directa com os elementos HTML dificultando a rápida criação e manutenção dos testes necessários.
Este artigo propõe uma solução para a geração automática de uma *framework* de teste tirando partido de um modelo de uma aplicação web definida com a OutSystems Platform.

Keywords: teste de software, aplicações web, qualidade de software, geração de código

1 Introdução

A necessidade de alterações frequentes às aplicações aumenta o risco de introdução de falhas [1], realçando a importância de testes capazes de detectá-las em fases iniciais do ciclo de desenvolvimento. Assim, evita-se a sua propagação, diminuindo-se consideravelmente o custo associado à sua correcção [3] e, por consequência, reduzindo o custo de desenvolvimento das aplicações.

A execução de testes funcionais em aplicações web passa pela interacção com as suas páginas de forma a verificar que uma determinada funcionalidade está de acordo com a sua especificação. Tradicionalmente, o uso de *frameworks* de teste web (e.g. Selenium *WebDriver*) implica a criação de dependências com a estrutura das páginas da aplicação. A forma como pretendemos resolver este problema 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 sejam robustos à estrutura HTML usada. A abordagem a este problema inclui tirar partido da OutSystems Platform.

Figura 1. Páginas envolvidas na realização da história de utilização e respectivos *inputs*

A OutSystems Platform é uma plataforma de desenvolvimento rápido que viabiliza o desenvolvimento de aplicações web no ambiente de desenvolvimento ServiceStudio, disponibilizando uma linguagem de modelação visual para definição do modelo da aplicação. Este modelo é utilizado no processo de compilação e geração automática de código, dando origem a diferentes artefactos como código Java ou .NET e páginas HTML. Do modelo da aplicação fazem parte *WebScreens* e *WebBlocks*, que representam a interface de utilização, constituídos por *widgets* como *inputs*, tabelas ou *combo boxes*, dando origem, no processo de geração automática de código, às páginas da aplicação e aos elementos HTML que as constituem.

Tirando partido do modelo da aplicação definido no ServiceStudio introduzimos os princípios para a geração automática de uma *framework* que mapeie os conceitos alto-nível de uma aplicação web definida na OutSystems Platform nos elementos HTML necessários e assim facilitar a criação de testes. A utilização desta *framework* permitirá abstracção sobre a estrutura HTML das páginas e seus elementos, facilitando a criação e manutenção de testes funcionais.

2 Motivação

Com o objectivo de demonstrar os problemas na criação de testes com recurso a ferramentas utilizadas na indústria, considere-se a seguinte história de utilização obtida da aplicação Directory[4] que permite a gestão centralizada de colaboradores: *Como colaboradora, a Alice precisa de saber quem é o colega responsável pelo website, para que consiga dar a conhecer uma anomalia.*

A Figura 1 apresenta as páginas da aplicação, e respectivos *inputs*, necessárias à verificação da história de utilização proposta, entre as quais: (1) Login - Autenticação do utilizador tendo como entradas o nome de utilizador 'alice.andrews' e a password 'aa123'; (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'.

[4] `http://www.outsystems.com/apps/Directory`

Na listagem 1.1 é mostrada a implementação da história de utilização apresentada utilizando a *framework* Selenium *WebDriver*.

Listing 1.1. Implementação de teste funcional para consulta de dados relativos a gestor de área organizacional com recurso a Selenium WebDriver

```
1  @Test
2  public void searchForOnlineOperationsManager() {
3    WebDriver driver = new FirefoxDriver();
4    driver.get("http://localhost/Directory");
5    doLogin(driver, "alice.andrews", "aa123");
6    assertEmployeeAreaAndPosition(driver, "Larry R. Logan", "
         Online Operations", "Online Operations Manager");
7  }
8
9  private void doLogin(WebDriver driver, String username, String
       password) {
10   WebElement inputUsername = driver.findElement(By.id("
         wtUserNameInput"));
11   inputUsername.sendKeys(username);
12   WebElement inputPassword = driver.findElement(By.id("
         wtPasswordInput"));
13   inputPassword.sendKeys(password);
14   WebElement buttonLogin = driver.findElement(By.cssSelector("
         input[value='Login']"));
15   buttonLogin.click();
16 }
17
18 private void assertEmployeeAreaAndPosition(WebDriver driver,
     String name, String area, String position) {
19   // Selecção de área da organização
20   driver.findElement(By.linkText(area)).click();
21   // Apresentação de dados do colaborador
22   assertEquals(area, driver.findElement(By.cssSelector("div.
         BreadcrumbText a span")).getText());
23   assertEquals(name, driver.findElement(By.id("
         wtEmpTbl_ctl04_wtEmpName")).getText());
24   assertEquals(position, driver.findElement(By.id("
         wtEmpTbl_ctl04_wtEmpPos")).getText());
25 }
```

O teste que valida a história de utilização definida anteriormente segue três etapas. Na primeira etapa, linha 3, inicializa-se o browser *Firefox* e, na linha 4, faz-se o pedido da página inicial a testar. Na segunda etapa faz-se a autenticação, invocando na linha 5 o método `doLogin` com os argumentos adequados. A linha 10 faz a pesquisa do elemento HTML de input do nome de utilizador e a linha 11 introduz o valor utilizado no teste. As linhas 12 e 13 repetem estes passos para a password. Na linha 14 é feita a pesquisa do elemento que representa o botão de

Login e na linha 15 é invocado o evento de *click* sobre este elemento. Na terceira etapa do teste é seleccionada a área organizacional e verificam-se os dados apresentados, invocando na linha 6 o método `assertEmployeeAreaAndPosition`. Na linha 20 selecciona-se a área da organização a que corresponde o teste verificando-se por fim, nas linhas 22 a 24, os dados do colaborador apresentado na página com os valores utilizados no teste.

O problema desta aborgadem, também identificado em outras *frameworks* de teste (e.g. HttpUnit[5] e Cucumber[6]), é a dependência explícita a elementos específicos de HTML. Esta dependência não só dificulta a criação dos testes (utilizando identificadores ou navegando directamente na estrutura HTML) como também dificulta a manutenção decorrente de mudanças ao nível da estrutura na página. Considere-se, no exemplo apresentado, o valor dos identificadores dos elementos que recebem o nome de utilizador e password na página de *Login*. Modificando na aplicação o seu valor, de `wtUserNameInput` e `wtPasswordInput` para `txtUsername` e `txtPassword` respectivamente, provocaria falhas na execução dos testes com dependências para os mesmos, exigindo esforço na sua adaptação e reduzindo a visibilidade sobre o impacto das alterações na funcionalidade.

3 Abordagem Proposta

Com este trabalho pretende-se 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. A Figura 2 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* 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. O seu contributo é fundamental na geração da *framework* de testes por nele se encontrarem descritos, com todo o detalhe, elementos como as páginas na aplicação, os comportamentos que expõem e as

[5] `http://httpunit.sourceforge.net`
[6] `http://cukes.info`

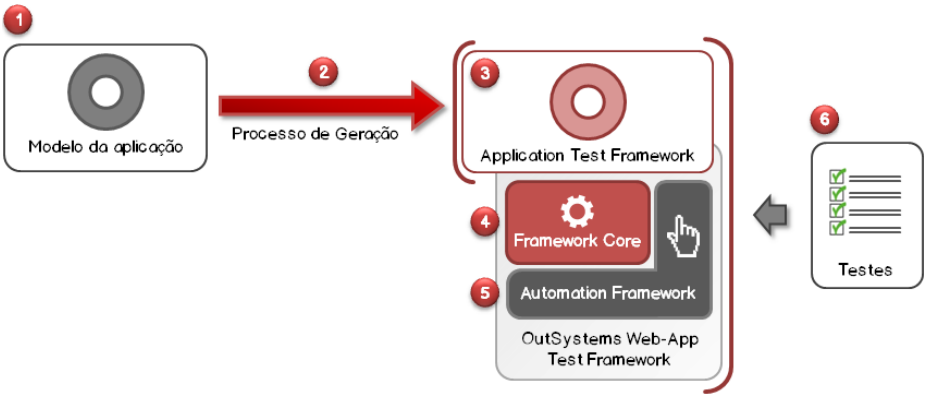

Figura 2. Arquitectura da solução onde se identificam: (1) o modelo da aplicação em ServiceStudio; (2) o processo de geração da *framework*; (3) a *framework* de testes obtida do modelo da aplicação; (4) o *core* da *framework* de testes; (5) a *framework* de automação e; (6) os testes

wigets que as constituem, que dão origem ao HTML com o qual o utilizador 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. 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 obtida do processo de geração, representando em API própria o modelo da aplicação definido no ServiceStudio. A sua função é a de facilitar a criação de testes, disponibilizando uma API composta por objectos que representam a aplicação, as suas páginas e os elementos com os quais os utilizadores 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. *widgets*, localizadores), 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 deixa de existir, uma vez que referências ao mesmo serão aqui centralizadas. A API

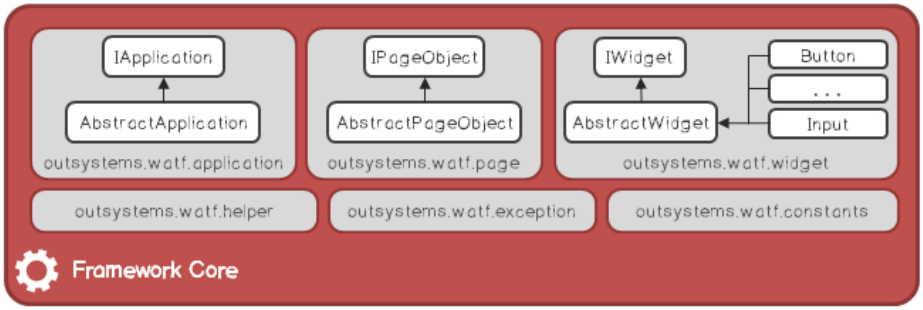

Figura 3. Estrutura interna de *Framework Core* onde são mostrados os *packages* que a compõem

disponibilizada facilita não só a criação de testes, mas também a sua legibilidade, disponibilizando estruturas que representam entidades do domínio da aplicação.

4 - Framework Core

A *Framework Core* é a camada que dá suporte à *Application Test Framework*. A sua função é disponibilizar definições para os objectos que representam a aplicação, as páginas e as *widgets*, independentemente da aplicação. A Figura 3 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 garantir a interacção dos testes com a aplicação através da manipulação de uma instância de um browser, tendo-se optado pela utilização da *framework* Selenium *WebDriver*. Esta camada dá suporte à *Framework Core* e à *Application Test Framework*, permitindo a manipulação de diferentes browsers e localização e manipulação de elementos expostos pelas páginas da aplicação através de uma API rica. É nesta camada que existem mecanismos, específicos da *framework* Selenium *WebDriver*, capazes de lidar com aspectos relativos à introdução de tempos de espera no carregamento de páginas ou de conteúdo dinâmico que as compõem (no caso de pedidos AJAX) e execução de JavaScript.

6 - Testes

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

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

A listagem 1.2 mostra uma proposta de implementação de um teste que faz referência a um protótipo da *framework* de testes criado manualmente.

Listing 1.2. Teste de consulta de dados com recurso a protótipo da *framework* de testes

```
1  @Test
2  public void searchForOnlineOperationsManager() {
3    Directory app = new Directory();
4    doLogin(app, "alice.andrews", "aa123");
5    assertEmployeeAreaAndPosition(app, "Larry␣R.␣Logan", "Online
         ␣Operations", "Online␣Operations␣Manager");
6  }
7
8  private void doLogin(Directory app, String username, String
       password) {
9    LoginPage loginPage = app.getLoginPage();
10   loginPage.setUsername(username);
11   loginPage.setPassword(password);
12   loginPage.clickLogin();
13 }
14
15 private void assertEmployeeAreaAndPosition(Directory app,
       String name, String area, String position) {
16   MainPage main = app.getMainPage();
17   main.clickDepartment("Online␣Operations");
18   assertEquals(main.getEmployeeByName("Larry␣R.␣Logan").length
         (), 1);
19   assertEquals(main.getEmployeeByPosition("Online␣Operations␣
         Manager").length(), 1);
20 }
```

Na linha 3 é instanciado o objecto que representa a aplicação que implementa *IApplication*. A autenticação é feita pela invocação do método `doLogin` na linha 4. Na linha 9 é invocado um método do objecto *Application* que devolve uma instância de um objecto do tipo *LoginPage*, que representa a página de Login. Nas linhas 10 e 11 é feito o acesso aos elementos de input do nome de utilizador e password, expostos aqui sobre a forma de métodos. Na linha 12 é chamado o método que representa a operação de login, exposta na página da aplicação através de um botão. A selecção da área organizacional e verificação dos dados apresentados é feita no método `assertEmployeeAreaAndPosition`, invocado na linha 5. Nas linhas 16 e 17 é obtido o objecto que representa a página principal da aplicação e invocado o método `clickDepartment`, que invocará o evento de click sobre um dos objectos de domínio expostos na página. Por fim verificam-se, linhas 18 e 19, os dados do colaborador apresentado na página com os valores

utilizados no teste.

O objecto do tipo *IApplication*, instanciado na linha 3, é responsável pela instanciação do browser a ser utilizado no teste, bem como do *driver* que o irá manipular, retirando do código de teste referências à *framework* de automação. Na geração de objectos deste tipo pretende-se tirar partido da informação presente no modelo de navegação da aplicação com o objectivo de retornar instâncias de objectos que representam as páginas da aplicação, objectos estes do tipo *IPage-Object*, centralizando o acesso às páginas no objecto *Application*, retirando do código de teste referências a caminhos da aplicação.

Na solução proposta tira-se partido do padrão *PageObject*, identificado em [2]. A sua utilização tem como objectivo minimizar o esforço de manutenção envolvido na adaptação dos testes a alterações à aplicação, reduzindo a duplicação de código, centralizando num objecto os comportamentos expostos e elementos presentes na interface de utilização. Os testes passam a invocar métodos expostos pelo objecto *PageObject* como forma de interagir com a interface de utilização ou executar acções disponiveis na pagina. Nas linhas 10 e 11 são invocados métodos que representam os elementos de input da página que guardam o nome de utilizador e password. Estes métodos, além de viabilizarem a alteração e obtenção de estado dos elementos na página, permitem a validação dos inputs na *framework* de testes, uma vez que o tipo de dados recebido, ou devolvido, corresponde ao tipo de dados definido no modelo da aplicação no ServiceStudio (e.g. inteiro, decimal).

A utilização de conceitos relacionados com o domínio da aplicação, como é o caso de 'Departamento' e 'Colaborador', facilita a legibilidade do teste. Desta forma, pretende-se incluir na geração da *framework* métodos capazes de tratar entidades, individualmente ou em grupo, como é o caso de `getEmployeeByName` na linha 17, facilitando a criação de testes por facilmente permitir a identificação dessas entidades nas páginas da aplicação. Nas linhas 18 e 19 exemplifica-se o uso de métodos de entidades de domínio capazes de aceder a uma lista de entidades com mapeamento directo no HTML da página da aplicação.

4 Conclusões e trabalho futuro

A solução proposta permite remover dos testes dependências com o HTML das páginas da aplicação, facilitando de uma forma geral a manutenção da *test suite*. A extensão do mecanismo de geração automática de código, baseado no modelo da aplicação definido no ServiceStudio, irá permitir que a API da *framework* de testes, também gerada automaticamente, se mantenha consistente com o código da aplicação, facilitando a criação de novos testes ou a alteração de testes já existentes. Do ponto de vista da visibilidade sobre alterações à aplicação, a quebra de compatibilidade entre testes e aplicação é possível de ser verificada em tempo de compilação, evitando tempo gasto num ciclo de verificação.

Os próximos passos envolvidos no desenvolvimento deste trabalho consistem essencialmente em explorar a linguagem da OutSystems Platform, identificando a forma como os elementos que compõem a aplicação se mapeiam na *framework* de testes. Após a definição da API com que cada elemento irá mapear, será possível codificar o gerador automático da *framework*, que permitirá aplicar a solução proposta a aplicações empresariais reais. Pretende-se também investigar formas de tirar partido da linguagem visual oferecida pela OutSystems Platform, permitindo a criação de testes no ServiceStudio.

Referências

1. Glenford J. Myers and Corey Sandler, *The Art of Software Testing*, John Wiley & Sons, 2004.
2. Selenium Browser Automation Framework, *Selenium browser automation framework*, https://code.google.com/p/selenium/.
3. Gregory Tassey, *The Economic Impacts of Inadequate Infrastructure for Software Testing*, Diane Pub Co, 2002.