4
Ferramentas

Neste capítulo serão apresentadas as duas ferramentas desenvolvidas para auxiliar o processo da geração de testes: o editor de tabela de decisão (que também gera os casos de teste), e o gerador de scripts de teste.

4.1.
Editor de tabela de decisão

Foi desenvolvida uma ferramenta para as tarefas de criação e edição da tabela de decisão e geração dos casos de teste. Esta ferramenta foi desenvolvida com base na ferramenta criada por (Lachtermacher 2010). As condições da tabela de decisão devem ser inseridas manualmente pelo usuário da ferramenta a partir de sua interpretação da descrição do caso de uso.

Uma tabela de decisão pode ser dividida em três partes principais: condições, ações e regras. Na ferramenta desenvolvida é possível adicionar cada uma destas partes, respectivamente, através das opções “Adicionar Condição”, “Adicionar Ação” e “Adicionar Regra” contidas no menu “Operações com Tabela”.

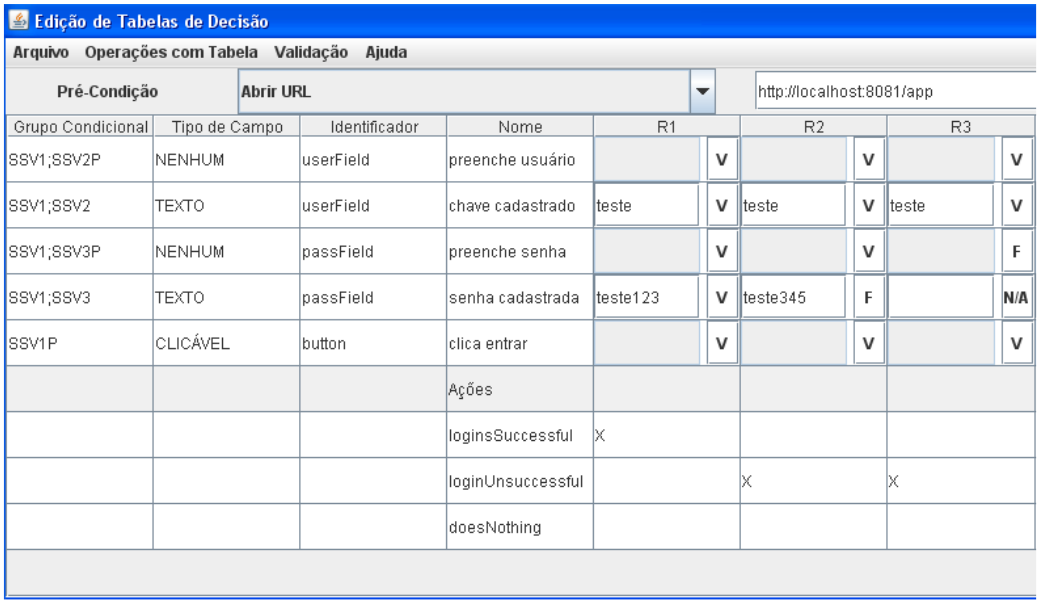

Figura 21 – Tela principal do Editor de tabela de decisão.

4.1.1. Edição da tabela

4.1.1.1. Condições

Uma condição na tabela de decisão está dividida em quatro partes: Grupo Condicional, Tipo de Campo, Identificador e Nome.

Grupo Condicional	Tipo de Campo	Identificador	Nome
MEOB1	SELETOR	id=sel1	condição1

Figura 22 – Elementos constituintes de uma condição.

Cada um destes quatro campos será explicado a seguir.

- **Tipo de Campo**: cada condição pode se traduzir em uma determinada ação a ser executada sobre um componente da interface. Para gerar corretamente o script de testes precisamos saber de qual tipo é este componente, como, por exemplo, se é um botão ou um campo de texto, etc. Esta informação serve para escolher qual comando da ferramenta usada para simular as ações do usuário, Selenium (Selenium, 2009) no caso deste trabalho, será codificado na geração do script de testes. Pelo motivo acima citado, o grupo de opções de Tipo de Campo é dependente da ferramenta de automação utilizada. Neste trabalho, com a utilização de Selenium para automatizar a interação com o navegador e elementos de interface, o elemento **Tipo de Campo** pode assumir seis valores:
  - **Seletor**: o elemento fornece uma lista de opções, onde uma delas, e somente uma, pode ser selecionada. Semelhante a um JComboBox de Java, é usado para indicar que o elemento é do tipo de *tag* de HTML “<select>” com atributo “multiple” valorado como **falso**.
  - **Seletor Múltiplo**: o elemento fornece uma lista de opções, onde uma ou mais opções podem ser selecionadas simultaneamente. Semelhante a uma JList de Java, é usada para indicar que o elemento é do tipo de *tag* de HTML

"<select>" com atributo "multiple" valorado como **verdadeiro**. Como a ferramenta Selenium utiliza para a interação com elementos do tipo de *tag* <select> dois comandos diferentes, dependendo do valor do atributo "multiple", foi preciso realizar essa distinção também na interface de edição do "Tipo de Campo", para sabermos qual comando utilizar na escrita automática dos scripts de testes.

  - **Marcador**: o elemento é um *checkbox* ou um *radio button*.
  - **Texto**: elemento onde o usuário pode entrar com informações textuais.
  - **Clicável**: elemento que responde a um clique do mouse como botões, *links*, etc.
  - **Nenhum**: este tipo pode ser usado no caso onde a condição não se refere a nenhum elemento da interface. Neste caso o valor para o Identificador não precisa ser preenchido e nenhum código será gerado para esta condição.

- **Identificador**: para simular a interação do usuário com o sistema a ser testado, a ferramenta responsável por tal tarefa deve ser capaz de identificar os elementos da interface que serão exercitados. A ferramenta utilizada neste trabalho, o Selenium (Selenium, 2009), pode identificar um elemento através de diversas estratégias:
  - **id**: localizar o elemento HTML pelo valor do atributo "id". Ex: "id=form1".
  - **name**: localizar o elemento HTML pelo valor do atributo "name". Ex: "name=telField".
  - **identifier**: localizar o elemento HTML pelo valor do atributo "id" e, caso não ache, localizar pelo valor do atributo "name". Ex: "identifier=form1".
  - **xpath**: localizar o elemento HTML usando uma expressão da forma XPath (XPath, 1999). Ex: "xpath=//input[@type=submit]".
  - **css**: localizar o elemento HTML através do "*cascading style sheets*" (CSS, 1999) aplicado a ele. Ex: "css=input[name="user"]"

- **link**: localizar o *link* HTML através do texto contido nele. Ex: “link=Pesquisa avançada”.
- **dom**: localizar um elemento HTML através de interpretação de expressão JavaScript utilizando *Document Object Model* (DOM, 1997). Ex: “dom=document.images[10]”.

Uma destas estratégias deve estar presente como valor do campo “Identificador” ao qual a condição se refere.

Como alternativa ao preenchimento do campo “Identificador” com um localizador utilizado pelo Selenium, podemos substituí-lo por uma palavra qualquer que será interpretado como uma variável pela ferramenta. Esta variável deve estar definida em um arquivo de configuração chamado “elements.map”, localizado na pasta “mapping”, e deve ter como valor o localizador que será utilizado pelo Selenium. Cada linha neste arquivo de configuração diz respeito a uma variável e seu valor, que devem estar separados por “:” (dois pontos).

Para ser interpretado como variável, o valor do campo “Identificador” na interface da tabela de decisão deve começar com um “:”(dois pontos), caso contrário, será interpretado diretamente como uma estratégia de localizador de Selenium (veja lista acima).

- **Nome**: é o nome do condicional e, caso o “Tipo de Campo” não seja “Seletor” ou “Seletor Múltiplo”, serve apenas para auxiliar o usuário na identificação da condição. No caso do Tipo de Campo ser Seletor ou Seletor Múltiplo, o campo Nome serve também para indicar qual item será selecionado. A última palavra, ou o último texto entre aspas (simples ou duplas), presente no campo “Nome” da condição será o valor que será selecionado no seletor e que deve fazer parte da lista de opções desse seletor. Neste caso o campo “Identificador” serve para detectar qual o seletor que terá uma de suas opções selecionadas. Por exemplo, suponhamos uma lista de opções com o Identificador “id1” e com as opções “Futuro de Dólar” e “Debênture”. Neste caso teríamos uma condição com Nome “selecionar ‘Futuro de Dólar’” (repare as aspas simples) para

selecionar a primeira condição da lista “id1”, ou com Nome “selecionar Debênture” para selecionar a segunda opção da lista.

- **Grupo Condicional**: O número de combinação de valores das condições cresce exponencialmente com o número de condições de acordo com a fórmula $m^n = c$, onde ‘m’ é o número de valores que as condições podem assumir, ‘n’ é o número de condições existentes e ‘c’ é o número de colunas (ou regras). Esta fórmula é válida se todas as condições puderem assumir o mesmo número de valores, por exemplo, se cada condição puder assumir o valor “V” (verdadeiro) ou “F” (falso) e existirem três condições, então teremos $2^3=8$ combinações possíveis. Se acrescentarmos mais duas condições, o número de combinações salta para 32. Entretanto pode ocorrer de algumas condições não admitirem certas combinações de valores entre si, como, por exemplo, duas condições que não podem ser verdadeiras ao mesmo tempo.

  Um exemplo deste último caso seria, dado o preenchimento de um formulário, a existência de uma condição “clicar botão ‘salvar’” e de outra condição “clicar botão ‘cancelar’”. Se ambas puderem assumir valor verdadeiro e falso, então o número de combinações possíveis entre estas duas condições é quatro. Entretanto um formulário não permitiria que as duas condições fossem verdadeiras ao mesmo tempo. Da mesma forma, para fins de teste, pode não ser desejável que as duas condições assumam valores falsos simultaneamente. Retirando-se estes dois casos, o número de combinações cai pela metade, de quatro para dois. Para deixar explícitas as relações restritivas entre as condições, como as do exemplo acima, existe o conceito de grupos de condicionais. A ideia é que certas condições podem pertencer a determinados grupos de condicionais que dividem entre si certas restrições. Estas restrições são usadas no momento da validação da tabela quanto à completeza e corretude, e servem para indicar ao computador se certas combinações de valores deveriam ou não estar presentes. Eliminando-se estas combinações de valores que não são permitidas, é possível diminuir significativamente o número de colunas da tabela de decisão.

Os tipos de grupos de condicionais disponíveis são:

- **Condição mutuamente exclusiva**: nas condições que pertencem a este grupo, apenas uma pode assumir o valor verdadeiro, sendo que todas as condições podem ser falsas. A sigla usada para este tipo de grupo condicional na interface é "ME".
- **Condição mutuamente exclusiva obrigatória**: uma, e apenas uma, condição deste grupo deve ser verdadeira. Todas as condições não podem ser falsas simultaneamente. A sigla usada para este tipo de grupo condicional na interface é "MEOB".
- **Condição de conjunto obrigatória**: dentre as condições deste grupo deve haver pelo menos uma com o valor verdadeiro. A sigla usada para este tipo de grupo condicional na interface é "OBR".

Para o entendimento dos próximos três tipos de condicionais é importante a introdução de dois conceitos: o de condição principal e o de condições associadas. Dentre as condições destes últimos três grupos haverá uma condição marcada como principal e as demais condições do grupo da condição principal são denominadas de condições associadas. Estas por sua vez, têm seus valores restringidos de acordo com o valor assumido pela condição principal e do grupo de condicionais ao qual pertencem.

- **Condição de mascaramento**: para as condições pertencentes a este grupo, se a condição principal for verdadeira, então todas as condições associadas também serão verdadeiras. A abreviação usada para este tipo de grupo condicional na interface é "Masc".
- **Somente se verdadeiro**: neste grupo, se a condição principal for falsa então as condições associadas não se aplicam. A sigla usada para este tipo de grupo condicional na interface é "SSV".

- **Somente se falso**: semelhante à anterior, com a diferença que as condições associadas não se aplicam somente se a condição principal for verdadeira. A sigla usada para este tipo de grupo condicional na interface é “SSF”.

Uma condição pode pertencer a mais de um grupo condicional. Cada grupo condicional distinto é identificado através de um índice inteiro ao final de sua sigla.

Grupo Condicional	Tipo de Campo	Identificador	Nome
ME1;ME2	TEXTO	id1	condição1
ME1;ME3	CLICÁVEL	id2	condição2
ME2;ME3	SELETOR	id3	condição3

Figura 23 – condições com mais de um grupo condicional.

No caso de grupos condicionais que admitem condições principais, a sigla do grupo condicional desta condição termina com a letra “P” para indicar qual condição é a principal.

Grupo Condicional	Tipo de Campo	Identificador	Nome
SSV1P	CLICÁVEL	id1	condição1
SSV1	TEXTO	id2	condição2
SSV1	SELETOR	id3	condição3

Figura 24 – grupo condicional com condição principal.

Para atribuir um grupo condicional a uma condição é necessário primeiramente adicionar uma condição à tabela. Após esse passo há dois caminhos que se pode tomar, descritos a seguir.

Para atribuir um grupo do tipo mutuamente exclusivo, mutuamente exclusivo obrigatório ou conjunto obrigatório é utilizada a interface da Figura 25. É possível acessar esta interface através da opção “Adicionar Grupo Condicional” no menu “Operações com Tabela”. Nessa tela, a primeira lista apresenta todas as condições existentes na tabela e a segunda lista apresenta as condições que serão

atribuídas ao grupo de condicional desejado. O tipo de grupo de condicional pode ser selecionado na *combo* presente na interface.

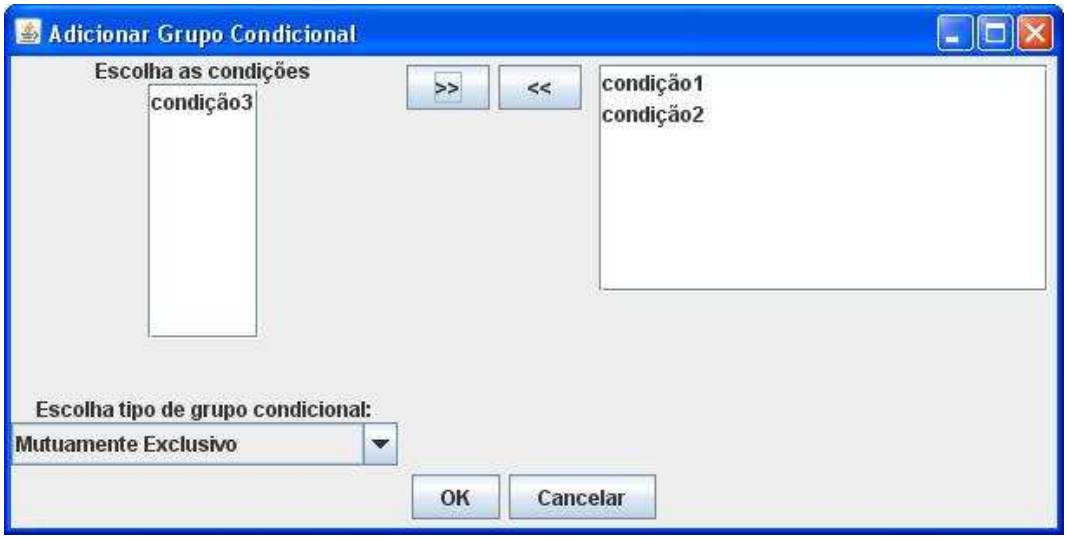

Figura 25 – Tela de adição de grupo condicional sem condição principal.

Para os tipos de grupos de condicionais que admitem condições principais, ou seja, “condição de mascaramento”, “somente se verdadeiro” e “somente se falso” deve ser acessada outra interface através da opção “Adicionar Grupo Condicional com Principal” presente no menu “Operações com Tabela”.

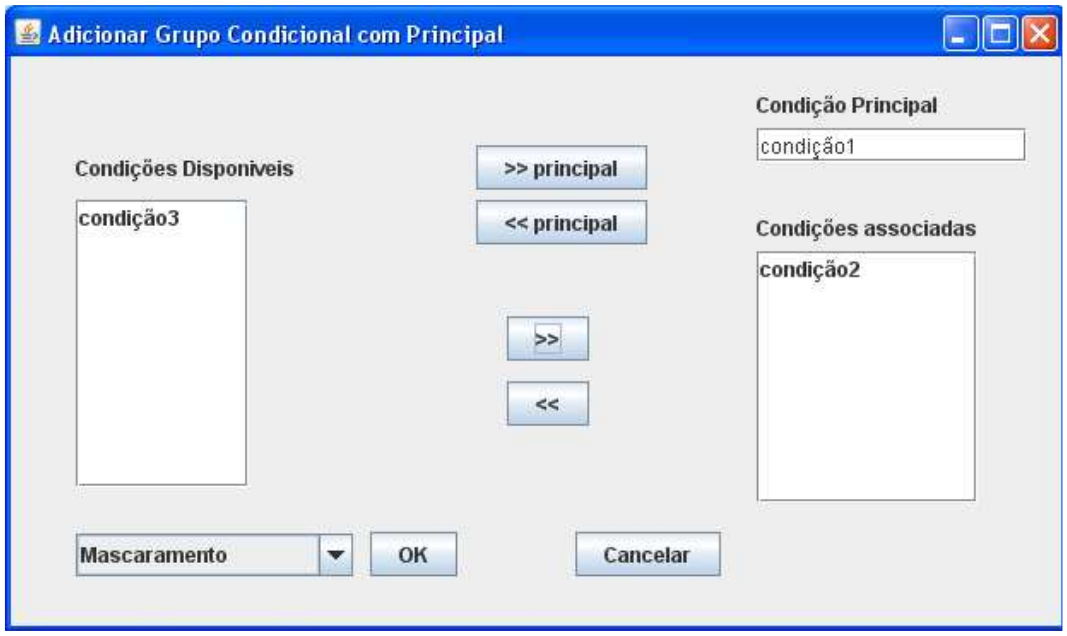

Figura 26 – Tela de adição de grupo condicional com condição principal.

Nessa tela, é necessário selecionar uma condição principal da lista de condições disponíveis (lista à esquerda da tela) e clicar no botão “>> principal”. Para retirar a condição principal, clica-se no botão “<< principal”. As condições associadas são exibidas na lista à direita da tela. Estas condições também são escolhidas a partir das condições disponíveis na lista à esquerda, e são manipuladas através dos botões “>>” e “<<” que, respectivamente, incluem e excluem uma condição à lista de condições associadas. O tipo de grupo condicional deve ser selecionado na *combo* no canto inferior esquerdo da tela.

4.1.1.2. Ações

Na parte inferior da tabela encontram-se as ações. Estas correspondem aos oráculos dos casos de teste a serem gerados. Para cada combinação possível de valores para as condições, deve estar associada pelo menos uma ação a ser executada. A indicação que uma ação deve ser tomada para determinada regra se dá através da inserção do caractere “X” na interseção da linha da Ação com a coluna da Regra.

Na ferramenta desenvolvida, cada ação corresponde a um método que deve ser implementado em um arquivo à parte na linguagem Java. O nome deste arquivo pode ser configurado através do arquivo “config.properties”.

```
RESULTS_CLASS=Results
```

Figura 27 – configuração de arquivo de ações.

O nome da ação na tabela de decisão deve corresponder exatamente ao nome do método usado para implementá-la. O código necessário para implementar as ações não é gerado automaticamente e deve ser codificado pelo usuário, pois não há como prever que passos serão necessários para verificar o resultado de certa combinação de valores de condições.

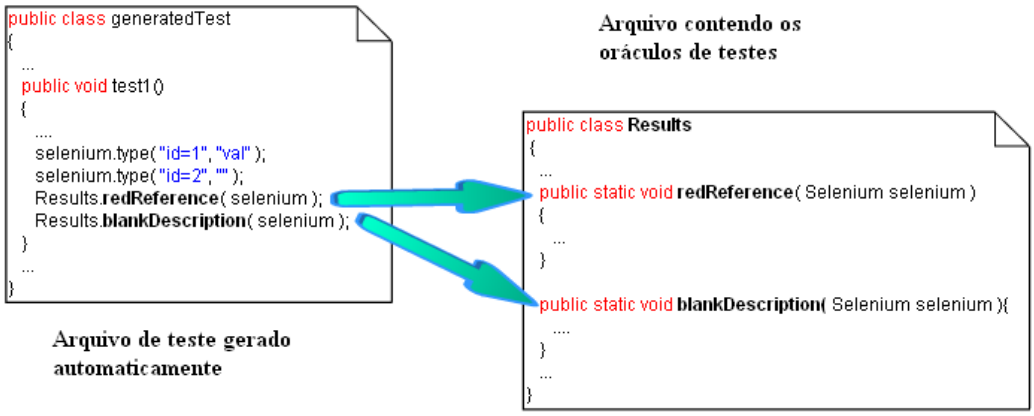

Figura 28 – Esquema de dependência entre o arquivo de teste gerado e o arquivo de oráculos.

4.1.1.3. Regras

Cada regra refere-se a uma combinação de valores das condições. As regras são representadas na forma de colunas na tabela de decisão, onde cada coluna é uma regra. A altura da coluna, ou seja, o número de linhas, será igual ao número de condições existentes na tabela, e cada célula desta coluna será preenchida unicamente com um valor que será referente à condição da mesma linha da célula. Idealmente, para a total cobertura das condições, as regras devem esgotar todas as possibilidades de combinação de valores entre as condições.

Os valores disponíveis que uma célula de regra para uma condição pode assumir são “V” (verdadeiro), “F” (falso) e, em alguns casos, “---” (indiferente) e “N/A” (não aplicável). Estes valores estão disponíveis através de uma *combo* para cada célula da regra.

É comum o uso dos valores “---” ou “-” para indicar que é indiferente se determinada condição é verdadeira ou falsa, ou seja, o resultado esperado é o mesmo para estes valores. Este recurso é bastante utilizado para simplificar e facilitar a construção da tabela de decisão, uma vez que duas colunas em potencial tornam-se uma só. Entretanto, para fins de teste, é importante que tal condição seja testada tanto com o valor verdadeiro quanto falso, a fim de verificar se realmente o resultado esperado foi o resultado obtido para ambas as valorações. Assim, uma regra com este valor será desmembrada em duas: uma com o valor

"V" e outra com o valor "F" quando da transformação da tabela para os casos de teste.

Caso o valor desta condição não satisfaça alguma restrição de algum tipo de condicional ao qual pertence, a validação da tabela acusará erro para esta condição.

Os valores assumidos por condições que pertençam aos grupos condicionais "Condição mutuamente exclusiva", "Condição mutuamente exclusiva obrigatória", "Condição de conjunto obrigatória" e "Condição de mascaramento" podem ser "V" (verdadeiro) e "F" (falso) e, com exceção do tipo "Condição de mascaramento", também pode assumir o valor "---" (indiferente). No entanto, se a condição pertencer também a um grupo "Somente se falso" ou "Somente se verdadeiro" e for uma condição associada a uma condição principal de alguns destes grupos, então poderá assumir também o valor "N/A", dependendo do valor da condição principal.

Uma condição principal do grupo de condicional "Somente se falso" ou "Somente se verdadeiro" pode assumir os valores "V" e "F". Entretanto se ela for uma condição associada de outro grupo de condicional do tipo SSV ou SSF, também poderá assumir o valor "N/A". Uma condição associada pode assumir os valores "V", "F" ou "N/A".

Para condições de "Tipo de Campo" como "Texto", onde dados textuais podem ser fornecidos pelo usuário, a célula destas condições na regra permite que o usuário preencha a informação necessária. Para condições com outros tipos de campos, esta parte da célula da regra é desabilitada.

Grupo Condicio...	Tipo de Campo	Identificador	Nome	R1		R2	
	TEXTO	id=senha	preecher senha corretamente	teste123	V	erro	F

Figura 29 – Células de regra para condição do tipo Texto.

Grupo Condicio...	Tipo de Campo	Identificador	Nome	R1		R2	
	SELETOR	id=sel1	selecionar Futuro de Juros		V		F

Figura 30 – Células de regra para condição que não é do tipo Texto.

Duas ou mais condições podem conter o mesmo valor para o campo "Identificador" e neste caso elas se referem ao mesmo elemento na interface. Se estas condições também forem do tipo de campo "Texto" então, ao preencher o

dado textual de uma condição, as outras condições referentes ao mesmo elemento terão seus dados textuais para esta regra preenchidos automaticamente com o mesmo valor fornecido à célula referente à condição sendo preenchida.

Cada regra gera um caso de teste representado como um método no arquivo de testes. Cada método gerado contém os comandos necessários para implementar as ações descritas pelas condições e a chamada dos métodos referentes às ações especificadas na tabela de decisão. O nome do método gerado será o mesmo nome da coluna que o deu origem. Este nome é “R” acrescido de um índice para cada coluna, por exemplo, para a primeira coluna (ou regra), este nome é “R1”. O nome padrão das colunas pode ser alterado no arquivo “msg.properties” através da chave “TABLE_COL_NAME_PATTERN”.

4.1.1.4. Pré-condição

Neste trabalho, os testes gerados por uma tabela de decisão devem refletir os testes necessários para um caso de uso. Assim, como um caso de uso pode depender da execução de outro caso de uso como pré-condição, os testes derivados de uma tabela de decisão podem depender de uma sequência de passos executada em um teste de outra tabela de decisão. Tomemos como exemplo uma página de vendas onde, para comprar um produto o usuário precisa autenticar-se no sistema como demonstra o diagrama abaixo.

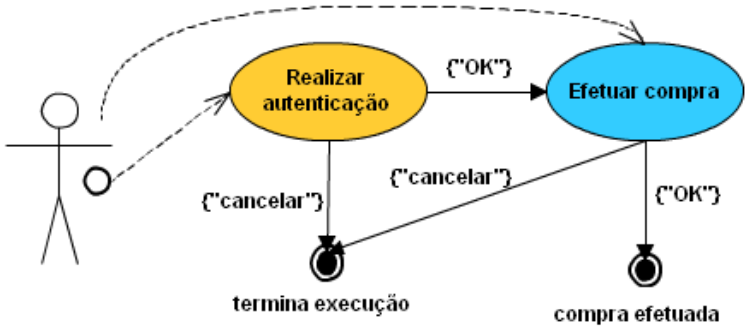

Figura 31 – Diagrama de estado para o caso hipotético.

Com a utilização do processo e da ferramenta sugeridos, os dois casos de uso possuirão suas próprias tabelas de decisão que darão origem aos testes.

O caso de uso “Realizar autenticação” só pode ser executado após o cliente autenticar-se com sucesso no sistema de vendas. Podemos então dizer que o caso de uso “Efetuar compra” tem como pré-condição o caso de uso “Realizar autenticação”. Mais especificamente, o caso de uso “Efetuar compra” precisa que o caminho do caso de uso “Realizar autenticação” que faz a correta autenticação no sistema seja executado. Podemos representar essa dependência de execução dos casos de uso como uma máquina de estados como na figura acima. Cada estado tem a sua tabela de decisão e a transição de estados é definida pelas ações do usuário e pré-condições.

Na ferramenta desenvolvida, essa dependência fica representada pelo campo “Pré-condição”. Para a tabela de decisão do caso de uso “Efetuar compra” teremos o campo “Pré-condição” preenchido com uma chamada para o método que implementa o caso de teste no qual o usuário faz a autenticação com sucesso, método esse oriundo da tabela de decisão para o caso de uso “Realizar autenticação”. Assim, o valor deste campo seria a chamada para um método no estilo Java, ou seja, “<Nome da classe>.<nome do método>”, onde <Nome da classe> é a classe que contém os testes para o caso de uso “Realizar autenticação” e <nome do método> é o método desta classe que realiza a autenticação com sucesso.

Figura 32 – Campo pré-condição com teste.

Porém no primeiro caso de uso do diagrama, como estamos tratando de sistemas web, teríamos como pré-condição o carregamento de algum endereço no navegador. Assim, alternativamente, o campo pré-condição permite especificar que devemos carregar algum endereço antes de começar os testes.

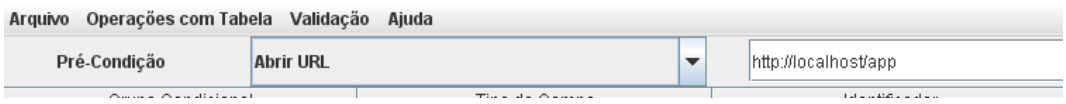

Figura 33 - Campo pré-condição com URL.

4.1.2.
Validação da tabela

Pode-se fazer a validação da tabela de decisão automaticamente através da opção “Validar Tabela” presente no menu “Validação”. Na validação da tabela são feitas as seguintes verificações:

- Verificar se o campo pré-condição foi preenchido.
- Verificar se todas as regras possuem pelo menos uma ação associada.
- Verificar se não existem duas colunas com valores iguais, levando-se em conta o valor “indiferente” (---) e “não aplicável” (N/A).
- Verificar se os grupos condicionais estão consistentes entre si, isto é, se a correta valoração de todas as condições de um grupo condicional ainda torna possível a correta valoração das condições de outro grupo condicional. Por exemplo: não podem existir duas condições “c1” e “c2” mutuamente exclusivas e uma terceira condição “c3” mutuamente exclusiva com “c2”. Para realizar essa verificação, primeiramente, para cada condição, são montadas todas as relações de valores que esta condição pode ter com as outras na forma de um grafo, levando-se em consideração os grupos de condicionais de cada condição. Em seguida são montadas todas as combinações possíveis de valores (colunas) das condições. Por último é checado se todas as relações estão presentes nas colunas de valores geradas para esta tabela e se existe alguma coluna que não está contemplada pelas relações geradas.
- Verificar se todas as combinações de valores das condições foram formuladas, levando em conta as restrições dos grupos condicionais existentes. Para isto, a ferramenta monta todas as combinações possíveis, dada a configuração de condições e grupos condicionais, e compara este resultado com os presentes na tabela de decisão.

Se, ao tentar salvar a tabela de decisão ou gerar os casos de teste, a tabela não tiver sido validada ainda, será perguntado ao usuário se ele deseja fazê-lo. Ao final da validação é exibida uma mensagem ao usuário informando sobre o sucesso ou não da validação, neste último caso identificando e informando quais

foram as razões da tabela de decisão não estar válida. Entretanto, mesmo que a tabela de decisão não passe em algum passo da validação, ainda será possível salvá-la ou gerar os casos de teste. Isto porque ela pode ainda não estar completa e será continuada no futuro, ou podem existir alguns casos de teste (regras da tabela de decisão) que o usuário não está interessado em gerar.

4.1.3. Persistência da tabela

Cada tabela de decisão pode ser salva no formato XML (XML, 2008) através da opção “Salvar” presente no menu “Arquivo”. Toda informação necessária para restaurar o estado da tabela é salva neste arquivo. Abaixo temos a estrutura de um arquivo gerado para uma tabela contendo uma condição, uma ação e uma regra.

```
<TableData>
    <actions>
        <TableAction>
            <identifier>Ação</identifier>
            <actionsRules>
                <string>X</string>
            </actionsRules>
        </TableAction>
    </actions>
    <comboFieldValue>comando do pré-requisito</comboFieldValue>
    <comboValue>Tipo de pré-requisito</comboValue>
    <conditions>
        <TableCondition>
            <actionType>Tipo de Campo</actionType>
            <condType>Grupos Condicionais separados por ';'</condType>
            <fieldName>Nome</fieldName>
            <identifier>Identificador</identifier>
            <rules>
                <CellValue>
                    <option>V</option>
                    <text>Dados</text>
                <CellValue>
            </rules>
        </TableCondition>
    </conditions>
</TableData>
```

Figura 34 – Estrutura do XML de uma tabela de decisão.

A descrição das *tags* geradas é apresentada abaixo:

- **<TableData>**: início dos dados da tabela de decisão. Há somente uma *tag* desta por tabela de decisão, e somente uma tabela de decisão por arquivo.

- **<actions>**: lista de ações da tabela de decisão. Há somente uma *tag* desta por tabela de decisão. Pode conter zero ou mais *tags* <TableAction> de acordo com o número de ações existentes na tabela de decisão.
- **<TableAction>**: refere-se à uma ação e contém os dados desta. Para cada ação na tabela de decisão haverá uma *tag* <TableAction> no arquivo XML gerado.
- **<identifier>**: quando esta *tag* estiver dentro da *tag* <TableAction> o seu valor se refere ao nome da ação. Este deve ser o nome de um método que servirá de oráculo e deve estar presente no arquivo de oráculos (seção 4.1.1.2). Quando esta *tag* estiver dentro da *tag* <TableCondition> o seu valor será interpretado como um identificador utilizado pela ferramenta que automatiza a interação do usuário com o sistema sob testes. Nesta implementação a ferramenta é o Selenium (Selenium, 2009) e o valor dentro da *tag* <identifier> deve ser um identificador válido para o Selenium.
- **<actionRules>**: é a lista de *tags* <string>. O tamanho desta lista, isto é, o número de *tags* <string>, é igual ao número de regras existentes na tabela de decisão.
- **<string>**: é o valor para uma célula da ação referente a uma determinada regra. O primeiro <string> da lista <actionRules> refere-se à célula da tabela de decisão produzida pela interseção da linha da ação com a primeira regra; o segundo <string> refere-se à interseção da linha da ação com a segunda regra, e assim por diante. Seu valor deve ser vazio, no caso da ação não ser executada para alguma regra, ou um “X”, no caso da ação ser executada para a regra.
- **<comboValue>**: tipo da pré-condição. Deve assumir um de dois valores: “Caso de Teste” ou “Abrir URL”.
- **<comboFieldValue>**: comando da pré-condição. Caso o valor de <comboValue> seja “Abrir URL” então o valor de <comboFieldValue> deve ser um endereço HTTP. Caso o valor de <comboValue> seja “Caso de Teste” então o valor de

<comboFieldValue> deve estar no formato “<Classe>.<método>”, onde <Classe> é a classe onde o método <método> está implementado. Este método contém os passos necessários como pré-condição para os testes da tabela de decisão expressa neste arquivo XML. Para maiores detalhes consulte a seção 4.1.1.4.

- **<conditions>**: lista de condições da tabela de decisão. Há somente uma *tag* desta por tabela de decisão. Pode conter zero ou mais *tags* <TableCondition> de acordo com o número de condições existentes na tabela de decisão.
- **<TableCondition>**: refere-se a uma condição e contém os dados desta. Para cada condição na tabela de decisão haverá uma *tag* <TableCondition> no arquivo XML gerado.
- **<actionType>**: é o valor do “Tipo de Campo” da condição. Nesta implementação, conforme explicado na seção 4.1.1.1, esta *tag* pode assumir um dos seguintes valores: “Seletor”, “Seletor Múltiplo”, “Texto”, “Clicável”, “Marcador” e “Nenhum”.
- **<condType>**: contém os grupos condicionais ao quais esta condição pertence. No caso de mais de um grupo, estes serão separados por ‘;’ (ponto e vírgula).
- **<fieldName>**: é o nome da condição.
- **<rules>**: é a lista de *tags* <CellValue>. O tamanho desta lista, isto é, o número de *tags* <CellValue>, é igual ao número de regras existentes na tabela de decisão.
- **<CellValue>**: representa uma célula de uma regra na tabela decisão. Contém o valor da condição e um possível dado textual fornecido pelo usuário. Ela representa a célula da tabela de decisão formada pela interseção da linha da condição com a coluna de uma regra. No caso de mais uma regra, a primeira *tag* <CellValue> representa a interseção da condição com a primeira regra, a segunda *tag* <CellValue> representa a interseção da condição com a segunda regra, e assim por diante. Haverá tantas *tags* <CellValue> para uma condição quanto o número de regras existentes na tabela de decisão.

- <**option**>: É o valor da condição em uma determinada regra. Pode assumir um dos quatro valores: “V”, “F”, “---”, “N/A”.
- <**text**>: No caso onde “Tipo de Campo” da condição é “Texto”, esta *tag* pode conter um valor fornecido pelo usuário. Caso contrário ela deve ter valor vazio.

4.1.4. Geração dos casos de teste

A geração da massa de testes é feita através da opção “Gerar Suite” do menu “Arquivo”. A massa de casos de teste é gerada e salva em disco no formato de um arquivo XML. Este arquivo servirá de entrada para a ferramenta que gera os scripts de teste.

Caso a tabela de decisão ainda não tenha sido validada, será perguntado ao usuário se ele deseja fazê-lo, entretanto a bateria de testes pode ser gerada mesmo se a tabela de decisão não passar ou não sofrer o processo de validação. Optou-se por esta abordagem para que o usuário, caso deseje, possa gerar casos de teste mesmo sem a tabela de decisão estar completa, pois alguns casos de teste (regras na tabela de decisão) podem não ser de interesse imediato.

Cada regra da tabela de decisão transforma-se em um caso de teste na bateria de testes. Cada caso de teste contém, como oráculos, as ações marcadas para ele na tabela de decisão.

4.1.4.1. Persistência dos casos de teste

Abaixo temos a estrutura de um caso de teste no formato XML gerado pela ferramenta:

```
<Suite>
    <suiteName>Nome da bateria de testes</suiteName>
    <preTest>Classe e método do pré-requisito</preTest>
    <url>URL a ser carregada</url>
    <testCases>
        <TestCase>
            <elements>
                <Element>
                    <identifier>Identificador</identifier>
                    <type>Tipo de Campo</type>
                    <value>Dados</value>
                </Element>
            </elements>
            <name>Nome do caso de teste</name>
            <resultsNames>
                <string>Nome do oráculo</string>
            </resultsNames>
        </TestCase>
    </testCases>
</Suite>
```

Figura 35 - Estrutura do XML de uma *suite* de testes derivada da tabela de decisão.

As descrições de cada *tag* do arquivo XML da figura acima são dadas a seguir:

- **<Suite>**: início dos dados da *suite* de testes. Há somente uma *tag* desta por *suite* de testes, e somente uma bateria de testes por arquivo.
- **<suiteName>**: nome da *suite* de testes. Tem o mesmo valor que o nome do arquivo XML ao qual pertence.
- **<preTest>**: se a pré-condição do caso de uso representado pelos testes descritos no XML for a execução de outro caso de uso, então esta *tag* terá o nome da classe que representa este caso de uso e o nome do método que representa o caminho a ser executado deste caso de uso no formato <Classe>.<método>. Se esta *tag* existir, a *tag* <url> não estará presente neste XML e vice-versa.
- **<url>**: esta *tag* contém um endereço HTTP se a pré-condição for carregar um endereço no navegador, caso contrário esta *tag* não estará presente.
- **<testCases>**: lista de casos de teste. Contém zero ou mais *tags* <TestCase>.

- **<TestCase>**: representa um caso de teste.
- **<elements>**: lista de *tags* <Element>. Contém zero ou mais elementos <Element>.
- **<Element>**: refere-se a um elemento da interface que será exercitado, como botões, campos de texto, etc.
- **<identifier>**: identificador utilizado pela ferramenta que automatiza a interação do usuário com o sistema sob testes. Nesta implementação a ferramenta é o Selenium (Selenium, 2009) e o valor dentro da tag <identifier> deve ser um identificado válido para o Selenium.
- **<type>**: tipo do campo na interface. Pode assumir os valores "CLICK", "INPUT", "SELECT", "MULTISELECT" e "CHECK", que se referem, respectivamente, aos campos do tipo "Clicável", "Texto", "Seletor", "Seletor Múltipo" e "Marcador".
- **<value>**: dependendo do tipo de campo do elemento, ou seja, do valor da *tag* <type>, pode ser necessária alguma informação adicional. Esta informação estará no valor da tag <value>. Se a *tag* <type> tiver como valor "INPUT" então <value> terá como conteúdo o dado textual fornecido pelo usuário. Caso <type> seja do tipo "CHECK" então <type> terá como valor "CHECK" ou "UNCHECK" que significam, respectivamente, que o elemento de interface, que é do tipo *checkbox* ou *radio button,* deve ser marcado ou desmarcado.
- **<name>**: Nome do caso de teste.
- **<resultsNames>**: lista de ações para este caso de teste.
- **<string>**: cada tag "string" representa uma ação associada ao caso de teste.

4.1.5.
Diagrama de classes

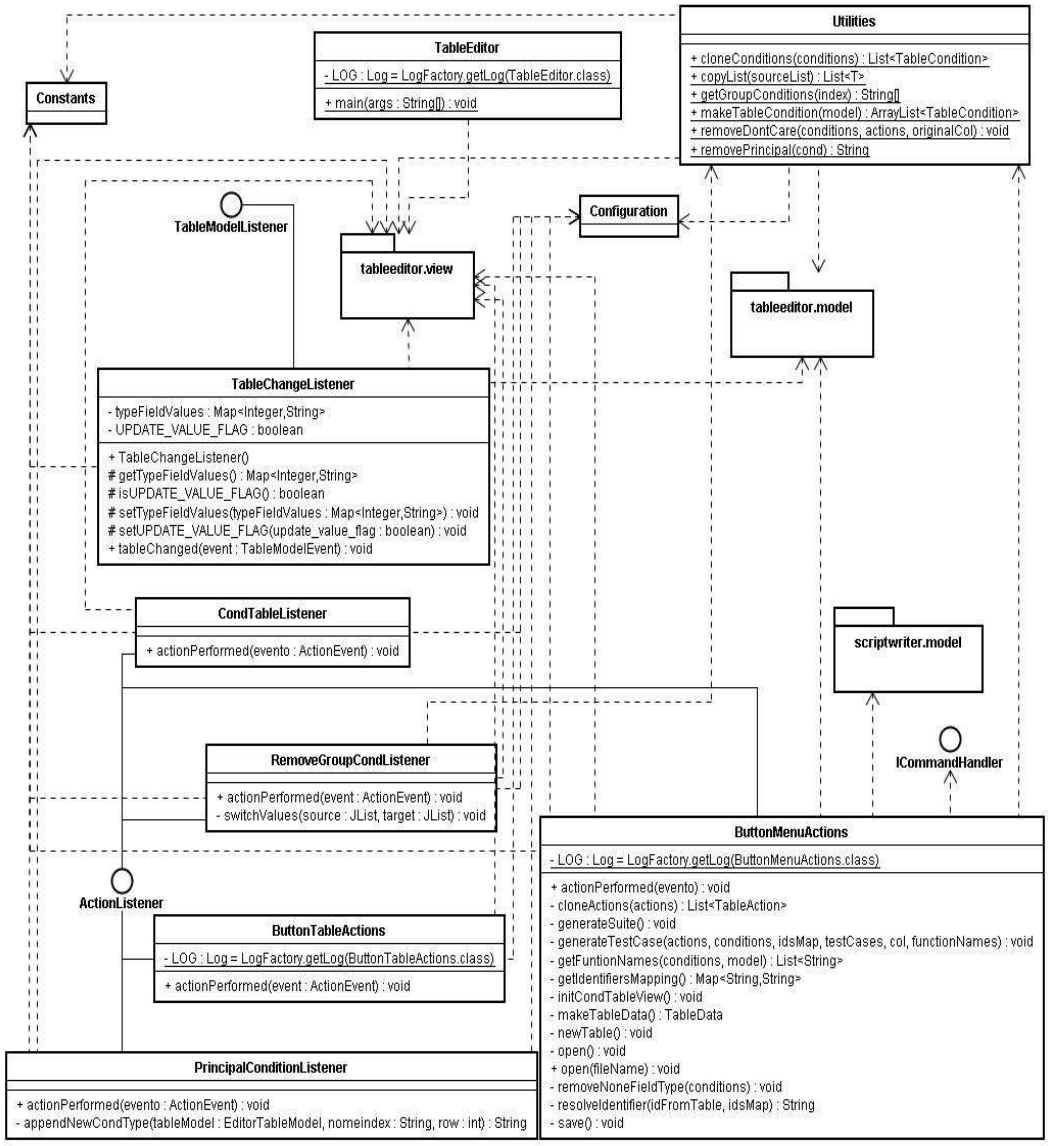

Figura 36 – Diagrama de classes dos pacotes tableeditor e tableeditor.control.

A figura acima contém as classes que formam os pacotes **tableeditor** e **tableeditor.control**, onde estão localizados a classe principal do aplicação, a **TableEditor,** e classes que implementam a interface **ActionListener**, responsáveis por implementar as funcionalidades disparadas por ações como cliques de botões ou edições de dados. Nestes pacotes também encontram-se classes utilitárias como **Constants**, com constantes utilizadas no sitema; **Utilities**, com funções utilitárias ao sistema; e **Configuration,** que fornece variáveis configuráveis a partir de um arquivo de propriedades.

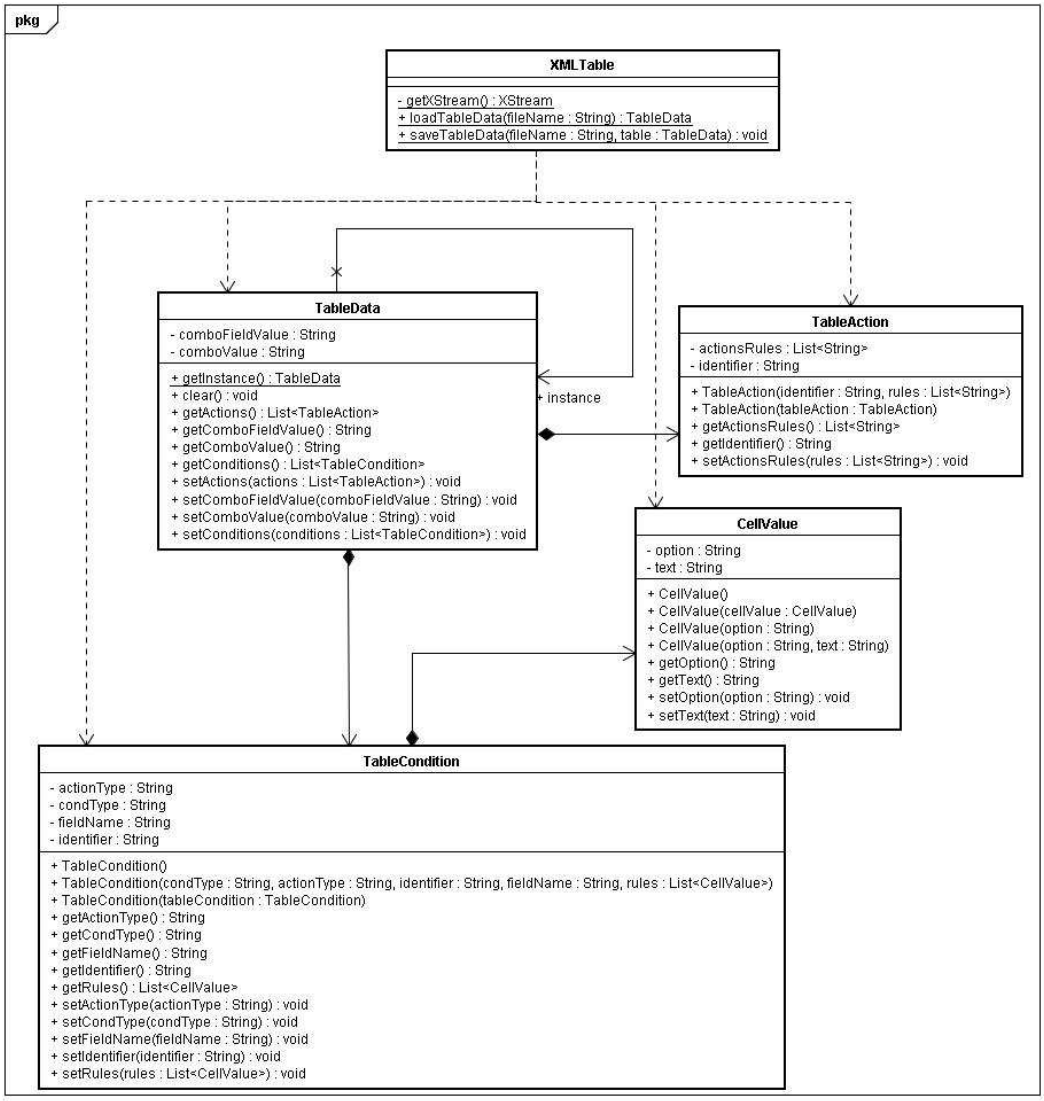

Figura 37 – Diagrama de classes do pacote tableeditor.model.

A figura 37 mostra a representação lógica de uma tabela de decisão. A classe **TableData** é carregada pela classe **XMLTable** a partir de um arquivo XML. Cada objeto **TableData**, que é a tabela propriamente dita, pode ter vários objetos **TableCondition**, representando condições da tabela, e vários objetos **TableAction**, representando as ações presentes na tabela. Cada **TableCondition** pode conter vários objetos **CellValue**, que representam os dados de uma célula de uma Regra da tabela de decisão, a saber: o valor da condição (ver seção 4.1.1.1) e um eventual dado para a condição. Os dados da classe **TableData** são usados para a geração do XML contendo os casos de teste semânticos, que, posteriormente, serão traduzidos em scripts de testes executáveis.

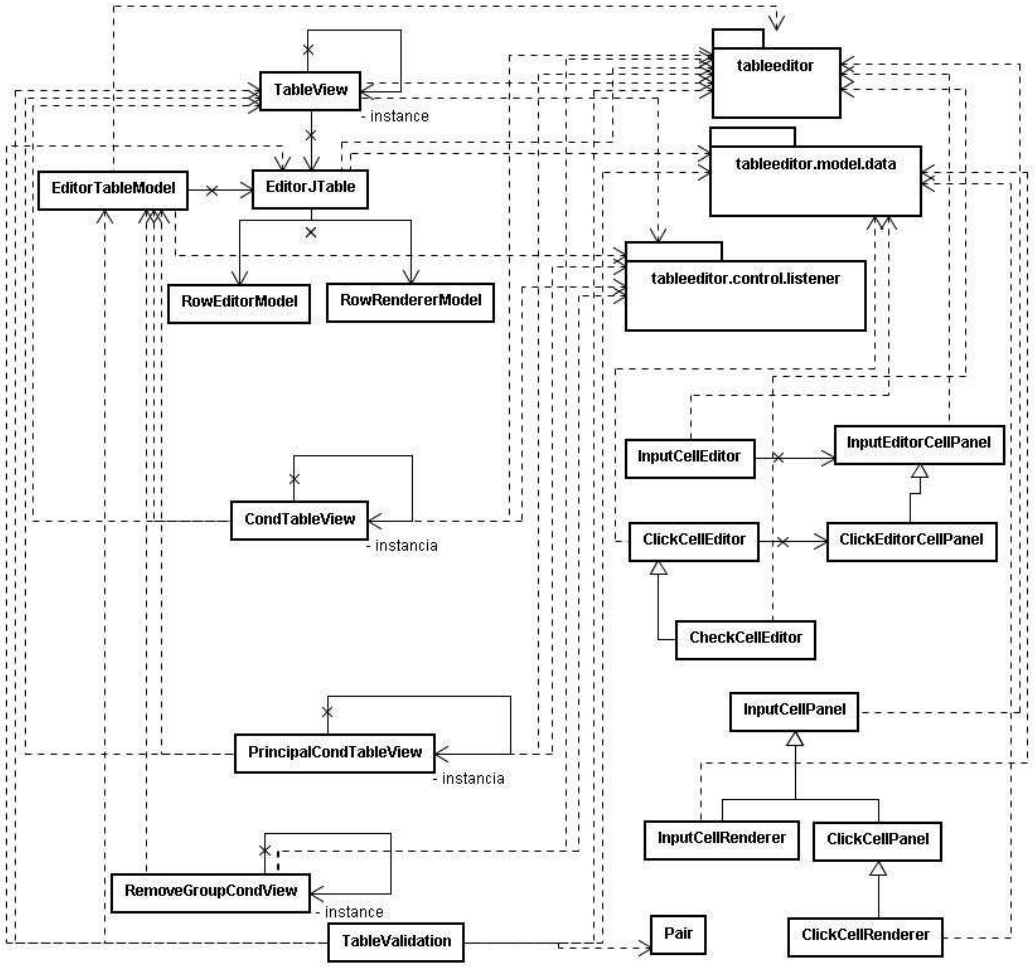

Figura 38 - Diagrama de classes do pacote tableeditor.view.

A figura acima mostra as classes reponsáveis pela representação visual da ferramenta, em especial, a classe **TableView** representa a janela principal da aplicação; **EditorJTable** representa a tabela de decisão que o usuário manipula e da qual a classe **TableData** retira os dados para armazenamento em XML; a classe **TableValidation** contém todo o código necessário para a validação da tabela.

4.2. Gerador de scripts de teste

A geração automática dos scripts de teste é feita através de outra ferramenta. Esta ferramenta recebe como entrada o XML descrevendo os casos de teste (seção 4.1.4.1), gerado pela ferramenta de edição da tabela de decisão, e gera, como saída, o script de teste.

Neste trabalho o script de teste é gerado na linguagem Java e a automação do navegador é feita a partir da biblioteca do Selenium (Selenium, 2009) para Java. Para a automação da execução dos testes é usada a biblioteca JUnit (JUnit, 2009) e, consequentemente, o arquivo gerado é escrito em conformidade com as regras desta biblioteca.

4.2.1. Diagrama de classes

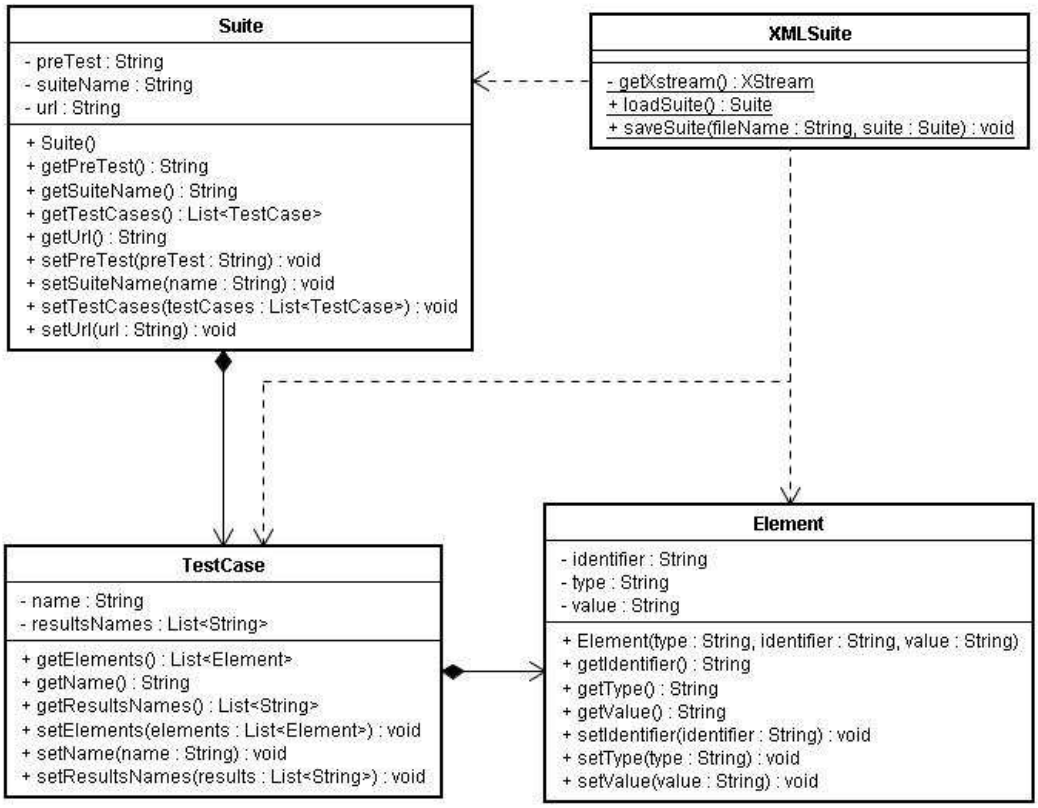

Figura 39 - Diagrama de classes do pacote scriptwriter.model.

A figura 39 mostra a representação lógica de uma *Suite* de testes. A classe **Suite** é carregada pela classe **XMLSuite** a partir de um arquivo XML. Cada objeto **Suite** pode ter vários objetos **TestCase**, que são as representações dos casos de uso. Cada **TestCase**, por sua vez, pode conter vários objetos **Element**, que são os menores comandos possíveis executados na interface, como clicar em um botão ou preencher um campo. Os dados da classe **Suite** são usados para a geração do script de teste executável.

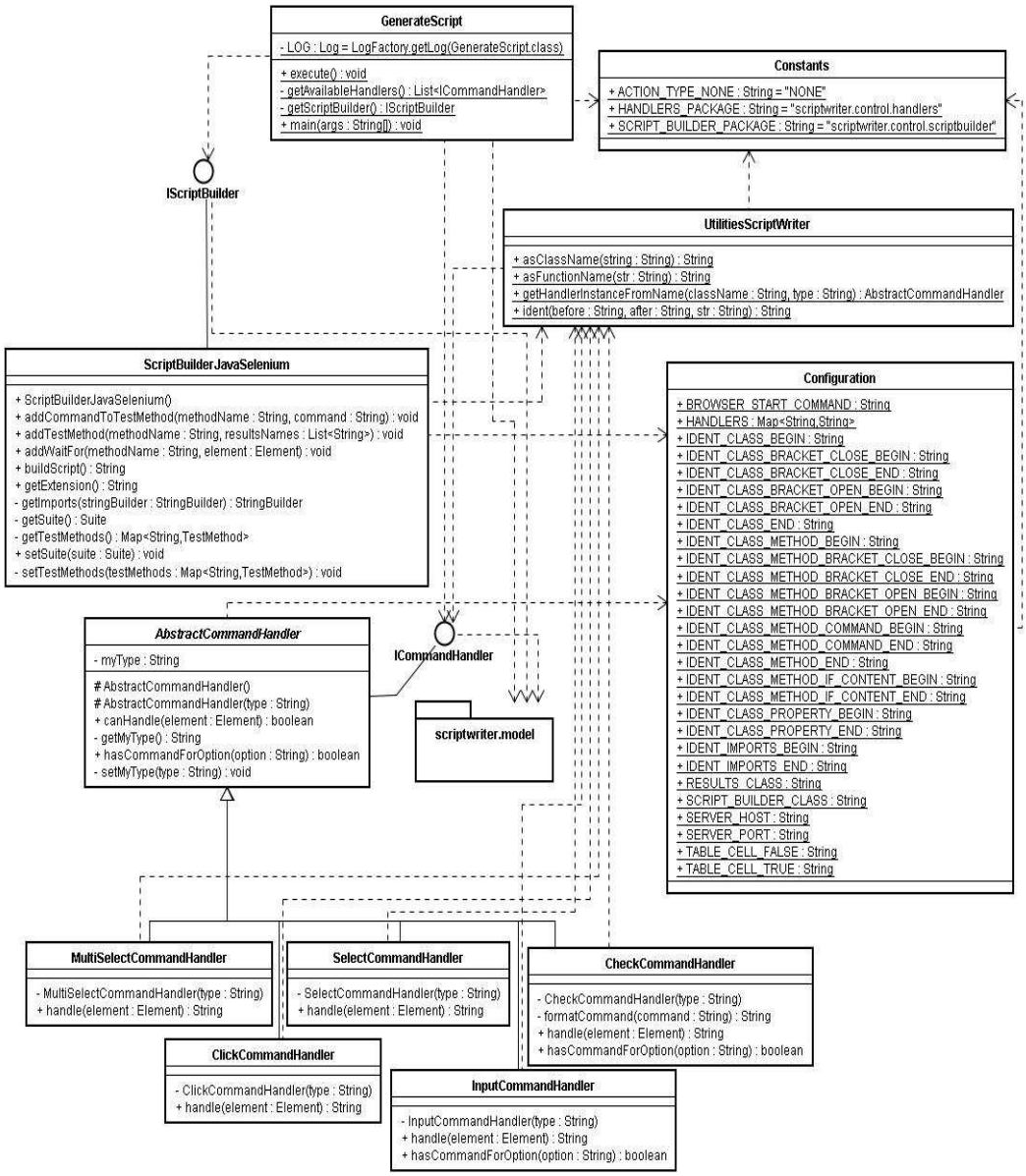

Figura 40 - Diagrama de classes dos pacotes scriptwriter e scriptwriter.control.

A figura acima mostra as classes envolvidas na geração dos scripts de testes executáveis. A classe **GenerateScript** recebe uma representação dos casos de teste semânticos lidos a partir de um XML (ver figura 39) e gera o arquivo contendo os testes através da classe que implementa **IScriptBuilder** e através de uma *Chain of Responsibility* (Gamma et al., 1980) formada pelas classes que implementam **ICommandHandler**. Cada classe que implementa **ICommandHandler** é responsável pelo tratamento de algum elemento de interface. Esse tratamento é feito pelo método ***handle*** que retorna o código necessário para a automação deste elemento. Esse código é então passado para uma classe que implementa a interface **IScriptBuilder**, que é responsável por

gerar as outras partes código do script de testes, como cabeçalhos de funções e declarações de atributos, entre outros. Nesse trabalho o código é gerado em Java e utilizando a API do Selenium para esta linguagem. Assim, as classes que estendem **ICommandHandler** deve estar de acordo com a classe utilizada para escrever os testes, neste caso **ScriptBuilderJavaSelenium**, para que haja consistência entre o código utilizado para a automação de cada elemento e o automatizador utilizado. Pode-se adicionar tratamentos a diferentes elementos simplesmente fornecendo uma nova classe que implemente **ICommandHandler**. Essas classes são lidas de um arquivo de propriedades e instanciadas dinamicamente pela ferramenta.