# 4 Ferramentas

Neste capítulo serão apresentadas as três ferramentas construídas para auxiliar o processo de teste, são elas: o editor da tabela de decisão, o gerador dos casos de teste e o gerador de scripts de teste. Cada ferramenta é apresentada em uma seção deste capítulo e cada seção apresenta as funcionalidades e a arquitetura da ferramenta.

## 4.1.Editor de Tabela de Decisão

Como descrito anteriormente, uma tabela de decisão é uma ferramenta utilizada direta ou indiretamente em diversas técnicas de geração de casos de testes. Por essa razão, esta será utilizada como ferramenta auxiliar para a geração tanto dos casos de teste quanto dos dados para esses testes.

Uma tabela de decisão é preenchida utilizando como base a especificação do artefato ou funcionalidade. A geração automática da tabela de decisão a partir da especificação do artefato não está dentro do escopo desse trabalho. Nesse trabalho o preenchimento da tabela será realizado manualmente.

Essa ferramenta tem como objetivo realizar o preenchimento de uma tabela de decisão e também atribuir informações adicionais aos campos. O *output* dessa ferramenta será um arquivo XML com uma tabela de decisão tipada, ou seja, uma tabela de decisão com informações adicionais sobre os tipos de dados dos campos.

### 4.1.1.Processo do Editor

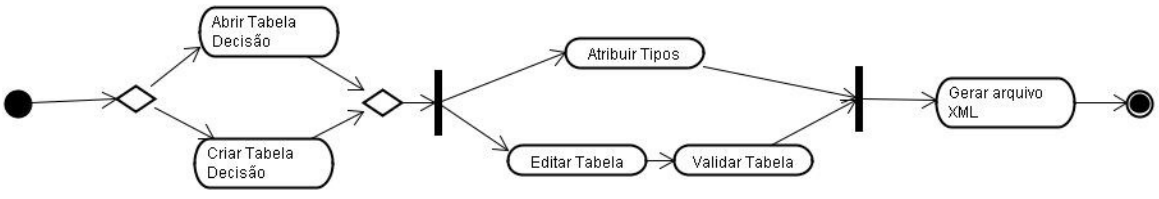


Figura 5 – Interface processo do editor

A figura 5 apresenta um diagrama de atividades que mostra o funcionamento da ferramenta. Desse diagrama podemos destacar quatro atividades: Editar Tabela, Atribuir Tipos, Validar Tabela e Gerar arquivo XML.

#### 4.1.1.1.Edição da Tabela

Uma tabela de decisão, como explicada anteriormente, possui três conceitos principais: Ação, Condição e Regra.

Na ferramenta desenvolvida é possível adicionar e remover cada um desses conceitos. Para realizar a adição de uma condição é necessário selecionar a opção **`Adicionar Condição`** dentro do menu **`Operações com Tabela`** que irá acrescentar uma linha dentro da seção de condições na tabela criada. Para adicionar uma regra, seleciona a opção **`Adicionar Regra`** contido dentro do menu **`Operação de Tabela`** e uma coluna será acrescentada na tabela. Para inserir uma ação na tabela, é só selecionar a opção **`Adicionar Ação`** e numa linha é inserida na seção das ações. Para associar uma regra a uma ação é só preencher o campo correspondente a interseção deles com o caractere **`X`**. Cada uma das células correspondentes a uma condição só pode preenchida com um dos seguintes valores: V (verdadeiro), F (falso) e – (indiferente). A figura 6 mostra um exemplo de uma tabela de decisão completa criada dentro da ferramenta.

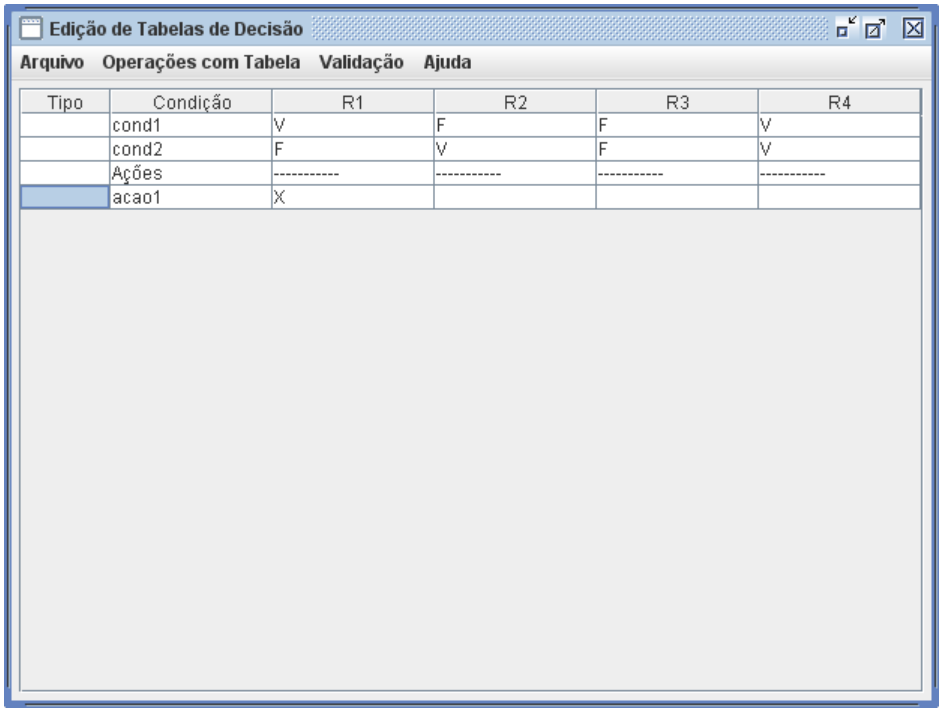


Figura 6 – Tabela de decisão completa.

A condição mencionada anteriormente não possui nenhum tipo de relacionamento. Existem quatro tipos de condições com relacionamento, são eles:

- Condição Mutuamente Exclusiva – deve possuir no máximo uma condição com o valor verdadeiro, sendo que todas as condições podem ser falsas.
- Condição Mutuamente Exclusiva Obrigatória – de possuir uma e exatamente uma condição com o valor verdadeiro.
- Condição de Conjunto Obrigatória – de possuir pelo menos uma condição com o valor verdadeiro, no entanto é permitido que mais de uma condição seja verdadeira.
- Condição com Mascaramento - Para essa opção existem dois conceitos importantes: a condição principal e as condições associadas. A única relação que existe é que se a condição principal for verdadeira, todas as condições associadas também devem ser.

Essas condições são inseridas de uma forma diferente das condições sem relacionamento. A inserção de condições do tipo mutuamente exclusivo, mutuamente exclusivo obrigatório e conjunto obrigatório são realizados através da interface apresentada na figura 7. Essa interface é disponibilizada ao selecionar a

opção **Adicionar Condições Especiais** disponível no menu **Operações com Tabela**. Nessa interface, a primeira lista apresenta todas as condições sem relacionamento já inseridas na tabela e a segunda lista a apresenta as condições que serão transformadas no tipo selecionado na combo da interface.

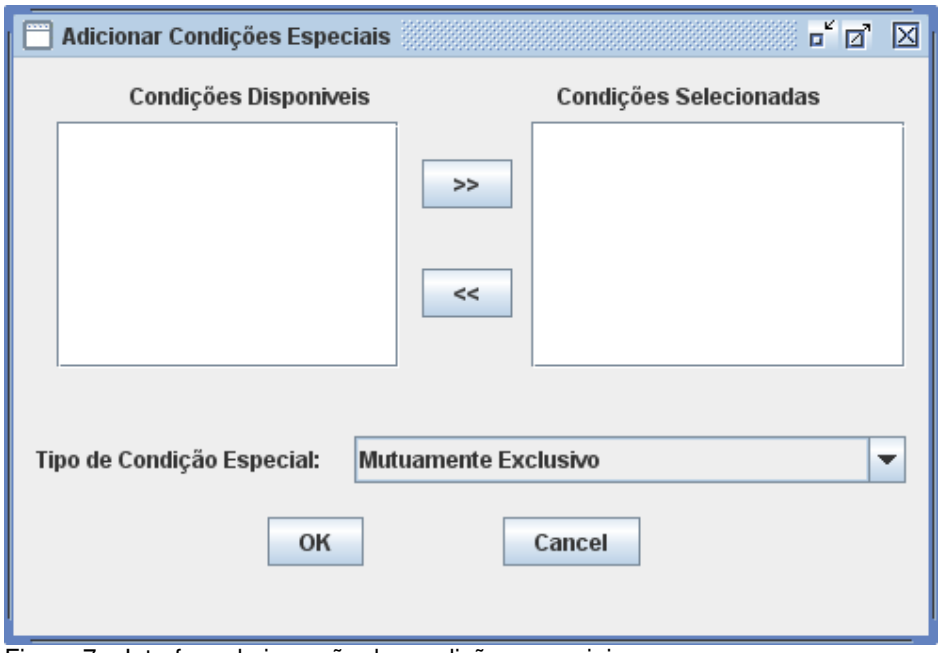


Figura 7 – Interface de inserção de condições especiais

É importante ressaltar que para criar uma condição de algum desses tipos é necessário primeiramente inserir uma condição sem relacionamento na tabela e depois transformá-la em uma condição com relacionamento.

Para selecionar as condições que desejar transformar, é necessário marcar a condição e clique no botão **>>** para copiar a condição para a segunda lista. Selecione o tipo de condição que deseja criar na *combo* **Condicao Especial**. Ao selecionar o botão **OK** todas as condições que estão na segunda lista serão apresentadas na interface principal com o tipo escolhido descriminada na coluna **Tipo** da tabela de decisão. O mesmo processo é feito com condições do tipo mutuamente exclusiva obrigatória e conjunto obrigatório, só que ao apresentar o relacionamento na interface são apresentados os tipos **MEOB** e **OBR**, respectivamente.

Por último, para o tipo de condição com mascaramento é utilizada a interface apresentada na figura 8. Para esse tipo é necessário definir qual é a condição principal, através da seleção de uma condição na lista de condições

disponíveis e clicar no botão **"Adicionar"** e a opção "**Principal**" e as condições associadas através da seleção de uma ou mais condições na lista de condições disponíveis e clicar no botão **"Adicionar"** e a opção **"Associadas"**. Para retirar um elemento da lista de condição associada ou uma condição principal é só selecionar a opção **"Remover"** e a opção **"Associados" ou "Principal"** respectivamente.

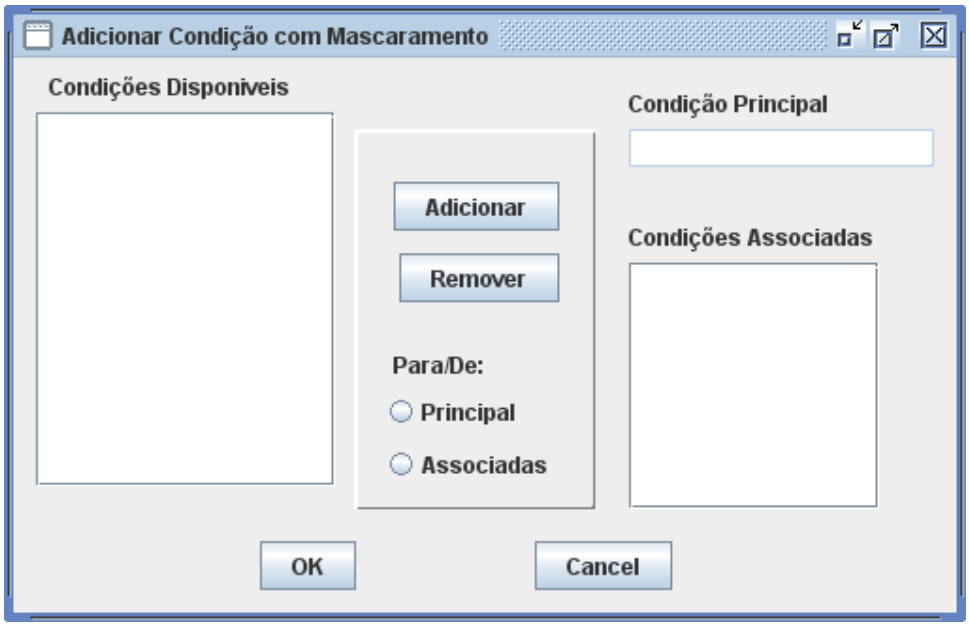


Figura 8 – Interface de inserção de condições com mascaramento.

Para excluir uma condição sem relacionamento ou uma ação é só selecionar a linha e a opção **"Remover Linha",** que está contido dentro do menu "Operações Tabela". Já para excluir uma regra, uma coluna deve ser selecionada e depois a opção **"Remover Regra"**. Para a exclusão das condições com relacionamento, é necessário selecionar a linha que contenha a condição que queira remover e selecionar a opção **"Remover Condição Especial"** ou **"Remover Condição com Mascaramento"**. Será apresentada a interface utilizada para adicionar esse tipo de condição, e nela é possível retirar a condição desejada. Ao realizar essa ação, as condições voltam a ser condições sem relacionamento, permanecendo na tabela de decisão. Todas essas opções descritas estão apresentadas no menu "Operações com Tabela".

Existem algumas considerações sobre a forma de preenchimento da tabela de decisão para que facilite o processo, entre elas podemos destacar:

- A primeira palavra da condição sempre corresponde ao nome do campo.

- A condição que determina o tipo do campo deve ser descrita da seguinte forma: **<nome_campo> é do tipo <nome_do_tipo>.** Esses tipos podem ser: numéricos, string, inteiros ou reais. Quando for utilizado o tipo inteiro ou real, é assumido que o campo é do tipo numérico.
- Quando um campo é obrigatório, a condição é descrita assim: **<nome_campo> é preenchida**
- Quando um campo precisa definir uma gramática a condição deve ser escrita da seguinte forma: **<nome_campo> está no formato**. Esse formato é informado pelo usuário que está preenchendo a tabela de decisão. Por exemplo: endereço está no formato, quando está se tratando do campo endereço.
- Para os componentes *radio Button, check Box*, botão, lista e combo as condições devem ser preenchidas da seguinte forma: **<nome_campo> é selecionado.** Por exemplo: feminino é selecionado, quando está se tratando da opção feminino.
- O preenchimento das ações irá permitir identificar como a mensagem de erro ou sucesso irá aparecer. Quando a mensagem de erro ocorre em uma tela *pop-up,* a ação deve ser preenchida da seguinte forma: **mensagem <texto da mensagem>.** Quando a mensagem de erro aparecer em um rótulo, a ação deve ser preenchida: **label <nome_do_label> <texto da mensagem>.**



#### 4.1.1.2. Validação da Tabela

Para fazer a validação da tabela é necessário selecionar a opção **"Validar Tabela"** dentro do menu **"Validação"**. Na atividade de validação da tabela é verificado se a tabela está completa e não ambígua. Para poder realizar esses cálculos, primeiro é necessário ver as particularidades de alguns tipos de condições. As condições que possuem tratamentos especiais são aquelas que possuem relacionamentos uma com as outras já apresentadas anteriormente.

Para essas condições foi necessário fazer uma validação de preenchimento dos valores, para que os relacionamentos fossem respeitados.

Após a criação do relacionamento entre as condições é possível fazer a validação de não ambigüidade e completeza da tabela. Para a validação de não ambigüidade, foram feitos duas verificações: se a quantidade de colunas é maior do que $2^n$ onde n é o número de condições e se a utilização do indiferente causa a igualdade de duas ou mais regras.

Para a validação de completeza, é necessário fazer um cálculo pelo número de condições e outro cálculo pelo número de regras. Para a contagem do número de regras, é necessário percorrer todas as colunas e para cada valor indiferente na coluna, o valor dessa coluna é multiplicada por dois. Senão tiver nenhum indiferente, o valor da coluna é um. Depois são somados os valores de todas as colunas. Já para o cálculo pelo número de condições, todas as condições que não possuem nenhum relacionamento (mutuamente exclusivo, mutuamente exclusivo obrigatórios , etc.) tem o valor dois. Para condições mutuamente exclusivas o valor é o número de condições que pertencem ao relacionamento mais 1. Para condições mutuamente exclusivas obrigatórias o valor é o número de condições que pertencem ao relacionamento. Nas condições obrigatórias o valor é $2^m - 1$, onde m é número de condições do relacionamento. Por último, o relacionamento de mascaramento o valor é $2^{m-1} + 1$, onde m é número de condições do relacionamento. Para o resultado final é multiplicado o valor que cada condição. Para a tabela estar completa o valor encontrado pelas condições e pela regra devem dar o mesmo resultado.



#### 4.1.1.3.Atribuir Tipos

Para a atividade de atribuição dos tipos é necessário primeiramente escolher o tipo do componente que um determinado campo assume na interface que está sendo testada. A escolha é feita por intermédio de uma interface em que cada campo descrito na tabela decisão deverá ser definido. Um exemplo da interface responsável por essa atividade é apresentada na figura 9.

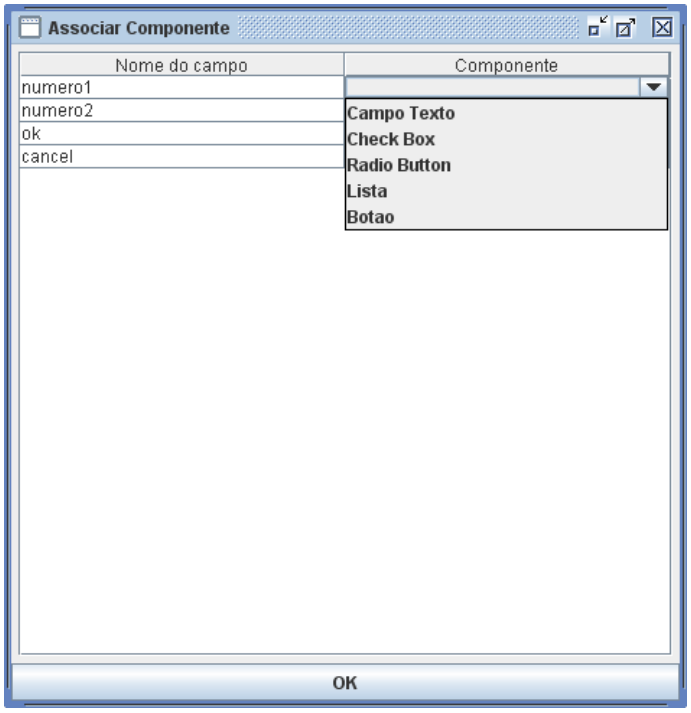


Figura 9 – Interface de escolha do tipo de componente.

Além da informação do tipo de componente, existem outras informações que são adicionadas, mas essas dependem do tipo de componente. Quando o componente é do tipo **`campo texto`**, duas informações adicionais podem ser descritas: o **`tipo`** e a **`regra`**. O atributo **`tipo`** informa qual é o tipo de valor que esse **`campo texto`** pode assumir como, por exemplo, numérico ou string. O atributo **`regra`** somente é preenchido quando o **`campo texto`** assumir valores do tipo string e quando esse campo possuir uma formatação específica. Essa formatação será informada pelo usuário através de uma expressão regular que permita gerar dados válidos para esse campo. A interface que recupera essa informação está descrita na figura 10. Essa interface é apresentada para cada campo que necessite informar uma formatação para o campo.

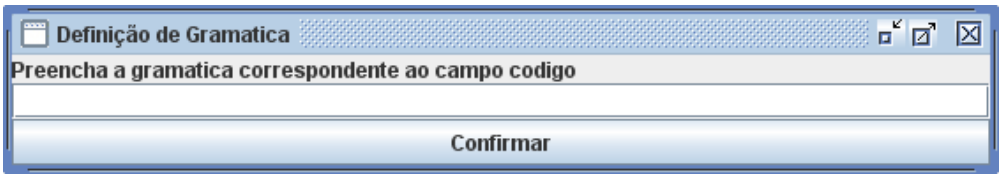


Figura 10 – Interface de definição de uma expressão regular.

Uma expressão regular é "Um método formal de se específicar um padrão de texto" [JARGAS, 2001]. Para ser possível específicar o padrão, existem alguns caracteres especiais. São eles:

- **. (ponto)** – representa um caractere qualquer
- **[...] (lista)** – representa uma lista de caracteres permitidos. Nesse caso podem ser descritos caracteres específicos como **[ABC]** ou um intervalo como **[A-Z]**
- **[^...] (lista negada) –** representam uma lista de caracteres proibidos, que podem ser descritos da mesma forma que a lista.
- **? (opcional)** – representa que o caractere ou lista que antecede esse símbolo pode ser utilizado uma ou nenhuma vez.
- *** (asterisco)** – representa que o caractere ou lista antes desse símbolo pode ser utilizado zero ou mais vezes. Quando esse símbolo é utilizado depois do símbolo ponto (.), pode ser gerado qualquer caractere a quantidade de vezes desejada.
- **+ (mais) –** tem a representação parecida com o asterisco sendo que neste caso, a repetição deve acontecer uma ou mais vezes.
- **{n,m} –** esse símbolo limita a quantidade de vezes que os caracteres podem se repetir. Existem algumas restrições:
  - {3,} significa pelo menos 3
  - {3} significa exatamente 3
  - {,3}- significa de 0 a 3.
- **\ (escape)** - esse símbolo é utilizado para garantir que o caractere que vier depois será o próprio caractere e não será utilizado como um símbolo.
- **| (ou) -** o conjunto de caracteres antes ou o conjunto de caracteres depois podem ser utilizados. O presente gerador limita que esse símbolo só pode ser utilizado dentro de um grupo.
- **(...) (grupo) -** delimita um grupo. A diferença entre um grupo e uma lista é que da lista só é extraído um caractere e do grupo são extraídos todos os caracteres contidos dentro dele.

Será apresentada a seguir uma tabela que contenha expressões regulares e alguns exemplos de string correspondentes a essas expressões.

| Expressão Regular | Strings |
|---|---|
| [a-z]+ | abgec |
| | uv |
| a.a | aba |
| | a{a |
| | aPa |
| [^A-Z]* | +{0ay |
| | a$ |
| | |
| AB? | A |
| | AB |
| AB\\?[A-Z0-9]{1,4} | AB?9AT |
| | AB?ZB |
| | AB?0 |
| (PQR){1,2} | PQR |
| | PQRPQR |
| boa-(noite \| tarde) | boa-noite |
| | boa-tarde |

Tabela 1 – Exemplos de expressões regulares

Os símbolos que representam uma repetição com uma quantidade indefinida de vezes terão essa quantidade definida através da geração de um número aleatório.

É importante ressaltar que existem alguns outros caracteres que podem ser utilizados [JARGAS, 2001] em uma expressão regular, mas não serão considerados já que os caracteres apresentados são suficientes para expressar como um campo deve ser preenchido.

Quando o componente é do tipo lista ou *combo,* existem três atributos que poderão ser descritos: **`regra`**, **`restrição`** e **`seleção múltipla`**. O atributo **`regra`** representa os valores válidos que esse campo pode assumir. O atributo **`restrição`** contém somente os valores que não são válidos e a **`seleção múltipla`** se é possível selecionar mais de um item. Todas essas informações estão descritas em um arquivo de configuração. É obrigatório que a lista de valores válidos esteja descrita no arquivo de configuração. Caso o arquivo de configuração não possua a lista de valores não válidos, o atributo **`restrição`** é deixado sem preencher.

#### 4.1.1.4. Gerar arquivo XML

A última atividade é a geração do arquivo XML, contendo as informações da tabela de decisão. Esse arquivo servirá de entrada para a geração dos casos de teste.

Toda tabela de decisão criada a partir da ferramenta é armazenada em formato XML e cada arquivo XML representa uma única tabela de decisão. O

arquivo XML que armazena a tabela possui uma *tag* para cada conceito importante da tabela de decisão e uma última *tag* para mostrar informações adicionais sobre os campos. Os conceitos são: **`Condição`**, **`Estado`**, **`Ação`** e **`Regra`**. Para estruturar o arquivo XML é criada a *tag* **`TabelaDecisão`**, que contém um ou mais conceitos essenciais. A *tag* **`Condição`** contém nome, índice, número de estados e a lista de estados. Essa *tag* é utilizada para condições sem relacionamento, quando a condição possuir algum relacionamento, a *tag* utilizada no arquivo XML é diferente. A *tag* **`CondicaoME`** é usada para representar uma condição do tipo mutuamente exclusivo, **`CondicaoMEOB`** para condições mutuamente exclusiva obrigatória, **`CondicaoOBR`** para condições de conjunto obrigatório e **`CondicaoMasc`** para condições que possuem mascaramento. A *tag* **`Estado`** é composta pelo valor e o seu índice. A *tag* **`Ação`** é composta do nome e índice. Por último, a *tag* **`Regra`** é composta pelo seu valor. O valor da regra representa qual é o estado de cada condição e quais são as ações associada a essa regra. Para a construção do valor da regra foi estabelecido o seguinte padrão: **`#índice_acao #índice_acao; índice_condicao +índice_estado; índice_condicao2 + índice_estado2`**. A *tag* **`Campo`** possui todas as informações adicionais sobre os campos, que foram informadas na atividade de atribuir tipos. Essa *tag* possui os seguintes atributos: **`nmcampo`**, **`componente`**, **`tipo`**, **`regra`**, **`restrição e seleção_multipla`**. O uso de cada um desses atributos está descrito na seção anterior.

A figura 11 apresenta um arquivo XML com todos os conceitos de uma tabela de decisão.

```
<TabelaDecisaoTipada>
 <name>Teste</name>
 <verificada>true</verificada>
 <condicoes>
  <CondicaoME>
   <nm__condicao>opcao1 é selecionado</nm__condicao>
   <indice>0</indice>
   <indexME>1</indexME>
  </CondicaoME>
  <CondicaoME>
   <nm__condicao>opcao2 é selecionado</nm__condicao>
   <indice>1</indice>
   <indexME>1</indexME>
  </CondicaoME>
  <CondicaoMEOB>
   <nm__condicao>ok é selecionado</nm__condicao>
   <indice>2</indice>
   <indexMEOB>2</indexMEOB>
  </CondicaoMEOB>
  <CondicaoMEOB>
   <nm__condicao>cancel é selecionado</nm__condicao>
   <indice>3</indice>
   <indexMEOB>2</indexMEOB>
  </CondicaoMEOB>
 </condicoes>
 <acoes>
.....
 </acoes>
 <regras>
.....
 </regras>
 <campos>
.......
 </campos>
</TabelaDecisaoTipada>
```

Figura 11 – Arquivo XML de uma tabela de decisão tipada

A figura 12 mostra um arquivo XML que contém uma condição que possui um relacionamento.

```
<TabelaDecisaoTipada>
 <name>Teste</name>
 <verificada>true</verificada>
 <condicoes>
  <CondicaoME>
   <nm__condicao>opcao1 é selecionado</nm__condicao>
   <indice>0</indice>
   <indexME>1</indexME>
  </CondicaoME>
  <CondicaoME>
   <nm__condicao>opcao2 é selecionado</nm__condicao>
   <indice>1</indice>
   <indexME>1</indexME>
  </CondicaoME>
  <CondicaoMEOB>
   <nm__condicao>ok é selecionado</nm__condicao>
   <indice>2</indice>
   <indexMEOB>2</indexMEOB>
  </CondicaoMEOB>
  <CondicaoMEOB>
   <nm__condicao>cancel é selecionado</nm__condicao>
   <indice>3</indice>
   <indexMEOB>2</indexMEOB>
  </CondicaoMEOB>
 </condicoes>
 <acoes>
.....
 </acoes>
 <regras>
.....
 </regras>
 <campos>
.......
 </campos>
</TabelaDecisaoTipada>
```

Figura 12 – Arquivo XML com tabela de decisão com condição com relacionamento

Por fim, a figura 13 mostra as informações adicionais dos campos através da *tag* **`Campo`**.

```
<TabelaDecisaoTipada>
 <name>Teste</name>
 <verificada>true</verificada>
 <condicoes>
  .....
 </condicoes>
 <acoes>
   .......
 </acoes>
 <regras>
  ....
 </regras>
 <campos>
   <Campo>
    <nmcampo>lista1</nmcampo>
    <componente>Lista</componente>
    <regra>{Joao,Maria,Joaquim}</regra>
    <restricao>{Pedro, Henrique, Alfredo}</restricao>
    <selecao__multipla>true</selecao__multipla>
   </Campo>
   <Campo>
    <nmcampo>ok</nmcampo>
    <componente>Botao</componente>
   </Campo>
   <Campo>
    <nmcampo>cancel</nmcampo>
    <componente>Botao</componente>
   </Campo>
 </campos>
</TabelaDecisaoTipada>
```

Figura 13 – Tabela de decisão com a definição dos campos

### 4.1.2.Implementação

A ferramenta foi desenvolvida utilizando o padrão *Model*/*View*/*Control* (MVC). A figura 14 mostra um diagrama de pacotes com a organização dos pacotes e seus sub-pacotes. A posterior cada um dos pacotes serão detalhados.

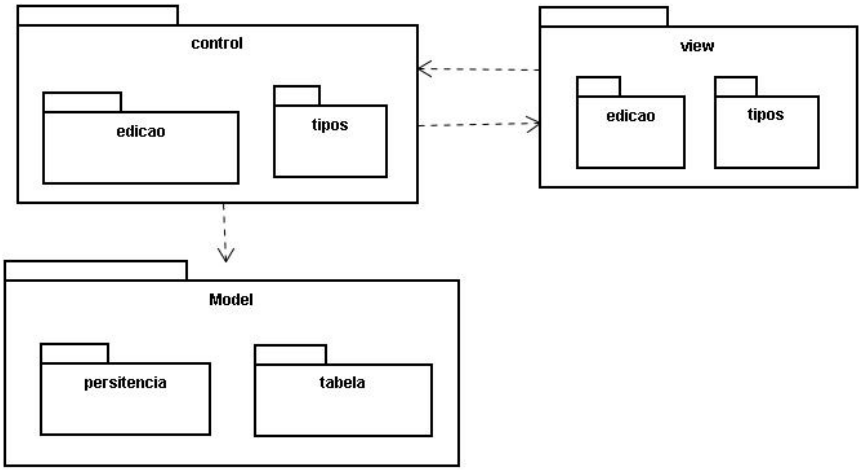


Figura 14 – Diagrama de pacotes do ETD

O pacote *model* contém duas subdivisões: **`tabela`** e **`persistência`**, como apresentado na figura 15.

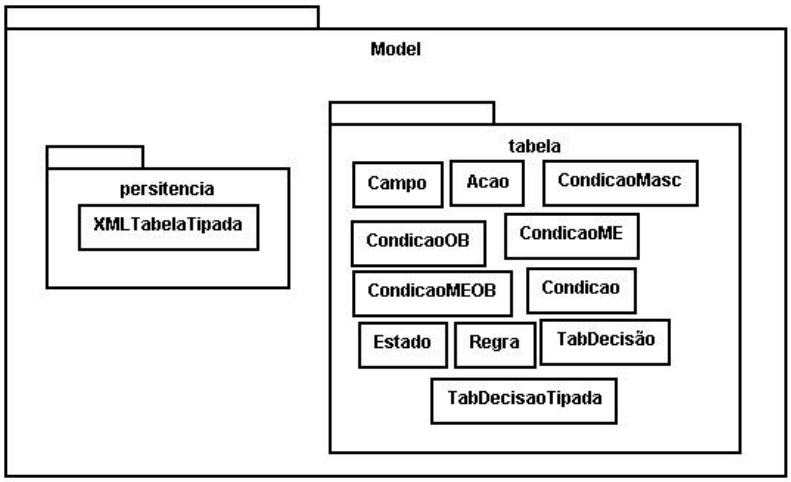


Figura 15 – Diagrama de pacotes do pacote model

Um diagrama de classes do pacote **`tabela`** está apresentado na figura 16. A classe **`TabDecisão`** descreve uma tabela de decisão com seus conceitos básicos: condição, ação e regra. Como ainda tinha as informações sobre os campos, foi criada a classe **`TabDecisãoTipada`** que estende a classe tabela de decisão só que possui a referência para a classe **`Campo`**. Cada um dos conceitos básicos da tabela foi desenvolvido como uma classe que possui no mínimo um atributo que é o nome daquele conceito. No caso da **`regra`**, o valor que é armazenado na variável nm_regra é o padrão mostrado na seção anterior. A classe **`TabDecisãoTipada`** que representa uma tabela de decisão tipada usa o padrão de projeto **`Singleton`** que garante que só terá uma instância dessa classe. Esse padrão de projeto foi

utilizado porque só pode ser tratada uma tabela de decisão a cada momento. Cada tipo de condição foi descrita como uma classe filha da classe **`Condição`** sendo que a única diferença entre elas é como as outras classes as tratam. Como cada condição possui um estado, foi criada uma classe que representasse isso que se relaciona com a classe **`Condição`**. A classe **`Estado`** possui uma lista que contém os possíveis valores que um estado pode assumir, são eles: verdadeiro, falso e indiferente. A classe **`Campo`** representa as informações adicionais de tipo dos campos que compõem a tabela de decisão.

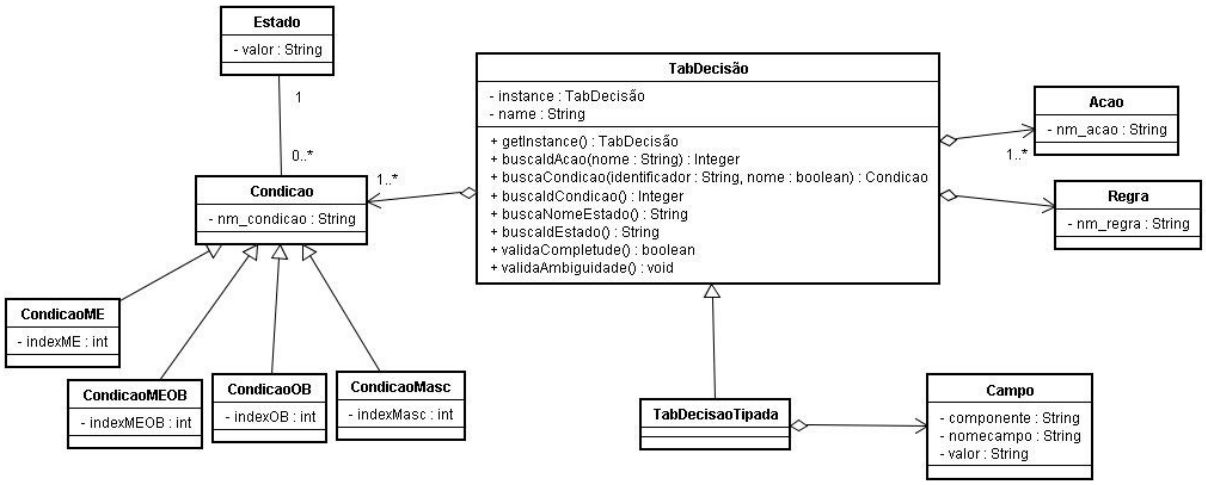


Figura 16 – Diagrama de classes do pacote model.

No pacote **`persistência`** ficam as classes que fazem o armazenamento das informações. A classe **`XMLTabelaTipada`** é responsável por armazenar a tabela de decisão tipada em arquivos XML. Para isso é utilizada a biblioteca **`xstream`**.

A figura 17 apresenta um diagrama de pacote com todas as classes que pertencem ao pacote *control.*

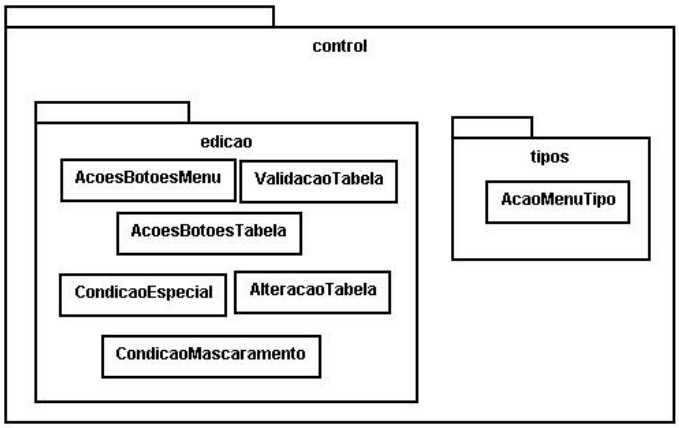


Figura 17 – Diagrama de pacotes do pacote control

A figura 18 apresenta o diagrama do pacote *view*.

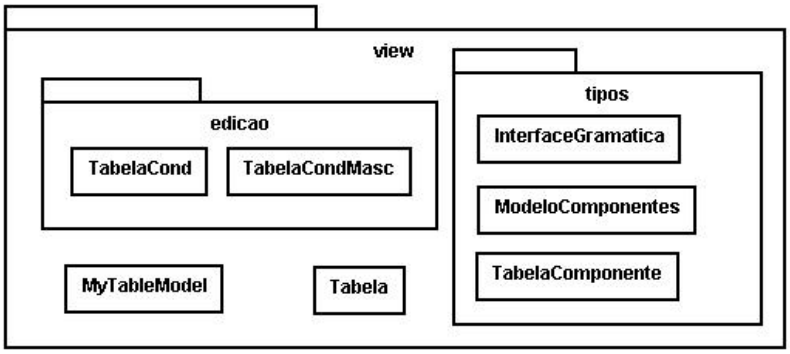


Figura 18 – Diagrama de pacotes do pacote view

A figura 19 apresenta o diagrama de classes onde são mostrados os elementos pertencentes ao pacote *control* na cor cinza claro, os elementos pertencentes ao pacote *view* na cor branco e os elementos pertencentes ao pacote *model.persistencia* na cor cinza escuro.

As classes que pertencem ao pacote *view* têm o intuito de criar interfaces para que o usuário consiga interagir com a ferramenta. As imagens das interfaces estão apresentadas na seção anterior. As classes **`Tabela`**, **`TabelaComponente`** e **`InterfaceGramatica`**, **`TabelaCond`** e **`TabelaCondMasc`** são classes que estendem a classe **`JFrame`** para apresentar as interfaces para os usuários, sendo que a primeira apresenta a tabela de decisão, a segunda possibilita ao usuário escolher qual o tipo de componente cada campo possui, a terceira permite definir uma gramática para um campo e as duas últimas são responsáveis pela inclusão de condições especiais e de condições de mascaramento respectivamente.

O pacote *control* contém as classes em que foram implementadas as funcionalidades. A classe **`AcoesMenu`** e **`AcoesBotoesTabela`** implementam a interface **`ActionListener`** que permite que, ao selecionar um botão em qualquer menu da interface principal, a correspondente classe será acionada. A classe **`AcoesBotoesTabela`** é responsável por disponibilizar a interface de seleção de componentes através do método **`gerarComponentes()`** e por gerar o arquivo XML com a tabela de decisão tipada através do método **`gerarXML()`**.

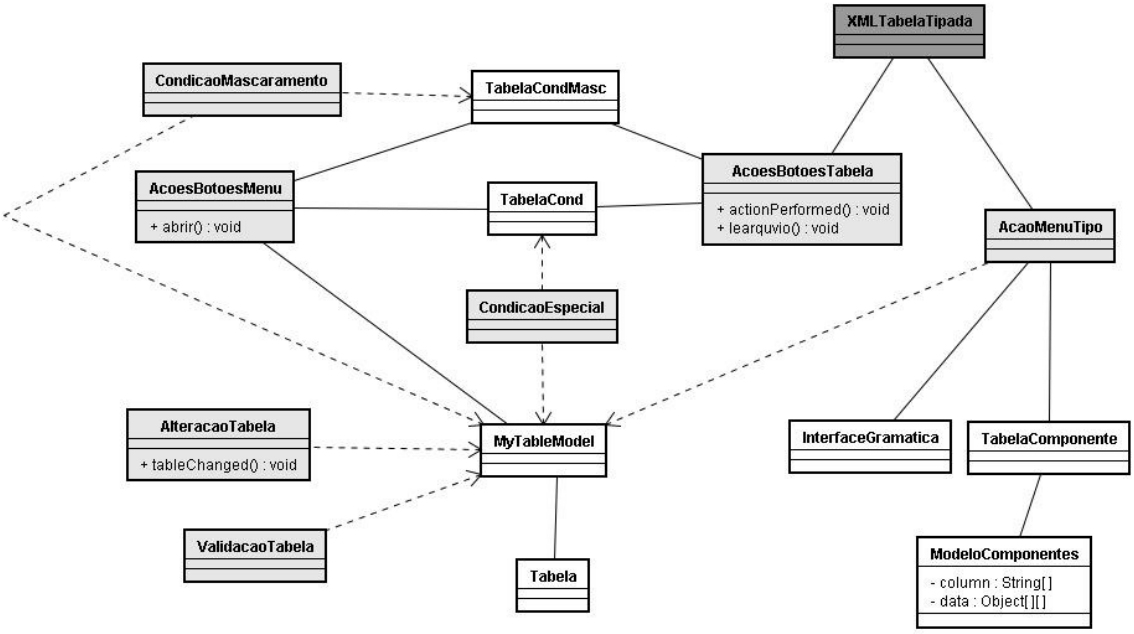


Figura 19 – Diagrama de classe de *control* e *interface*

## 4.2.Gerador de Casos de Teste

Para gerar os casos de teste, é necessário ter como input o arquivo XML contendo uma tabela de decisão tipada. A ferramenta de geração de casos de teste tem como objetivo gerar um arquivo XML com a massa de testes. Essa ferramenta não possui nenhuma interação com o usuário. O gerador foi desenvolvido utilizando a linguagem Java.

### 4.2.1. Processo do Gerador


Figura 20 - Processo do Gerador de casos de teste

A figura 20 apresenta um diagrama de atividades que mostra o funcionamento do gerador. Deste diagrama podemos destacar duas atividades: Gerar Dados e Gerar XML com Casos Teste.

A etapa de geração dos dados irá gerar valores para cada campo em cada caso de teste. A geração dos dados é diferenciada de acordo com o tipo do

componente utilizado na interface e com o tipo de valor do campo. As particularidades de cada uma das gerações serão detalhadas a seguir.

O primeiro componente a ser elaborado foi o campo texto. Este é o único componente que requer dois tipos de geração, cada um de acordo com os tipos de valores que são preenchidos. Os tipos de valores possíveis utilizados nesse trabalho foram *numérico* e *alfanumérico*.

O primeiro caso elaborado foi para o tipo de valor numérico. Para esse, existem dois tipos de condições possíveis: faixa limite e obedecer ao tipo do campo (por exemplo, ser do tipo inteiro). No primeiro tipo é feita a geração aleatória do número levando em consideração a faixa limite descrita. Para encontrar a faixa de limite são lidas todas as condições da tabela de decisão que possuem o valor verdadeiro. Esse procedimento é sempre válido porque quando um campo tem um limite, sempre existem 3 condições mutuamente exclusivas obrigatórias na tabela de decisão referente a esse limite, são elas: campo > X, campo < X e campo = X. Isso garante que não é necessário validar as condições que são falsas. Já para o segundo tipo, é visto que tipo de dado deve ser gerado, por exemplo, se é um valor inteiro ou não. Essa informação está descrita no arquivo XML contendo a tabela de decisão tipada. Para esse trabalho está sendo considerado que, quando um valor não é numérico, ele é uma *string* e vice-versa. Para uma tabela de decisão que contenha uma condição do tipo "o campo x é do tipo inteiro" será gerado para os casos de teste que possuírem essa condição falsa um valor do tipo real.

O segundo caso do componente campo de texto é quando é preenchido utilizando caracteres alfanuméricos. Para esse caso existem duas condições: o tamanho e a formatação deste campo. A primeira condição é tratada igual à faixa de valores do campo numérico, só que neste caso o valor gerado representa a quantidade de caracteres da string. Quando o campo possuir uma formatação específica, será recuperado do arquivo XML com a tabela de decisão tipada o atributo **`regra`** que possui a expressão regular que determina este campo.

Os componentes do tipo *radio Button* e *check Box* são compostos de diversas opções, sendo que o primeiro permite a seleção de apenas uma opção e o segundo de uma ou mais opções. A geração de dados para esses componentes só está associada à seleção ou não de uma determinada opção. Cada opção é descrita como uma condição na tabela de decisão.

Os componentes do tipo lista e *combo* são parecidos com os componentes anteriores já que só permitem a seleção de elementos. A diferença entre esses componentes e o *radio Button e check Box* é que para esses componentes é necessário saber valores possíveis a serem selecionados. Para isso é recuperado do arquivo de entrada os valores que esse componente pode e não pode assumir, através dos atributos **`regra`** e **`restrição`** respectivamente. Para cada teste que possuir a condição com esse componente marcada verdadeira, é gerado um índice aleatório no intervalo do tamanho da lista. Para criar o caso de teste valorado será o utilizado o valor que estiver nessa posição da lista recuperada do arquivo de XML de entrada. É obrigatório que a lista de valores válidos esteja no arquivo de entrada. Para os casos de teste que possuírem essa condição como falso, será recuperado um índice da lista de valores não válidos, e esse índice é gerado aleatoriamente. Caso não possua essa lista o elemento é deixado sem preencher. Caso o campo permita a seleção de mais de um elemento, expressa através do atributo seleção múltipla, para cada caso de teste, será gerado um número aleatório para ver a quantidade de elementos que serão selecionados.

A segunda atividade, pega os valores gerados e cria um arquivo XML contendo a massa de teste necessário para realizar cada caso de teste apresentado na tabela de decisão.

A estrutura do arquivo XML gerado é sempre a mesma, independente dos tipos de dados que foram gerados. A figura 21 mostra um caso de teste descrito no arquivo XML.

```
<Dados>
 <casoteste>
  <CasoTeste>
   <campos>
    <Campo>
     <componente>Campo Texto</componente>
     <nomecampo>numero1</nomecampo>
     <valor>1</valor>
    </Campo>
    <Campo>
     <componente>Campo Texto</componente>
     <nomecampo>numero2</nomecampo>
     <valor></valor>
    </Campo>
    <Campo>
     <componente>Botao</componente>
     <nomecampo>ok</nomecampo>
     <valor></valor>
    </Campo>
   </campos>
   <oraculos>
    <Oraculo>
     <nome>mensagem "Numero1 invalido"</nome>
    </Oraculo>
   </oraculos>
  </CasoTeste>
```

Figura 21 – XML dos dados gerados.

Esse XML possui as seguintes *tags* principais:

- **`Caso de Teste`**: que representa o inicio e fim de cada caso de teste
- **`Campo:`** contém os dados para um determinado campo da interface. Os dados são:
  - **`Componente:`** contém o tipo de componente utilizado na interface
  - **`NomeCampo:`**é o nome do campo. Essa informação é muito importante para possibilitar a geração automática do script de teste e ela é recuperada dos nomes das condições da tabela de decisão.
  - **`Valor:`** o dado gerado para esse determinado campo.
- **`Oráculo:`** é o resultado esperado do teste.

### 4.2.2.Implementação

A ferramenta foi desenvolvida utilizando o padrão *Model*/*View*/*Control* (MVC). Serão apresentados os diagramas de classe separadamente para cada pacote desse padrão.

O pacote *model* contém três subdivisões: tabela, dados e persistência. O pacote **`tabela`** possui a mesma estrutura que foi utilizada para representar uma tabela de decisão tipada utilizada no editor de tabela de decisão. Já no pacote de **`dados`**, existe uma classe para cada conceito importante para a geração dos dados, que são: **`CasoTeste, Dados, Campo e Oraculo.`** A classe **`Dados`** é responsável por armazenar todos os casos de testes necessários para a tabela de decisão apresentada. Um objeto da classe **`CasoTeste`** representa um caso de teste, ou seja, uma coluna da tabela de decisão que é composta por um conjunto de campos e um conjunto de oráculos. A classe **`Oraculo`** possui apenas um atributo nome para armazenar qual a ação que aquele determinado caso de teste deve realizar. Por último, a classe **`Campo`** possui três atributos importantes: **`componente, nomecampo e valor`**. O atributo **`componente`** armazena o tipo de elemento utilizado na interface para esse campo especifico, e ele pode ser: campo texto, check box, radio button, lista, *combo box* e botão. O atributo **`valor`** possui o dado que foi gerado aleatoriamente. A figura 22 mostra o relacionamento entre essas classes.

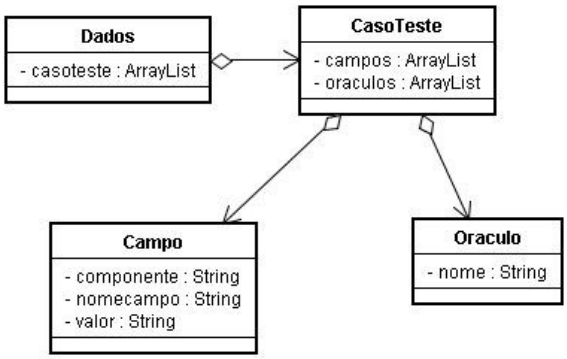


Figura 22 – Diagrama de classe de domínio

A figura 23 apresenta o diagrama de classes em que são mostrados os elementos pertencentes ao pacote *control* na cor cinza e os elementos pertencentes ao pacote *model.persistencia* na cor branca.

No pacote *model.persistencia* ficam as classes que fazem o armazenamento das informações. As classes **`XMLTabela`** e **`XMLDados`** são responsáveis por armazenar a tabela de decisão tipada e os casos de teste respectivamente em arquivos XML. Para isso é utilizada a biblioteca **`xstream`**.

O pacote *control* contém as classes em que foram implementadas as funcionalidades. A classe **`GeracaoDados`** é responsável por gerar o arquivo XML com os casos de teste através do método **`gerarXML()`**. O método **`gerarXML()`** utiliza o método **`geraAleatorio()`** da classe **`GeracaoGramatica`** para gerar um número aleatório de acordo com as restrições já recuperadas da tabela de decisão e esse método usa a biblioteca **`Random`** para possibilitar essa geração. O método **`gerarXML()`** também utiliza o método **`geraGramatica`**() para gerar valores possíveis para uma determinada expressão regular.

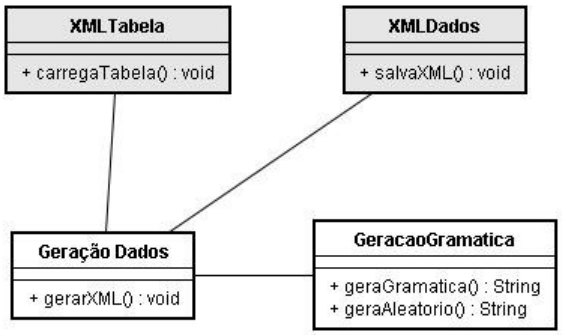


Figura 23 – Diagrama de classe de *control*

## 4.3.Gerador de Scripts de Teste

Para a geração dos scripts, é necessário dispor do arquivo XML com os casos de teste. A ferramenta de geração de scripts tem como objetivo gerar um arquivo Java com os scripts em JUnit. Cada script de teste gerado contém o preenchimento dos campos e o oráculo do teste.

O gerador foi desenvolvido utilizando a linguagem Java.

### 4.3.1.Processo do Gerador

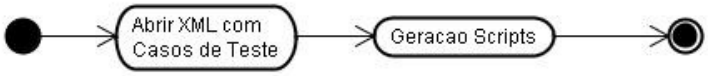


Figura 24 - Processo do Gerador de scripts de teste

A figura 24 apresenta um diagrama de atividades que mostra o funcionamento do gerador. Deste diagrama podemos destacar uma principal atividade que é gerar scripts teste.

O script é dividido em duas partes: o ambiente dos testes e os testes propriamente ditos. A primeira parte é descrita através dos métodos **`setUpOnce(),setUp() e tearDown()`**. Dentro do método **`setUp()`** a linha que não é possível ser gerada automaticamente é a instanciação da interface a ser testada. A figura 25 mostra a primeira parte de um script. O nome do arquivo java que contém o script é fornecido pelo testador e este também é o nome da classe, no exemplo apresentado o nome do arquivo java seria **`MassaTesteNumerico`**

```
public class MassaTesteNumerico {
private FrameFixture window;

 @Before public void setUpOnce() {
FailOnThreadViolationRepaintManager.install();  }

 @Before public void setUp() {
 /*instanciar a interface a ser testada */
 window = new FrameFixture(/*nome do obejto da interface*/)
window.show(); /* shows the frame to test */}

 @After public void tearDown() {    window.cleanUp();  }
```

Figura 25 – Parte inicial do script gerado

Na segunda parte do script, é gerado um método para cada caso de teste descrito no arquivo XML de entrada. Em cada um desses métodos são geradas as ações, que correspondem ao preenchimento dos campos com os dados gerados e a seleção de botões, e o resultado esperado do teste, ou seja, o oráculo. O oráculo é representado através da apresentação de uma mensagem de erro ou de sucesso. Esses métodos já estão prontos para serem executados pelo teste utilizando o *framework* JUnit.

```
@Test public void caso_teste_1(){
window.textbox("numero1").enterText("1");
window.textbox("numero2").enterText("");
window.button("ok").click();
window.optionPane().requireMessage("Numero1 invalido");
}

@Test public void caso_teste_2(){
window.textbox("numero1").enterText("0");
window.textbox("numero2").enterText("");
window.button("ok").click();
window.optionPane().requireMessage("Numero1 invalido");
}
```

Figura 26 – Script gerado com os casos de teste.

Como essa ferramenta só trata da conversão do arquivo XML com os casos de teste para um tipo de script de teste automatizado específico, a implementação foi feita apenas em uma classe que faz a leitura do arquivo e gera um arquivo Java.