7
Estudo de Caso Usando UTP-C com JAAF+T

Neste capítulo, apresentamos um mercado virtual (*Virtual Market Place - VMP*) para ilustrar o uso da linguagem UTP-C e do JAAF+T em uma aplicação prática. Um exemplo de comércio eletrônico foi escolhido, porque há referências na literatura (He et al., 2003; Jennings et al., 1998; Lomuscio et al., 2003) como sendo um benchmark de SMAs.

Com o intuito de auxiliar os usuários da UTP-C e do JAAF+T, definimos uma abordagem (Seção 7.1) para facilitar o uso conjunto de tais propostas. Além disso, este capítulo apresenta em detalhe uma descrição textual do sistema multiagente modelado e desenvolvido (Seção 7.2). Em seguida (Seção 7.3), são mencionados outros exemplos de SMAs modelados a partir do UTP-C e criados a partir do JAAF+T. Por fim, uma discussão é apresentada sobre o mercado virtual desenvolvido (Seção 7.4).

7.1.
Abordagem Adotada

O objetivo da abordagem é ajudar o testador a trabalhar de forma conjunta com a UTP-C e o JAAF+T. Essa abordagem descreve o que é necessário ser especificado a fim de representar completamente um design por meio de diagramas UTP-C e criar agentes autoadaptativos usando o JAAF+T.

7.1.1.
Diagramas Estáticos UTP-C

Inicialmente, torna-se necessário realizar a modelagem estática dos testes de software da aplicação desejada. Portanto, devem ser identificados quais testes serão criados, os casos de testes, suítes e critérios de seleção para executar os testes. A modelagem estática da UTP-C é realizada a partir da criação de diagramas de classe da UML.

Como testes são compostos por casos de teste, cada teste é representado por uma classe com estereótipo <<TestContext>> que possui um ou mais métodos com estereótipo <<TestCase>>, representado assim os casos de teste.

Cada caso de teste possui um conjunto de informações úteis para sua identificação, como, por exemplo, seu tipo de teste, prioridade de execução e se é obrigatório para execução (ver subseção 6.2.3). Já as informações que devem ser comuns a todos os casos de teste de um mesmo teste são definidas no test context modelado, como, por exemplo, nível de teste, ferramenta usada para executá-los, se são executados de forma automática ou manual, além da versão do SUT que eles devem estar atualizados.

Assim que os test contexts são modelados, as dependências entre eles também devem ser identificadas. A fim de ajudar a expressar a semântica de cada dependência, estereótipos propostos pela UTP-C (ver subseção 6.2.8) podem ser utilizados.

Após a modelagem dos test contexts e das suas dependências, critérios de seleção devem ser identificados. Diferentes critérios podem ser definidos, como, por exemplo, executar exclusivamente testes unitários e que também sejam de regressão. Na subseção 6.2.4 são apresentadas em detalhe quais informações podem ser usadas para modelar os critérios desejados.

Por fim, torna-se importante modelar quais suítes deverão ser criadas no sistema. Para expressar quais testes são executados por cada suíte, basta modelar as dependências dos suítes em relação aos test contexts modelados (ver subseção 6.2.8).

7.1.2. Diagramas Dinâmicos UTP-C

A fim de modelar os fluxos de execução representados nos suites de teste, diagramas de atividades devem ser utilizados. Um diagrama de atividade modela a ordem de execução dos test contexts ou de casos de teste (ver subseções 6.2.2 e 6.2.3).

Assim que um suíte for identificado, ele deve ser modelado no diagrama de atividade e deve ser informado qual artefato está sendo testado a partir dele. Para que seja possível representar essa informação, uma entidade comentário com o estereótipo <<ArtifactUnderTest>> deve ser incluída no diagrama. Tal entidade deve ser composta pelas seguintes informações: (i) nome do artefato, (i) tipo de artefato, (iii) caminho onde o log com os resultados dos testes serão armazenados, e (iv) o critério de seleção relacionado. Para mais detalhes ver subseção 6.2.7.

7.1.3. Processo de Adaptação no JAAF+T

Toda vez que um agente autoadaptativo capaz de realizar autoteste for criado, deve ser definido o processo de autoadaptação que ele deverá executar. Atualmente, o JAAF+T oferece um processo padrão que pode ser utilizado por diferentes agentes de software. No entanto, isso não impede que novos processos de autoadaptação possam ser criados.

A partir do momento que algum agente autoadaptativo deve realizar autoteste, o processo a ser usado deve conter as atividades Teste e Validação. Tais atividades ajudam, respectivamente, a selecionar e executar os testes desejados, assim como tomar decisões a partir dos resultados gerados pelos testes. No entanto, para que o JAAF+T consiga auxiliar a execução dos agentes autoadaptáveis, os arquivos XML oferecidos pelo framework (TF.xml, DF.xml, CFF.xml e CEF.xml) devem ser preenchidos. Nas próximas subseções esses arquivos são abordados em mais detalhe.

7.1.4. Testes no JAAF+T

A partir do diagrama de classe da UTP-C que ilustra todos os testes (test contexts) de algum sistema, o arquivo TF.xml deve ser preenchido. Esse arquivo é oferecido pelo JAAF+T e possui como responsabilidade descrever quais testes poderão ser executados em algum processo de autoadaptação. A partir dessa descrição o JAAF+T ajuda agentes autoadaptáveis a coordenarem a execução dos testes. As informações colocadas nesse arquivo são equivalentes às informações dos test contexts modeladas pelos diagramas estáticos da UTP-C, como, por exemplo, ferramenta usada para executar o teste, nível de teste, tipo de teste, identificador do teste, etc. Assim, o testador pode usar como base os modelos da UTP-C para preencher mais facilmente o arquivo TF.xml.

7.1.5. Dados de Entrada e Saída para Testes no JAAF+T

Após preencher o TF.xml com os testes a serem executados, os dados de entrada e saída que serão usadas por eles devem ser fornecidos ao arquivo DF.xml. Esse arquivo permite que diversas condições de teste sejam definidas, permitindo que um mesmo teste possa ser executado de n maneiras diferentes.

Para permitir que os testes possam utilizar o arquivo DF.xml, suas classes de teste devem implementar a interface *TestUsingDFInterface*, pois essa interface define dois métodos responsáveis por informar quais dados do DF.xml serão utilizados por cada teste. A partir desses métodos pode ser informado o uso de somente uma condição específica de teste, ou um conjunto delas para um mesmo teste. Detalhes são apresentados na Seção 4.2.

7.1.6. Fluxos de Execução dos Testes no JAAF+T

Para que agentes JAAF+T possam saber quais testes deverão ser executados para validar algum artefato do sistema, fluxos de execução (suites de teste) devem ser definidos no arquivo CFF.xml. Como tais fluxos foram modelados a partir dos diagramas de atividade UTP-C, usaremos como base esses diagramas para preencher o CFF.xml, que deve conter as seguintes informações: (i) nome do artefato a ser testado, (ii) tipo de artefato, (ii) testes a serem executados para validar algum artefato, (iii) log que terá o resultado das execuções, e (iv) critério de seleção a ser respeitado quando for executar os testes. Detalhes do CFF.xml são apresentados na Seção 4.2.

7.1.7. Critérios de Seleção dos Testes no JAAF+T

Critérios de seleção ajudam a selecionar quais testes deverão ser executados para validar algum artefato. Perceba que o arquivo CFF.xml informa o id do critério relacionado a algum artefato. Esse identificador é definido no arquivo CEF.xml, cujos critérios do SUT devem ser representados.

O arquivo CEF.xml permite a definição de critérios simples até mais elaborados. Um exemplo de critério seria permitir a execução de testes que sejam funcionais, relacionados ao nível de teste de sistema, possuam alta prioridade, sejam executados de forma automática e que tenham o risco de produto médio ou alto. Consequentemente, caso algum teste definido em um suíte do CFF.xml não atenda o critério definido, esse teste não é executado. Visando facilitar o preenchimento do arquivo CEF.xml, usaremos como base um diagrama de classe da UTP-C criado e que modela esses critérios.

7.2. O Exemplo do Mercado Virtual

A fim de exemplificar o uso da linguagem de modelagem UTP-C e do framework JAAF+T, usamos o exemplo de mercados virtuais, que representa mercados na Web, onde usuários podem comprar itens. Na instância criada, cada usuário é representado por um agente comprador, que solicita a compra de livros a um agente vendedor (representante de um mercado virtual, como, por exemplo, Amazon, Ebay, Saraiva, etc.).

Quando o usuário comprador realiza seu cadastro no sistema, ele deve informar qual seu mercado preferido, já que a partir dessa informação, o agente comprador procura realizar a compra com o agente vendedor representante desse mercado. Após o cadastramento, o usuário poderá solicitar a compra de livros a partir do fornecimento das seguintes informações: (i) título(s) do(s) livro(s), (ii) nome(s) do autor(es), (iii) se o(s) livro(s) deve(m) ser novo(s) ou se pode(m) ser usado(s). A partir dessas informações, o agente comprador verifica se o agente vendedor possui o item disponível. Perceba que cada solicitação de compra pode ser composta por informações de um ou mais livros.

Caso o agente vendedor tenha o(s) livro(s) desejado(s), as informações do(s) item(ns) são(sejam) apresentadas ao usuário para que a compra possa ser efetivada. No entanto, quando o agente vendedor não possui disponibilidade de algum item para venda, ou quando o preço sugerido pelo mercado é maior do que o usuário está disposto a pagar, tal usuário pode solicitar a compra para outro mercado. Para que isso seja possível, o usuário deve definir um critério de compra que será usado por seu agente comprador a fim de encontrar outro agente vendedor (mercado) que o atenda. Um critério de compra é composto pelas seguintes informações:

- **Valor máximo de pagamento:** O usuário deve informar o valor máximo em dinheiro que está disposto a pagar para cada livro que faz parte da sua solicitação de compra.
- **Livros novos e usados:** O usuário deve informar se deseja comprar livros novos ou usados.
- **Reputação do agente vendedor:** Como cada mercado possui um agente vendedor representante, o usuário pode solicitar que seu agente comprador negocie exclusivamente com agentes vendedores que possuam reputação maior ou igual à reputação informada pelo usuário.

Como o sistema armazena e disponibiliza a informação de quais compradores negociaram com quais vendedores, o agente comprador pode solicitar a outros agentes compradores a reputação que deles em relação a algum vendedor. Essa reputação é chamada de reputação de testemunho, apresentada pelo modelo FIRE (Huynh et al., 2004). Nesse modelo, o valor possível das reputações é de [-1,+1], onde -1 equivale a uma reputação absolutamente negativa, +1 absolutamente positiva, enquanto que 0 (zero) é uma reputação neutra ou incerta. Esses valores foram considerados no sistema.

A ideia geral da reputação de testemunho é ilustrada na Figura 37, onde um agente A (comprador) solicita a reputação a outros agentes compradores (agentes C, D e E) em relação ao agente vendedor B. A partir das informações recebidas, o agente A encontra a reputação final do vendedor a partir da média das reputações fornecidas pelos agentes C, D e E.

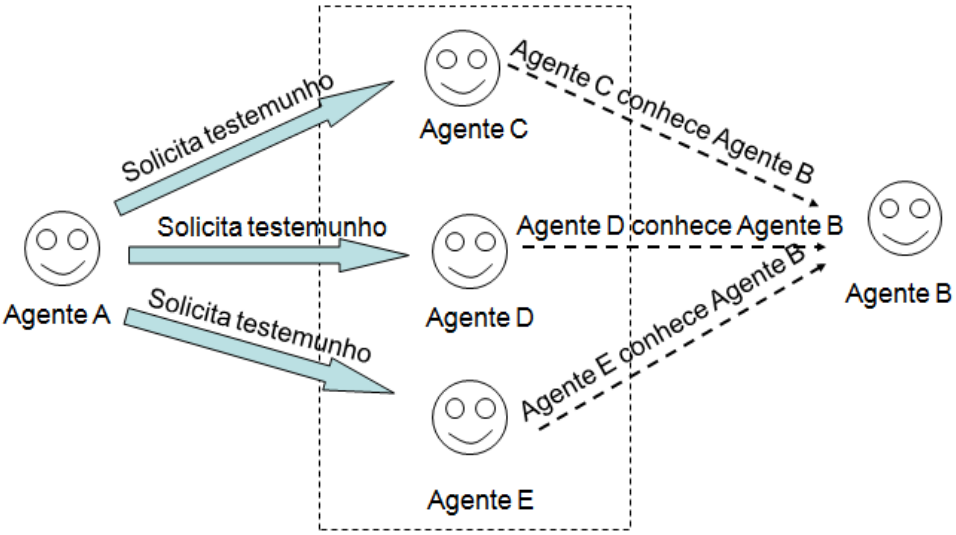

Figura 37. Visão geral da reputação de testemunho.

Quando o agente comprador não encontra ao menos um vendedor que atenda 100% do critério de compra definido pelo usuário, o vendedor que chegou mais próximo em atendê-lo é escolhido. Ao encontrá-lo, a proposta de venda é apresentada ao usuário, aguardando a confirmação da compra. Após a confirmação, o agente comprador conclui a negociação com o agente vendedor e em seguida incrementa a reputação do vendedor em +0.1. Todo vendedor inicia com reputação 0 (zero) para cada agente comprador.

A decisão de escolha de qual vendedor será usado para atender o critério de compra fornecido pelo usuário, é feita a partir de um processo de autoadaptação executado pelo agente comprador. Para entendermos quais testes são executados, assim como o processo de autoadaptação adotado pelo agente, a seguir é apresentada a modelagem UTP-C realizada, seguida pela explicação detalhada desse processo.

7.2.1. Modelagem a partir da UTP-C

Nesta subseção são apresentados os diagramas UTP-C criados seguindo a abordagem apresentada na Seção 7.1. Inicialmente são apresentados os diagramas de classe estáticos, seguido pelo diagrama de atividade dinâmico definido para o sistema.

7.2.1.1. Diagramas Estáticos

Na Figura 38 é apresentado um diagrama de classes com cinco test contexts que poderão ser executados por agentes compradores no mercado virtual para avaliar agentes vendedores. Os test contexts modelados são os seguintes:

1. **TestInteraction:** Verifica se a conexão com algum agente vendedor está sendo possível.
2. **TestAvailableItem:** Testa se o agente vendedor possui o item desejado para compra.
3. **TestVerifyIfItemIsNewOrUsed:** Analisa se o item disponível para venda é usado ou novo, atendendo assim o critério fornecido pelo usuário comprador.
4. **TestVerifyValueItem:** Testa se o valor do item disponível para venda é menor ou igual ao valor informado no critério de compra.

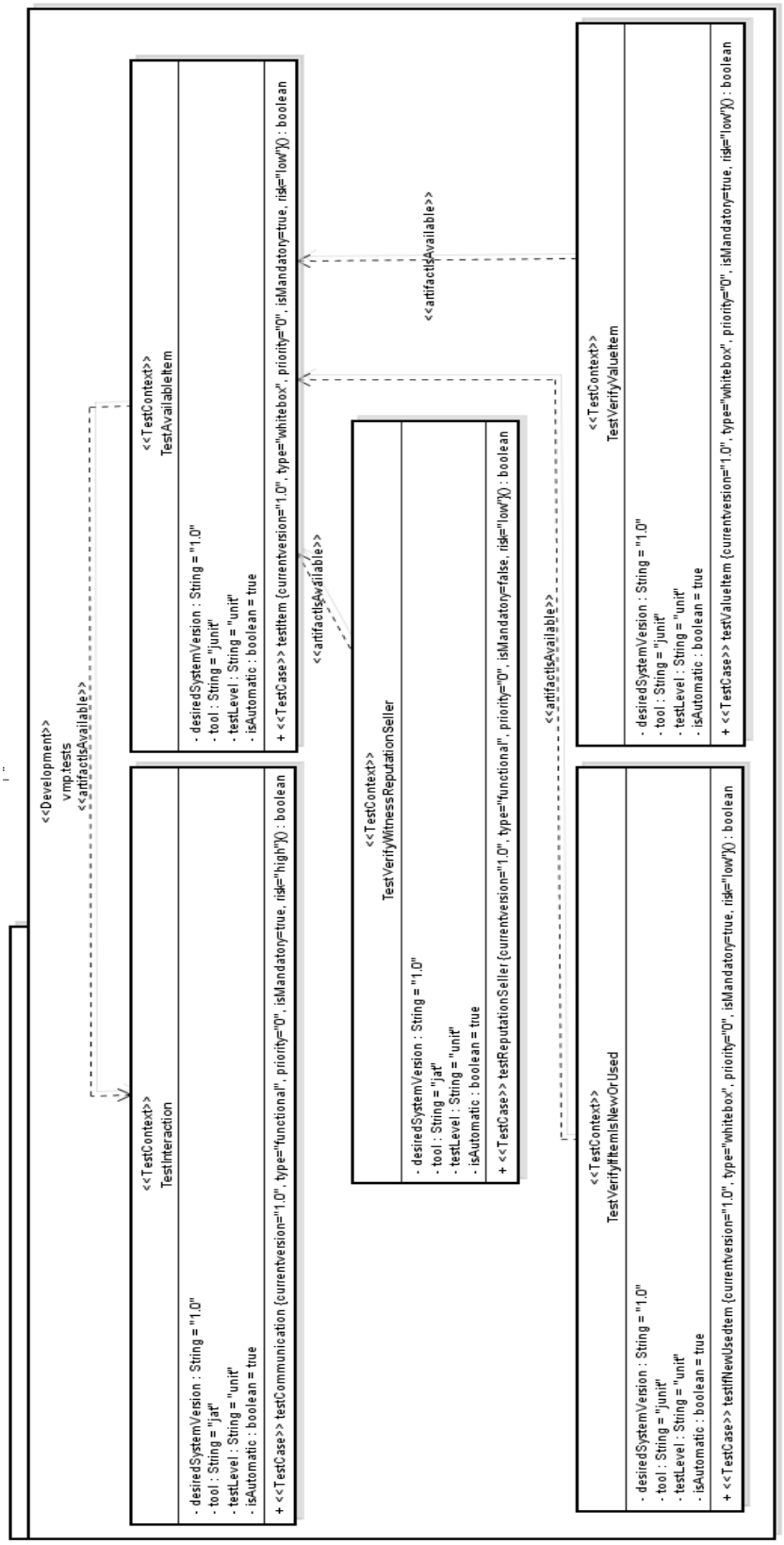

Figura 38. Test Contexts do mercado virtual.

5. **TestVerifyWitnessReputationSeller:** Analisa se a reputação do agente vendedor é maior ou igual à reputação fornecida no critério de compra. Para realizar tal teste são consultadas as reputações de testemunho dos últimos três compradores do agente vendedor em análise. Caso a média das reputações fornecidas seja maior ou igual à reputação definida no critério de compra, o vendedor está aprovado para negociação.

Perceba que no diagrama de classes da Figura 38 são apresentadas informações detalhadas de cada test context, como, por exemplo, o pacote que eles estão armazenados (*vmp.tests*), seu nível de teste, ferramenta usada para executá-lo, versão do SUT que o teste deve estar atualizado e seu caso de teste.

Já na Figura 39 é apresentada uma visão que permite identificar quais testes são novos (uso do conceito classificação de teste) para o sistema em teste. Nesse exemplo, os cinco test context apresentados na Figura 33 são considerados novos. Quando houver a evolução desse sistema, novas classificações de teste podem ser definidas, gerando mudanças de classificação para esses testes modelados.

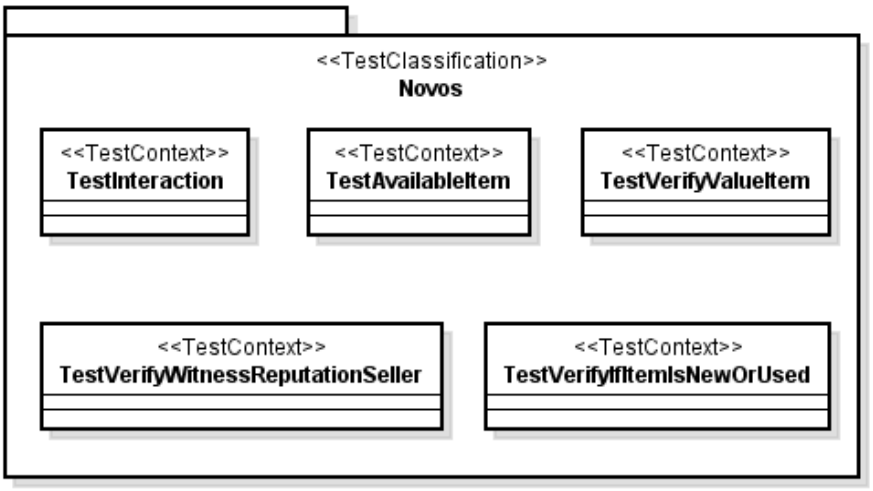

Figura 39. Modelagem de classificação de teste no mercado virtual.

Já a Figura 40 ilustra os critérios de seleção de teste definidos para o SUT. Dois critérios foram definidos: Criterion1 e Criterion2. Criterion2 permite a execução de testes unitários com prioridade 0 (zero), e que sejam funcionais ou

caixa branca. Já Criterion1 considerada que além das restrições definidas no Criterion2, os testes também devem ser obrigatórios. Vale ressaltar que o sistema considera que quanto menor o valor definido no atributo priority, maior a prioridade do teste. Logo, testes com o valor 1 possuem menor prioridade de execução do que os testes com o valor 0.

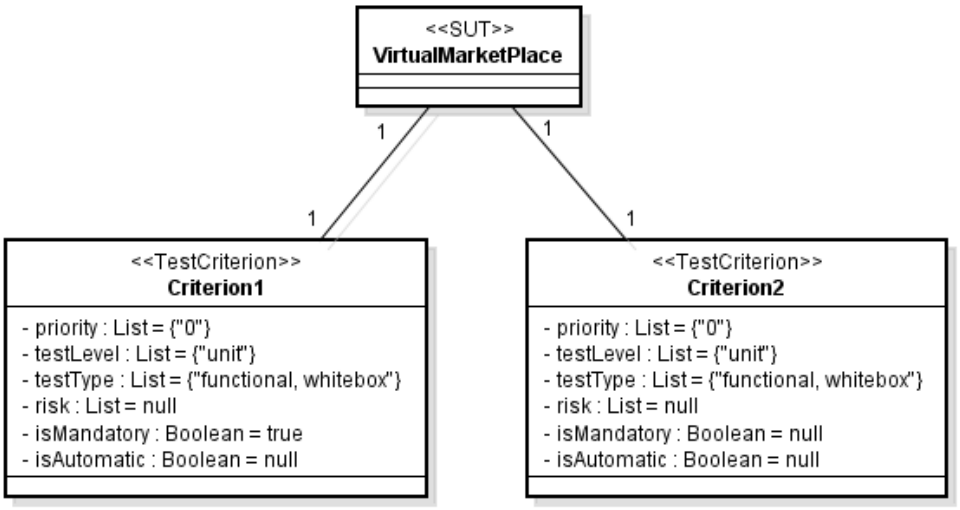

Figura 40. Critérios do mercado virtual, modelados a partir da UTP-C.

Por fim, a Figura 41 apresenta um diagrama simplificado com a suíte de teste responsável por executar os cinco testes definidos no sistema. A ordem de execução desse suíte é detalhada no diagrama de atividades apresentado na próxima subseção.

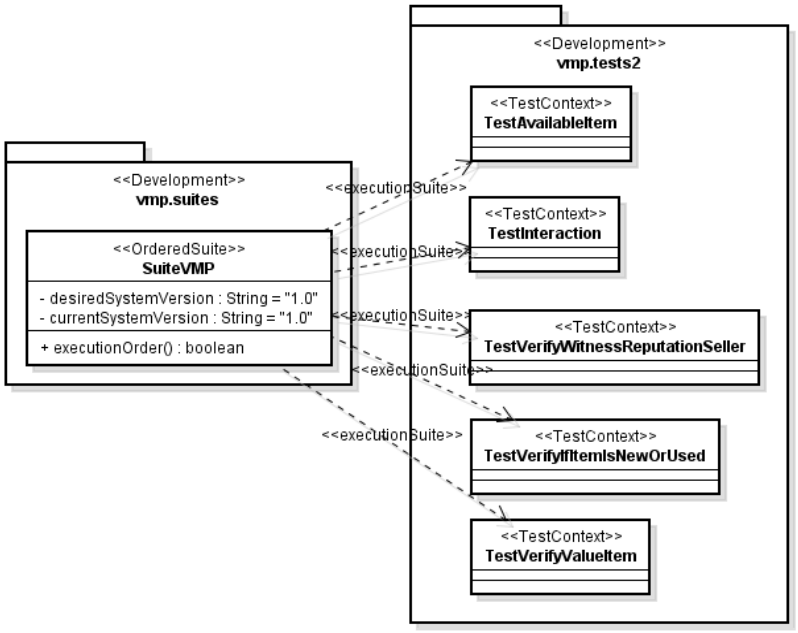

Figura 41. Suite do mercado virtual modelada em um diagrama de classes.

7.2.1.2. Diagrama Dinâmico

Na Figura 42 é apresentado o diagrama de atividade responsável por descrever o fluxo de execução dos test contexts executados a partir do SuiteVMP apresentado no diagrama de classe da Figura 41. Para informar que o diagrama de atividade está relacionado com uma suíte de teste, o estereótipo <<OrderedSuite>> foi incluído no nome do diagrama.

No diagrama há uma entidade comentário responsável por apresentar informações referentes ao artefato a ser testado (agente vendedor). Apesar de cinco testes serem executados a partir desse suite, todos somente serão executados caso atendam o critério de seleção definido no atributo criterionid. Na modelagem ilustrada, consideramos que o critério adotado é o Criterion1 (ver Figura 40). Portanto, só serão executados os testes que sejam unitários com prioridade média ou alta, funcionais ou caixa branca, possuam prioridade zero e sejam considerados obrigatórios.

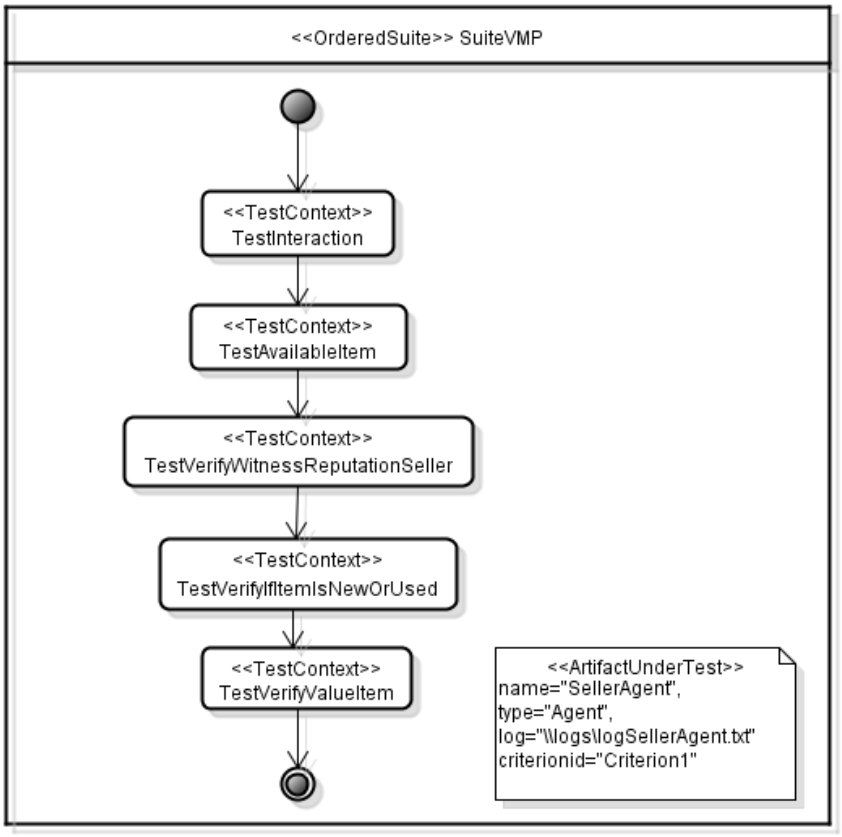

Figura 42. Diagrama de atividades do suíte do sistema mercado virtual.

7.2.2. Autoadaptação Realizada pelo Agente Comprador

Nesta subseção é apresentado o processo de autoadaptação executado pelos agentes compradores, seguido pela descrição dos arquivos TF.xml, DF.xml, CFF.xml e CEF.xml usados nesse processo.

7.2.2.1. Processo de Autoadaptação

Quando um agente comprador não consegue realizar a compra com um agente vendedor representante de um mercado, previamente escolhido pelo usuário no seu cadastramento, o processo padrão de autoadaptação oferecido pelo JAAF+T (ver Figura 5) é executado pelo agente comprador. Esse processo procura por outro agente vendedor que consiga atender a solicitação de compra respeitando o critério de compra informado pelo usuário. Esse processo é composto pelas seguintes atividades: Coleta, Análise, Decisão, Teste, Validação e Efetuar.

A atividade de Coleta é executada assim que o agente comprador é informado pelo usuário que deseja encontrar outro mercado que atenda o critério de compra fornecido. Dessa forma, as informações do critério são coletadas, assim como as informações da última opção de venda apresentada ao usuário.

Em seguida, a atividade Análise é executada a fim de encontrar a causa da compra não ter sido anteriormente realizada. Essa análise é feita a partir das informações coletadas, que permitem comparar o critério de compra do usuário com as informações da última tentativa de compra. Exemplos de possíveis causas que possam ter impedido a compra podem ser as seguintes: (i) indisponibilidade de algum item desejado, (ii) baixa reputação do agente vendedor, (iii) alto preço do item, ou (iv) problema na comunicação com o agente vendedor. A partir desse diagnóstico, o agente comprador decrementa em -0.1 a reputação do agente vendedor envolvido quando mais de uma dessas causas são identificadas.

Logo depois a atividade Decisão é executada para escolher um agente vendedor candidato que possa atender a solicitação de compra. Tal identificação é feita pelo serviço de páginas amarelas, oferecido pelo framework JADE (Bellifemine et al., 2012a; Bellifemine et al., 2012b), que permite conhecer todos os agentes vendedores do sistema. Assim que um agente vendedor é escolhido, a atividade Teste é executada.

A atividade Teste executa os testes modelados no diagrama de classes ilustrado na Figura 38. Assim que os testes são executados, a atividade Validação analisa os resultados gerados para decidir se o agente vendedor em análise pode ser utilizado para atender a compra solicitada. Caso possa ser usado, a atividade Efetuar é executada para realizar as configurações necessárias no agente comprador para que possa utilizá-lo. Caso contrário, a atividade Decisão é executada novamente para escolher outro agente vendedor candidato a fim de avaliá-lo. Esse ciclo de execução continua até que um agente vendedor seja considerado apto para participar da negociação desejada.

7.2.2.2. Testes Executados na Aplicação

Para que os testes modelados a partir da UTP-C possam ser manipulados a partir do framework JAAF+T, eles devem ser representados no arquivo TF.xml. Código 4 mostra o arquivo TF.xml usado na aplicação. Esse arquivo possui dois conjuntos de teste (“setJATTests” e “setJUnitTests”) e cinco testes

(testInteraction, testAvailableItem, testVerifyifitemisNewOrUsed, testVerifyValueItem e testWitnessReputrationSeller). Os modelos da UTP-C foram usados como base para preencher esse arquivo, já que possuem informações equivalentes, como, por exemplo, id do teste, nível e tipo de teste, se serão executados de forma automática ou manual, ferramenta usada para executá-los (JAT ou JUnit), prioridade, risco do produto, e se são obrigatórios para execução.

```
<tf>
  <setTest id="setJATTests" >
     <test id="testInteraction" isAutomatic="true"
testLevel="unit" testType="functional" isMandatory="true"
context="VMP v1.0">
            <classpath>
            vmp.tests.TestInteraction
            </classpath>
            <priority>0</priority>
            <risk>high</risk>
            <tool>JAT</tool>
      </test>
  </setTest>
  <setTest id="setJUnitTests" >
      <test id="testAvailableItem" isAutomatic="true"
testLevel="unit" testType="whitebox" isMandatory="true"
context="VMP v1.0">
            <classpath>
            vmp.tests.TestAvailableItem
            </classpath>
            <priority>0</priority>
            <risk>low</risk>
            <tool>JUnit</tool>
      </test>
      <test id="testVerifyIfItemisNewOrUsed" isAutomatic="true"
testLevel="unit" testType="whitebox" isMandatory="true"
context="VMP v1.0">
            <classpath>
            vmp.tests.TestVerifyIfItemisNewOrUsed
            </classpath>
            <priority>0</priority>
            <risk>low</risk>
            <tool>JUnit</tool>
        </test>
        <test id="testVerifyValueItem" isAutomatic="true"
testLevel="unit" testType="whitebox" isMandatory="true"
context="VMP v1.0">
            <classpath>
            vmp.tests.TestVerifyValueItem
            </classpath>
            <priority>0</priority>
            <risk>low</risk>
            <tool>JUnit</tool>
        </test>
        <test id="testVerifyWitnessReputationSeller"
isAutomatic="true" testLevel="unit" testType="whitebox"
isMandatory="false" context="VMP v1.0">
            <classpath>
```

```
            vmp.tests.TestVerifyWitnessReputationSeller
            </classpath>
            <priority>0</priority>
            <risk>low</risk>
            <tool>JUnit</tool>
        </test>
 </setTest>
</tf>
```

Código 4. TF.xml da instância mercado virtual do JAAF+T.

7.2.2.3. Dados Usados pelos Testes

O JAAF+T oferece a classe *ParserDF* para permitir a recuperação das informações representadas no DF.xml, assim como permitir a adição de novos elementos *data* e condições de teste em tempo de execução. Dessa forma, foi definido que o sistema mercado virtual adiciona e atualiza elementos *data* toda vez que o usuário comprador informa quais livros devem ser comprados e qual o critério de compra a ser respeitado.

No Código 5 é apresentado parte do arquivo DF.xml usado no sistema. A mesma ideia apresentada nesse exemplo é aplicada nos outros testes executados pelos agentes compradores.

Cada elemento *data* do DF.xml está relacionado a algum teste definido no arquivo TF.xml. Perceba que o elemento *data* do Código 5 está relacionado ao teste *testVerifyValueItem* e a um comprador chamado Roberto. Essa identificação torna-se imediata, devido à estrutura adotada pelo sistema em relação o identificador (atributo *id*) do elemento *data*. Veja a seguir tal estrutura:

"data<ID do Teste definido no TF.xml>_<Nome do comprador>".

Caso o sistema identifique a existência de um elemento *data* com o mesmo id, esse elemento *data* é substituído de forma automática, assim que o mesmo usuário informa um novo critério de compra e/ou outros livros a serem comprados.

TestVerifyValueItem, por exemplo, é o teste responsável por validar se o valor do livro desejado é maior ou igual ao valor informado pelo agente vendedor. Quando mais de um livro é solicitado para compra, uma condição de teste (uso do elemento condition) para cada livro é criada no DF.xml, assim

como apresentado no Código 5. Perceba que nesse exemplo dois livros foram solicitados para compra, sendo assim fornecido um valor máximo aceitável para compra em cada livro (informação fornecida pelo usuário).

```
<df>
  ...
  <data id="dataTestVerifyValueItem_Roberto">
        <condition id="condition1">
              <setInputs>
                    <input id="titleBook" value="Advanced Software Testing"/>
                    <input id="Author" value="Rex Black"/>
              </setInputs>
              <setOutputs>
                    <output id="maxPrice" value="100"/>
              </setOutputs>
        </condition>
        <condition id="condition2">
              <setInputs>
                    <input id="titleBook"
                    value="Model-Driven Testing: Using the UML Testing Profile"/>
                    <input id="Author" value="Baker"/>
              </setInputs>
              <setOutputs>
                    <output id="maxPrice" value="80"/>
              </setOutputs>
        </condition>
  </data>
  ...
</df>
```

Código 5. DF.xml da instância mercado virtual do JAAF+T.

O código fonte do teste *TestVerifyValueItem* é apresentado no Código 6. Perceba que a classe correspondente implementa a interface *TestUsingDFInterface* para permitir a manipulação dos dados definidos no DF.xml. O método *getIdData()* informa o id do elemento *data* a ser usado, enquanto que o método *getIdCondition()* retorna o valor null, a fim de informar que todas as condições que fazem parte do elemento *data* devem ser executadas. Esses dois métodos são definidos pela interface *TestUsingDFInterface.*

InfoRequestBuyer é uma classe composta por informações do usuário comprador, como, por exemplo, nome, endereço e data de nascimento. Essa classe é usada no método *getIdData()* para informar o id do elemento *data* a ser usado.

Já a classe *AvailableBooksBySeller* contêm todos os livros disponíveis para venda a partir de um agente vendedor. A partir dessa classe torna-se possível recuperar o preço oferecido pelo vendedor passando como parâmetro o nome do livro e o nome do autor a partir do método *getAvailablePriceToSell*. Com o preço em mãos, finalmente o teste pode comparar o preço obtido com o

preço esperado (definido no DF.xml), e assim concluir se o agente vendedor pode ser escolhido para a negociação desejada.

```
public class TestVerifyValueItem extends TestCase implements TestUsingDFInterface{

    @Override
    public String getIdData(){
        // TODO Auto-generated method stub
        return "dataTestVerifyValueItem_"+InfoRequestBuyer.getInstance().getNameBuyerUser();
    }
    @Override
    public String getIdCondition(){
        // TODO Auto-generated method stub
        return null;
    }

    public void testValueItem(){
        boolean isPossibletoBuyExpected = true;
        boolean isPossibletoBuyResult=false;
        Condition condition = CurrentConditionDFToExecute.getInstance().getCondition();

        String titleBook = condition.getValueByInputId("titleBook");
        String nameAuthor = condition.getValueByInputId("Author");
        Double expectedResult = Double.parseDouble(condition.getValueByOutputId("maxPrice"));

        AvailableBooksBySeller availableBooksBySeller = AvailableBooksBySeller.getInstance();
        Double resultObtained = availableBooksBySeller.getPriceAvailableBookToSell(titleBook, nameAuthor);

        if (expectedResult <= resultObtained)
               isPossibletoBuyResult = true;

        assertEquals(isPossibletoBuyExpected,isPossibletoBuyResult);
    }
}
```

Código 6. Código fonte do teste *TestVerifyValueItem*.

7.2.2.4. Fluxo de Execução dos Testes

No arquivo CFF.xml foi definida a suíte de execução ilustrada no Código 7 para testar os agentes vendedores. Esse arquivo foi preenchido a partir do diagrama de atividade UTP-C ilustrado na Figura 42. Vale ressaltar que a ordem colocada dos elementos *test* no CFF.xml define a ordem de execução do suíte. Além disso, as informações dos atributos do elemento *artifact* estão modeladas na entidade comentário do diagrama da atividade da Figura 42.

```
<cff>
      <artifact id="SuiteVMP" type="Agent" logPath="\\logs\logSellerAgent.txt"
      criterionID="Criterion1">
            <test type="test" id="testInteraction"/>
            <test type="test" id="testAvailableItem"/>
            <test type="test" id="testVerifyWitnessReputationSeller"/>
            <test type="test" id="testVerifyIfItemisNewOrUsed"/>
            <test type="test" id="testVerifyValueItem"/>
      </artifact>
</cff>
```

Código 7. CFF.xml da instância mercado virtual do JAAF+T.

7.2.2.5. Critérios de Seleção de Testes para Execução

No Código 8 é apresentado o arquivo CEF.xml com os critérios de seleção que o sistema pode considerar quando for executar testes para validar algum agente vendedor. Para preenchê-lo, usamos como base o diagrama de classe da Figura 40, que ilustra os critérios definidos para o sistema.

A principal diferença dos dois critérios definidos no CEF.xml é que o Criterion1 permite para seleção somente de testes obrigatórios, enquanto que no Criterion2 tanto obrigatórios como opcionais podem ser executados. Dessa forma, o arquivo CFF.xml ilustrado no Código 7, impede que o teste *TestVerifyWitnessReputationSeller* seja executado, já que o artefato em teste está relacionado com o Criterion1 (ver Código 4). Tal critério foi definido inicialmente enquanto não há um número substancial de negociações que permitam definir reputações aos agentes vendedores. Assim que esse número for alcançado, os desenvolvedores/testadores podem alterar o critério no arquivo CFF.xml para Criterion2.

```
<CEF>
    <criterion id="Criterion1">
        <priority>0</priority>
        <testLevel>unit</testLevel>
        <testType>functional</testType>
        <testType>white box</testType>
        <isMandatory>true</isMandatory>
        <context>VMP v1.0</context>
    </criterion>
    <criterion id="Criterion2">
        <testLevel>unit</testLevel>
        <testType>functional</testType>
        <testType>white box</testType>
        <priority>0</priority>
        <context>VMP v1.0</context>
    </criterion>
</CEF>
```

Código 8. CEF.xml da instância mercado virtual do JAAF+T.

Para ajudar a esclarecer os critérios definidos pelo CEF.xml, as condições definidas a partir deles são apresentadas a seguir.

Criterion1:

SE (teste tiver prioridade zero) **E**
(teste está relacionado ao nível de teste unitário) **E**
(teste for do tipo funcional **OU** caixa branca) **E**
(teste for obrigatório) **E**
(teste estar relacionado ao contexto “VMP v1.0”) **ENTÃO**
TESTE VÁLIDO PARA EXECUÇÃO

Criterion2:
SE (teste tiver prioridade zero) **E**
(teste está relacionado ao nível de teste unitário) **E**
(teste for do tipo funcional **OU** caixa branca) **E**
(teste estar relacionado ao contexto “VMP v1.0”) **ENTÃO**
TESTE VÁLIDO PARA EXECUÇÃO

7.3. Outros Exemplos Usando UTP-C e JAAF+T

Outros sistemas foram instanciados a partir do JAAF+T para a criação de agentes autoadaptativos que fossem capazes de validar suas adaptações. O primeiro deles foi relacionado ao domínio de deslizamento de terra apresentado em (Costa et al., 2010a). Deslizamentos são fenômenos naturais difíceis de predizer, já que dependem de muitos fatores imprevisíveis. O número anual de deslizamentos é na ordem dos milhares, enquanto que os danos de infraestrutura estão na ordem dos bilhões de dólares (Karam, 2005). Como há a necessidade de sistematicamente lidar com tais fatores, um dos principais desafios enfrentados pelos especialistas é decidir o modelo de configuração mais apropriado para gerar mapas de suscetibilidade (MSs). MSs são mapas responsáveis por indicar os lugares com riscos de deslizamento em uma área específica. A partir do mapa gerado, torna-se possível identificar as áreas com maiores riscos em uma dada região.

Considerando esse contexto, implementamos um sistema multiagente (SMA) composto por agentes que possuíam como objetivo central gerar um bom MS que apresentasse os lugares com risco de deslizamento na cidade do Rio de Janeiro, Brasil. O SMA procurou encontrar um modelo de suscetibilidade [Soeters e Westen, 1996] que fosse capaz de criar um bom MS baseado em dados providos por um usuário do sistema. Cada agente adapta seu comportamento enquanto procura o modelo mais apropriado a partir de uma família de modelos que cada agente possui. Um modelo é implementado como

um serviço web (*web service*), e cada agente usa arquivos OWL-S (Martin et al., 2012) para encontrar tais modelos. OWL-S é uma ontologia, no âmbito do framework de web semântica baseado na OWL, para descrever serviços de web semântica. Exemplos de testes executados pelos agentes geradores de MS estão relacionados à: (i) verificação se algum web-service está online, (ii) comparação de uma parte de um mapa gerado por um web-service com uma base histórica da região analisada para decidir se o serviço em análise será ou não usado, etc. Pudemos sem dificuldades preencher os arquivos XML do framework JAAF+T com as informações necessárias para que agentes pudessem coordenar a execução dos seus testes.

Já o outro sistema instanciado pelo JAAF+T e que também foi modelado a partir do UTP-C está relacionado ao controle de suprimento e estoque de petróleo e derivados do petróleo (ex: querosene, gasolina, óleo lubrificante, etc.). Detalhes da instanciação do JAAF+T são apresentadas em (Costa et al., 2011a), enquanto que a modelagem UTP-C em (Costa et al., 2012a) Algumas das principais funcionalidades do sistema são as seguintes: (i) registrar rotas (ex: caminhos) baseadas em dutos e navios que possam transportar produtos derivados do petróleo; (ii) predizer quando tais produtos irão chegar em pontos estratégicos localizados no Brasil (ex: terminais ou refinarias); (iii) planejar as melhores rotas que permitam realizar o transporte dos produtos desejados; etc.

Visando identificar um conjunto de caminhos que consiga atender as necessidades de transporte dos produtos no Brasil, um dos módulos da aplicação foi modelado para aplicar o paradigma de autoadaptação com autoteste considerando um sistema multiagente. A ideia central do módulo é utilizar um agente Manager que busque rotas que consigam atender demandas de transporte de produtos informadas por usuários do sistema. Essas rotas são buscadas a partir da comunicação com agentes especialistas. Agentes especialistas são aqueles que conhecem em detalhes as rotas que envolvem específicos tipos de pontos estratégicos (como, por exemplo, terminais e refinarias). Assim, a autoadaptação é realizada pelo agente Manager a fim de decidir quais agentes especialistas devem ser usados para recuperar e apresentar rotas que atendam o usuário do sistema. Exemplos de testes executados são os seguintes: (i) verificar se está sendo possível trocar mensagens com algum agente especialista, (ii) testar se a base de dados usada pelo agente especialista está online, e (iii) se o agente especialista encontra alguma rota que ligue de forma direta um ponto de origem e destino solicitado pelo usuário. Assim como no sistema de deslizamento de terra, não houve

dificuldades em preencher os XMLs usados pelo JAAF+T. No entanto, como foi inicialmente realizada a modelagem UTP-C, achamos mais fácil a criação dos arquivos XML usando como base esses diagramas criados.

7.4. Discussão

Para exemplificar o uso da linguagem UTP-C e do framework JAAF+T, foi modelado e desenvolvido um sistema multiagente para o domínio mercado virtual. Em relação à modelagem, diagramas de classe estáticos e de atividade dinâmico foram criados. Os diagramas estáticos foram usados para modelar as informações estáticas dos testes, casos de teste, suítes, e critérios de seleção. Já o diagrama dinâmico permitiu modelar o fluxo de execução de suítes de teste.

Como os modelos UTP-C foram criados a partir de uma ferramenta gratuita de modelagem chamada de Astah community (Astah, 2012), todas as propriedades dos casos de teste ficaram expostas no diagrama de classe ilustrado na Figura 38. No entanto, algumas dessas propriedades poderiam ser omitidas para evitar que diagramas do tipo fiquem extremamente grandes em outros projetos. Uma solução alternativa é o uso de outras ferramentas de modelagem oferecidas na literatura e que permitam esconder essas informações. Um exemplo de ferramenta que permite a aplicação dessa ideia é a Rational Software Architecture (RSA, 2012).

Em relação à criação do mercado virtual a partir do JAAF+T, não foram representados no arquivo TF.xml testes manuais. No entanto, qualquer situação que precise da intervenção humana para completar a validação de algum artefato podem ser adicionadas facilmente no arquivo TF.xml. O JAAF+T considera que em testes manuais, o agente autoadaptativo fica em estado de espera até que seja fornecido um feedback a partir de alguém. Tal feedback é fornecido a partir do método “finishedExecution(Boolean result)” da classe *WaintingResponseManualTest*. Assim que esse método é chamado passando como parâmetro o resultado do teste (sucesso - true ou falha - false), os testes pendentes são executados.

Tanto a UTP-C como o JAAF+T podem ser usados, estendidos e aplicados para uma variedade de domínios, assim como apresentado na Seção 7.3. Além disso, a UTP-C pode ser usada para sistemas desenvolvidos a partir de diferentes tecnologias (e.g. J2EE, .NET), já que a modelagem é baseada em UML e portanto voltada ao paradigma orientado a objetos.

Diversos trabalhos propostos voltados para autoadaptação (Dobson et al., 2006; Garlan et al., 2004; Kaiser et al., 2003; Neto et al., 2009a) e trabalhos que também incluem a técnica de autoteste (Denaro et al., 2007; Stevens et al., 2007; Wen et al., 2005) são criados de forma *ad hoc*. Dessa forma, essas abordagens pré-definem processos de autoadaptação amarrados ao domínio da aplicação, gerando carência de abordagens genéricas na literatura. Visando atender essa carência, o JAAF+T foi proposto. Tal framework oferece uma solução flexível que permite a criação de agentes de software capazes de executar diferentes processos de autoadaptação independente do domínio de aplicação. Adicionalmente, o framework permite a manipulação de diversas informações relacionadas à área de teste para que agentes coordenem a execução dos testes responsáveis por validar suas adaptações.

Em relação à modelagem de testes, (Baker et al. 2007) apresenta uma abordagem de como modelar testes de software a partir da UML Testing Profile (UTP) original. Para que sua abordagem seja atendida, diversos diagramas e entidades devem ser modelados gerando grande esforço do designer. Já a linguagem UTP-C direciona a modelagem das informações realmente voltadas à gerência de testes, evitando que modelos fiquem poluídos com informações adicionais.

Como o JAAF+T permite a representação e manipulação de informações úteis para a coordenação de testes a partir de arquivos XML, usar como base diagramas UTP-C ajuda no trabalho de criação e edição desses arquivos. Isso acontece porque a UTP-C oferece uma visão gráfica e mais simplificada das informações contidas em tais arquivos, como, por exemplo: (i) propriedades dos testes, e (ii) relacionamentos dos testes com outras entidades do projeto (suítes, artefatos em teste, dependências com outros testes).

Quando várias partes de um sistema são desenvolvidas em paralelo e integradas quando completadas (comum no modelo de desenvolvimento incremental), diagramas UTP-C voltados em modelar de forma separada testes responsáveis por validar partes desse sistema (ex: features, componentes, etc.) podem ser criados. Seguindo essa ideia, cada testador responsável por testar alguma dessas partes, pode receber um modelo UTP-C já criado com os testes a serem contemplados no SUT. Outra alternativa seria o próprio testador criar um novo modelo UTP-C. A partir disso, assim que cada parte do sistema for finalizada e testada, espera-se que os modelos correspondentes da UTP-C estejam criados e assim sejam integrados ao projeto. Portanto, ao realizar uma apropriada separação de preocupações do que deve ser testado, diagramas

UTP-C podem ser criados e editados de forma separada sem que afete o trabalho de outros testadores. Lucidchart (Lucidchart, 2012), Collaborative UML Designer (CUD, 2012) e Rational Software Architecture (RSA, 2012) são exemplos de ferramentas que permitem essa forma de trabalho colaborativa em um mesmo projeto de modelos.

Como modelos ajudam e permitem a abstração de informações sem que haja a necessidade de representar detalhes de desenvolvimento, esses modelos podem ser usados como base para tomadas de decisão envolvendo membros de diferentes equipes (teste, requisitos, desenvolvimento, etc). Logo, essas decisões podem estar relacionadas a diversas atividades, como, por exemplo, planejamento e acompanhamento de testes.

Boa parte dessas informações manipuladas pela UTP-C não são consideradas por conhecidas linguagens de modelagem de teste oferecidas na literatura, como, por exemplo, a UTP original (Baker et al., 2007), AML (Trost e Cavarra, 2012) e UTML (UTML, 2012). Mesmo manipulando importantes conceitos de teste, elas não proveem a modelagem de relevantes informações para gerêcia dos testes e que permitam, por exemplo, a identificação de (i) qual versão do SUT cada teste é capaz de testar, (ii) quais testes são obrigatórios, (iii) quais tipos de teste foram criados (ex: funcional, performance, etc.), (iv) quais níveis de teste estão sendo contemplados (unitário, integração, sistema e aceitação) (v) quais tipos de dependências existem entre os testes modelados (ex: criação de algum artefato, tempo de execução, etc.), (vi) quais testes são automatizados e manuais, (vii) quais critérios de seleção de testes algum SUT possui, (viii) quais artefatos do SUT devem ser validados por quais testes, (viii) quais ferramentas de teste são usadas, etc.

Ferramentas podem ser desenvolvidas para gerar de forma automática relatórios a partir de modelos UTP-C a fim de facilitar e agilizar a leitura e identificação de informações úteis para a atividade de coordenação. Um exemplo seria a geração de relatórios responsáveis por informar quais e quantos testes estão pendentes de atualização para alguma versão do SUT. Na Seção 8.3 é apresentada uma nova ferramenta desenvolvida que permite a geração desse tipo de relatório a partir de modelos UTP-C.Sabe-se que ferramentas de controle de versão, como, por exemplo, CVS (CVS, 2012) e SVN (SVN, 2012b) não realizam versionamentos adequados de modelos UML, pois geralmente o conteúdo desses modelos é armazenado em um único arquivo, similar a um XMI. Como o versionamento é realizado por arquivo, certas informações modeladas não podem ser versionadas de forma independente, como, por

exemplo, métodos de uma classe. Em particular, esse tipo de versionamento é importante, pois permitiria o controle de versão de casos de teste na UTP-C, já que são representadas como métodos em diagramas de classe. Conhecer qual caso de teste está atualizado para qual versão do SUT, é uma informação extremamente importante para o acompanhamento de tarefas. Portanto, tratamentos podem ser incluídos em alguma ferramenta existente de controle de versão (ex: SVN, CVS, etc) para permitir o uso de informações modeladas pela UTP-C e assim realizar um melhor versionamento dos testes.

Além de ferramentas de controle de versão, outras ferramentas voltadas para integração contínua (ex: Hudson, Cruise Control e Continuum) podem ter tratamentos adicionais para permitir o uso de informações representadas em modelos UTP-C. A partir dessas informações tanto relatórios poderiam ser gerados, assim como o envio de notificações relevantes sobre os testes modelados aos envolvidos do projeto.

7.5. Considerações Finais

Neste capítulo foi apresentado o estudo de caso mercado virtual a fim de ilustrar o uso da linguagem UTP-C e do framework JAAF+T em uma aplicação prática. Inicialmente (Seção 7.1) foi definida uma abordagem de uso para facilitar o uso conjunto dessas propostas. Em seguida, foi apresentada em detalhe uma descrição textual do sistema multiagente modelado e desenvolvido (Seção 7.2), além de outros exemplos de SMAs que também usaram a UTP-C e o JAAF+T (Seção 7.3). Por fim, foi apresentada uma discussão (Seção 7.4) a fim de esclarecer quais as principais diferenças e vantagens da abordagem adotada no estudo de caso em relação a outras propostas oferecidas na literatura.

No próximo capítulo são apresentadas ferramentas desenvolvidas para ajudar na geração automática de arquivos XML do JAAF+T, assim como na geração de outros artefatos úteis para equipes de teste a partir de modelos UTP-C. Essas ferramentas não foram mencionadas neste capítulo 7, pois quisemos ilustrar que a UTP-C e o JAAF+T podem ser usados em conjunto sem a necessidade delas, mesmo que tais ferramentas ofereçam bons benefícios.