Manual do gerador de aplicação Captor

Sumário

1 Introdução

Os geradores de aplicação são ferramentas projetadas para receber uma especificação de software, validar essa especificação e gerar artefatos, tais como: código-fonte, casos de teste, diagramas UML e outros tipos de documentos (Czarnecki e Eisenercker, 2002). O processo de uso de geradores de aplicação é apresentado na Figura 1.

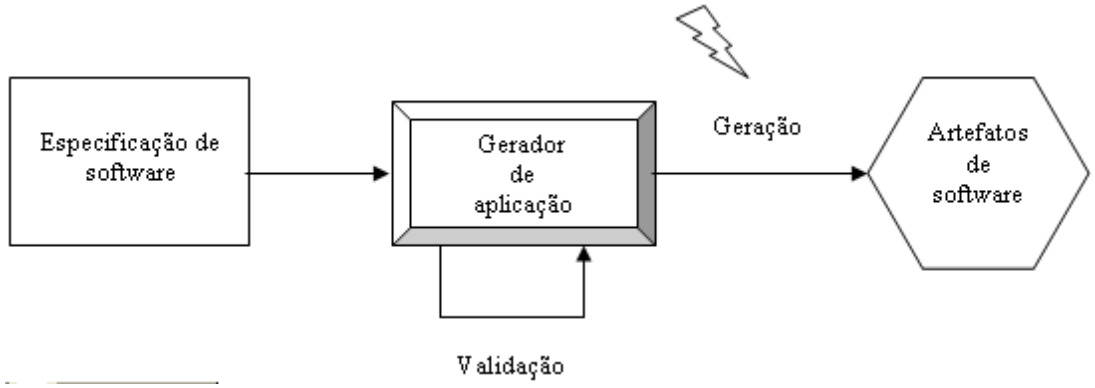

Figura 1: Utilização de geradores de aplicação

1.1 A ferramenta Captor

A ferramenta Captor é um gerador de aplicação configurável, ou seja, ela pode ser configurada para receber especificações de diferentes domínios e gerar artefatos de software para diversas aplicações nesse domínio.

Na Figura 2 ilustra-se o processo de configuração e utilização da ferramenta Captor para as instâncias I1 e I2. A Instância I1 pode receber especificações para o domínio T e pode produzir artefatos para as aplicações T1, T2 e Tn. A Instância I2 pode receber especificações para o domínio P e pode produzir artefatos para as aplicações P1, P2 e Pn.

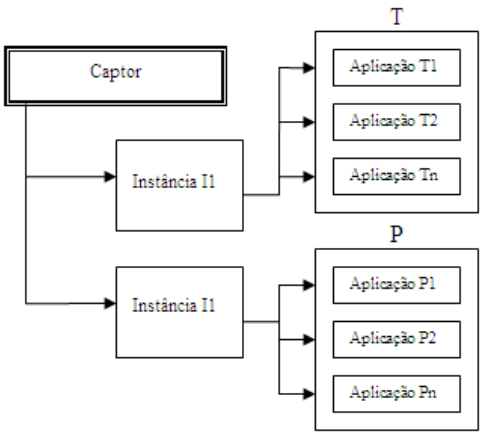

Figura 2: Processo de utilização da ferramenta Captor

Conforme ilustrado na Figura 2, a ferramenta Captor precisa ser configurada antes de ser utilizada para gerar artefatos em um domínio específico. O usuário que interage com a ferramenta durante a etapa de configuração é chamado engenheiro de domínio e o usuário que interage com a ferramenta durante a etapa de utilização é chamado engenheiro de aplicação.

1.2 Terminologia utilizada

Na Tabela 1 é apresentada a terminologia utilizada neste manual.

Tabela 1: Terminologia

Termo	**Descrição**
Gerador de aplicação	Uma ferramenta que utiliza uma especificação, valida essa especificação e produz artefatos de software.
Arquivos fonte	Arquivos que são utilizados por um gerador para armazenar a especificação de uma aplicação.
Arquivos de saída	Arquivos gerados por um gerador de aplicação.
Arquivos de desenvolvimento	Arquivos que o engenheiro de aplicação utiliza para construir aplicações de software. Esses arquivos podem ser gerados por uma ferramenta ou podem ser editados manualmente.

2 Configuração da ferramenta

A realização do processo de configuração tem o objetivo de fazer que a ferramenta seja capaz de aceitar uma especificação, realize diversas validações para assegurar que essa especificação está correta e gere artefatos de software.

A ferramenta Captor pode ser configurada para um domínio por meio de um conjunto de arquivos. Na Figura 3 são apresentados os arquivos necessários para a configuração da ferramenta Captor.

O arquivo de configuração principal, explicado em detalhes na Sub-seção 2.1, contém o meta-modelo da linguagem da especificação das aplicações e as regras de validação dessa especificação. Os *templates* XSL, explicados em maior detalhes na Sub-seção 2.2, são documentos de texto que contém marcações especiais que são substituídas pelos dados da especificação da aplicação durante o processo de geração de artefatos. O arquivo de mapeamento de transformação de *templates*, explicado em detalhes na Sub-seção 2.3, contém a informação de quais *templates* devem ser transformados no processo de geração de artefatos.

2.1 Arquivo de configuração principal

Na Figura 4 apresenta-se a interface gráfica da ferramenta Captor.

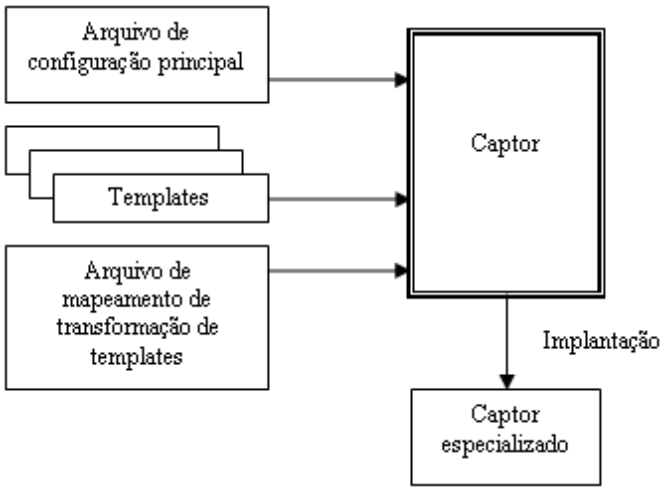

Figura 3: Arquivos necessários para configurar a ferramenta Captor

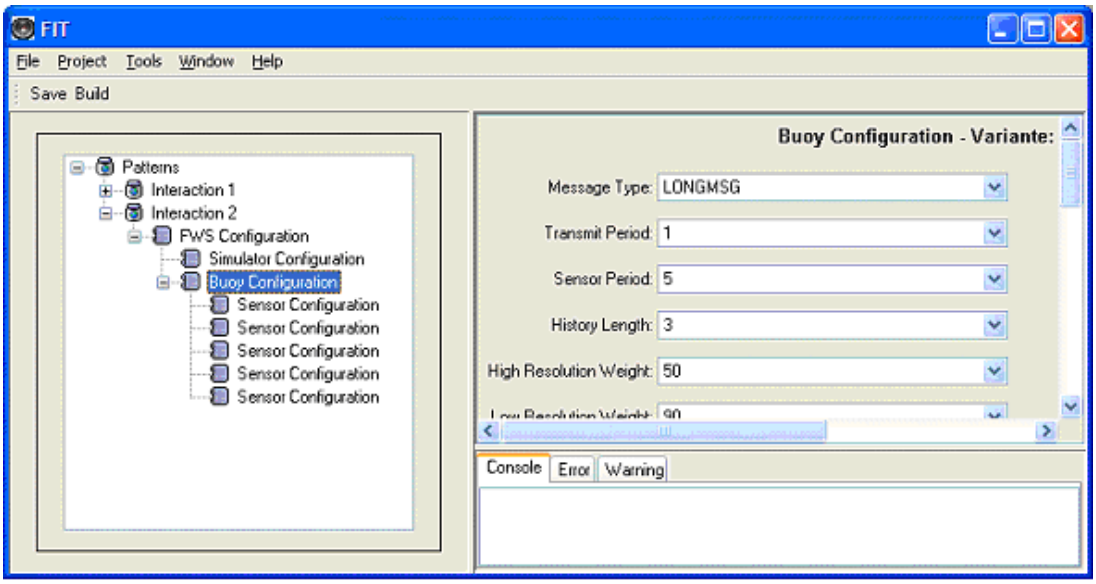

Figura 4: Interface gráfica da ferramenta Captor

A ferramenta pode ser configurada para receber especificações que são armazenadas em formulários. Os formulários são organizados em forma de árvore e podem ser acessados no painel esquerdo da ferramenta. Cada formulário possuí um ou mais componentes gráficos que armazenam os dados da especificação. Esses componentes gráficos podem conter caixas de texto, campos de seleção, tabelas, entre outros e são disponibilizados no painel central-direito da ferramenta.

O arquivo de configuração principal contém o meta-modelo da especificação. Esse meta-modelo descreve quais são os formulários que a ferramenta precisa apresentar, quais componentes gráficos cada formulário contém e quais são as regras de validação que devem ser verificadas nos dados da especificação inseridos nos formulários pelo engenheiro de aplicação.

A ferramenta disponibiliza diversos componentes gráficos para o engenheiro de domínio compor os formulários de maneira apropriada. Se o engenheiro de domínio precisar de um componente que não é parte da biblioteca de componentes de formulário padrão, ele pode utilizar o "Manual de desenvolvimento de componentes de formulário" para criar novos componentes de formulário personalizados. Para realizar a validação, a ferramenta disponibiliza dois mecanismos: o mecanismo de validação sintática e o mecanismo de validação estrutural de formulários. Os mecanismos de validação são apresentados nas Subseções 2.1.2 e 2.1.3 respectivamente.

O primeiro passo na configuração da ferramenta para um domínio específico é a criação do arquivo de configuração principal. Esse arquivo deve ser armazenado no caminho de diretório: /install_dir/domains/nome_do_dominio/nome_do_dominio.domain, onde /install_dir representa o caminho de diretório da instalação da ferramenta e /nome_do_dominio é o nome do domínio da nova configuração.

Na Listagem 1 é apresentado o conteúdo de um arquivo de configuração simplificado que contém o meta-modelo de uma especificação que utiliza dois formulários. O primeiro formulário contém uma caixa de texto e uma caixa de seleção e o segundo formulário contém uma tabela com quatro colunas.

Listing 1: Arquivo de configuração da ferramenta Captor

```
1  <?xml version=''1.0'' encoding=''UTF-8''?>
2
3  <forms>
4
5    <name>SimpleExample</name>
6
7    <form isRoot=''true''>
8      <id>1.1</id>
9      <enabled>true</enabled>
10
11     <name>Form1</name>
12     <variant>Variante1</variant>
13
14     <nextForms>
15       <nextForm>
16         <id>2.*</id>
17         <multiplicity>1</multiplicity>
18       </nextForm>
19     </nextForms>
```

```
21    <formComponents>

23      <formComponent>
          <fullname>
25          captor.windowsystem.formcomponent.textpanel.TextPanel
          </fullname>
27
          <parameters>
29
            <parameter>
31            <name>id</name>
              <value>textId</value>
33          </parameter>
            <parameter>
35            <name>label</name>
              <value>Caixa de texto</value>
37          </parameter>

39        </parameters>

41      </formComponent>

43      <formComponent>
          <fullname>
45    captor.windowsystem.formcomponent.comboboxpanel.ComboBoxPanel
          </fullname>
47
            <parameters>
49
              <parameter>
51              <name>id</name>
                <value>comboId</value>
53            </parameter>
              <parameter>
55              <name>label</name>
                <value>Caixa de selecao</value>
```

```
57              </parameter>
            <parameter>
59              <name>elements</name>
                <value>1:2:3:4:5</value>
61          </parameter>

63        </parameters>

65      </formComponent>

67    </formComponents>

69  </form>

71  <form>
    <id>2.1</id>
73
    <enabled>true</enabled>
75
    <name>Form2</name>
77  <variant>Variante1</variant>

79  <formComponents>

81    <formComponent>
        <fullname>
83        captor.windowsystem.formcomponent.tablepanel.TablePanel
        </fullname>
85
        <parameters>
87
          <parameter>
89          <name>id</name>
            <value>tableId</value>
91        </parameter>
          <parameter>
93          <name>colname1</name>
```

```
            <value>col1</value>
95          </parameter>
            <parameter>
97            <name>colname2</name>
              <value>col2</value>
99          </parameter>

101       </parameters>

103     </formComponent>

105   </formComponents>

107 </form>

109 </forms>
```

O documento é formado por uma marcação “forms” que contém uma marcação “name” e uma ou mais marcações “form”. A marcação “name” contém o nome do projeto. Cada marcação “form” representa um formulário e pode possuir as marcações “id”, “enabled”, “name”, “variant”, “help”, “require”, “nextForms” e “formComponents” que devem ser estruturadas nessa ordem respectivamente. As marcações “require” e “nextForms” são opcionais e as restantes são de uso obrigatório.

O formulário definido na linha 7 possuí o atributo “isRoot=true”. Apenas um formulário pode ter esse atributo com o valor “true”. Esse atributo indica para a ferramenta que esse será o primeiro formulário da especificação.

Todos os formulários devem possuir um identificador único definido na marcação “id” (linhas 8 e 72). O identificar utilizado nos formulários deve estar no formato “NUMERO.NUMERO”. Esse formato é utilizado pela ferramenta no gerenciamento de formulários durante o ciclo de vida do projeto.

A marcação “variant´´ define o nome da variante desse formulário. Os formulários variantes são apresentados na subseção 2.1.1.

A marcação “help” é utilizada para disponibilizar uma breve descrição de ajuda para o usuário engenheiro de aplicação.

A marcação “nextForms” definida na linha 14 indica que o formulário corrente possuí um formulário filho identificado pelo id “2.*” (o asterisco indica que o formulário filho pode conter os valores: 2.1, 2.2, 2.3 e assim sucessivamente). A marcação multiplicidade (linha 17) indica qual o número máximo de filhos que o formulário corrente pode possuir.

A marcação “formComponents” (linha 21) indica quais componentes gráficos o formulário vai conter. Os componentes de formulário são definidos em uma ou mais marcações “formComponent” (linhas 23, 43 e 81) e são parâmetrizados para apresentar comportamento específico. Como exemplo, o componente de formulário definido na linha 21 é implementado pela classe: “captor.windowsystem.formcomponent.textpanel.TextPanel” e é parametrizado por dois valores (linhas 30 e 34). O parâmetro “id” representa um identificador único para esse componente e o parâmetro “label” indica qual será a etiqueta que será disponibilizada ao lado esquerdo do componente gráfico na interface do formulário.

Na Figura 5 é apresentado a interface gráfica da ferramenta Captor configurada com o arquivo apresentado na Listagem 1.

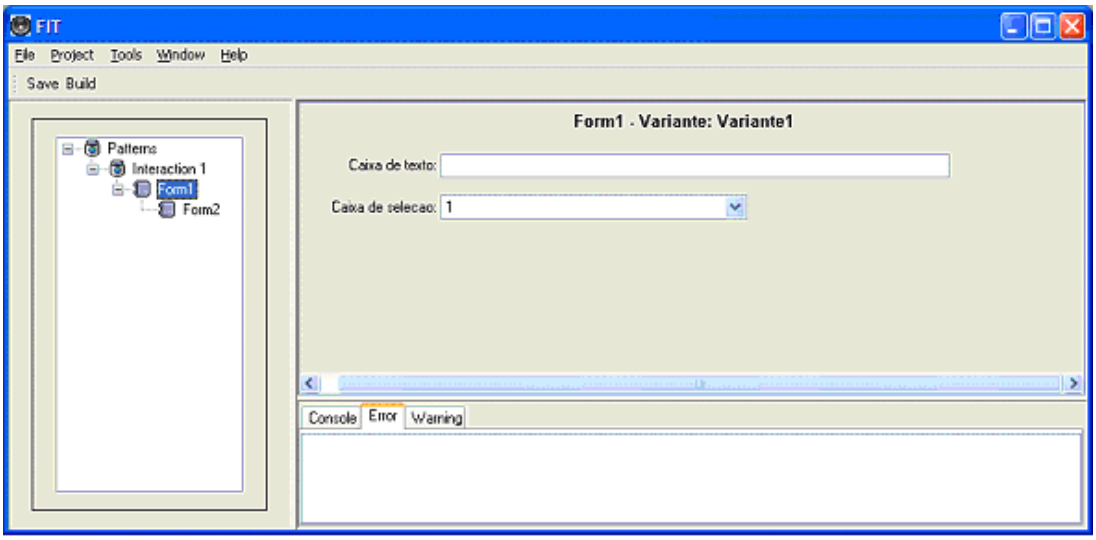

Figura 5: Interface gráfica da ferramenta configurada com os dados da Listagem 1

Os formulários “Form1” e “Form2” podem ser acessados pela árvore de formulários contida no painel esquerdo da ferramenta e os dados dos formulários podem ser editados no painel central superior da ferramenta.

Existem diversos componentes gráficos de formulários disponibilizados junto a ferramenta para compor formulários. Cada componente possuí um nome único e é desenvolvido para receber diversos parâmetros (opcionais e obrigatórios). A documentação desses componentes pode ser encontrada no manual de formulários de componentes.

2.1.1 Variantes

Um formulário pode ter uma variante, ou seja, uma outra versão desse formulário com pequenas modificações. Durante a edição da especificação, o engenheiro de aplicação, baseado nos requisitos de software pode selecionar qual variante ele necessita editar.

A Listagem 2 apresenta o trecho do arquivo de configuração principal que define uma variante para o formulário “Form1” apresentado na Listagem 1.

Listing 2: Definição de variantes

```
1 <form>
    <id>1.2</id>
3
    <enabled>true</enabled>
5
    <name>Form1</name>
7   <variant>Variante2</variant>

9   <extends>1.1</extends>

11  <formComponents>

13    <formComponent>
        <fullname>
15        captor.windowsystem.formcomponent.textpanel.TextPanel
        </fullname>
17    </formComponent>

19  </formComponents>

21 </form>
```

O nome das variantes de um formulário deve ter o mesmo nome do formulário principal. Como exemplo, os formulários identificados por 1.1 e 1.2 possuem o nome “Form1”. Recomenda-se que o identificador dos formulários e seus variantes devem ser iniciados pelo mesmo número, por exemplo, o primeiro variante do formulário com identificador 1.1 deve possuir identificador igual a 1.2 e assim sucessivamente.

Na linha 9 da Listagem 2, a marcação “extends” indica que esse formulário vai estender o formulário 1.1. As marcações abaixo da linha 9 sobre-escrevem as marcações contidas no formulário com identificador igual a 1.1. As variantes são acessíveis em tempo de execução por meio do painel esquerdo da ferramenta.

2.1.2 Validação sintática

Os dados inseridos nos componentes dos formulários podem necessitar validações específcas de domínio. Alguns componentes de um formulário podem possuir preenchimento

obrigatório e outros podem ser opcionais. Alguns componentes devem aceitar que o usuário insira determinados tipos de dados (inteiro, booleanos, expressões regulares) em seus *widgets* gráficos e outros podem aceitar qualquer tipo de valor.

O engenheiro de domínio deve parametrizar os componentes de formulário para que a ferramenta seja capaz de validar a especificação da aplicação de forma apropriada. Como exemplo, se o engenheiro quiser validar o componente "captor.windowsystem.formcomponent.textpanel.TextPanel" da Listagem 1, ele tem as alternativas apresentadas na Listagem 3.

Listing 3: Parâmetros de validação sintática do componente TextPanel

```
1 <parameter>
    <name>use</name>
3   <value>required</value>
  </parameter>
5
  <parameter>
7   <name>regexp</name>
    <value>[0-9]+</value>
9 </parameter>
```

O parâmatro "use" (linha 1) é utilizado para indicar se é necessário algum valor na caixa de texto. Se o valor desse parâmetro for igual a "required", então a ferramenta deve emitir um alerta nos casos em que o usuário não preencher esse campo.

O parâmetro "regexp" (linha 6) indica que o valor inserido nesse campo de texto deve ter o mesmo valor da expressão regular contida na marcação "value". No exemplo da Listagem 3, a expressão regular "[0-9]+" indica que o campo de texto deve possuir um ou mais números inteiros. Qualquer outra combinação de caracteres é decalrada inválida e a ferramenta deve emitir alertas de erros nos casos em que o usuário inserir valores incorretos.

Os parâmetros de validação disponíveis nos diversos componentes de formulários são documentados no manual de componentes de formulário.

2.1.3 Validação estrutural dos formulários

Para obter uma especificação completa, o engenheiro pode definir que é necessário preencher um número mínimo de formulários. No exemplo da Listagem 1, se o engenheiro de domínio definir que a especificação da aplicação necessita do preenchimento dos formulários "Form1" e "Form2", ele pode inserir a marcação apresentada na Listagem 4 dentro da especificação da formulário "Form1":

Listing 4: Validação da estrutura de formulários

```
<require>
  <or>
    <formPath>child(2.*)</formPath>
  </or>
</require>
```

A marcação require define quais são as dependências entre os formulários e deve ser inserida dentro da marcação “form”, entre as marcações “variant” e “nextForms” do arquivo de configuração principal (seria na linha 13 da Listagem 1).

A marcação “require” pode ter uma ou mais marcações “or”. Cada marcação “or” pode ter um ou mais marcações “formPath”. A marcação “formPath” deve conter uma expressão que indica o caminho do formulário corrente até o formulário que deve estar presente para a validação estar completa. Todas as marcações “or” são avaliadas em tempo de execução e pelo menos uma marcação “formPath” deve ser avaliada como verdadeira para que o formulário seja declarado estruturalmente correto. Na tabela 2 são apresentados alguns exemplos de expressões que podem definir a hierarquia dos formulários.

Tabela 2: Validação da estrutura de formulários

child(2.1)	Este formulário deve possuir um formulário filho com id igual a 2.1
parent(1.*)	Este formulário deve possuir um formulário pai com id igual a 1.*
child(2.*)->child(3.*)->child(4.*)	Este formulário deve possuir um formulário filho com id igual a 2.* que deve possuir o filho com id igual a 3.* que deve possuir o filho com id igual a 4.*
child(2.*)-> child(3.*)->parent->(2.*)	Este formulário deve possuir um formulário filho com id igual a 2.* que deve possuir um filho com id igua a 3.* que deve possuir um pai com id igual a 2.* (o último passo foi propositalmente redundante)

2.1.4 Validação do arquivo de configuração principal

Após o término da edição do arquivo de configuração principal, o engenheiro de domínio pode validar o documento com a ferramenta “Meta-model validator”. Essa ferramenta lê o arquivo de configuração principal e caso existam, relata erros e inconsistências. Se o arquivo for validado corretamente pela ferramenta de validação de meta-modelos, o gerador pode ser utilizado com a nova configuração. A ferramenta de validação de meta-modelos está disponível no menu Tools->Meta-Model Validator da janela principal do gerador Captor.

Após a configuração do meta-modelo (formulários e validações), a ferramenta ja é capaz de receber a especificação da aplicação, validar essa especificação, salvar essa especificação em formato XML (esses arquivos são chamados de arquivos-fonte) e recuperar essa especificação dos arquivos XML e disponibiliza-las nos formulários (atividades de criar um projeto, editar o projeto, validar um projeto, fechar o projeto e abrir o projeto de novo).

2.1.5 Geração automática do arquivo de configuração principal

A ferramenta Captor foi configurada para receber a especificação necessária para criar o arquivo de configuração principal e gerar o arquivo de XML com essa especificação.

O usuário deve abrir a ferramenta, selecionar o menu "File->Application Project". Na tela de seleção de domínios o usuário deve escolher "New Captor Project". Nas telas seguintes o usuário deve escolher um diretório para os arquivos fontes e um diretório para os arquivos de saída.

Após o projeto ser criado, o usuário pode criar novas interações, novos formulários e novas variantes. Após a edição dos formulários o usuário pode clicar no botão de acesso rápido "Build". Ao término do processo de transformação de artefatos, a ferramenta invoca a ferramenta Ant para realizar a instalação automática dos novos artefatos.

Após esse processo, o usuário pode testar a nova configuração, selecionando o menu "File->Application project" e na primeira tela do *Wizard*, selecionar o domínio recém criado.

2.2 Arquivos de *templates* XSL

Um *template* XSL é uma folha de estilos que especifica como um documento de XML deve ser transformado em outro documento. O processo de transformação é apresentado na Figura 6.

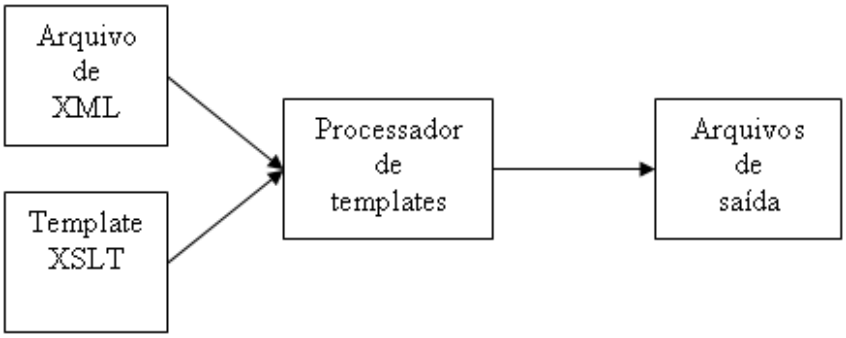

Figura 6: Transformação de *templates* XSL

O processador de *templates*, baseado nas instruções dos *templates*, pode gerar diversos tipos de arquivos de saída, dentre eles estão: arquivos XML, código-fonte, documentos de UML, arquivos de PDF, planilhas e casos de teste.

A ferramenta Captor, armazena as informações inseridas na sua interface gráfica em arquivos no formato XML. Quando o usuário solicita a transformação da especificação em artefatos, a ferramenta recupera os arquivos de XML com o conteúdo da especificação (arquivos-fonte) e seleciona os *templates* necessários para realizar a geração de artefatos (o processo de seleção de *templates* é apresentado na seção 2.3).

A estrutura do XML gerado pela ferramenta Captor a partir dos dados da especificação possuí uma parte da sua estrutura fixa e uma parte variável. A parte fixa do XML gerado pela ferramenta configurada com a Listagem 1 é apresentado na Listagem 5.

Listing 5: Estrutura fixa da especificação armazenaa em XML

```
<?xml version=''1.0'' encoding=''UTF-8''?>

<formsData>

 <project>
   <name>Nome do projeto</name>
 </project>

 <forms>

   <form id=''1.1'' variant=''Default''>

     <data>
       <!--
       Os dados armazenados pelos componentes do
       formulário 1.1 são armazenados nesta linha
       -->
     </data>

     <form id=''2.1'' variant=''Default''>

       <data>
         <!--
         Os dados armazenados pelos componentes do
         formulário 1.2 são armazenados nesta linha
         -->
       </data>
```

```
29          </form>

31        </form>

33      </forms>

35  </formsData>
```

A ferramenta Captor armazena a especificação inserida em sua interface gráfica em uma estrutura de XML que começa com a marcação raiz “data”. Dentro dessa marcação existem duas marcações filhas: “project” e “forms”. A marcação “project” (linha 5) contém o nome do projeto e a marcação “forms” (linha 9) contém os dados da árvore de formulários da especificação. A marcação “forms” contém uma marcação raíz “form” (linha 11). Essa marcação contém uma marcação filho “data” (linha 13) e zero ou mais marcações filhas “forms” (nesse caso, uma marcação na linha 20).

A parte variável da estrutura de diretórios é armazenada nas marcações “data” filhas da marcação “form” (linhas 11 e 20). Dependendo do número e do tipo de componentes de formulário que são utilizados para definir um formulário, o conteúdo dessa marcação pode variar. Como apresentado na Listagem 1, o formulário “Form1” foi configurado para apresentar uma caixa de texto e uma caixa de seleção (linha 21 e 43 da Listagem 1). A estrutura de XML que esses dois componentes armazenam as informações é apresentado na Listagem 6.

Listing 6: Estrutura variável do formulário 1.1 definido na Listagem 1

```
1 <textatt name=``textId''>
    Valor inserido na caixa de texto
3 </textatt>
  <combo name=``comboId''>
5   Valor selecionado da caixa de seleção
  </combo>
```

Essa estrutura de XML gerada pelos componentes do Formulário “1.1” deve ser armazenada na Linha 13 da Listagem 5. Os formulários armazenam os dados de seus componentes em tempo de execução. Alguns componentes permitem que a geração de XML seja parametrizada. No exemplo da Listagem 6, o componente TextPanel e o componente ComboPanel utilizam o parâmetro “id” (linhas 30 e 50 da Listagem 1 respectivamente) para indicar que a ferramenta deve gerar XML personalisado a partir da especificação. Esses parâmetros foram utilizados pelos componentes de formulário para definir o valor do atributo “name” (linhas 1 e 4 da Listagem 6).

Para a obtenção da transformação da especificação em artefatos de software, o engenheiro de domínio que implementa a biblioteca de *templates* deve ter o conhecimento da estrutura do XML gerado pela ferramenta (arquivos-fonte). Após esse conhecimento analisando os arquivos-fonte armazenados em /diretorio_do_projeto/input, o engenheiro de domínio pode iniciar o processo de desenvolvimento dos *templates*. Os *templates* devem ser armazenados em qualquer caminho de diretório abaixo do caminho: /install_dir/domains/nome_do_dominio /templates e devem ser estruturados de acordo com as regras de trasnformação da linguagem XSL.

A linguagen de transformações XSL está fora do escopo desse manual. Instruções detalhadas da linguagem de transformação XSL podem ser encontradas no site: `http://www.w3c.org/xslt` e em diversos manuais e livros disponíveis livremente ou para compra na internet.

2.2.1 Desenvolvimento de *templates* com Zonas de Segurança

Uma zona de segurança (Jack Herrington, 2005) é uma região delimitada por comentários onde o desenvolvedor de aplicações pode inserir código personalisado sem perder as modificações com a re-geração dos artefatos.

As marcações das zonas de segurança devem ser escritas nos templates. Após a geração dos artefatos, o desenvolvedor pode inserir dados manuais entre as marcações da zona de segurança sem perder os dados com a re-geração dos artefatos. Na Listagem 7 é apresentado um exemplo de zona de segurança que pode ser adicionado a qualquer template processado pela ferramenta Captor.

Listing 7: Exemplo de criação de zonas de segurança nos arquivos de *template*

```
// START-SAFE(someSafeZoneId)
2  <xsl:value-of select=''/data/safezone[@id= someSafeZoneId]''/>
// END-SAFE
```

A cadeia de caracteres "someSafeZoneId" deve ser substítuída por um identificador de zonas de segurança único. Esse identificador é utilizado pela ferramenta para recuperar as informações dos arquivos de saída.

Na Listagem 8 é apresentado um exemplo de zona de segurança produzida em um arquivo com código Java gerado pela ferramenta. As modificações manuais devem ser inseridas pelo engenheiro de aplicação abaixo da linha 3 e acima da linha 5.

Listing 8: Exemplo de zonas de segurança em arquivos Java

```
1 public void someMethod() {

3 // START-SAFE(someSafeZoneId)
```

```
  System.out.println(``Hello safe-zone!'');
// END-SAFE

  return;
}
```

A ferramenta Captor analisa o projeto que está sendo gerado e caso existam arquivos com zonas de segurança que devem ser sobre-escritos no novo processo de geração, o conteúdo das zonas de seguraça é extraído dos arquivos que ja foram gerados e é armazenado na árvore de XML junto com a especificação. Durante o processo de transformação dos *templates*, esses dados são inseridos novamente nos arquivos de saída permitindo que o engenheiro de aplicação personalise o código gerado sem perder essas modificações com a re-geração dos artefatos.

2.3 Arquivo de mapeamento da transformação dos *templates*

O arquivo de mapeamento de transformação de *templates* é utilizado pela ferramenta para indicar quais *templates* devem ser utilizados no processo de geração de artefatos. Esse arquivo deve ser armazenado no caminho de diretório /install_dir/domains/nome_do_dominio/rules.xml, deve ser definido em XML e pode possuir diversas marcações para controlar o processo de geração de artefatos. O conjunto de todas as marcações possíveis da ferramenta Captor é chamado de MTL (do inglês *Mapping trasnformation language*)

Na Listagem 9 é apresentado um exemplo simples da estrutura do arquivo de mapeamento para os formulários definidos na Listagem 1.

Listing 9: Arquivo de mapeamento de *templates*

```
<composer name=``FIT''>

  <statements>

    <callTask id=``processForm1''/>
    <callTask id=``processForm2''/>

  </statements>

  <tasks>

    <task id=``processForm1''>

```

```
      <compose>
16      <template>grn.st</template>
        <newFilename>\${/data/project/name}_GRN.st</newFilename>
18    </compose>

20  </task>

22  <task id=''processForm2''>

24    <compose>
        <template>sql.sql</template>
26      <newFilename>sql.sql</newFilename>
      </compose>
28

    </task>
30

  </tasks>
32

</composer>
```

O arquivo de mapeamento de *templates* possuí duas marcações abaixo da marcação raíz "composer": a marcação "statements" e a marcação "tasks".

A marcação "tasks" pode ter uma ou mais marcações "task". A marcação "task" é utilizada para definir a transformação de um arquivo de template em um arquivo de saída. Essa marcação deve possuír uma marcação "compose". Cada marcação "compose" é formada pelo nome do arquivo de *template* utilizado na transformação (linhas 16 e 25) e pelo nome do arquivo de saída que esse *template* transforma (linhas 17 e 26). O caminho de diretório dos arquivos de *template*, devem ser indicados de forma relativa ao diretório de *templates* do domínio (ex: /install_dir/domains/nome_do_dominio/templates). O nome do arquivo que deve ser gerado (marcação "newFileName") pode conter expressões XPATH (XSL, 2005) para gerar nomes de arquivos personalisados. Por exemplo, na linha 17 o nome do arquivo gerado será o nome contido na marcação /data/project/name do arquivo fonte concatenado com a cadeia de caracteres "_GRN.st".

A marcação "statements" pode conter três tipos de marcações: marcação "callTask", "if" ou "for-each". As marcação "callTask" indica para a ferramenta que uma determinada tarefa deve ser executada (chamadas para a marcação "task" nas linhas 13 e 22).

As marcações "if" são utilizadas para realizar assertivas sobre o conteúdo do arquivo fonte que está sendo processado. Na Listagem 10 são apresentados três exemplos de clausulas condicionais.

Listing 10: Cláusulas condicionais no arquivo de mapeamento de *templates*

```
1 <statements>

3   <if test=``exist(/data/forms/form)''>
      <callTask id=``processForm1''/>
5   </if>
    <if test=``equal(/data/forms/form@id='1.1')''>
7     <callTask id=``processForm1''/>
    </if>
9   <if test=``not-equal(/data/forms/form/='1.1')''>
      <callTask id=``processForm2''/>
11  </if>

13 </statements>
```

A primeira clausula da Listagem 10 (linha 3) só é executada se existir o caminho "/data/forms/form" no arquivo-fonte (XML gerado a partir da especificação da aplicação), ou seja, a clausula só é executada se o formulário inicial foi preenchido pelo engenheiro de aplicação. A segunda clausula condicional (linha 6) só é executada se o atributo "id" da marcação "/data/forms/form" for igual a cadeia de caracteres "1.1". A terceira clausula condicional (linha 9) só é executada se o atributo "id" da marcação "/data/forms/form" for diferente da cadeia de caracteres "1.1".

O teste realizado pela função "exists" recebe como argumento uma expressão XPath. O teste realizado pela função "equal" e "not-equal" recebe como parâmtros dois argumentos. Os argumentos podem ser uma expressão XPath ou uma cadeia de caracteres delimitada por aspas simples. Se a função da clausula if que está sendo processada for avaliada como verdadeira, os statemements dentro do if são executadas, caso contrário as tarefas são ignoradas.

As marcações "for-each" são utilizadas para realizar interações durante o processo de geração de artefatos. Na Listagem 11 são apresentados exemplos dessas clausulas.

Listing 11: Cláusulas "for-each" no arquivo de mapeamento de *templates*

```
1 <statements>

3   <for-each select=``/data/forms/form/form''>
```

```
    <callTask id=''processSomeForm''/>
5   </for-each>

7   <for-each select=''/data/forms/form/form''>
    <if test=''equal(/data/current/form/@id,'2.1')''>
9     <callTask id=''processSomeForm''/>
    </if>
11  </for-each>

13 </statements>
```

A marcação na linha 3 da Listagem 11 indica que a ferramenta deve fazer um loop de transfomação em que a chamada callTask da linha 4 é executada o mesmo número de vezes que o número de formulários /data/forms/form/form do arquivo-fonte. Além da interação, essa clausula disponibiliza o nó corrente da árvore de XML do arquivo-fonte de maneira personalisada por meio do caminho "/data/current/". Por exemplo, em cada interação da linha 3 da Listagem 11, a ferramenta realiza uma cópia do nó /data/forms/form/form no caminho /data/current/form. Essa abordagem é utilizada para que o engenheiro de domínio tenha a informação, dentro dos templates e dentro do arquivo de mapeamento, sobre qual nó está sendo processado.

As marcações dentro da marcação "statement" devem obedecer uma ordem pré-determinada. Por exemplo, as marcações "callTask" devem ser definidas nas primeiras posições. Após as marcações "callTask" todas as marcações "ifTask" devem ser definidas. Após as marcações "ifTask" todas as marcações "for-each" devem ser definidas. Essa ordem é a mesma dentro das marcações "ifTask" ou "for-each", ou seja, dentro de uma marcação "ifTask" ou "for-each", os "statements" devem apresentar a mesma ordem (´´callTask", "ifTask" e "for-each").

2.4 Início rápido

Para o iniciar a configuração da ferramenta para um domínio específico, é necessário o planejamento prévio sobre qual vai ser a estrutura da especificação das aplicações nesse domínio. Após essa fase, execute as seguintes tarefas:

1. Faça uma cópia do diretório: /install_dir/domains/Blank para o diretório: /install_dir/domains/foo, onde "foo" é o nome do domínio escolhido.

2. Renomeio o arquivo: /install_dir/domains/foo/Blank_.domain para: /install_dir/domains/foo/foo.domain.

O arquivo /install_dir/domains/foo/foo.domain representa o arquivo de configuração principal e possuí a definição de um formulário com apenas um componente de formulário do tipo caixa de texto.

O arquivo /install_dir/domains/foo/rules.xml é o arquivo de mapeamento de transformações e contém apenas uma regra que transforma o *template*: /install_dir/domains/foo-/template/main.xsl em um arquivo de saída.

Os arquivos /install_dir/foo/pre-build.xml e /install_dir/foo/pos-build.xml contém os esqueletos de um *script* da ferramenta Ant que não realiza nenhuma ação. Esses arquivos podem ser utilizados dessa maneira enquanto não for necessário nem o pré e nem o pós processamento dos arquivos de saída.

Para testar a aplicação, selecione "File->Application Project". Na tela inicial do *wizard* selecione o domínio "foo" e siga as instruções. Após a criação do projeto, clique com o botão direito do mouse em cima do item "Forms" e selecione no menu o item "New Interaction". O primeiro e único formulário deve aparecer no painel central-direito da ferramenta.

Nesse momento o usuário já pode adicionar texto na caixa de textos do primeiro formulário, salvar a aplicação e gerar um arquivo pressionando o botão "Build".

Os arquivos gerados pela ferramenta são: /diretorio_do_projeto/input/interaction_0.fit e /diretorio_de_saida/interaction_0/nome_do_projeto.nome_do_projeto.

O arquivo /diretorio_do_projeto/input/interaction_0.fit contém o arquivo de XML gerado pela ferramenta à partir dos dados da especificação (arquivo-fonte) e o arquivo /diretorio_de_saida/interaction_0/nome_do_projeto.nome_do_projeto é o arquivo de saída gerado pela ferramenta.

Para continuar o processo de configuração são necessários os seguintes passos:

- Coloque em um editor de texto os seguintes arquivos:
    1. /install_dir/domains/foo/foo.domain
    2. /install_dir/domains/foo/rules.xml
    3. /install_dir/domains/foo/pre-build.xml
    4. /install_dir/domains/foo/pos-build.xml
- Alterar os dados do formulário raíz:
    - Alterar as definições do primeiro formulário.
    - Adicionar mais componentes no formulário raíz.
    - Adicionar mais formulários.
    - Adicionar as restrições de validação necessárias.

- Examinar a estrutura dos arquivos-fonte gerados pela ferramenta e realizar o desenvolvimento dos *templates*.
- Alterar o arquivo de mapeamento de *templates*.
- Testar a nova configuração.

Se o usuário desejar gerar automaticamente o arquivo de configuração principal, ele deve criar um novo projeto por meio do menu "File->Application Project" e na tela inicial do *wizard* selecionar "New Captor Project". No nome do projeto, o usuário deve digitar "foo" ("foo" é o nome do domínio escolhido).

Após a criação do projeto o usuário pode criar e editar novos formulários e ao fim do processo clicar no botão "Build". O processo de geração sobre-escreve o arquivo de configuração principal /install_dir/domains/foo/foo.domain pelo arquivo que foi gerado. Os arquivos de mapeamento e de pre e pos processamento não são alterados no processo automático.

3 Utilização da ferramenta

A ferramenta Captor fornece apoio para a criação e edição de projetos. Cada projeto é definido por um nome, um diretório onde a especificação é armazenada e um diretório onde os arquivos de saída são armazenados após a geração dos artefatos.

Na Figura 7 é apresentado a utilização da ferramenta Captor, configurada com o arquivo de configuração principal apresentado na Listagem 1.

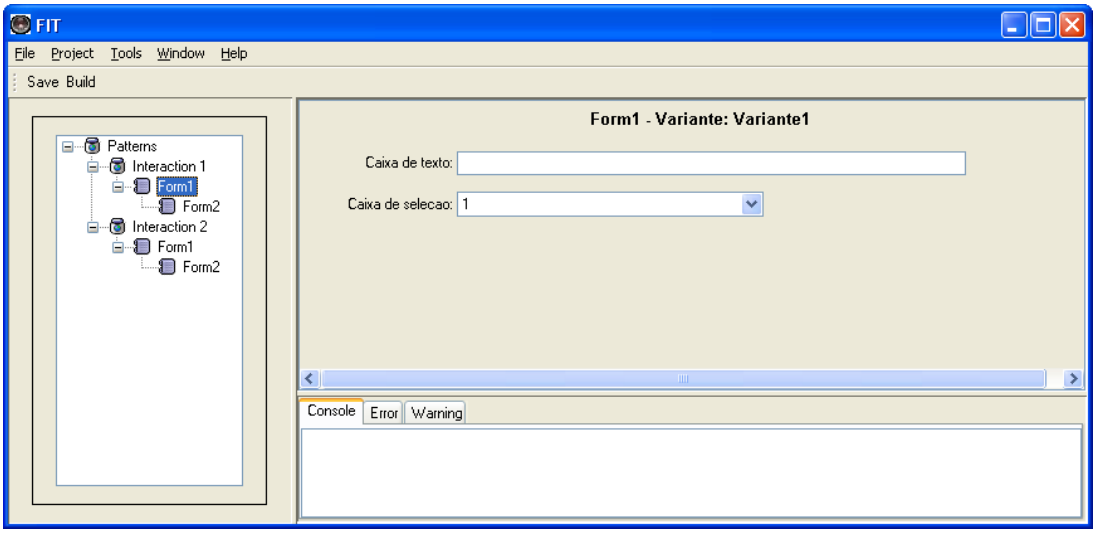

Figura 7: Utilizacão da ferramenta Captor

Em um projeto, o engenheiro de aplicação pode editar uma ou mais especificações por meio de uma ou mais árvores de formulários. O usuário pode editar diversas árvores

de formulários por meio de diversas interações. Cada interação possuí uma árvore de formulários que define a especificação de uma aplicação.

Na Figura 7 é apresentado a interface da ferraenta Captor sendo utilizado para editar duas especificações. O engenheiro de aplicação criou duas interações que podem ser acessadas pelos elementos "Interaction 1" e "Interaction 2" do menu da barra esquerda da ferramenta. Durante a transformação dos artefatos, os arquivos de saída dessas especificações são armazenados no caminho de diretório: /output_dir/interaction_0 e /output_dir/interaction_1, onde /output_dir é o diretório de saída que é definido no momento em que o engenheiro de aplicação cria um projeto na ferramenta Captor.

3.1 Pre e Pos transformação de artefatos

O Ant (Davidson et al., 2005) é uma ferramenta que auxilia na compilação e instalação de projetos Java. Essa ferramenta oferece diversas funcionalidades para a instalação de aplicações Java e não Java, tais como: copiar arquivos, apagar arquivos, mover arquivos, compactar arquivos em diversos formatos, integração com mecanismos de controle de versão, execução de aplicativos externos, compilação automática de aplicativos Java, entre outras funcinalidades.

A ferramenta Captor pode ser configurada para realizar o processamento dos arquivos com a ferramenta Ant antes e depois do processo de transformação de artefatos. Se existir um arquivo do Ant com o nome /install_dir/domains/nome_do_dominio/pre-build.xml o Ant será chamado com esse arquivo como parâmetro antes da ferramenta executar a transformação dos artefatos. Se existir um arquivo do Ant com o nome "/install_dir/domains/nome_do_dominio/pos-build.xml" o Ant será chamado com esse arquivo como parâmetro depois da ferramenta executar a transformação dos artefatos.

Se o engenheiro de aplicação achar necessário, ele pode criar os arquivos de pre e pos transformação no diretorio: /diretorio_do_projeto/pre-build.xml e "/diretorio_do_projeto/pro-build.xml", onde o diretório /diretorio_do_projeto representa o caminho de diretório selecionado para armazenar os arquivos do projeto durante a criação de projeto. Se esses arquivos existirem, a ferramenta cancela as execuções do Ant que utilizam os *scripts* contidos no diretório de domínio e executa o Ant com esses arquivos do diretório de projeto.

O *script* do Ant não pode obter informações sobre os dados do projeto que está sendo transformado previamente. Para poder copiar ou mover arquivos de posição, o Ant deve ter o conhecimento do local de saída dos diretórios transformados pela ferramenta Captor, entre outras informações. Para resolver esse problema, o usuário pode inserir as linhas contidas na Listagem 12 no *script* do Ant.

Listing 12: Pré-processamento do *script* do Ant

```
<!--PROJECT\_GENERATED\_DATA - DO\_NOT\_EDIT-->
<!--PROJECT\_GENERATED\_DATA - DO\_NOT\_EDIT-->
```

Antes de executar o Ant, a ferramenta Captor realiza o pre-processmento dos arquivo de pre e pos transformação para criar as seguintes variáveis:

Listing 13: Resultado do pré-processamento do *script* do Ant

```
<!--PROJECT_GENERATED_DATA - DO_NOT_EDIT-->
<property name=''install_path'' location=''/Captor''/>
<property name=''project_name'' value=''FWS1''/>
<property name=''project_path'' location=''/Captor/projects/foo''/>
<property name=''project_output_path'' location=''/output''/>
<property name=''interaction_0'' value=''0''/>
<property name=''interaction_1'' value=''1''/>
<!--PROJECT_GENERATED_DATA - DO_NOT_EDIT-->
```

A variavel "install_path" indica o local de instalação da ferramenta Captor. A variável "project_name" indica o nome do projeto, as variáveis "project_path" e "project_output_path" indicam o diretório do projeto e o diretório para onde os arquivos trasnformados são armazenados. As variáveis "interaction_X" indicam quantas interações foram editadas na ferramenta.

Uma das principais funcionalidades da ferramenta Ant utilizada em conjunto com a ferramenta Captor é a copia automática dos arquivos de saída para o diretório de arquivos de desenvolvimento ou, se o usuário utilizar zonas de segurança o Ant pode ser executado para copiar os arquivos modificados manualmente do diretório de desenvolvimento para o diretório de saída do gerador antes que o processo de transformação seja iniciado.

3.2 Mensagens de erro durante a validação da especificação e os seus significados

Diversas mensagens de erro de validação podem aparecer no painel de visualização de erros enquanto o engenheiro de domíno edita a especificação da aplicação. [1].

Existem dois tipos de erros de validação: os erros de falha na validação de um formulário e os erros de checagem estrutural. Nas Subsecções 3.2.1 e 3.2.2 são apresentados detalhes de como interpretar essas mensagens de erro.

[1]Se o painel de visuaalização de erros não estiver aparecendo, clique no menu Windows->Show View->Error.

3.2.1 Erros de validação de formulários

Os dados inseridos nos formulários precisam ser validados. Quando o usuário insere um dado incorreto em algum campo do formulário, a ferramenta emite uma mensagem de erro avisando o usuário que não foi possível validar a especificação corretamente.

Mensagem: Form validation error

Saída de exemplo:

Error in form: Interaction 1->Form1

Message error:

** Age is not a number.*

Significado:

A linha "Error in form: Interaction 1->Form1" indica em qual formulário ocorreu o erro. Nesse caso, o erro ocorreu no formulário raíz com nome igual a "Form1" localizado na interação 1.

A mensagem de erro "Age is not a number" indica que o campo de idade não possuí um valor válido.

3.2.2 Erros de checagem estrutural

Para obter uma especificação completa, alguns formulários podem ter o preenchimento obrigatório. Se o usuário não preencher todos os formulários obrigatórios, a ferramenta deve emitir uma mesangem de erro notificando o usuário.

Mensagem: Structural form validation error

Output de exemplo:

The form: Interaction 1->Form1 require one of these forms:

child(Form2) or child(Form3)->child(Form4)

AND

child(Form5)

Insert the required forms indicated above to get a valid specification.

Significado:

A linha "The form: Interaction 1->Form1 require one of these forms:" indica que o formulário raíz com nome igual a "Form1" da interação 1, possuí uma dependência obrigatória com os formulários indicados na mensagem de erro.

Os formulários que devem ter preenchimento obrigatório são especificados nas regras lógicas da mensagem. Nesse caso, o formulário raíz Form1 da interação 1 deve possuir:

(*um filho com nome igual a Form2* **OU** *um filho com nome igual a Form3 que deve possuir um filho com nome igual a Form4*) **E** (*um filho com nome igual a Form5*).

Referências

Czarnecki, K.; Eisenercker, U. W. *Generative programming.* Addison-Wesley, 2002.

Davidson, J. D.; Atherton, B.; Bailliez, S.; Benson, M. The Apache Ant Project. Available at: http://ant.apache.org. 2005.
Disponível em `http://ant.apache.org/contributors.html` (Acessado em 26/09/2005)

Jack Herrington Extensible code generation with java, part 2. 2005.
Disponível em `http://today.java.net/pub/a/today/2004/05/31/generation-pt2.html` (Acessado em 19/10/2005)

XSL The extensible stylesheet language family (xsl). 2005.
Disponível em `http://www.w3.org/Style/XSL/` (Acessado em 19/10/2005)