8
Ferramentas para Aplicar Testes Baseados em Modelos

Neste capítulo são apresentadas duas ferramentas desenvolvidas para ajudar na geração automática de artefatos a partir de diagramas UTP-C. Inicialmente é detalhado o mapeamento considerado para permitir a geração dos arquivos TF.xml e CFF.xml (usados pelo framework JAAF+T) a partir, respectivamente, de transformações baseadas em diagramas estruturais e dinâmicos da UTP-C. Todos os diagramas considerados para transformação são criados a partir da ferramenta Rational Software Architecture - RSA (RSA, 2012), ferramenta amplamente utilizada no mercado para criar modelos UML. Em seguida, o capítulo apresenta as ferramentas desenvolvidas para permitir essas transformações, que são as seguintes: (i) componentes desenvolvidos na linguagem Lua (Lua, 2012), e (ii) um plug-in para o RSA (Silva, Costa et al., 2012), que além de permitir a geração de arquivos XML para o JAAF+T também realiza a geração de outros artefatos (relatórios e comentários javadoc em scripts de teste desenvolvidos na linguagem Java) que ajudam a coordenar a execução dos testes.

8.1.
Mapeamento entre Arquivos XML do JAAF+T e Modelos UTP-C

Devido ao custo de manter e criar os arquivos XML do framework JAAF+T (detalhes no capitulo 4), percebeu-se que diagramas UTP-C poderiam ser usados para nos ajudar nessas atividades. Assim, inicialmente identificamos qual tipo de diagrama poderia ser utilizado para gerar de forma automática cada arquivo. Após essa análise, concluímos que diagramas de classes e diagramas de atividades baseados na UTP-C (ver capítulo 6) poderiam ser usados para gerar arquivos TF.xml e CFF.xml, respectivamente, já que informações contempladas nesses arquivos são modeladas em tais diagramas. Os arquivos DF.xml e CEF.xml, por enquanto, não foram considerados para geração. No entanto, em breve pretendemos oferecer alguma abordagem que permita gerá-los.

Para realizar transformações de diagramas baseados na UTP-C para arquivos TF.xml e CFF.xml, os diagramas devem ser criados na ferramenta Rational Software Architecture (RSA). O RSA considera que cada projeto de modelo criado possui um arquivo com a extensão "*.emx" (similar a um arquivo XMI), responsável por descrever todos os diagramas e elementos modelados. Assim, um parser no arquivo "*.emx" é realizado para que os arquivos XML possam ser gerados. A fim de esclarecer o mapeamento considerado para permitir a transformação de entidades modeladas em diagramas UTP-C para os arquivos TF.xml e CFF.xml, a Tabela 9 e 10 apresentam esses mapeamentos adotados.

Informação representada no arquivo TF.xml	**Informação modelada em diagramas de classe da UTP-C**
Elemento <setTest>.	Pacote com estereótipo <<Development>> que possui ao menos um test context.
Conteúdo do atributo id de um elemento <setTest>.	Nome do pacote com estereótipo <<Development>> que possui ao menos um test context.
Elemento <test>.	Classe de teste com estereótipo <<TestContext>>.
Conteúdo do atributo id de um elemento <test>.	Nome de uma classe com estereótipo <<TestContext>>.
Conteúdo do atributo *isAutomatic* de um elemento <test>.	Atributo *isAutomatic* de um test context.
Conteúdo do atributo *testLevel* de um elemento <test>.	Atributo *testLevel* de um test context.
Conteúdo do atributo *testType* de um elemento <test>.	Atributo *type* de um caso de teste modelado. Caso diferentes tipos pertençam a um mesmo test context, os diferentes tipos são recuperados e apresentados no TFF.xml separados por vírgula.
Conteúdo do atributo *isMandatory* de um elemento <test>.	Se houver algum caso de teste de um test context considerado obrigatório para execução é recuperado o valor

	true.
Conteúdo do atributo *context* de um elemento <test>.	Atributo *desiredSystemVersion* de um test context.
Conteúdo do elemento <classpath> que faça parte de um <test>.	Caminho completo onde um test context está armazenado, incluindo o nome da classe.
Conteúdo do elemento <priority>.	Maior prioridade identificada dentre os casos de teste que façam parte de um mesmo test context (uso do atributo priority). Prioridade com menor valor significa que possui maior prioridade, isto é, um caso de teste com valor zero possui uma prioridade maior do que um caso de teste com valor um.
Conteúdo do elemento <risk>.	Maior risco identificado dentre os casos de teste que façam parte de um mesmo test context (uso do atributo risk). Risco com menor valor significa que possui maior risco, isto é, um caso de teste com valor zero possui um risco maior do que um caso de teste com valor um.
Conteúdo do elemento <tool>	Atributo *tool* de um test context.

Tabela 9. Mapeamento para transformação de diagramas estruturais da UTP-C para o arquivo TF.xml.

Informação representada no arquivo CFF.xml	Informação modelada em diagramas de atividade da UTP-C
Elemento <artifact>.	Entidade comentário com estereótipo <<ArtifactUnderTest>>.
Conteúdo do atributo *id* de um elemento <artifact>.	Conteúdo do atributo *name* do comentário com estereótipo <<ArtifactUnderTest>>.
Conteúdo do atributo *type* de um elemento <artifact>.	Conteúdo do atributo *type* do comentário com estereótipo <<ArtifactUnderTest>>.
Conteúdo do atributo *logPath* de um elemento <artifact>.	Conteúdo do atributo *log* do comentário com estereótipo <<ArtifactUnderTest>>.
Conteúdo do atributo *criterionID* de um elemento <artifact>.	Conteúdo do atributo *criterionid* do comentário com estereótipo <<ArtifactUnderTest>>.
Elemento <test>.	Atividade modelada com estereótipo <<TestContext>>.
O conteúdo do atributo *type* de um elemento <test> será "*test*", pois irá referenciar um teste.	Para cada atividade test context modelada é fornecido o valor "*test*" para o atributo type do elemento <test> no CFF.xml.
Conteúdo do atributo *id* de um elemento <test>.	Nome da atividade que representa um test context.

Tabela 10. Mapeamento para transformação de diagramas dinâmicos da UTP-C para o arquivo CFF.xml.

8.2. Componentes LUA

Considerando a ideia de permitir a transformação de diagramas UTP-C para arquivos XML usados pelo JAAF+T, dois componentes foram criados na linguagem Lua (Lua, 2012): RSATF.lua e RSACFF.lua. O primeiro componente segue a ideia ilustrada na Figura 43, ou seja, um ou mais diagramas de classe UTP-C (etapa 1) são usados como base para gerar um arquivo TF.xml (etapa 2). Já o segundo componente permite a geração de um arquivo CFF.xml a partir de

um ou mais diagramas de atividade UTP-C, assim como ilustrado na Figura 44. O TF.xml possui exclusivamente informações estáticas, enquanto que o CFF.xml possui informações dinâmicas, pois descreve fluxos (ordens) de execução dos testes.

Assim como mencionado na Seção 8.1, para que cada componente execute as transformações mencionadas, eles realizam um parse no arquivo "*.emx" para obter as informações das entidades modeladas. Dessa forma torna-se possível a geração dos arquivos esperados. Para informar qual arquivo "*.emx" será usado, cada componente Lua possui uma variável "path" no método main. O Código 9 apresenta parte desse método de cada componente criado, onde o arquivo ".emx" usado é o "Pacote em Branco Simplificado.emx". O nome desse arquivo é o nome padrão fornecido pelo RSA, toda vez que um projeto de modelos é criado do zero. Portanto, caso seja necessário utilizar outro arquivo "*.emx", basta alterar o conteúdo da variável "path". Exemplos de diagramas UTP-C são apresentados na próxima seção, pois nela são apresentadas informações adicionais do RSA, facilitando assim o entendimento do leitor em relação à ferramenta de modelagem.

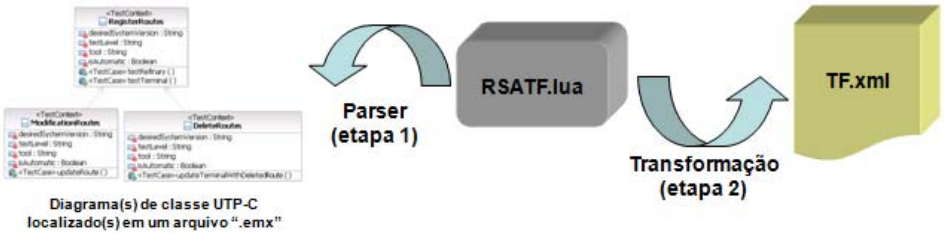

Figura 43. Transformação realizada pelo componente RSATF.lua.

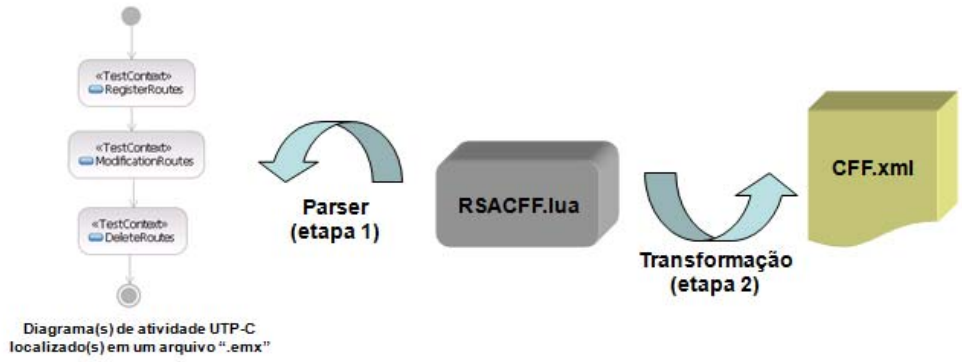

Figura 44. Transformação realizada pelo componente RSACFF.lua.

```
function main()
        ...
        -- Caminho que está o arquivo ".emx"
        path = "C:/.../Pacote em Branco Simplificado.emx"

        -- Permite a leitura do arquivo ".emx". Conexão com tal arquivo.
        fd = io.input(path)
        …
end
```

Código 9. Estrutura do método Main dos componentes desenvolvidos em LUA.

Após a criação dos dois componentes Lua, identificamos que outros artefatos úteis poderiam ser gerados a partir de diagramas UTP-C para ajudar na gerência dos testes. Para facilitar o uso dessas transformações foi criado um plug-in em Java para o RSA, detalhado na seção a seguir.

8.3. Plug-in para o RSA

Nesta seção é apresentado um plug-in desenvolvido (Silva, Costa et al., 2012) em Java para a ferramenta Rational Software Architecture 7.5 (RSA) da IBM que usa como base a IDE Eclipse. Tal plug-in permite que a partir de transformações de diagramas UML (baseados na UTP-C) para arquivos Java, os seguintes artefatos sejam gerados para ajudar na coordenação dos testes:

- Relatório descrevendo quais testes estão atualizados e pendentes de atualização para alguma versão de um sistema;
- Comentários no formato javadoc para os scripts de teste desenvolvidos em Java (ex: testes em DBUnit, JUnit e JAT);
- Arquivos TF.xml e CFF.xml para instâncias do framework JAAF+T.

Inicialmente esta seção apresenta a arquitetura do plug-in. Em seguida é realizada a descrição de como tal ferramenta realiza a geração automática do relatório responsável por indicar quais testes estão atualizados e pendentes de atualização para alguma versão de um sistema. Além disso, é apresentado como o plug-in gera comentários javadoc para scripts de teste desenvolvidos em Java.

Por último, a seção descreve como é realizada a geração dos arquivos TF.xml e CFF.xml para instâncias do framework JAAF+T.

8.3.1. Visão Geral da Arquitetura do Plug-in

Na Figura 45 é ilustrada uma visão geral do plug-in. Repare que ele foi criado para o Rational Software Architecture (RSA) 7.5, desenvolvido sobre a IDE Eclipse. Para que esse plug-in pudesse ser criado, três pontos principais foram considerados. O primeiro está relacionado à necessidade de criar diagramas de classes e atividades baseados na UTP-C. Somente a partir desses modelos, o plug-in realiza a geração de artefatos úteis para equipes de teste. O segundo ponto refere-se à necessidade de indicar no RSA quais pontos de extensão seriam adicionados e qual transformação iria executar essas novas funcionalidades oferecidas pelo plug-in. Por isso um arquivo chamado plugin.xml foi criado para definir tais informações. A transformação escolhida do RSA 7.5 foi referente aos diagramas UML para Java. Já o terceiro e último ponto é referente às novas classes criadas para realizar os tratamentos adequados dos dados recuperados pelas transformações.

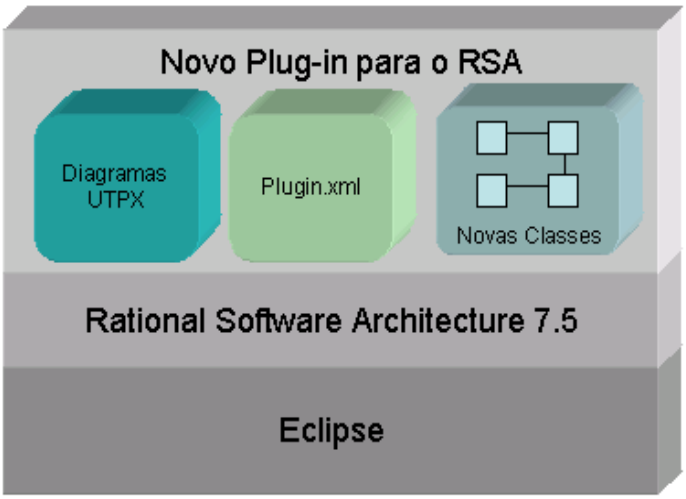

Figura 45. Modelo conceitual do plug-in para o RSA.

Na Figura 46 é apresentado um diagrama de classes com as principais classes criadas para o plug-in. As classes *PropertyKeywordJavadocRule*, *OperationKeywordJavadocRule* e *PackageKeywordJavadocRule* possuem os tratamentos adicionais para os dados recuperados de propriedades, operações e

pacotes, respectivamente, a partir de uma transformação solicitada pelo usuário da ferramenta. Essas classes estendem *ClassRule*, classe provida pelo RSA 7.5. Todas as classes pintadas de cinza são classes já oferecidas pelo RSA 7.5, enquanto que as outras entidades são novas classes definidas pelo plug-in.

PropertyKeywordJavadocRule e *OperationKeywordJavadocRule* geram comentários javadoc baseados em informações modeladas em propriedades e operações. Para incluir tais comentários é utilizada a classe *TagElement*. Já a classe *PackageKeywordJavadoc* é responsável por gerar o relatório dos testes atualizados e os arquivos XML para o framework JAAF+T. Para realizar essas gerações, a classe *GenerateFile* (aplica o padrão de projeto Singleton) (Gamma et al., 1994) é utilizada. A partir dela um conjunto de instâncias das classes *TestContext*, *TestCase* e *Artifact* são criadas para armazenar, respectivamente, dados recuperados dos test contexts, casos de teste, assim como as ordens de execução dos testes para validar artefatos do SUT. A partir dessas instâncias geradas os arquivos desejados (ex: relatórios e arquivos XML) são criados.

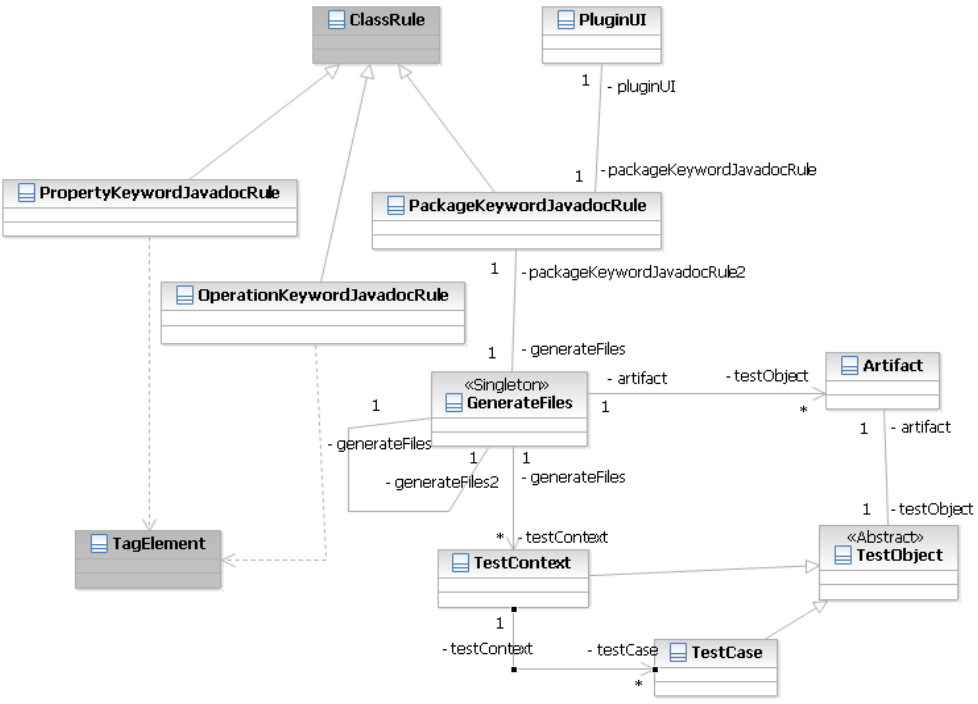

Figura 46. Diagrama de classes do novo plug-in para o RSA.

8.3.2. Geração de Relatórios e Comentários Javadoc

Uma das principais preocupações identificadas em projetos de teste é visualizar de forma fácil quais testes estão atualizados ou pendentes de atualização para alguma versão de um sistema que será ou está sendo testado. Visando atender tal necessidade o plug-in desenvolvido usa a UTP-C para modelar informações de teste a partir de diagramas de classe. Na Figura 47 é ilustrado um exemplo de diagrama de classes baseado na UTP-C com três test contexts.

Assim como apresentado na Seção 8.1, algumas das informações utilizadas pelo plug-in, a partir de diagramas UTP-C, são as seguintes: (i) versão desejada de um sistema que um test context deve estar atualizado, (ii) nível de teste relacionado, (iii) ferramenta usada para sua execução, (iv) se ele é executado de forma automática ou manual, etc.

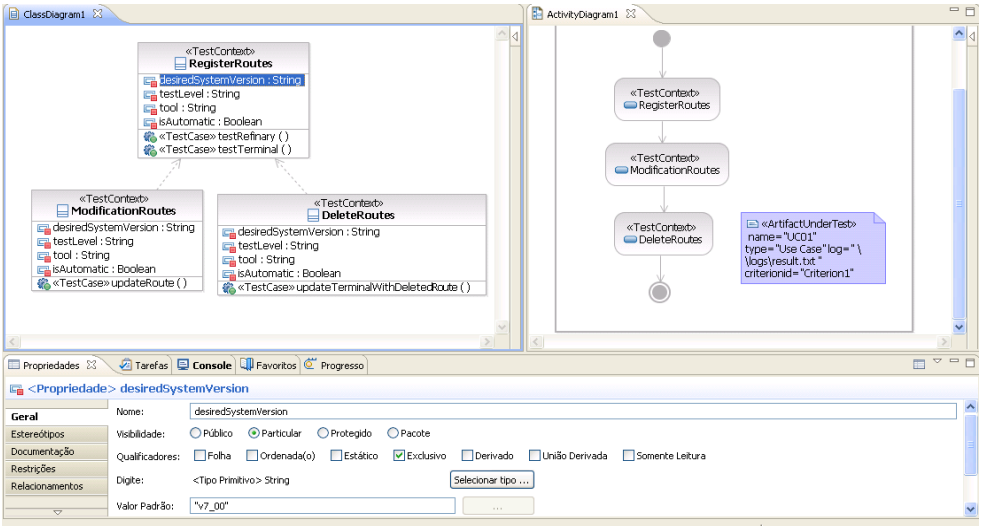

Figura 47. Diagrama de classes e atividades da UTP-C modeladas no RSA.

Cada test context da Figura 47 possui um atributo chamado *desiredSystemVersion*, responsável por informar a versão desejada de um sistema que tal classe deve estar atualizada. Caso o usuário selecione o atributo *desiredSystemVersion* a partir do RSA, o valor padrão definido para tal propriedade é apresentado na aba Propriedades, ilustrada na parte inferior da Figura 47. Nesse exemplo o valor definido foi “v7_00”. Tanto esse atributo como

os outros que compõem um test context (*testLevel*, *tool* e *isAutomatic*) também podem ser editados pela aba Propriedades.

Já a versão corrente do sistema que cada caso de teste está atualizado é representada por um atributo chamado "*currentVersion*". Esse atributo pode ser visualizado pelo RSA a partir da opção "Documentação" da aba Propriedades (ver Figura 48) quando um caso de teste (método com o estereótipo <<TestCase>>) é selecionado pelo usuário. Além dele, há outros atributos apresentados: *type*, *priority*, *isMandatory* e *risk* (ver subseção 6.2.3). A aba Propriedades é um recurso já oferecido pelo RSA 7.5, e é utilizado para modelar diagramas baseados na UTP-C.

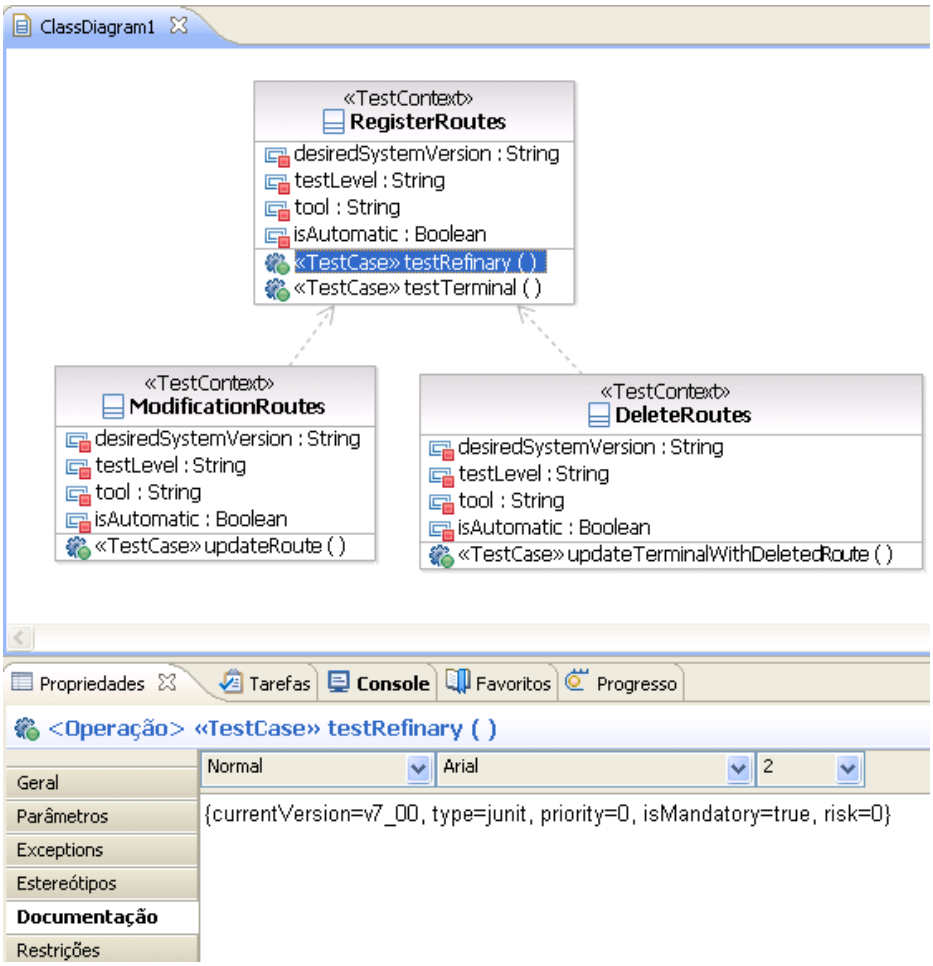

Figura 48. Atributos adicionais de um caso de teste modelado.

A partir das informações modeladas, um relatório similar ao ilustrado na Figura 49 pode ser gerado pelo plug-in toda vez que for solicitada uma

transformação de UML para Java. Tal relatório apresenta as seguintes informações:

- Listagem de todos os test contexts atualizados para a versão desejada. Nesse caso são considerados somente test contexts que possuem casos de teste obrigatórios.
- Listagem de todos os test contexts pendentes de atualização. Assim como a listagem anterior, são considerados, exclusivamente, os test contexts com casos de teste obrigatórios.
- Resumo geral dos test contexts e casos de teste analisados.
- Data e hora da geração do relatório.

```
--------- TestContexts atualizados para a versão v7_00: ---------
ModificationRoutes
RegisterRoutes

--------- TestContexts pendentes de atualização para a versão v7_00: ---------
DeleteRoutes - v6_00

--------- Resumo: ------------------------------------
Quantidade de test contexts atualizados para a versão v7_00: 2
Quantidade de test contexts pendentes de atualização para a versão v7_00: 1

Quantidade de casos de teste atualizados para a versão v7_00: 2
Quantidade de casos de teste pendentes de atualização para a versão v7_00: 1
Quantidade de casos de teste que não precisam ser atualizados para a versão v7_00: 1
------------------------------------------------

Automatic generation at: 2012-04-04 19:45:27
```

Figura 49. Relatório gerado a partir do plug-in.

Após a solicitação de uma transformação pelo RSA, uma tela similar a Figura 50 é apresentada para que um relatório (similar ao da Figura 49) seja gerado. Nessa tela o usuário deve fornecer a versão do sistema que os testes devem estar atualizados (a partir do campo de texto “*Desired System Version*”) e o caminho em que tal relatório será guardado (a partir do campo de texto “*Folder*”). Já o checkbox JAAF+T permite que o usuário do plug-in informe qual arquivo XML (TF.xml e CFF.xml) será gerado. Na subseção 8.3.3 são apresentados exemplos de XMLs gerados pelo plug-in.

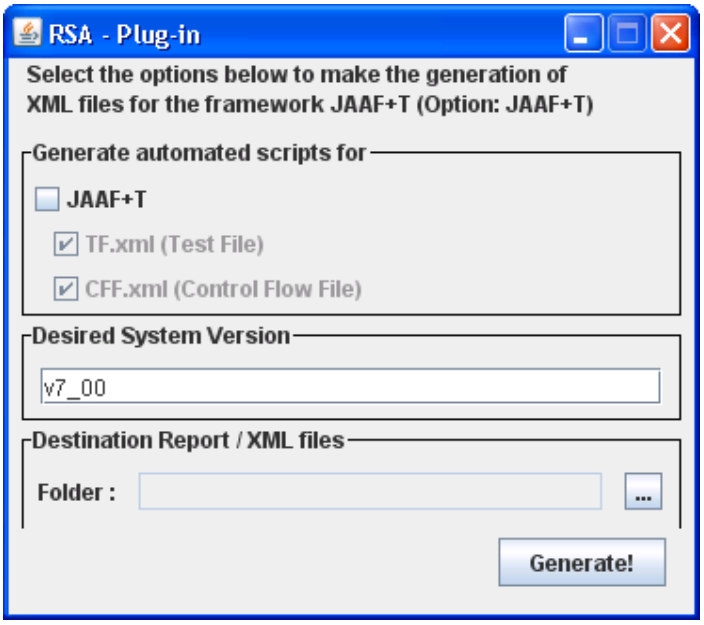

Figura 50. Tela para geração de artefatos de teste.

Além do relatório mencionado, o plug-in realiza a geração automática de comentários javadoc em scripts de teste desenvolvidos em arquivos Java. Essa geração é realizada quando a transformação de diagramas UML para Java é solicitada pelo usuário. Os comentários gerados são descritos a seguir e um exemplo de classe com comentários gerados é ilustrado no Código 10:

- Para o atributo *desiredSystemVersion* de um test context é escrito o seguinte comentário: “Versão que os casos de teste devem estar atualizados é <versão do sistema recuperada pelo atributo *desiredSystemVersion*>”.
- Para o atributo *testLevel* de um test context é escrito o seguinte comentário: “Nível de teste <nível de teste informado no atributo *testLevel*>”.
- Para o atributo *tool* de um test context é escrito o seguinte comentário: “Ferramenta de teste <nome da ferramenta informada no atributo *tool*>”.
- Quando o atributo *isAutomatic* tiver o valor true é apresentado o comentário “Teste automatizado”. No entanto, quando possuir o valor false é apresentado “Teste não automatizado”.
- Para cada método que for um caso de teste é incluído o comentário “TestCase”, além das informações relacionadas ao tipo de

obrigatoriedade do caso de teste, a versão corrente do sistema que ele está atualizado, seu tipo de teste, sua prioridade de execução e o risco de produto relacionado.

```
/**
 * <!-- begin-UML-doc -->
 * <!-- end-UML-doc -->
 * @author andrew
 * @generated "UML para Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
 */
public class DeleteRoutes {
	/**
	 * <!-- begin-UML-doc -->
	 * <!-- end-UML-doc -->
	 * Versão que os casos de teste devem estar atualizados é
v7_00
	 * @generated "UML para Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
	 */
	private String desiredSystemVersion = "v7_00";
	/**
	 * <!-- begin-UML-doc -->
	 * <!-- end-UML-doc -->
	 * Nível de teste unitário
	 * @generated "UML para Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
	 */
	private String testLevel = "unit";
	/**
	 * <!-- begin-UML-doc -->
	 * <!-- end-UML-doc -->
	 * Ferramenta de teste dbunit
	 * @generated "UML para Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
	 */
	private String tool = "dbunit";

	/**
	 * <!-- begin-UML-doc -->
	 * <!-- end-UML-doc -->
	 * Teste automatizado
	 * @generated "UML para Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
	 */
	private Boolean isAutomatic = true;

	/**
	 * <!-- begin-UML-doc -->
	 * {currentVersion=v6_00, type=white-box, priority=0,
isMandatory=true, risk=0}
	 * <!-- end-UML-doc -->
	 * TestCase
	 * @generated "UML para Java
(com.ibm.xtools.transform.uml2.java5.internal.UML2JavaTransform)"
	 */
	public void updateTerminalWithDeletedRoute() {
		// begin-user-code
		// TODO Stub de método gerado automaticamente
```

```
            // end-user-code
        }
}
```

Código 10. Exemplo de classe gerada pelo plug-in.

8.3.3. Geração dos arquivos TF.xml e CFF.xml

Para realizar a geração automática dos arquivos TF.xml e CFF.xml a partir de diagramas UTP-C, foi considerado o mapeamento descrito na Seção 8.1. Visando ilustrar exemplos de arquivos XML gerados, o Código 11 e 12 mostram os arquivos TF.xml e CFF.xml gerados a partir das classes e atividades modeladas e ilustradas na Figura 47, respectivamente.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Automatic generation at: 2012-04-30 09:58:23 -->
<TF>
 <setTest id="GUI.UC01">
        <test id="DeleteRoutes" isAutomatic="true" testLevel="unit"
                testType="white-box" isMandatory="true" context="v7_00">
                <classname>GUI.UC01.DeleteRoutes</classname>
                <priority>0</priority>
                <risk>0</risk>
               <tool>dbunit</tool>
        </test>
        <test id="ModificationRoutes" isAutomatic="true" testLevel="unit"
                testType="white-box" isMandatory="true" context="v7_00">
                <classname>GUI.UC01.ModificationRoutes</classname>
                <priority>0</priority>
                <risk>0</risk>
               <tool>junit</tool>
        </test>
        <test id="RegisterRoutes" isAutomatic="true" testLevel="unit"
                testType="white-box" isMandatory="true" context="v7_00">
                <classname>GUI.UC01.RegisterRoutes</classname>
                <priority>0</priority>
                <risk>0</risk>
```

```
                <tool>junit</tool>
        </test>
 </setTest>
</TF>
```

Código 11. Exemplo de TF.xml gerado pelo plug-in.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Automatic generation at: 2012-04-30 11:14:05 -->
<CFF>
        <artifact id="UC01" type="Use Case" logPath="\\logs\result.txt"
                criterionID="Criterion1">
                <test type="test" name="RegisterRoutes"/>
                <test type="test" name="ModificationRoutes"/>
                <test type="test" name="DeleteRoutes"/>
        </artifact>
</CFF>
```

Código 12. Exemplo de CFF.xml gerado pelo plug-in.

No Apêndice B são apresentados modelos UTP-C criados no RSA para o sistema mercado virtual descrito no capítulo 7. Já no Apêndice C são ilustrados os arquivos TF.xml e CFF.xml gerados pelo plug-in a partir dos diagramas apresentados no Apêndice B.

8.3.4. Avaliação da Ferramenta de Teste

Nesta seção é apresentada uma avaliação qualitativa do plug-in desenvolvido. O objetivo da avaliação foi verificar até que ponto as funcionalidades oferecidas por ele auxiliam os desenvolvedores na aplicação de testes baseados em modelos considerando o uso da UTP-C. Doze profissionais da indústria que trabalham diretamente com testes de sistemas de larga escala foram entrevistados. O roteiro da avaliação consistiu de uma série de mudanças aplicadas a um modelo baseado na UTP-C. Nas próximas subseções apresentamos em detalhes o material utilizado (subseção 8.3.4.1), o procedimento de execução da avaliação (subseção 8.3.4.2) e finalmente os resultados obtidos (subseção 8.3.4.3).

8.3.4.1. Material

Como material utilizamos um modelo com cinco test contexts. Cada test context é composto por um ou mais casos de teste. Além disso, cada caso de teste foi modelado como automático ou manual e obrigatório ou opcional. A partir desse modelo tornou-se possível exercitar boa parte do conjunto de conceitos oferecidos pela abordagem UTP-C.

Para controlar o grau de experiência dos participantes oferecemos um roteiro detalhado de uso da ferramenta. Neste caso, cada participante teve que seguir esse roteiro estruturado, no estilo passo-a-passo, durante a realização de um conjunto de tarefas. Foram definidas oito tarefas subdivididas em duas atividades.

A primeira atividade foi composta por quatro tarefas de manutenção dos test contexts, onde cada tarefa descrevia um cenário particular de manutenção dos modelos. Desta forma conseguimos observar até que ponto as funcionalidades oferecidas pela ferramenta auxiliavam na compreensão e futura manutenção dos modelos. Cenários de evolução são particularmente importantes de serem avaliados, pois como vem sendo observados em diversos trabalhos, tarefas de compreensão/manutenção de artefatos de teste são recorrentes em qualquer projeto de desenvolvimento (Kaner et al., 2001). Além disso, as evoluções consideradas são em geral desafiadoras para os desenvolvedores envolvidos, assim como apresentado em (Costa et al., 2010b).

A segunda atividade foi composta por quatro tarefas relacionadas ao relatório gerado pela ferramenta. Os objetivos das tarefas foram: (i) medir a expressividade do relatório gerado (similar ao ilustrado na Figura 49); e (ii) seu papel na compreensão dos cenários de teste selecionados para alguma versão de um sistema em teste. A correta compreensão de certas propriedades desses cenários de teste, como sua obrigatoriedade e prioridade, são fundamentais para garantir qualidade na aplicação de testes.

Assim, um questionário composto por questões objetivas e perguntas abertas de esclarecimento e aprofundamento também foi aplicado aos participantes. Questões abertas e que comportam qualquer tipo de resposta são fundamentais para que possamos compreender melhor o ponto de vista dos desenvolvedores. No entanto, qualificações objetivas permitem confrontar as opiniões abstratas com itens que geram informações precisas a respeito dos mesmos tópicos. Por fim, esse questionário solicitou a cada participante que

escrevesse sobre sua expectativa de uso da ferramenta em cenários reais de teste.

8.3.4.2. Execução da Avaliação

As entrevistas foram conduzidas no Laboratório de Engenharia de Software da PUC-Rio de forma individual. Cada participante teve uma breve introdução de uso da ferramenta e tópicos relativos à abordagem da UTP-C foram esclarecidos. Essa etapa durou de cinco a quinze minutos devido ao conhecimento que cada participante tinha da UTP-C e do RSA. Em seguida, os participantes passaram a trabalhar nas tarefas seguindo o roteiro de uso da ferramenta. Ao fim de cada tarefa eles opinaram sobre o uso da ferramenta de acordo com o questionário aplicado. Como o exemplo que acompanha o roteiro não representa um cenário real, ao final do questionário foi solicitado a cada participante que escrevesse sobre sua expectativa de uso da ferramenta em situações reais.

8.3.4.3. Resultados e Discussões

A análise dos resultados foi dividida de acordo com as atividades definidas: (i) manutenção de test contexts; e (ii) análise exclusiva dos relatórios gerados pela ferramenta. Em seguida é apresentada uma análise geral do uso da ferramenta.

Manutenção de test contexts. Uma das atividades solicitadas aos participantes foi à realização de um conjunto de tarefas de evolução em um modelo simples. Durante a execução das tarefas os participantes de forma geral não encontraram dificuldades em alterar as informações requeridas, como: *DesiredSystemVersion*, obrigatoriedade, dentre outras, conforme destacado por um dos participantes na citação a seguir:

"A proposta para criação/manutenção de modelos é bem intuitiva."

Podemos destacar a adoção de uma interface padrão, isto é, semelhante à de outras ferramentas, como um dos fatores que influenciaram na facilidade de

uso do plug-in no RSA. Por exemplo, o uso de uma visão referente às propriedades dos objetos do modelo em separado do modelo propriamente dito permitiu a edição rápida das evoluções necessárias para realização das tarefas (ver Figura 48). Isto acontece porque tal recurso evita a navegação entre janelas de edição de propriedades. Essa navegação geralmente aumenta o tempo requerido para certas edições e deixa os desenvolvedores relativamente perdidos durante tarefas simples de edição. Além disso, o RSA permite a edição de certas informações diretamente nos objetos modelados, agilizando assim edições recorrentes, como o caso da ferramenta de teste adotada por algum test context.

No entanto, os participantes também destacaram alguns problemas de usabilidade da ferramenta. Um dos participantes destacou a necessidade de realizar a edição de algumas informações (e.g., prioridade) exclusivamente nas abas de Propriedades e não nas classes modeladas e apresentadas visualmente no diagrama. Outros destacaram como um fator negativo da ferramenta o tempo adicional requerido àqueles com pouco conhecimento no RSA para a fixação dos seguintes pontos: (i) das etapas de edição dos modelos; e (ii) dos caminhos para realização destas edições. Observamos que estes problemas exigiram um esforço cognitivo maior aos desenvolvedores do que o esperado. A seguir algumas citações são transcritas:

> *"O modelo é fácil de entender, apenas algumas informações ocultas no diagrama (e acredito que o RSA é responsável) que dificulta as atividades. Isso é difícil de separar."*
>
> *"Só soube como editar a versão corrente do SUT com um caso de teste (apesar de saber onde está) através de ajuda."*
>
> *"... para quem está iniciando existe um tempo para fixar as etapas."*

Todos os modelos modificados pelos participantes foram comparados com modelos previamente criados pelos organizadores da avaliação. Dessa forma, pudemos constatar que os modelos criados pelos participantes estavam corretos segundo a especificação da manutenção fornecida.

Análise dos relatórios gerados pela ferramenta. A segunda atividade solicitada aos participantes foi analisar o relatório gerado a partir do diagrama alterado pelas mudanças aplicadas na etapa anterior. Nesta segunda atividade nosso objetivo foi mostrar aos participantes que, apesar do formato proposto no

relatório, existem outras informações que poderiam ser apresentadas no relatório. Desta forma, os participantes foram induzidos a procurar essas informações que não estavam no relatório. Neste caso, alguns participantes realmente observaram que tais dados não estavam presentes.

"Necessário mais clareza."
"Esta informação não consta no relatório ou não está clara."

Com isso, observamos que apenas cinco dos doze participantes relataram alguma dificuldade para realizar as tarefas. No entanto, no final somente em 25% dos casos, as tarefas não foram qualificadas como fácil ou foi relatado algum obstáculo na execução da tarefa. Com isso, conseguimos observar que os participantes rapidamente assimilaram a natureza do relatório. Isso é reforçado pelo fato de que em todos os casos, as tarefas iniciais que questionavam sobre informações que realmente se encontravam no relatório foram identificadas com sucesso.

Análise geral. Finalmente, a partir da experiência relatada pelos participantes conseguimos observar que as funcionalidades oferecidas pela ferramenta, no geral, tendem a facilitar e incentivar o uso de modelos de testes em projetos de desenvolvimento. Essa conclusão é confirmada por citações de alguns participantes quando questionados sobre os seguintes itens: (i) se o uso da ferramenta em cenários reais é recomendável; e (ii) se é recompensador o uso do plug-in proposto entre o esforço da criação e manutenção dos modelos e os resultados obtidos com o uso do relatório. Dentre elas, podemos destacar:

"Não senti qualquer dificuldade tanto na criação/manutenção do modelo, pois como já mencionado, a proposta é bem intuitiva. Mesmo não dominando o RSA pude gerar/alterar o modelo."
"O exemplo utilizado era simples, mas é fácil imaginar um exemplo mais complexo. Neste caso, obter as informações sem o uso da ferramenta seria muito trabalhoso e não geraria resultados confiáveis."
"Não só recomendada como essencial para a utilização da UTP-C."

8.4. Discussão

Neste capítulo foram apresentadas ferramentas que exemplificam como modelos UTP-C podem ser usados como base para gerar artefatos úteis voltados para a coordenação dos testes. Um exemplo apresentado na Seção 8.3 foi à geração de relatórios que informam quais testes estão atualizados ou pendentes de atualização para alguma versão do SUT. A vantagem dessa geração é que o plug-in proposto recupera de forma automática informações espalhadas em diferentes diagramas, evitando que alguém do projeto tenha que identificá-las de forma manual. Além disso, o plug-in permite que a partir desse relatório, o acompanhamento dos testes possa ser feita de forma mais fácil. Outros relatórios úteis, que usem como base modelos UTP-C, poderiam ser criados para informar, por exemplo, quais e quantos testes foram criados por cada nível de teste ou por tipo de teste, quais testes com prioridade alta ainda não foram atualizados para alguma versão do SUT, etc.

Outro ponto considerado no capítulo e importante para a área de desenvolvimento e coordenação dos testes é a inclusão de comentários relevantes em scripts de teste a fim de ajudar no entendimento do código criado. Como modelos UTP-C trazem informações importantes sobre testes, essas informações podem ser incluídas como comentários para ajudar na atividade de desenvolvimento dos scripts de teste. Seguindo essa linha, o plug-in apresentado na Seção 8.3 recupera de forma automática informações de modelos UTP-C, e em seguida gera comentários javadoc em scripts desenvolvidos em Java. Esses comentários são apresentados em linguagem natural. Dessa forma, toda vez que um script automatizado for criado a partir do plug-in proposto, comentários relevantes são adicionados na classe do teste, como, por exemplo, qual é o nível e os tipos de teste relacionados, qual ferramenta é utilizada para executá-lo, qual versão do SUT o teste deve estar atualizado, etc.

Já o terceiro ponto abordado pelo capítulo foi a geração dos arquivos TF.xml e CFF.xml a partir de modelos UTP-C. Como o framework JAAF+T permite a criação de agentes de software capazes de coordenar a execução de testes a partir desses arquivos, utilizar alguma ferramenta capaz de varrer todas entidades modeladas nos diagramas UTP-C e gerar tais arquivos, torna o trabalho do desenvolvedor mais fácil. As ferramentas propostas no capítulo garantem a consistência dos modelos UTP-C com os arquivos TF.xml e CFF.xml

quando esses modelos são usados como base para gerar ou editar os arquivos XML, isto é, os modelos UTP-C são editados e em seguida é solicitada a transformação para que as mudanças ncessárias sejam aplicadas aos arquivos. No entanto, se os arquivos TF.xml e CFF.xml forem editados manualmente, essas mudanças não serão refletidas nos modelos. Para atender essa preocupação, pretendemos incluir tratamentos nas ferramentas apresentadas para que permitam manter a consistência dos arquivos e modelos, mesmo com edições manuais nesses arquivos.

8.5. Considerações Finais

Neste capítulo foram apresentadas duas ferramentas voltadas para a geração automática de artefatos a partir de diagramas UTP-C. Inicialmente, na Seção 8.1 foi detalhado o mapeamento adotado pelas ferramentas para permitir a geração do TF.xml e CFF.xml, arquivos usados pelo framework JAAF+T. Na Seção 8.2 foram apresentados componentes desenvolvidos em Lua para permitir a geração desses arquivos XML, enquanto que na Seção 8.3 foi apresentado um novo plug-in para o Rational Software Architecture (RSA, 2012). Esse plug-in além de gerar arquivos XMLs para o JAAF+T, também gera relatórios e comentários javadoc em scripts de teste desenvolvidos na linguagem Java. Por fim, na Seção 8.4 foi apresentada uma discussão sobre os artefatos gerados a partir das ferramentas propostas. Tal discussão esclarece as principais vantagens das ferramentas oferecidas, assim como possíveis extensões a serem aplicadas.