Uma Metodologia para Verificação de Modelos de Sistemas de Comércio eletrônico

Adriano Pereira Mark Song Gustavo Gorgulho
Wagner Meira Jr. Sérgio Campos
{adrianoc, song, gorgulho, meira, scampos}@dcc.ufmg.br

Departamento de Ciência da Computação
Universidade Federal de Minas Gerais
Caixa Postal 702 - CEP 30.123-970
Belo Horizonte – Minas Gerais – Brasil
Telefone: 55-31-34995860 Fax: 55-31-34995858

Resumo Comércio eletrônico é uma importante área de aplicação da ciência da computação que tem evoluído significativamente nos últimos anos. Entretanto, sistemas de comércio eletrônico são complexos e difíceis de serem projetados corretamente. Atualmente, a maioria das abordagens é *ad-hoc*, o que normalmente torna os sistemas menos confiáveis e implica em alto custo em termos de tempo e financeiros. Neste trabalho propõe-se uma metodologia que utiliza métodos formais, especificamente verificação simbólica de modelos, para projetar aplicações de comércio eletrônico e verificar automaticamente se suas regras de negócio são satisfeitas. Usando a metodologia proposta, o projetista é capaz de identificar, antecipadamente, erros no processo de desenvolvimento do projeto e corrigí-los antes que se propaguem a estágios posteriores da implementação. Dessa forma, torna-se possível gerar aplicações mais confiáveis, desenvolvidas mais rapidamente e a baixo custo. A fim de demonstrar a aplicabilidade e a praticabilidade da técnica proposta, modelou-se e verificou-se uma loja virtual, na qual múltiplos compradores competem para adquirir itens de um produto. O modelo verificado possui mais de 10^{23} estados e a verificação terminou em poucos minutos em um PC com processador AMD Atlhon de 1 GHz e 512 MB de memória. A utilização de verificação automática se mostrou de extrema importância, pois indicou erros difíceis de serem detectados durante o projeto da aplicação como, por exemplo, uma falha do controle de concorrência que permitia que o mesmo artigo fosse vendido para clientes distintos.

Palavras-chave: verificação de modelos, métodos formais, verificação formal, padrões de propriedade, comércio eletrônico.

1 Introdução

Aplicações de comércio eletrônico têm se tornado cada dia mais populares. De uma forma abrangente, podemos definir comércio eletrônico como uso de recursos de rede e tecnologia da informação para facilitar processos centrais ao funcionamento de uma organização [9].

À medida que novos serviços de comércio eletrônico são desenvolvidos, surgem novos tipos de erros, alguns inaceitáveis. Definimos erro como um comportamento inesperado de um programa de computador. Um erro típico que pode ocorrer em um *site* de comércio eletrônico é permitir que dois clientes comprem o mesmo item.

Entretanto, garantir a correção de um sistema de comércio eletrônico não é uma tarefa fácil, dada a variedade de situações que podem causar um erro, muitos deles bem sutis. Tal tarefa é complexa e trabalhosa se somente testes e simulações, que são técnicas comuns de validação de sistemas, são usados.

Métodos formais consistem basicamente do uso de técnicas matemáticas para ajudar na documentação, especificação, projeto, análise e certificação de sistemas de computação. O uso de métodos formais em comércio eletrônico, em especial verificação de modelos, é promissor uma vez que compõe uma técnica robusta e eficiente para verificar a correção de várias propriedades do sistema, principalmente ligadas à detecção precoce de erros.

Este artigo apresenta uma nova metodologia para projetar sistemas de comércio eletrônico aplicando verificação de modelos. A Seção 2 define alguns conceitos importantes sobre verificação automática. Na Seção 3, a metodologia proposta é apresentada, sendo mostrado um exemplo do seu uso na Seção 4. A Seção 5 analisa os trabalhos correlatos e, finalmente, a Seção 6 apresenta algumas conclusões e trabalhos futuros.

2 Verificação de Modelos de Comércio Eletrônico

Garantir a correção do projeto nos estágios iniciais ainda é um desafio no processo de desenvolvimento de qualquer sistema. Os métodos correntes utilizam, em geral, as técnicas de *simulação* e *teste* para validar o projeto. Embora efetivo nos estágios iniciais, seu uso se torna ineficiente à medida que o projeto evolui e os erros vão sendo removidos. Um sério problema com essas técnicas é analisar apenas *alguns* dos possíveis comportamentos do sistema. Não existe a certeza da remoção de todos os erros, visto que um comportamento não explorado pode conter erros fatais. Uma alternativa para o problema é o uso de métodos formais, especificamente *verificação formal*, que permite explorar exaustivamente todos os estados e execuções possíveis da aplicação.

Verificação simbólica de modelos [2] é uma técnica de verificação formal na qual propriedades comportamentais do sistema podem ser verificadas com o uso de um modelo que enumera todos os possíveis estados alcançados pela aplicação. O sistema que está sendo verificado é representado por um grafo de transição de estados (modelo) e as propriedades (comportamento) por fórmulas em lógica temporal. Rótulos são associados aos valores das variáveis do sistema enquanto as transições correspondem aos passos no modelo.

Sistemas de comércio eletrônico podem ser modelados com o uso de um número reduzido de componentes: os produtos comercializados, tais como livros e DVDs; os agentes que atuam sobre esses produtos, como os consumidores e vendedores; e as ações que modificam o estado de um produto, tais como a

reserva ou venda de um item. Assim como nos sistemas comerciais tradicionais, nos sistemas de comércio eletrônico o principal componente continua sendo o produto negociado. Para cada produto comercializado existe um ou mais itens, que são instâncias desse produto.

Cada item de um produto pode ser caracterizado por seu ciclo de vida, podendo ser representado por um diagrama de transição de estados - os estados alcançados durante sua comercialização e as possíveis transições entre eles. Exemplos de estados são *Reservado* ou *Vendido*.

As entidades que interagem com o sistema de comércio eletrônico são denominadas agentes. Exemplos de agentes são os compradores e vendedores. Os agentes realizam ações que podem mudar o estado de um item. Estas correspondem às transições no ciclo de vida do item. Colocar um item na cesta de compra ou cancelar a reserva do mesmo são exemplos de ações.

Serviços, por sua vez, são seqüências de ações sobre os produtos. Enquanto uma ação está associada ao item e geralmente é representada por uma única operação, serviços lidam com o produto realizando uma seqüência de ações transacionais. A compra de um livro é um exemplo de um serviço que envolve ações como reservar o livro, confirmar a reserva e atualizar o estoque.

2.1 Regras de Negócio

A fim de modelar e verificar um sistema de comércio eletrônico, deve-se expressar seu comportamento, que pode ser descrito por suas *regras de negócio*. Uma regra de negócio é uma descrição que especifica o comportamento da aplicação, e é definida usando os componentes previamente citados - produto, agente e ação.

A especificação das regras de negócio não é tarefa trivial, pois demanda o uso de uma notação formal. Em [4,5] foi desenvolvido um sistema de padrões para a especificação de propriedades baseadas na ocorrência de eventos, na ordem dos mesmos e no escopo da aplicação. A fim de facilitar a tradução das regras de negócio para uma notação formal decidimos adotar um sistema de padrões.

Em nosso sistema de padrões, definido para sistemas de comércio eletrônico, um evento consiste de um estado do produto e/ou um conjunto de ações. A ordem define a seqüência na qual os eventos acontecem. E o escopo é a extensão de execução do programa no qual um determinado padrão deve ser satisfeito. Existem basicamente cinco tipos de escopos:

- Global: toda a execução do programa;
- Anterior: a execução até a ocorrência de um dado evento;
- Entre: qualquer execução entre dois eventos pré-determinados;
- Após: a execução após a ocorrência de um dado evento;
- Após-até: como o anterior, mas limitado pela ocorrência de um segundo evento.

Cada padrão possui um escopo, que é definido especificando um evento inicial e final para o padrão. Uma lista com alguns padrões e uma breve descrição segue abaixo:

– Ausência: um dado evento não ocorre no escopo;
– Existência: um dado evento tem que ocorrer no escopo;
– Universalidade: um evento ocorre em todo o escopo;

Na Seção 4 demonstraremos a validação da nossa metodologia através de um estudo de caso e a utilização dos padrões será ilustrada através de exemplos. A próxima seção apresenta resumidamente os conceitos de verificação de modelos e lógica CTL. Estes conceitos serão aplicados na especificação de sistemas de comércio eletrônico.

2.2 Lógica de Árvore de Computação - CTL

A lógica de árvore de computação é usada para expressar as propriedades que serão validadas por um verificador de modelos. Ela é classificada como lógica de ramificação, pois os operadores descrevem uma estrutura ramificada da árvore. A árvore de computação é resultado direto do grafo de transição de estados. Cada caminho na árvore representa uma possível computação do sistema sendo modelado, conforme ilustra a figura 1.

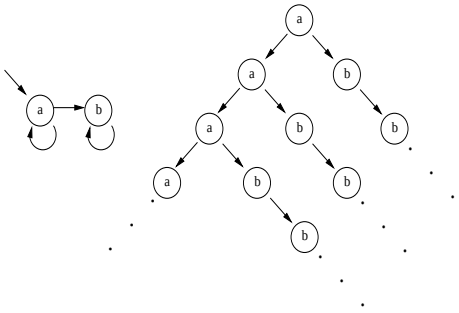

Figura 1. Grafo de transição de estados e sua correspondente árvore de computação.

As fórmulas em CTL se referem a caminhos na árvore de computação derivada do modelo e são construídas a partir de proposições atômicas, conectores lógicos como $\neg$ e $\wedge$, e *operadores temporais*.

Cada operador *CTL* consiste de duas partes: um quantificador de caminho seguido por um quantificador temporal. O quantificador de caminho descreve a estrutura ramificada permitindo identificar: todos os caminhos (A) ou algum caminho na estrutura (E). Os quantificadores temporais descrevem a ordem dos eventos em um dado caminho, por exemplo: (F) em algum estado no caminho, (G) em todos os estados no caminho e (X) no próximo estado. A seguir, apresentamos alguns exemplos de fórmulas *CTL* e sua interpretação:

– $\mathbf{AG}(reserva \rightarrow \mathbf{AF}\ confirma)$: É sempre verdade que a reserva de um item é confirmada.

– **EF**($estoque < 0$): É possível que o estoque de um determinado produto seja negativo.

2.3 Propriedades de um Sistema de Comércio Eletrônico

Propriedades podem ser descritas como fórmulas em *CTL* (Subseção 2.2). Por exemplo, uma propriedade pode descrever que um item pode ser reservado somente se ele estiver disponível. A fim de especificar esta propriedade, um desenvolvedor poderia traduzir este requisito para a seguinte fórmula *CTL*:

```
AG (((estado_item=Disponível) & (serviço=Reservar) & (estoque = v) ->
AX ((estado_item=Reservado) & estoque = v - 1)),
onde v representa um valor inteiro maior do que zero.
```

A partir da análise de aplicações de comércio eletrônico, percebe-se que uma propriedade importante a ser verificada é completeza. Essa propriedade garante que o modelo é consistente, assegurando que todos os estados e ações são alcançados. Para expressar a propriedade de completeza pode-se utilizar o padrão de *Existência*. Adicionalmente, é necessário definir o escopo como *após*, considerando que "Existe no futuro" significa "após o estado/evento corrente".

Outra propriedade importante é a transitividade, que define o próximo estado a ser alcançado após a ocorrência de um evento no estado corrente. É necessário verificar essa propriedade para garantir a correta execução dos serviços que satisfazem as regras de negócio.

A maioria das propriedades a serem verificadas em sistemas de comércio eletrônico estão relacionadas às transações. Uma transação é uma abstração de uma seqüência de operações executadas de forma atômica e confiável. Processamento de transações é importante em quase todos os ambientes de computação modernos que suportam processamento concorrente. Em um servidor de comércio eletrônico, uma transação consiste de uma seqüência de ações que afetam os items existentes, cada ação podendo potencialmente afetar o estado do item. Uma das propriedades mais importantes que deve ser satisfeita neste contexto é assegurar a consistência da transação, isto é, demonstrar que o mecanismo de controle de concorrência implementado é correto e que transações concorrentes não resultam em estados inválidos. Em outras palavras, devemos verificar se as transações são atômicas.

Há três tipos de propriedades relacionadas à consistência das transações que verificamos, a saber:

– Atomicidade: uma transação deve ser executada em sua totalidade, caso contrário, não deve executar absolutamente nenhuma de suas operações. Melhor dizendo, se a mesma não for concluída conforme previsto, seus efeitos devem ser revertidos.
– Consistência: transformações preservam a consistência do estado do produto, isto é, uma transação transforma um estado consistente do produto em outro estado consistente.

– Isolamento: as transações executadas que alteram o estado do produto devem estar isoladas uma das outras, ou seja, uma transação não deve afetar o resultado de outra executada simultaneamente.

A próxima seção detalha a nossa metodologia incorporando o uso de verificação de modelos em uma metodologia incremental para a concepção de sistemas de comércio eletrônico.

3 Metodologia Proposta

Existem várias aplicações de comércio eletrônico, tais como biblioteca digital, livraria virtual e *sites* de leilão. A diferença entre elas são sua natureza e as respectivas regras de negócio. Algumas destas regras são simples, como por exemplo: um item não deve ser vendido para mais de um consumidor. Por outro lado, existem regras específicas à aplicação, como permitir ou não a reserva de um item, prover controle de estoque, ou definir prioridades para transações executadas concorrentemente.

A metodologia proposta, uma extensão da metodologia *CAFE* [9], consiste de uma técnica para se aplicar verificação de modelos no projeto de sistemas de comércio eletrônico. A metodologia *CAFE* explica como especificar um sistema de comércio eletrônico e consideramos que o projetista deve conhecer alguma linguagem formal, tal como SMV [7], para construir seu modelo.

Nossa metodologia é incremental e dividida em 4 níveis. É relevante ressaltar que esta organização foi adotada com o intuito de simplificar a especificação do projeto, mas o projetista pode adotar outra organização, conforme julgar mais adequado ao seu projeto.

O primeiro nível, definido como conceitual, engloba as regras de negócio e a definição do sistema de comércio eletrônico a ser projetado. Quanto mais detalhes o projetista especificar, mais fácil deverá ser aplicar a metodologia e alcançar bons resultados no processo de verificação. O segundo nível, denominado aplicação, modela o ciclo de vida do item a ser comercializado, identificando os tipos de operações (definidas como ações) que são executadas sobre ele e que alteram seu estado. O terceiro deles, conhecido como funcional, modela os serviços providos pelo sistema e introduz o conceito de múltiplos itens. O último nível contempla os componentes do sistema de comércio eletrônico. Ele complementa o escopo da aplicação, modelando sua arquitetura, razão pela qual denomina-se nível arquitetural.

Nas próximas subseções são detalhados os níveis da metodologia formal proposta, utilizando exemplos reais de regras de negócio de comércio eletrônico para explicar como as mesmas podem ser verificadas.

3.1 Nível Conceitual

Da mesma forma que modelos de comércio tradicionais, o objeto central do comércio eletrônico é o *produto*[1] comercializado. Para cada produto comerciali-

[1] Onde produto pode ser um bem ou serviço.

zado, há um ou mais *itens*, que são instâncias de produto. Cada item é caracterizado por um grafo que descreve seu *ciclo de vida*, ou seja, os estados pelos quais um item pode passar. Exemplos de estados são *disponível*, *reservado* ou *vendido*. O *domínio* de um item são os estados nos quais o item pode estar.

As entidades ativas que participam da aplicação de comércio eletrônico são chamadas *agentes*. Exemplos de agentes são os clientes de uma loja e seus vendedores. Os agentes executam *ações* que modificam o estado de um item, ou seja, ações são transições no grafo de ciclo de vida do item. Alocar um item ou comprá-lo são exemplos de ações.

Serviços são seqüências de ações que atuam sobre produtos. Enquanto ações tratam de itens específicos e executam tarefas simples (como por exemplo alocar um item), serviços tratam do produto como um todo e de transações completas. Por exemplo, um serviço pode ser comprar um exemplar de um livro específico. Este serviço pode ser decomposto nas ações: pagar o livro, despachar o livro, e atualizar o estoque.

Formalmente, caracteriza-se uma aplicação de comércio eletrônico por uma tupla $< P, I, D, Ag, Ac, S >$ onde P é o conjunto de produtos da aplicação, I o conjunto de itens comercializados, D os domínios possíveis para os itens (seus ciclos de vida), Ag o conjunto de agentes, Ac o conjunto de ações, e S o conjunto de serviços [9].

Produtos são conjuntos de itens, ou seja, $i \in I$ significa que $i \in p, p \in P$. Os produtos particionam os itens, ou seja, todo item pertence a um e somente um produto. Formalmente, $I = \bigcup_{\forall p \in P} p$ e $p_i \cap p_j = \emptyset$ para $i \neq j$. Domínios são associados a itens, ou seja, cada item i tem como domínio, D_i. Dois itens do mesmo produto têm o mesmo domínio, ou seja, para todos os itens $i, j \in I$, existe um produto p tal que se $i \in p$ e $j \in p$, então $D_i = D_j$.

3.2 Nível de Aplicação

Este nível descreve o sistema de comércio eletrônico em termos do ciclo de vida dos items. É necessário identificar os estados de um item, seus atributos, o conjunto de ações que podem ser executadas sobre ele e os efeitos causados por elas, e os agentes que executam essas ações. Aqui, ainda não estamos interessados nas funcionalidades do *web site* e sua arquitetura.

Os items são modelados pelos seus grafos de ciclo de vida, os quais representam o estado em que cada item pode estar durante a execução do sistema de comércio eletrônico. Um exemplo de grafo de ciclo de vida é ilustrado pela Figura 2. Neste grafo, podem ser vistos estados do item, tais como *Disponível*, ou *Reservado*. Transições representam o efeito das ações tais como reserva de um item ou sua compra.

Cada ação gera uma transição no grafo de ciclo de vida do item e é definida por uma tupla $< a, i, tr > \in Ac$, onde $a \in Ag$ o agente que efetuou a ação, e $i \in I$ o item ao qual a ação foi feita, e $tr \in D_i X D_i$ é a transição associada à ação. Serviços são definidos pela tupla $< p, A >$ onde $p \in P$ e $A = a_1, a_2, \ldots$ é uma seqüência de ações tal que se $a_i = (d_1, d_2)$, $a_{i+1} = (d_3, d_4)$ então $d_2 = d_3$ e $\forall i, d_i \in D_j$ onde D_j é um domínio de um item de p.

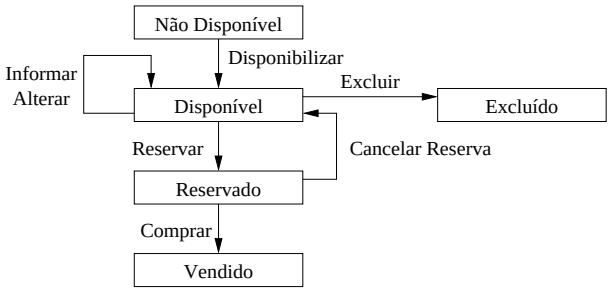

Figura 2. Grafo do ciclo de vida de um item do produto

Cada item de I tem diversos atributos, incluindo o produto a que está associado, seu estado e outras características. Finalmente, os agentes são representados por processos concorrentes que executam serviços, que são seqüências de transições efetuadas no grafo de transição de estados.

Neste modelo, cada estado global representa um estado em cada grafo de ciclo de vida, e transições modelam os efeitos das ações no sistema. Por conseguinte, caminhos no grafo global representam eventos que podem ocorrer no sistema. O ciclo de vida do produto corresponde ao conjunto de todos os ciclos de vida de seus items.

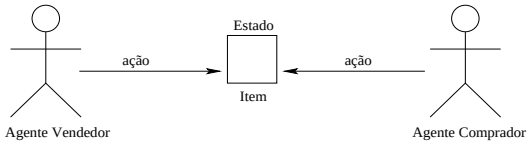

Figura 3. Segundo Nível - Aplicação

Neste nível, é importante verificar a propriedade de completeza do modelo de comércio eletrônico. É importante observar que, portanto, existem apenas ações e estados. Ações, por definição, são transacionais porque representam a menor unidade de execução do sistema, logo as propriedades de atomicidade, consistência e isolamento estão garantidas. A transitividade está relacionada às funcionalidades, logo será importante no nível seguinte, onde serviços passam a ser executados no servidor de comércio eletrônico.

Para verificar a propriedade de completeza do modelo de comércio eletrônico usa-se as fórmulas CTL a seguir descritas, onde S consiste de todos os estados do grafo de ciclo de vida do item comercializado e A, o universo de ações.

```
EF (estado = <S>)        EF (ação = <A>)
```

A Figura 3 ilustra o segundo nível da nossa metodologia, o nível de aplicação. Conforme ela mostra, existem agentes (comprador e vendedor) que representam,

respectivamente, o consumidor e o fornecedor do sistema, e um item descrito por um conjunto de estados. Os agentes executam ações que podem afetar o estado do item.

3.3 Nível Funcional

Este nível introduz o conceito de produto, composto por zero (o produto não está disponível) ou mais itens. O projetista determina as operações que os agentes podem realizar, que são denotadas aqui como serviços. Um serviço é executado sobre produtos e seus efeitos podem alterar ou não o seu estado e o de seus items. O objetivo deste nível é definir com clareza como os serviços são executados e como afetam o produto e o ciclo de vida de seus items.

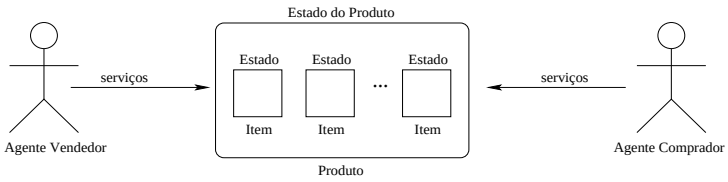

Figura 4. Terceiro Nível - Funcional

Neste nível, é relevante verificar a propriedade da transitividade do modelo. Os agentes executam serviços que alteram o estado do item. Esse estado deve estar consistente com o ciclo de vida do item e com a regra de negócio relacionada a ele. Um exemplo de transitividade pode ser visto a seguir:

```
AG ((estado_item=Não_Disponível & serviço=Disponibilizar) ->
AX (estado_item=Disponível)
```

Também é importante verificar as propriedades de atomicidade, consistência e isolamento. É essencial garantir a consistência entre o estado do produto e de seus itens em dado momento. Mais do que isso, existem agentes realizando serviços concorrentemente, o que pode levar o sistema a um estado inválido. Portanto deve-se garantir a propriedade do isolamento.

Para tornar a explicação mais clara, são dados alguns exemplos da verificação destas propriedades. A primeira propriedade é atomicidade. Se um item está disponível e um serviço *Reservar* é executado por um agente comprador, o item deverá estar reservado no momento seguinte e o estoque do produto deverá ser decrementado.

```
AG ((estado_item=Disponível & serviço=Reservar & estoque=1)
-> AX (estado_item=Reservado & estoque=0))
```

Um exemplo da propriedade de consistência pode ser visto a seguir:

– Se o estoque é igual a zero, então nenhum item deve estar disponível.

```
AG (estoque = 0 -> !estado_produto = Disponível)
```

Um exemplo da propriedade de isolamento é: se há dois itens disponíveis e dois compradores reservam uma unidade deste produto simultaneamente, então, no estado seguinte, o estoque deste produto deverá ser nulo.

```
AG ((comprador1_serviço=Reservar & comprador2_serviço=Reservar
& estoque=2) -> AX (estoque=0))
```

O nível funcional é ilustrado pela Figura 4. Conforme exibido, existem agentes que executam serviços, que podem alterar o estado do produto. Alguns desses serviços como o que reserva um item do produto (serviço *Reservar*), altera também o estado do item.

É importante notar que as propriedades validadas no nível anterior devem ser garantidas também neste nível, e assim por diante. A verificação das propriedades deve ser incremental assim como o processo proposto pela nossa metodologia.

3.4 Nível Arquitetural

Este nível especifica o sistema de comércio eletrônico em termos de seus componentes e o modo como eles interagem. É fundamental enfatizar que ele complementa os níveis previamente descritos, completando a especificação do sistema e descrevendo sua arquitetura.

Neste estágio, o modelo é mais complexo, pois passa a contemplar os componentes da arquitetura do sistema. Por conseguinte, as propriedades transacionais, assim como a transitividade e completeza do modelo, devem ser novamente verificadas para que o projetista se certifique de que as propriedades previamente verificadas em sua aplicação continuam sendo satisfeitas na versão atual.

Conforme mostra a Figura 5, os componentes típicos do sistema de comércio eletrônico são introduzidos: o servidor *web*, o servidor de transação e o banco de dados. Existem agentes que submetem requisições para o servidor *web*, que as traduz em operações e repassa para o servidor de transação. Essas operações, denominadas serviços, são executadas por este servidor, podendo resultar em ações sobre um ou mais itens de um produto. Este último nível é importante porque ele possibilita ao projetista desenvolver uma especificação mais próxima da implementação real que ele espera obter.

4 Estudo de Caso: Uma Loja Virtual

Nesta seção apresentaremos o estudo de caso de uma aplicação muito popular em comércio eletrônico: uma loja virtual. O objetivo é demonstrar como a metodologia proposta pode ser utilizada no projeto de sistemas mais confiáveis. Esta aplicação é um exemplo típico de comércio eletrônico onde está presente a maioria dos aspectos que tornam difícil o projeto desses sistemas, tais como múltiplos agentes competindo por produtos, múltiplos itens, entre outros. Utilizou-se o verificador de modelos SMV [7] para aplicação da metodologia neste estudo de caso.

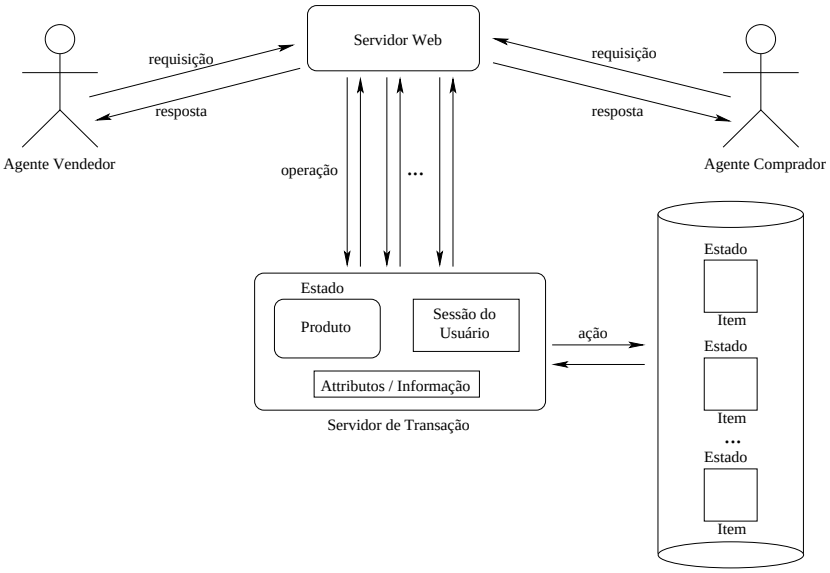

Figura 5. Quarto Nível - Arquitetural

4.1 Nível Conceitual

Conforme definido na metodologia, Seção 3.1, o nível conceitual detalha as exigências de um sistema de comércio eletrônico. Listamos algumas das regras de negócios identificadas no nosso estudo de caso:

- Se um item está indisponível e for disponibilizado, no momento seguinte deverá estar disponível;
- Se um item está disponível e for reservado, no momento seguinte deverá estar reservado;
- Se um item está reservado e for cancelada sua reserva, então ele deverá estar disponível no momento seguinte;
- Se um item está reservado e sua compra for confirmada, então deverá estar vendido no momento seguinte;
- O estoque de um produto tem que ser sempre positivo;
- Se o estoque for positivo, pelo menos um item deve estar disponível;
- Se o estoque for nulo, então o produto não pode estar disponível;
- As ações de *Reservar e Cancelar Reserva* devem ser atômicas;
- Se existirem agentes executando concorrentemente, as ações devem ser isoladas.

Todas as regras de negócio devem ser satisfeitas nos níveis seguintes da metodologia para que a correção do sistema de comércio eletrônico seja confirmada.

4.2 Nível de Aplicação

Exemplificamos aqui a elaboração do segundo nível da nossa metodologia, o nível de aplicação. Os agentes, em verificação, são modelados como processos concorrentes que executam ações. No modelo existe um agente vedendor representando o administrador da loja e um ou mais compradores atuando como clientes. Com o objetivo de ilustrar o uso da metodologia, apresenta-se a seguir partes do código SMV utilizado na modelagem da loja virtual.

Um módulo em SMV consiste de um grupo de variáveis e atribuições as mesmas. O módulo principal consiste da composição paralela de cada módulo. Isso pode ser realizado através da instanciação de cada módulo no módulo principal, como mostrado a seguir:

```
MODULE main

VAR ac1: agente_comprador();
    ac2: agente_comprador();
    av1: agente_vendedor();
    it1: item();
```

Como descrito na Seção 3.2, a primeira propriedade a ser verificada é a completeza, que pode ser expressada através do padrão de *Existência* e escopo *Após*. Portanto, a propriedade é facilmente descrita através das seguintes fórmulas CTL:

```
EF (it1.estado = Não_Disponível)
...
EF (it1.estado = Excluído)
EF (ac1.ação = Comprar)
EF (ac2.ação = Comprar)
EF (av1.ação = Disponibilizar)
```

Estas especificações devem estar consistentes com o grafo de ciclo de vida do item, como ilustrado na Figura 2. No nosso modelo todas foram verificadas verdadeiras, garantindo a completeza do modelo.

No nível de aplicação, o programa SMV produzido tinha 3 módulos (sendo 3 processos), 94 linhas de código e foram verificadas quatorze propriedades.

4.3 Nível Funcional

Dando continuidade ao processo definido pela metodologia, novos módulos foram adicionados ao modelo, os quais representam o produto e seus itens. Neste nível, conforme já explicado, o interesse é por verificar as propriedades inerentes às regras de negócio relacionadas aos serviços.

Inicialmente, conforme descrito na Seção 3.3, deve ser verificada a propriedade de transitividade do modelo. Para isso, pode-se utilizar o padrão de *Universalidade* e escopo *Global*. As fórmulas CTL listadas a seguir visam verificar essa propriedade:

```
AG(estado= Disponível & serviço= Excluir) -> AX(estado= Excluído)
...
AG(estado= Disponível & serviço= Reservar) -> AX(estado= Reservado)
AG(estado= Reservado & serviço= Comprar) -> AX(estado= Vendido)
```

A seguir, são explicadas algumas propriedades transacionais, iniciando pela atomicidade. Se um item estiver disponível e o serviço *Reservar* for executado pelo comprador, o item deverá estar reservado no estado seguinte e o estado estar consistente com isso, ou então o serviço não será completado e o estado não deverá ser alterado.

```
AG ((estado = Disponível & serviço = Reservar & estoque = v) ->
AX ((estado = Disponível & estoque = v) |
    (estado = Reservado & estoque = v-1)))
```

Note que a utilização da variável v é apenas para efeito de simplificação da fórmula, pois em SMV deveria ser expandida para todos os possíveis valores do estoque.

De maneira análoga, há outro exemplo: se o estado é reservado e o serviço que cancela a reserva *Cancelar Reserva* é executado, conforme pode ser visto a seguir.

```
AG ((estado= Reservado & serviço= Cancelar_Reserva & estoque= v) ->
AX ((estado= Disponível & estoque= v+1) |
    (estado= Reservado & estoque= v)))
```

As fórmulas a seguir ilustram alguns exemplos da propriedade de consistência, verificados na loja virtual modelada.

- O estoque do produto nunca pode ser negativo.

  ```
  AG !(estoque < 0)
  ```

- Se o estoque do produto é positivo, então deve existir pelo menos um item no estado disponível.

  ```
  AG ((estoque > 0) -> (estado_produto = Disponível))
  ```

Finalmente, apresentamos exemplos da propriedade de isolamento. Se existem dois compradores, um tentando reservar um item e o outro visando cancelar a reserva de um item do mesmo produto, o estoque do produto deverá manter-se consistente após a execução de ambos.

```
AG ((comprador1_serviço= Reservar & comprador2_serviço=
    Cancelar_Reserva & estoque = v) -> AX (estoque = v))
```

No caso do estoque ser nulo, o serviço de reserva não pode ser precedido pelo serviço de cancelamento de reserva. Portanto, para resolver esse problema decidiu-se por priorizar o agente comprador que deseja cancelar a reserva, mantendo assim a robustez da aplicação. De forma similar, todas as demais regras de negócio foram especificadas e verificadas.

Neste nível foi possível identificar um grave problema, que violava a propriedade de isolamento, possibilitando que um mesmo item de um produto fosse vendido duas vezes. O erro ocorria porque dois agentes compradores tentavam adquirir ao mesmo tempo um item de um mesmo produto e existia apenas um item disponível. O fragmento de código a seguir ilustra parte do código do nível funcional, onde foi identificada o erro. Este código ilustra a sessão de cada usuário comprador na loja virtual, que é quem mantém as informações importantes associadas a ele.

```
next(sessao_usuario) := case
    ...
    sessao_usuario = selecao_de_produto & servico_comprador =
    Reservar & next(disponivel)=1 : carrinho_de_compras;
    ...
    sessao_usuario = selecao_de_produto & servico_comprador =
    Reservar & next(disponivel)=0 : erro;
    ...
    1: sessao_usuario;
esac;
```

Nesta situação, o servidor de transação permitia que ambos os clientes reservassem o mesmo item, o que possibilitava que no futuro o item fosse vendido para os dois. A fim de solucionar isso, introduzimos um semáforo, que garantiu a exclusão mútua e, consequentemente, evitou a violação da propriedade de isolamento. Adicionamos a variável *em_uso* ao nosso modelo para garantir o isolamento no procedimento de reserva do produto, o que assegurou a consistência do modelo.

```
next(em_uso) := case
    em_uso=0 & servico_comprador_1=Reservar & servico_comprador_2=
    Reservar & estoque > 0 & estoque <= 2: {1,2};
    ...
    em_uso = 1 & servico_comprador_1 = Cancelar_Reserva : 0;
    em_uso = 1 & servico_comprador_1 = Comprar : 0;
    em_uso = 2 & servico_comprador_2 = Comprar : 0;
    em_uso = 2 & servico_comprador_2 = Cancelar_Reserva : 0;
    1: em_uso;
esac;
```

Com isso, quando um comprador solicitar a reserva de um item, caso a variável *em_uso* contenha a identificação de outro comprador, então a adição do item ao seu carrinho de compras não é efetuada.

Neste nível de aplicação, o programa SMV produzido tinha 4 módulos (sendo 5 processos), 211 linhas de código e foram verificadas 30 propriedades.

4.4 Nível Arquitetural

Neste estágio, novos módulos foram inseridos ao modelo de comércio eletrônico a fim de representar da melhor maneira possível a arquitetura real do sistema. Portanto foi criado o módulo do servidor *web*, servidor de transação e servidor de banco de dados no modelo, adequando as especificações à nova implementação. Desta forma, as propriedades passaram a se relacionar às requisições, ao invés de serviços.

Neste nível não foram identificadas novas propriedades relacionadas às regras de negócio, visto que todas as regras previamente identificadas haviam sido verificadas no níveis anteriores. Entretanto tornou-se necessário verificar o correto funcionamento dos componentes da arquitetura, representando novas propriedades a serem verificadas. Com isso, foram implementados 5 módulos (sendo 7 processos), o que demandou 700 linhas de código SMV e 71 propriedades foram verificadas.

O modelo completo implementado consiste de mais de 10^{23} estados, dos quais 10^{14} são alcançáveis. Para sua execução foram utilizados aproximadamente 17MB de memória e a verificação demandou apenas nove minutos para validação de todas as propriedades. Utilizamos um PC com processador AMD Atlhon de 1 GHz, 512 MB de memória e sistema operacional *linux*.

5 Trabalhos Correlatos

Há poucos trabalhos sobre análise e verificação formal de sistemas de comércio eletrônico até o momento. Diversos trabalhos, tal como [6,1,3,10] se concentram em verificar propriedades de protocolos específicos e não indicam como essas técnicas podem auxiliar no projeto de novos sistemas. Esses trabalhos utilizam desde técnicas para provar teoremas [6,1] que são tradicionalmente menos eficientes (ainda que mais expressivas), até verificação de modelos [3,10]. Além disso, os trabalhos mencionados são capazes de verificar somente sistemas menores e consomem mais recursos que nosso método.

6 Conclusões e Trabalhos Futuros

Neste artigo propõe-se uma metodologia para especificação e verificação de modelos de sistemas de comércio eletrônico. Através do uso de métodos formais torna-se possível a formalização das especificações do sistema, assim como a verificação automática de propriedades que devem ser satisfeitas. Essa técnica possibilita aplicações mais confiáveis, desenvolvidas mais rapidamente e com custos reduzidos.

Foi modelada e verificada uma loja virtual para demonstrar o funcionamento da metodologia. Como resultado do estudo de caso, tornou-se possível a detecção

de um erro grave, que violava a propriedade transacional de isolamento, fazendo com que o mesmo item fosse vendido duas vezes. Isso acontecia porque dois agentes compradores tentavam adquirir o produto ao mesmo tempo e havia somente um item disponível. Durante a verificação foi identificada o erro que estava acontecendo na loja virtual e também suas causas, o que poderia ser muito difícil de descobrir no sistema em produção.

Atualmente, estamos estudando outras características de sistemas de comércio eletrônico que ainda não foram formalizadas, bem como a possibilidade de gerar o código atual que irá implementar o sistema a partir de sua especificação formal.

Referências

1. Bolignano. Towards the formal verification of electronic commerce protocols. In *PCSFW: Proceedings of The 10th Computer Security Foundations Workshop* (1997), IEEE Computer Society Press.
2. Clarke, E. M., Grumberg, O., and Peled, D. A. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
3. Department, S. L. Model checking the secure electronic transaction (set) protocol. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems* (1998).
4. Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. Property specification patterns for finite-state verification. In *2nd Workshop on Formal Methods in Software Practice* (March 1998).
5. Dwyer, M. B., Avrunin, G. S., and Corbett, J. C. Patterns in property specifications for finite-state verification. In *21st International Conference on Software Engineering* (May 1999).
6. Gurgens, S., Lopez, J., and Peralta, R. Efficient detection of failure modes in electronic commerce protocols. In *DEXA Workshop* (1999), pp. 850–857.
7. K.L. McMillan. *Symbolic Model Checking: An Approach to the State Explosion Problem*. PhD thesis, Carnegie Mellon University, Pittsburgh, PA, 1992.
8. K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publishers, Norwell Massachusetts, 1993.
9. Meira Jr., W., Murta, C. D., Campos, S. V. A., and Neto, D. O. G. *Sistemas de Comercio Eletronico, Projeto e Desenvolvimento*. Campus, 2002.
10. Wang, W., Hidvégi, Z., Bailey, A., and Whinston, A. E-process design and assurance using model checking. In *IEEE Computer* (Oct. 2000).