Universidade do Minho

HASLab — INESC TEC

Testes de interfaces gráficas com o recurso a mutações

Documentação de Bolsa de Investigação

Luís Miguel Pinto
pg27756@alunos.uminho.pt

Resumo

Versão 0.4

Braga, 30 de Outubro de 2015

Conteúdo

1 Introdução

Este trabalho insere-se no contexto do Projeto de investigação "*PBGT - Testes de interfaces gráficas com o utilizador baseado em padrões*" com a referência *BI1-2015_PTDC/EIA-EIA/119479/2010_UMINHO*, na Universidade do Minho, no ano civil de 2015.

Atualmente, existe um forte crescimento das aplicações web para as mais variadas utilizações. Estas estão maiores, mais complexas, sofisticadas, interativas e integradas em outros serviços, o que o torna a fase de execução de testes mais morosa, cara e pouco ágil. Por este motivo, torna-se fundamental existir uma ferramenta que permita não só validar as interfaces gráficas existentes como também encontrar possíveis erros no funcionamento.

A ferramenta que existe atualmente consegue automatizar a geração e execução de casos de teste às interface de uma aplicação web através da utilização de ficheiros (modelo da máquina de estados, mapeamento, valores e mutações) que são criados pelo utilizador e definem a plataforma que vai ser testada.

Assim, o presente relatório pretende documentar a validação da tecnologia utilizada, as alterações efetuadas e finalmente o resultado da aplicação da ferramenta a uma interface gráfica.

2 Análise da ferramenta

O processo de testar uma camada da interface é essencial para garantir a qualidade do software. Para assegurar que este processamento é menos demorado, mais económico e com uma fiabilidade superior foi construída esta ferramenta que pretende automatizar o processo de geração e execução de testes.

A ferramenta recorre a máquina de estados que representam a navegação da interface gráfica. Como a interface a testar é uma aplicação web, cada página pode ser representada como um estado que pode ter vários sub-estados. Cada botão ou *link* numa página representam ações de *input* ou comandos que são transições para novos estados. Em paralelo, é necessário criar um ficheiro de mapeamento entre este modelo e a interface gráfica para garantir a ligação aos elementos da aplicação web. Existe ainda um ficheiro para os valores de entrada na geração dos testes.

A partir do modelo da máquina de estados, que é carregado para a ferramenta, é gerado um grafo onde são aplicados algoritmos de travessia para gerar vários testes abstratos. Estes testes são gerados em *Java* e são executados com recurso à *framework Selenium*. Depois da aplicação dos testes à aplicação web é gerado um relatório com os resultados detalhados da execução.

Por vezes, os utilizadores tem comportamentos inesperados na utilização de uma plataforma web, por isso são introduzidos na geração de testes pequenos erros para simular tais situações. É também possível criar um ficheiro de mutações que para um dado elemento do modelo é introduzido um erro de utilização.

Para descrever a forma do processo de geração automático de testes foi criada a Figura1, onde é apresentada a arquitetura deste processo.

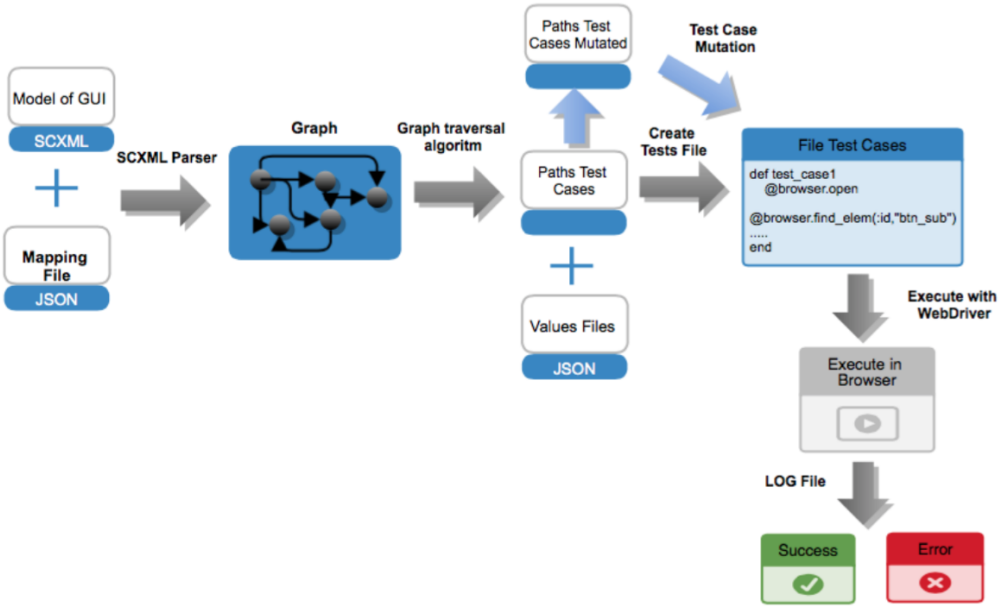

Figura 1: Arquitetura do processo de geração de testes

2.1 Máquina de estados

Máquina de estados são um padrão recorrente na engenharia de software. São vistas como uma maneira útil de pensar sobre o comportamento de sistemas reactivos, desde as fases iniciais de desenho, até ao testes de software [1].

Uma máquina de estado é um modelo que é constituído por um conjunto de estados e transições para alteração de estados. Estas transições podem acontecer derivado a eventos do utilizador ou do próprio sistema. Assim na construção de um modelo para uma interface gráfica é necessário perceber as transições que ocorrem entre estados e os motivos/ações de isso acontecer.

2.1.1 State Chart XML

O *State Chart XML* (SCXML) é uma linguagem para descrever máquinas de estados baseadas em eventos que combinam os conceitos de CCXML e tabelas de estado de Harel [2]. Permite descrever um modelo da máquina de estado na linguagem XML de forma limpa e organizada.

A semântica principal SCXML válida para se utilizar na definição do modelo de interfaces gráficas para esta ferramenta é a seguinte:

<state> Representa um estado, o elemento principal. Ainda dentro deste podem existir mais subestados e os restantes elementos semânticos desta lista. Cada estado é constituído por alguns atributos como o *id* e o *type*.

- ***id*** É utilizado para identificar estado.
- ***type*** Atributo opcional, utilizado quando se está na presença de algum formulário na página web. Ficando assim com o valor "*form*".

<transition> Representa uma possível transição entre estados. Normalmente acontece quando existe um clique num link ou botão que origina uma transição. Este elemento pode ter alguns dos próximos atributos, o *id*, o *target*, o *type* e o *label*. Fazem parte em alguns momentos os próximos elementos: <*step*>, <*submit*> e o <*error*>.

- ***id*** A váriavel que vai identifcar a ação do utilizador no ficheiro de mapeamento. Não é utilizado quando se está a representar um formulário da interface gráfica.
- ***target*** Onde será colocado o *id* do estado para o qual vai transitar.
- ***type*** Este atributo é utilizado na presença de um formulário, pode ter valores como *form* ou *ajax* (se o pedido for assíncrono). Outro valor que pode ser usado para definir este atributo é o *menu*.
- ***label*** Variável usada nos formulário para representar a ação do utilizador quando está a submeter este.

<step> É utilizado quando é necessário mais do que uma ação para passar a um novo estado. Por exemplo, quando existe um menu e é necessário clicar num link dentro deste.

<submit> Utilizado para transições em formulários. Possuí um atributo, *target* com o significado explicado anteriormente.

<error> Utilizado quando um formulário sofre mutações, transitando assim para um novo estado através do *target*.

<send> Corresponde a *inputs* de utilizadores no preenchimento de formulários. Tag constituída por dois atributos obrigatórios e um opcional.

label Mapeia a variável com os ficheiros de configuração.

type Serve para informar se um input é obrigatório ou opcional.

element Atributo opcional. Utilizado para referir se elemento é *select-box* ou *checkbox*.

<onentry> Validações que são sempre executadas quando se entra em um certo estado. Tem os seguintes atributos (válida também na próxima tag).

id Mapeia a variável com os ficheiros de configuração.

type O tipo de validação que pode ser efetuada. Mais informações sobre as validações a seguir.

<onexit> Validações que ocorrem sempre quando se está a sair de um determinado estado.

```
<!-- BEGIN REGISTER STATE -->
<state id="register">
  <state id="register_form" type="form">
    <send label="name" type="required"/>
    <send label="email" type="required"/>
    <send label="password" type="required"/>
    <send label="conf_password" type="required"/>

    <transition type="form" label="submit_register">
      <submit target="root" />
      <error target="error_register" />
    </transition>

    <onexit id="message_authentication" type="displayed?"/>
  </state>

  <onentry id="section_title_signup" type="default" />
  <onentry id="attr_username" type="attribute"/>
</state>
<!-- END REGISTER STATE -->
```

Figura 2: Exemplo de um estado em SCXML

2.1.2 Validações

Na geração de casos de teste é importante que sejam adicionadas as validações necessárias para perceber se existe alguma situação anômala na interface gráfica.

Na geração de casos de teste, é necessário verificar se as propriedades de saída de um caso respeitam um conjunto predefinido de dados. As validações são introduzidas dentro de um estado, no ficheiro do modelo da máquina de estados, e são representadas pelos elementos <*onentry*> e <*onexit*>. Posteriormente, quando se estiver a gerar os casos de teste estas serão convertidas em código, para que assim o *Selenium* consiga executar as validações.

Assim, são válidas as seguintes validações a uma aplicação web:

displayed?/not_displayed?	Verifica se um dado elemento está visível ou não.
is_selected/is_not_selected	Verifica se um elemento (selectbox/checkbox) está seleccionado.
enabled?/disabled?	Verifica se um elemento está ativo ou desativo. Por exemplo, os botões de submissão.
attribute	Verifica se um dado atributo de um elementopor exemplo. Por exemplo, o atributo *value* de um *input*.
css	Verifica uma determinada propriedade CSS de um elemento. Por exemplo, *background-color, position*.
contains	Verifica se um elemento contém uma *string*.
regex	Verifica se um elemento contém uma *string* que corresponda a uma expressão regular.
url	Verifica o url de uma página.
default	Verifica se um elemento tem uma determindade *string*.

2.2 Ficheiros de configuração

Terminado o modelo da máquina de estados, é importante definir que elementos da página web, que vai ser testada, correspondem aos elementos do modelo. É necessário criar dois ficheiros, em formato *JSON*, que serão posteriormente carregados na ferramenta. O ficheiro de mapeamento contém para cada variável da máquina de estados uma configuração para determinar qual o tipo de elemento HTML a encontrar, como encontrá-lo, a ação e o tipo de execução. O ficheiro de valores contém os dados que vão ser utilizados para *input* na geração dos casos de testes.

```
"name" : {
  "how_to_find" : "id",
  "what_to_find" : "user_name",
  "what_to_do" : "sendKeys",
  "type_of_action" : "textbox"
}
```

Figura 3: Exemplo de configuração no ficheiro de mapeamento

Na figura 3 segue um exemplo de configuração de uma variável no ficheiro de mapeamento. Mais abaixo é possível perceber que tipos de dados podem ser utilizados na configuração dos mapeamentos.

how_to_find O localizador que se pretende utilizar para encontrar o elemento na interface web. Este pode ter os próximos valores: *id, xpath, cssSelector, className, linkText, name, tagName, partialLinkText.*

what_to_find O valor do elemento a ser encontrado.

what_to_do Que tipo de ação será executada entre *sendKeys, click, submit, moveToElement.* Variável não obrigatória.

type_of_action Auxilia na determinação do tipo de elemento HTML. Não é uma variável obrigatória, quando utilizada pode ter os eguintes valores: *textBox, selectBox, checkBox.*

A figura 4 contém um exemplo de configuração para o ficheiro de valores. Podemos verificar que para a variável *user_name*, que vem do exemplo da figura 3, o valor que será enviado para o input do elemento HTML é *JohnDoe.*

```
[
  { "#model_name" : "#value"},
  { "name" : "JohnDoe" },
  ...
]
```

Figura 4: Exemplo de configuração no ficheiro de valores

#model_name Representa o elemento do modelo que necessita de estar associado a um valor.

#value O valor que será enviado para o input do elemento HTML ou usado nas validações para a geração dos testes.

2.3 Mutações

Uma mutação é um método de inserção de falhas em testes de software. É uma técnica para avaliar e melhorar um conjunto de testes[3]. Introduz-se pequenos erros afim de se entender se existem alterações no comportamento esperado da aplicação web. Estes erros são introduzidos nos caminhos em vez de ser no modelo, para simplificar o modelo.

Realizada uma análise foram encontrados os três tipos de erros base (*slip, lapse e mistake*) na interação de o utilizador com uma interface web[4]. Foram considerados dois grupos, o primeiro são as mutações para formulários e o segundo são mutações em cliques de botões ou links.

Seguem-se as mutações que são válidas para aplicar em formulários. Estas são introduzidas através de um ficheiro próprio, que será carregado para a ferramenta na mesma altura em que são carregados os ficheiros de mapeamento e o modelo.

Lapse Elimina uma ação de introdução de valores.

Slip Troca a ordem de execução das ações.

Mistake Troca o valor de um input.

Duplo clique num botão de submissão Realiza um duplo clique num botão de submissão. Por exemplo, quando uma página demora uma certo tempo a responder a um pedido, o utilizador carrega sucesssivas vezes no botão de submissão.

O grupo seguinte de mutações em cliques de botões ou links tem as próximas propriedades. Estes testes não são configuráveis, são gerados automaticamente mas com a introdução das propriedades nos caminhos.

Duplo clique em elementos Fazer duplo clique em um link ou botão simples da página.

Duplo clique em menus Realiza um duplo clique num menu em vez de um simples clique.

Carregar no botão back Quando a ligação está mais fraca o utilizador comum tende a forçar o retroceder. Assim torna-se importante preparar a aplicação a dar uma resposta a este tipo de eventos. A próxima propriedade também carece deste problema.

Refresh da página A espera é tanta para realizar o pedido que se faz vários refresh à página.

2.3.1 Ficheiro de mutações

Este ficheiro segue também o formato dos antecessores, isto é, semântica JSON. Como podemos verificar no exemplo de configuração da figura 5 existem alguns campos que necessitam de ser configurados e adaptados ao tipo de mutação.

```
[
  {
    "type" : "mistake",
    "model_element" : "name",
    "value" : "DoeJohn",
    "fail" : "0"
  },
  ...
]
```

Figura 5: Exemplo de configuração das mutações em ficheiro

type Utiliza-se para definir qual o tipo de mutação que se quer realizar. As propriedades válidas para utilização são: *lapse, slip, mistake e double_click.*

model_element Define em que variável do modelo se quer introduzir a mutação.

value Este campo só é obrigatório quando o tipo de mutação que se vai realizar é *mistake*. Significa que o valor aqui colocado será trocado pelo valor original do ficheiro de valores.

fail Permite determinar se com a introdução desta mutação o caso de teste vai falhar, gerando um teste inválido. Assim se o valor for "1" significa que a mutação vai causar falhas no teste, caso o valor seja "0" significa que esta mutação não interfirá no resultado final do caso.

Uma vez introduzido uma mutação para um formulário é necessário acrescentar o elemento <*error*> dentro do elemento <*transition*>. O valor do *target* no elemento <*error*> aponta para um novo estado. Assim o sistema está preparado para verificar o erro quando ele acontecer fazendo a transição de estados. Na figura 2 verifica-se que existe o elemento <*error*>, já a figura 6 contém a definição do estado de forma a que quando surja uma situação anômala na utilização da interface gráfica, se possa avaliar se é um erro esperado.

```
<state id="error_register">
  <onentry id="message_authentication_error" type="displayed?"/>
</state>
```

Figura 6: Exemplo da configuração do estado quando acontece um erro

3 Conclusão

Nesta secção é realizada uma análise ao trabalho desenvolvido ao longo destes cinco meses de projeto.

4 Referências

[1] F. Belli. *Finite-state testing and analysis of graphical user interfaces. In Proc. of the 12th ISSRE*. IEEE Computer Society Press, 2001.

[2] 'SCXML', http://www.w3.org/TR/scxml/

[3] R. Rodrigues. *Testes Baseados em Modelos*. Universidade do Minho, 2015.

[4] J. T Reason. *Human error*. Cambridge : Cambridge University Press, 1990.