UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE CIÊNCIA DA COMPUTAÇÃO

RAFAEL KINDLEIN DE ANDRADES

Geração de Testes a partir de Máquinas de Estados Hierárquicas Comunicantes

Monografia apresentada como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Dra. Érika Fernandes Cota

Porto Alegre
2016

AGRADECIMENTOS

Em primeiro lugar, gostaria de agradecer a minha mãe – Vera Lúcia Martins Kindlein – por ter me apoiado e sempre me incentivado a buscar desenvolvimento pessoal e conhecimento. Em seguida, agradeço ao meu pai – Gerson de Andrades - por ter sido o responsável pelo meu interesse em computação, pela minha formação como profissional e como cidadão. Quero, adicionalmente, agradecê-los pelos sensacionais momentos vividos juntos e por não terem medido esforços para realizar os meus sonhos.

Quero agradecer aos familiares, amigos e colegas que estiveram ao meu lado ao longo desta jornada. Eles foram essenciais para superar as dificuldades enfrentadas e, também, para comemorar as conquistas alcançadas.

Por fim, agradeço à direção, aos professores, aos técnicos, aos monitores, aos bibliotecários e a todos os colaboradores do Instituto de Informática da UFRGS, pois foram competentes, dedicados e essenciais para a minha formação como cientista da computação.

Termino com um agradecimento especial a minha orientadora - Prof. Dra. Érika Fernandes Cota - pela paciência, pelas horas de consultoria, pelos debates produtivos e por estes construtivos meses trabalhando juntos.

RESUMO

Em função da criticidade de algumas classes de sistemas, da necessidade de garantir o comportamento esperado em um software e de minimizar riscos e custos com tratamento de erros em sistemas, a fase de testes de software consolidou-se como etapa imprescindível na prática da engenharia de software.

Mais especificamente, técnicas para testes funcionais – testes caixa-preta – têm sido aprimoradas a fim de cobrir escopos de projetos de softwares e sistemas complexos com alta taxa de assertividade e de confiabilidade. Do ponto de vista de teste de software, sistemas baseados em módulos concorrentes, hierárquicos e comunicantes geralmente têm sua modelagem e cobertura de testes complexos.

Neste trabalho, apresentamos uma proposta viável para gerar casos de teste funcionais através da técnica de Teste de Transição de Estados – *State Trasition Test Technique* (STT) – adaptada ao conceito de Máquinas de Estados Hierárquicas Comunicantes – *Communicating Hierarchical State Machines* (CHMs). Por fim, apresentamos um experimento real desenvolvido especificamente para este trabalho.

Palavras-chave: Teste de Software, Testes Funcionais, Testes Caixa-Preta, Máquinas de Estados Finitos, Teste de Transição de Estados, Máquinas de Estados Hierárquicas Comunicantes.

Generation of Tests from Communicating Hierarchical State Machines

ABSTRACT

Depending on the criticality of some classes of systems, the need to ensure the expected behavior in a software and to minimize risks and costs of handling errors in systems, Software Testing phase was consolidated as an essential step to engineering software models.
More specifically, techniques for Functional Testing - Black Box Tests - have been improved to cover scopes of software projects and complex systems with high rate of reliability and assertiveness. From the software testing viewpoint, systems based on concurrent, hierarchical and communicating modules generally have a complex modeling and tests coverage.
In this paper, we present a viable proposal to generate functional tests cases through the State Transition Test technique (STT) adapted to the concept of Communicating Hierarchical State Machines (CHMs). Finally, we present a real experiment developed specifically to this work.

Keywords: Software Testing, Functional Testing, Back Box Tests, Finite State Machines, State Transition Test, Communicating Hierarchical State Machines.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CFSM	Communicating Finite State Machine (Máquina de Estados Finitos Comunicante)
CHM	Communicating Hierarchical States Machine (Máquina de Estados Hierárquica Comunicante)
FSM	Finite State Machine (Máquina de Estados Finitos)
STT	State Transition Test Technique (Técnica de Teste de Transição de Estados)
UFRGS	Universidade Federal do Rio Grande do Sul
UML	Unified Modeling Language (Linguagem de Modelagem Unificada)

SUMÁRIO

1 INTRODUÇÃO

Com o constante aprimoramento e diversificação das aplicações em computação, softwares tornam-se cada vez mais populares e necessários às pessoas atualmente. Nas mais diversas áreas da sociedade, sistemas estão presentes facilitando a gestão de empresas, operacionalizando indústrias, administrando redes, comunicando máquinas e pessoas.

Na mesma proporção, deseja-se que estes softwares não falhem e, por conseguinte, exige-se confiabilidade destes.

Assim, técnicas de teste de software tornaram-se imprescindíveis na produção de software de qualidade, confiável e sustentável. Elas visam a detectar prematuramente falhas e comportamentos inesperados. Portanto, proporcionam maiores garantias de correção. Contudo, modelar testes tem se mostrado um grande desafio por diversos motivos: a complexidade de projetos de sistemas mais elaborados, a falta de conhecimento e de domínio das técnicas de testes existentes, a dificuldade natural de se automatizar testes, entre outros. Dentro deste escopo, softwares organizados em módulos comunicantes com execução concorrente e hierárquica apresentam desafios específicos.

Esse trabalho adapta a técnica de Teste de Transição de Estados – STT (do inglês *State Transition Test Technique*) (BROEKMAN, 2003, p. 124) – para a geração de testes funcionais a partir de Máquinas de Estados Hierárquicas Comunicantes – CHM (do inglês *Communicating Hierarchical States Machine*) (ALUR, 1999) – aqui empregadas para modelar uma dada classe de softwares concorrentes, hierárquicos e comunicantes – e apresenta um experimento executado sobre um sistema real.

Por outro lado, a automação total do método não é o foco deste trabalho e, por conseguinte, é sugerida como trabalho futuro.

1.1 Motivação

Hoje, a geração de testes caixa preta para softwares baseados em módulos concorrentes ainda é uma tarefa difícil e onerosa em função da escassez de métodos e ferramentas próprios para esta finalidade. Em geral, métodos tradicionais são aplicados de forma pouco precisa sobre estes modelos mais complexos.

Originalmente concebida para sistemas embarcados, os principais benefícios da técnica STT são a praticidade para extração dos casos de teste, fácil automatização do algoritmo de geração e a cobertura total dos caminhos alcançáveis no modelo.

Este trabalho visa a formalizar um modelo para portar estes benefícios para testes de alto nível – funcionais – sobre softwares com organização modular, comunicante e hierárquica, neste experimento modelados como CHMs.

Além de preencher esta lacuna, objetiva-se abrir caminho para novas pesquisas e estratégias de automação de testes funcionais sobre softwares com estrutura hierárquica e comunicante e sobre modelos CHM em geral.

Por fim, apresentar o resultado deste estudo como proposta organizacionalmente e economicamente viável à empresa DECISION IT SUPORTE EMPRESARIAL S.A. – empresa voltada ao desenvolvimento de soluções de TI para o ramo fiscal-tributário brasileiro – para cobertura e geração de casos de teste funcionais sobre seus projetos de sistemas.

1.2 Estrutura

Este documento está estruturado para apresentar, no Capítulo 1, uma introdução sobre os desafios de teste de software sobre sistemas comunicantes e hierárquicos e sobre a proposta deste trabalho. No Capítulo 2, conceitos básicos sobre máquinas de estados, teste de software, cobertura de testes e notação UML são apresentados a fim de ilustrar o cenário necessário para a melhor compreensão da técnica desenvolvida. O Capítulo 3 descreve as etapas da técnica proposta enquanto o Capítulo 4 demonstra um experimento real da sua aplicação. Por fim, o Capítulo 5 apresenta a conclusão sobre o trabalho desenvolvido e os trabalhos futuros relacionados.

2 CONCEITOS BÁSICOS

Nessa seção são apresentados os conceitos básicos necessários ao entendimento sobre o método desenvolvido.

2.1 Máquina de Estados Finitos

Uma máquina de estados finitos – FSM (do inglês *Finite State Machine*) (GILL, 1962) – é um modelo lógico matemático utilizado para modelar sistemas de computação. Trata-se de uma máquina abstrata cujo processamento se dá através da transição entre seus estados. Um estado pode armazenar informações, disparar eventos, possuir dados ou subestados atrelados, dependendo do tipo de FSM. Uma transição indica uma mudança de estado e consiste de uma operação necessária para que a transição ocorra. Ainda, podem-se ter condições a serem satisfeitas para que uma transação ocorra (restrição).

FSMs são utilizadas para modelar uma vasta gama de problemas, sistemas e conceitos. Dentre aplicações populares na computação, exemplos clássicos são a descrição de gramáticas de linguagens e os projetos de protocolos de comunicação. Em projetos de software, FSMs modelam objetos em software orientados a objetos (WAGNER, 2006), por exemplo.

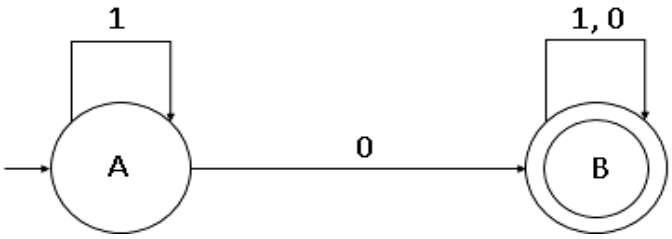
2.1.1 Definição

Há mais de um conceito de FSM existentes no campo de computação teórica. Dentre eles, segue a definição básica onde uma FSM é uma 5-upla $\{\Sigma, S, s_0, \delta, F)$ (GILL, 1962):

- Um alfabeto de entrada $\boldsymbol{\Sigma}$ | $\Sigma \neq \emptyset$
- Um conjunto finito de estados $\boldsymbol{S}$ | $S \neq \emptyset$
- Um estado inicial $\mathbf{s_0}$ | $s_0 \in S$
- A função de transição de estados $\boldsymbol{\delta}$ | $\delta: S \times \Sigma \rightarrow S$
- O conjunto de estados finais $\boldsymbol{F}$ | $F \subseteq S$

A Figura 2.1 apresenta um exemplo básico de FSM que processa números binários que contenham ao menos um zero onde FSM = {(0,1), (A,B), A, δ, B}.

Figura 2.1: Exemplo de FSM.



Fonte: O Autor.

2.2 Máquinas de Estados Finitos Comunicantes

Máquinas de Estados Finitos Comunicantes (CFSMs – do inglês *Communicating Finite State Machines)* são FSMs cuja definição contém as operações primitivas *send* e *receive* definidas sobre um canal de comunicação comum às FSMs (BRAND, 1983). Esta variação de máquina de estados introduz o conceito de máquinas comunicantes.

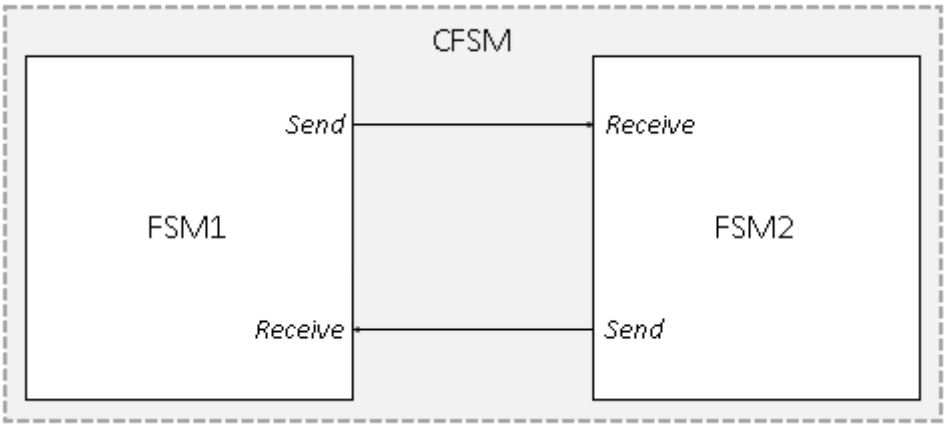
O conceito de CFSM estende as FSMs trazendo uma série de vantagens ao modelo tradicional. Dentre elas, a facilidade de verificar propriedades e falhas de comunicação, de concorrência e de sincronismo (ROSIER, 1983). Consoante a isto, atualmente, as principais aplicações de CFSMs em ciência da computação são a modelagem de processos concorrentes e a definição protocolos de comunicação.

Para um viés mais prático, um par de CFSMs pode ser interpretado como uma dupla de máquinas que podem enviar e receber sinais através de um meio de comunicação compartilhado – *buffers* não bloqueantes – entre elas. O processamento das máquinas é concorrente e o sincronismo entre elas é estabelecido por restrições de transição entre os eventos.

Para este trabalho, a introdução do conceito de máquinas comunicantes é determinante para a correta interpretação do sistema objeto, cujos módulos são máquinas de estados intercomunicantes entre si.

O funcionamento da comunicação entre duas CFSMs é ilustrado na Figura 2.2

Figura 2.2: Estrutura de uma CFSM.



Fonte: O Autor

2.2.1 Definição

Segundo GOUDA (1984):

- Seja uma **mensagem** uma *string* de caracteres.
- Sejam **M** e **N** duas FSMs comunicantes capazes de interpretar o mesmo conjunto **G** de mensagens, o par **(M, N)** é chamado **rede de M e N**.
- Um estado da rede (M, N) é uma 4-upla **[m, n, x, y]** onde **m** e **n** são dois nodos em M e N respectivamente e **x** e **y** são duas mensagens pertencentes ao alfabeto G.
- Um estado **[m, n, x, y]** alcançado indica que as execuções de M e N chegaram aos nodos **m** e **n** respectivamente.
- Um estado **[m, n, x, y]** alcançado indica que M e N receberam as sequências de mensagens **x** e **y** respectivamente.
- O estado inicial da rede (M, N) é **[m_0, n_0, E, E]** onde **m_0** e **n_0** são os nodos inicias em M e N respectivamente e **E** é a mensagem vazia.

As demais propriedades pertinentes a CFSMs estão detalhadas em GOUDA, (1984).

2.3 Máquinas de Estados Hierárquicas Comunicantes

Máquinas de Estados Hierárquicas Comunicantes (CHM – do inglês *Communicating Hierarchical State Machines*) são máquinas de estados cujos estados podem ser outras máquinas. Esse conceito estende as CFSMs através da introdução da propriedade de hierarquia (ALUR, 2002).

Hierarquia, em CHMs, é representada através de estados que correspondem a outra máquina de estados. Esses estados são popularmente chamados *superestados*.

O significado dos superestados, para o modelo, é alcançado recursivamente substituindo cada superestado pela máquina de estados correspondente.

2.3.1 Definição

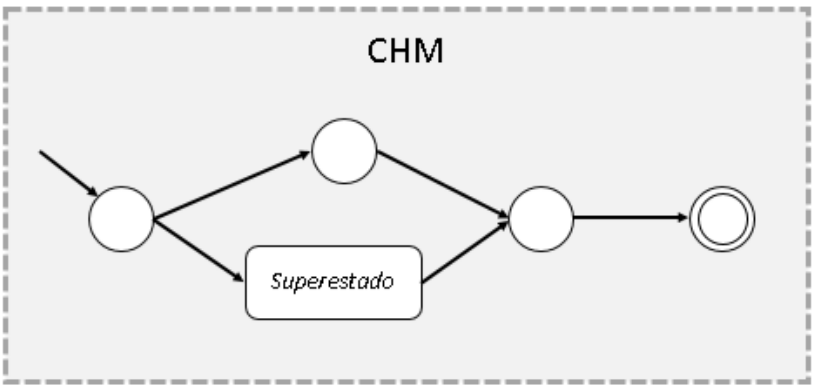
Assuma as definições de FSM e CFSM previamente apresentadas nas Seções 2.1.1 e 2.2.1 respectivamente. Uma CHM é definida pela adição da seguinte definição de hierarquia.

Conforme ALUR (2002), seja:

- **M** um conjunto de CHMs.
- **N** uma máquina de estados com estados **Q**.
- **α** a função **α : Q → M** que associa cada estado **q ∈ Q** com uma **CHM em M.**.

Então a tripla (N, M, α) é uma CHM. Um exemplo conceitual de CHM é apresentado na Figura 2.3:

Figura 2.3: Representação de hierarquia através de superestados em uma CHM.



Fonte: O Autor.

A representatividade, a complexidade e os desafios da combinação dos atributos de concorrência e hierarquia em máquinas de estados (CHMs) ainda são pouco estudados na ciência da computação. ALUR (2002) apresenta uma discussão mais aprofundada sobre o assunto.

2.4 Teste de Software

Cada vez mais exigem-se softwares com maior qualidade e confiabilidade. Para atingir os padrões de qualidade impostos, vários pesquisadores têm investigado diferentes critérios e técnicas de teste, visando a obter formas eficientes para detectar erros em projetos de software (AMMANN, 2008).

De uma maneira muito simples e geral, pode-se agrupar as técnicas de teste de software a partir de duas perspectivas:

- Testes baseados na estrutura, no código-fonte, nas características de construção do software – chamados testes estruturais ou caixa-branca;
- Testes baseados no comportamento externo, no funcionamento e nos resultados esperados conforme a especificação do projeto de software – chamados testes funcionais ou caixa-preta;

Este trabalho demonstra como adaptar a técnica STT para extrair teste funcionais a partir de um modelo de software baseado em CHMs.

2.4.1 Testes Funcionais

Também chamado de teste comportamental, teste caixa-preta, orientado a dado ou orientado a entrada e saída, a técnica de teste funcional avalia o comportamento externo do componente de software, sem considerar a estrutura interna do mesmo (MYERS, 2004, p. 9). Nesta categoria de teste, características estruturais, como o código-fonte, linguagem de programação, classes e métodos utilizados na confecção do software não são usados como base no projeto de testes.

Testes funcionais são de grande importância principalmente para os *stakeholders* do projeto de software porquanto eles detêm o conhecimento comportamental que o software deve apresentar. Assim, técnicas caixa-preta, normalmente, são as técnicas mais empregadas para gerar casos de teste. Em outras palavras, requisitos de funcionamento e casos de uso descritos no escopo do projeto serão o foco nesta categoria de testes.

Intuitivos, os testes funcionais também requerem mais domínio técnico e conhecimento sobre o software. Como são projetados a partir da especificação, podem ser usados, de forma eficaz, desde o início do projeto. Ainda, há a grande vantagem de não exigirem documentação complementar – adequam-se bem à metodologia ágil.

Uma forma de estruturar e mapear testes funcionais se dá através de modelos que representem o comportamento do software. A modelagem facilita a otimização de objetivos como a automatização, a manutenção dos testes, o reuso e a análise de cobertura.

2.5 Cobertura de Testes

A geração de testes funcionais pode ser feita a partir de diferentes artefatos de software. Uma especificação textual que explique o comportamento do sistema, ou mesmo o conhecimento tácito sobre o domínio da aplicação, podem servir como base para definição de estímulos de teste. Esse tipo de artefato de formato livre, porém, dificulta a análise da qualidade dos testes gerados. Em outros termos, nem sempre fica evidente quanto da especificação foi de fato exercitada por um dado conjunto de testes.

O uso de artefatos mais estruturados ou modelos que representem o comportamento descrito na especificação podem ajudar a estruturar e qualificar o processo de geração dos testes. Um modelo largamente usado em testes é o grafo, que pode representar tanto a estrutura de um código (como um grafo de fluxo de controle, seu uso mais comum), como seu comportamento de mais alto nível (uma máquina de estados, por exemplo).

Quando um grafo é usado como base de teste, é possível definir um conjunto de critérios que basicamente definem diferentes formas de explorar aquele modelo. Cada critério define um conjunto de obrigações ou requisitos que devem ser atendidos pelos testes. Assim, por exemplo, o critério de cobertura de nodos define que todos os nodos alcançáveis do grafo são obrigações de teste. Com as obrigações de teste definidas para um dado critério, pode-se avaliar a qualidade de um conjunto de testes em relação àquele critério (AMMANN, 2008).

Assim, se um determinado conjunto de testes atende a todas as obrigações de teste definidas por um critério, diz-se que aquele conjunto de testes satisfaz tal critério.

Alternativamente, quando o conjunto de testes não atende a todas as obrigações de teste listadas, pode-se calcular a taxa de obrigações atendidas. Essa noção de alcance dos testes a partir de um critério de teste é chamada cobertura e é calculada da seguinte forma:

$$\text{Cobertura} = \frac{\text{\# Obrigações Atendidas}}{\text{\# Total de Obrigações de Teste}}$$

Para a técnica empregada neste trabalho (STT), considera-se o critério de nível de profundidade do teste (em inglês, *test depth level*) para a avaliação de cobertura.

2.5.1 Critério de Nível de Profundidade do Teste (*test depth level*)

Em um software modelado sobre um grafo ou máquina de estados, podem-se testar transições individualmente. Por outro lado, também é possível testar sequências de transições. O número de transições consecutivas exercitadas em uma sequência é conhecido como nível de profundidade do teste (BROEKMAN, 2003, p. 131).

A STT é uma técnica de nível de profundidade de teste 1. Ou seja, ela define como obrigação de teste cada transição do grafo – equivalentemente ao critério de cobertura de arcos (AMMANN, 2008). Assim, podemos calcular a cobertura dos testes conforme a equação:

$$\text{Cobertura : } \textit{test depth level } 1 = \frac{\text{\# Transições Exercitadas pelos Testes}}{\text{\# Total de Transições no Modelo}}$$

Na técnica desenvolvida, o nível de profundida de teste se mantém 1 (sem alteração em relação à técnica STT original).

2.6 Técnica de Teste de Transição de Estados

Esta seção apresentará a técnica de Teste de Transição de Estados – STT (do inglês *State Transition Test Technique*) – introduzida por BROEKMAN (2003) e originalmente voltada à extração de casos de teste em sistemas embarcados com comportamento baseado em estados.

A STT implementa o critério de cobertura de nível de profundidade de teste 1 e utiliza modelos de estados planos. Isto é, modelos não hierárquicos.

O propósito da técnica consiste em verificar a relação entre eventos, ações, atividades, estados e transições. Através dessa técnica é possível determinar se um sistema com comportamento baseado em estados condiz com a sua especificação.

2.6.1 Etapas

A técnica STT foi proposta por BROEKMAN (p. 124, 2003) e foi escolhida para esse trabalho visto a possibilidade de automação e a sua fácil aplicação sobre máquinas e modelos de estados.

A geração de testes, seguindo a técnica STT, consiste em definir caminhos no modelo de estados através dos seguintes passos:

1. Compor a tabela estado-evento;
2. Gerar a árvore de transições;
3. Gerar o roteiro de testes legais;
4. Gerar o roteiro de testes ilegais;
5. Gerar o roteiro de testes de guardas.

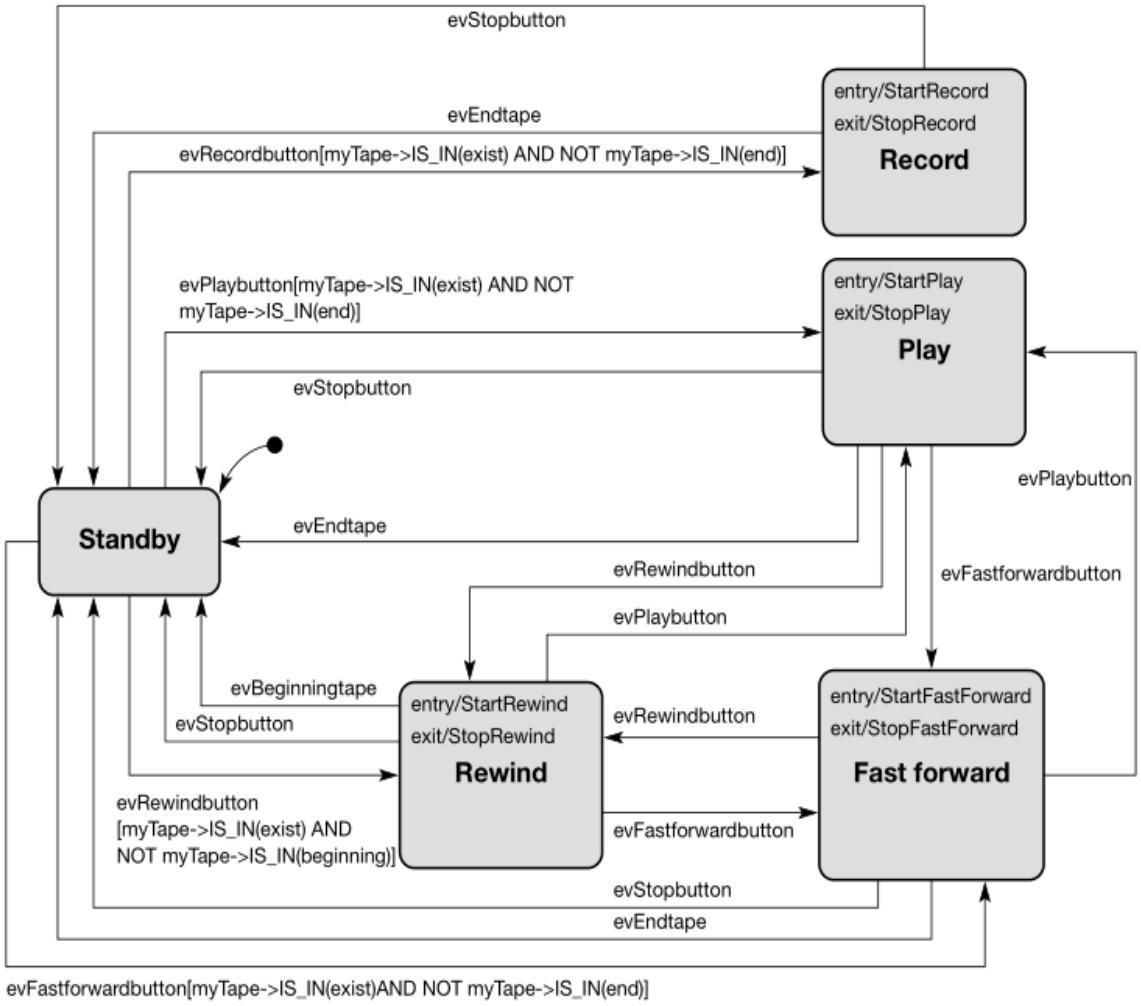
2.6.1.1 Tabela Estado-Evento

A tabela estado-evento explicita a relação entre estados e eventos. Isto é, se uma combinação estado-evento é válida, o resultado desta relação será adicionado à tabela (mesmo

auto relações sobre um estado). Na montagem da tabela, um número será associado a cada transição para uso posterior. A primeira coluna da tabela conterá o estado inicial. As próximas colunas serão formadas pelos estados que podem ser diretamente alcançados pelo estado inicial (uma transição de distância a partir do estado inicial). A seguir, são introduzidos os estados com duas transições de distância a partir do estado inicial. Assim, segue-se adicionando todos os estados pertencentes ao modelo.

Considere o exemplo mostrado na Figura 2.4 que modela um gravador de vídeos com uma máquina de estados. A tabela estado-evento resultante é mostrada na Tabela 2.1.

Figura 2.4: Representação de um sistema de gravação de vídeo através de uma máquina de estados.



Fonte: [BROEKMAN, 2003](p.125)

Tabela 2.1: Tabela estado-evento (combinações ilegais indicadas por um ponto).

	Standby	Rewind	Play	Fast forward	Record
evRewindbutton	1-Rewind	●	9-Rewind	13-Rewind	●
evPlaybutton	2-Play	5-Play	●	14-Play	●
evFastforwardbutton	3-Fast forward	6-Fast forward	10-Fast forward	●	●
evRecord	4-Record	●	●	●	●
evStopbutton	●	7-Standby	11-Standby	15-Standby	17-Standby
evEndtape	●	●	12-Standby	16-Standby	18-Standby
evBeginningtape	●	8-Standby	●	●	●

● Illegal combinations (sneak path)

Fonte: [BROEKMAN, 2003](p.126)

Através das guardas, é possível que um estado possa ter outros muitos estados resultantes. Todos estes estados resultantes são incluídos na tabela. Todas as transições recebem uma numeração única.

Ainda, como visto na Tabela 2.1, combinações ilegais (inexistentes) são marcadas com um ponto.

2.6.1.2 Árvore de Transições

A árvore de transições é gerada a partir da tabela estado-evento. O estado inicial é adicionado à árvore como sua raiz. Então, todas as transições e estados relacionado ao estado inicial são adicionados à árvore. As transições são rotuladas conforme a numeração atribuída na tabela estado-evento e são marcadas com "^" quando possuem guardas (restrições) vinculadas. A partir destes estados, o próximo nível de transições e estados é adicionado à árvore. Assim, sucessivamente os níveis são adicionados até que todos os caminhos alcancem um estado final ou o estado inicial novamente. Se, em qualquer momento durante a construção da árvore, um estado (não final ou inicial) é adicionado e já está presente em qualquer outro caminho da árvore, o caminho encerra sendo marcado com um ponto terminal temporário "*****".

Veja, na Figura 2.6, a árvore de transições resultante da Tabela 2.1.

No roteiro de testes derivados a partir da árvore de transições, cada caminho é percorrido através do percurso mais curto possível. Adicionalmente, garante-se a cobertura de todos os caminhos de nível de profundidade de teste 1 na máquina de estados.

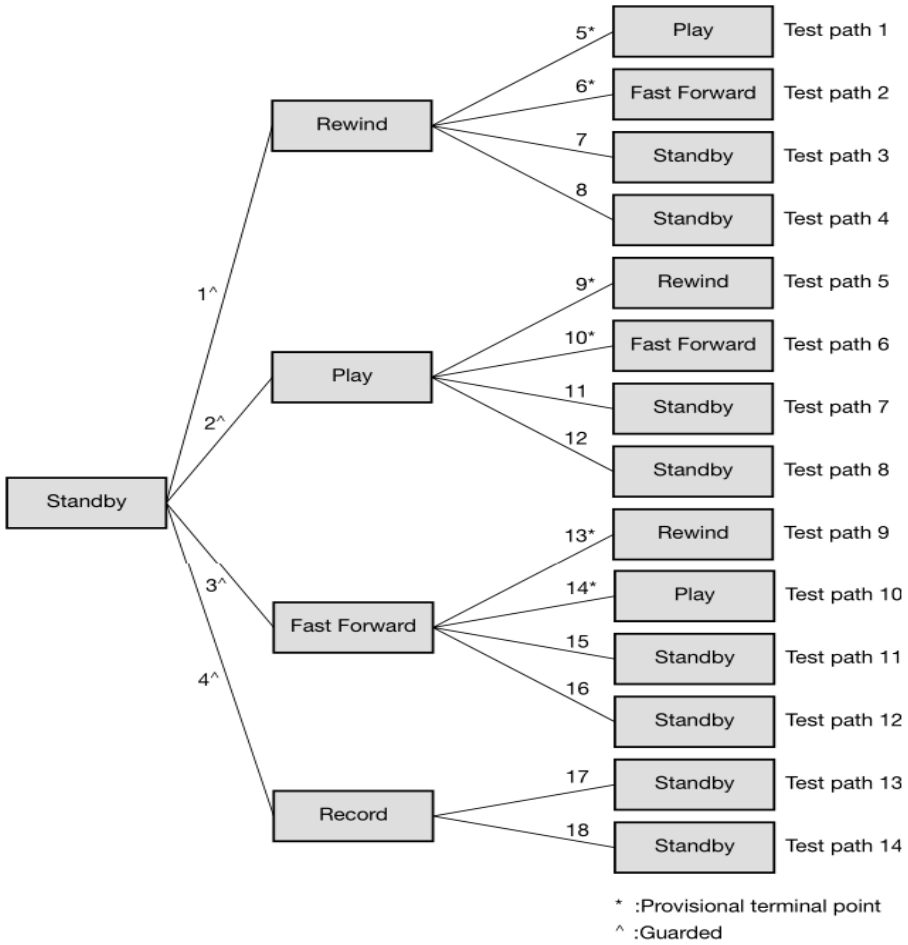
As guardas vinculadas às transições do modelo são elucidadas na descrição adicional da árvore de transições representada na Figura 2.5.

Figura 2.5: Descrição adicional da árvore de transições ilustrada na Figura 2.6.

```
1  [myTape->IS_IN(exist)] AND NOT myTape->IS_IN(beginning)
2  [myTape->IS_IN(exist) AND NOT myTape->IS_IN(end)]
3  [myTape->IS_IN(exist) AND NOT myTape->IS_IN(end)]
4  [myTape->IS_IN(exist) AND NOT myTape->IS_IN(end)]
```

Fonte: [BROEKMAN, 2003](p.126)

Figura 2.6: Árvore de transições resultante da Tabela 2.1.



Fonte: [BROEKMAN, 2003](p.127)

2.6.1.3 Casos de Teste Legais

Dispondo da tabela estado-evento e da árvore de transições, é possível determinar um roteiro de casos de teste legais. Para construir o roteiro, cada caminho na árvore de transições deve ser interpretado como um caso de teste. Assim, cada linha do roteiro será composta por: evento (transição), guardas, ações esperadas e o estado resultante. Conforme mostrado na Tabela 2.2.

Tabela 2.2: Roteiro de casos de teste legais.

	Input		Expected result	
ID	**Event**	**Guard**	**Action**	**State**
L1.1	evRewindbutton	Tape exists, tape not at the beginning	StartRewind	Rewind
L1.2	evPlaybutton		StopRewind; StartPlay	Play
L1.3	evStopbutton		StopPlay	Standby
L2.1	EvRewindbutton	Tape exists, tape not at the beginning	StartRewind	Rewind
L2.2	evFastforwardbutton		StopRewind; StartFastForward	Fast forward
L2.3	evStopbutton		StopFastForward	Standby
L3.1	EvRewind	Tape exists, tape not at the beginning	StartRewind	Rewind
L3.2	EvStopbutton		StopRewind	Standby
L4.1	EvRewind	Tape exists, tape not at the beginning	StartRewind	Rewind
L4.2	EvBeginningtape		StopRewind	Standby
L5.1	EvPlay	Tape exists, tape not at the end	StartPlay	Play
L5.2	evRewind		StopPlay; StartRewind	Rewind
L5.3	evStopbutton		StopRewind	Standby
L6.1	evPlaybutton	Tape exists, tape not at the end	StartPlay	Play
L6.2	evFastforwardbutton		StopPlay; StartFastForward	Fast forward
L6.3	evStopbutton		StopFastForward	Standby
L7.1	evPlaybutton	Tape exists, tape not at the end	StartPlay	Play
L7.2	evStopbutton		StopPlay	Standby
L8.1	evPlaybutton	Tape exists, tape not at the end	StartPlay	Play
L8.2	evEndtape		StopPlay	Standby
L9.1	evFastforwardbutton	Tape exists, tape not at the end	StartFastForward	Fast forward
L9.2	evRewind		StopFastForward; StartRewind	Rewind
L9.3	evStopbutton		StopRewind	Standby
L10.1	evFastforwardbutton	Tape exists, tape not at the end	StartFastForward	Fast forward
L10.2	evPlaybutton		StopFastForward; StartPlay	Play
L10.3	evStopbutton		StopPlay	Standby
L11.1	evFastforwardbutton	Tape exists, tape not at the end	StartFastForward	Fast forward
L11.2	evStopbutton		StopFastForward	Standby
L12.1	evFastforwardbutton	Tape exists, tape not at the end	StartFastForward	Fast forward
L12.2	evEndtape		StopFastForward	Standby
L13.1	evRecord	Tape exists, tape not at the end	StartRecord	Record
L13.2	evStopbutton		StopRecord	Standby
L14.1	evRecordbutton	Tape exists, tape not at the end	StartRecord	Record
L14.2	evEndtape		StopRecord	Standby

Fonte: [BROEKMAN, 2003](p.128)

2.6.1.4 Casos de Teste Ilegais

Casos de teste ilegais correspondem àquelas situações onde o sistema não foi especificado para responder a um dado evento a partir de um determinado estado. Isto é, espera-se que, para cada caso de teste ilegal, o sistema não responda.

O roteiro de casos de teste ilegais pode ser obtido lendo-se a tabela estado-evento. Para cada relação estado-evento marcada com um ponto "●" na tabela, será criado um caminho no roteiro de casos de teste ilegais. A linha da tabela será definida por ID – identificação do caso de teste, *setup* – caminho percorrido para alcançar o estado em teste, estado em teste, evento e resultado – coluna em branco para registrar respostas inesperadas do sistema. Este roteiro é mostrando na Tabela 2.3:

Tabela 2.3: Roteiro de casos de teste ilegais.

ID	Setup	State	Event	Result
I1		Standby	evStopbutton	
I2		Standby	evEndtape	
I3		Standby	evBeginningtape	
I4	L1.1	Rewind	evRewind	
I5	L1.1	Rewind	evRecord	
I6	L1.1	Rewind	evEndtape	
I7	L5.1	Play	evPlay	
I8	L5.1	Play	evPlay	
I9	L5.1	Play	evBeginningtape	
I10	L9.1	Fast forward	evFastforwardbutton	
I11	L9.1	Fast forward	evRecordbutton	
I12	L9.1	Fast forward	evBeginningtape	
I13	L13.1	Record	evRewindbutton	
I14	L13.1	Record	evFastforwardbutton	
I15	L13.1	Record	evRecordbutton	
I16	L13.1	Record	evBeginningtape	

Fonte: [BROEKMAN, 2003](p.129)

2.1.6.5 Casos de Teste de Guarda

Esta etapa objetiva gerar um roteiro de testes para testar as guardas – restrições – do software modelado. Basicamente, casos de teste legais são replicados e diferentes valores são usados para que a guarda seja avaliada para todos os valores possíveis. Técnicas de teste baseadas em classes de equivalência podem ser usadas nesta etapa, como detalhado em BROEKMAN (2003).

A geração do roteiro de testes de guarda não é relevante para a técnica proposta neste trabalho.

2.7 UML – Diagramas de Máquinas de Estados

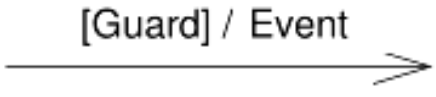
Nesta seção, a notação básica para a modelagem de máquinas de estados em UML é apresentada.

A técnica STT é aplicada sobre modelos de estados. Contudo, software é modelado através da notação UML *Statechart*. Adicionalmente, UML *Statechart* aborda a modelagem de máquinas de estados agregando notações para atributos como concorrência e hierarquia, que são ênfase deste trabalho. Assim, estes conceitos serão importantes para a leitura e a interpretação do experimento apresentado neste trabalho.

A especificação completa para modelagem UML está disponível em UML.org (2015).

Seta de Transição: Utilizada para indicar a transição de um estado para outro. Uma transição pode possuir restrições de execução definidas (guardas) e eventos atribuídos.

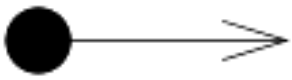
Figura 2.7: Seta de transição



Fonte: O Autor.

Estado Inicial: É o estado inicial de uma máquina; de onde parte a primeira transição.

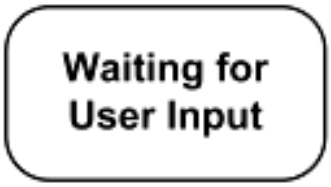
Figura 2.8: Estado inicial.



Fonte: O Autor.

Estado Simples: Representa um estado que não possui regiões ou subestados.

Figura 2.9: Estado simples.



Fonte: <UML-DIAGRAMS.org>.

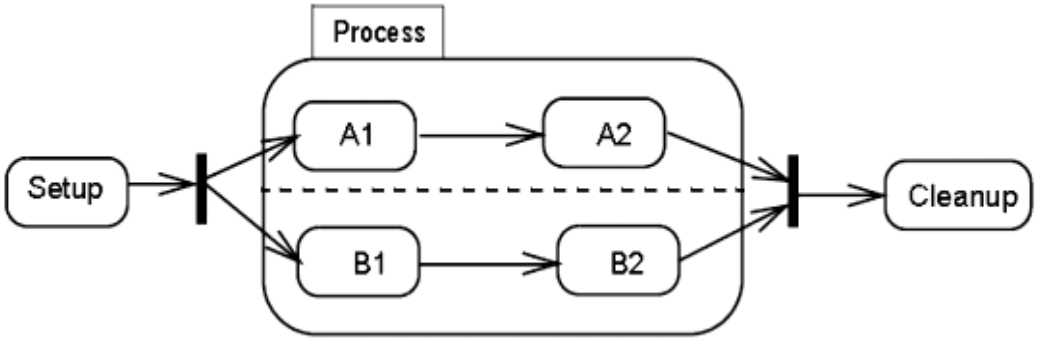
Estados simples também podem apresentar eventos internos. A notação prevê a especificação de um gatilho associado ao evento (ambos opcionais).

Figura 2.10: Estado simples com eventos internos.

Fonte: <UML-DIAGRAMS.org>.

Estado Composto: Um estado composto é um estado que possui estados internos (subestados). Os subestados podem ser sequenciais (disjuntos) ou concorrentes (ortogonais). Ainda, estados compostos podem ser subdivididos em mais de uma região.

Figura 2.11: Estado composto com duas regiões concorrentes.



Fonte: <UML.org>.

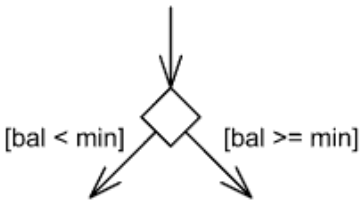
Superestado: Um superestado é definido como um estado que possui uma máquina de estados embarcada dentro de si. Algoritmicamente, um superestado pode ser recursivamente substituído pela máquina de estados equivalente.

Figura 2.12: Notação de superestado.

Fonte: <UML-DIAGRAMS.org>.

Pseudoestado de Decisão: Realiza uma ramificação condicional. Esta notação valida guardas condicionais para decidir qual transição de saída tomar a partir de um dado estado.

Figura 2.13: Notação de pseudoestado de decisão.



Fonte: <UML-DIAGRAMS.org>.

Estado Final: Quando alcançado, o estado final indica que, dentro da região especificada, o processamento da máquina terminou.

Figura 2.14: Notação de estado final.

Fonte: <UML-DIAGRAMS.org>.

3 GERAÇÃO DE TESTES A PARTIR DE MÁQUINAS DE ESTADOS HIERÁRQUICAS COMUNICANTES

Nesta seção, a STT é adaptada para gerar casos de teste sobre CFSMs e CHMs. Recursos sintáticos e semânticos são agregados à técnica visando a contemplar, de forma harmônica e funcional, atributos como sincronismo, concorrência, comunicação e hierarquia. Analogamente à técnica original, a técnica proposta mantém o nível de profundidade de teste 1.

3.1 Técnica Proposta

A técnica proposta está estruturada da seguinte forma:

1. **Modelagem CHM;**
2. **Composição da tabela estado-transição;**
3. **Composição da árvore de transições;**
4. Geração do roteiro de testes legais;
5. Geração do roteiro de testes ilegais;
6. **Geração do roteiro de testes hierárquicos.**

Em negrito estão marcadas as etapas novas ou que sofreram adaptação relevantes em relação à técnica STT original.

3.1.1 Modelagem CHM

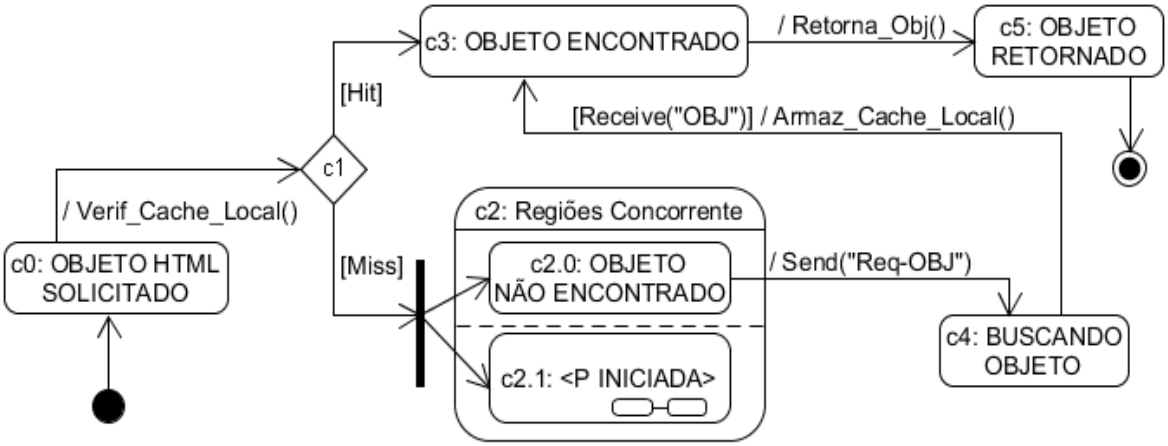
Não há um algoritmo exato e universal para modelar software como máquina de estados. Uma mesma especificação pode ser modelada em diversas máquinas de estados estruturalmente diferentes sem perda de correção semântica. A qualidade do modelo dependerá do nível de domínio técnico e do conhecimento, do profissional habilitado, sobre o projeto e sobre o comportamento do software.

A modelagem deve partir da especificação de software levando em consideração os conceitos básicos apresentados nas Seções 2.1, 2.2, 2.3 e 2.7. Documentos adicionais do projeto de software não são necessários. Portanto, o roteiro de testes pode ser gerado na fase

inicial do projeto permitindo atualizações e manutenções concomitantes ao desenvolvimento do software – prática comum em métodos ágeis (COHEN, 2004, p. 1-66).

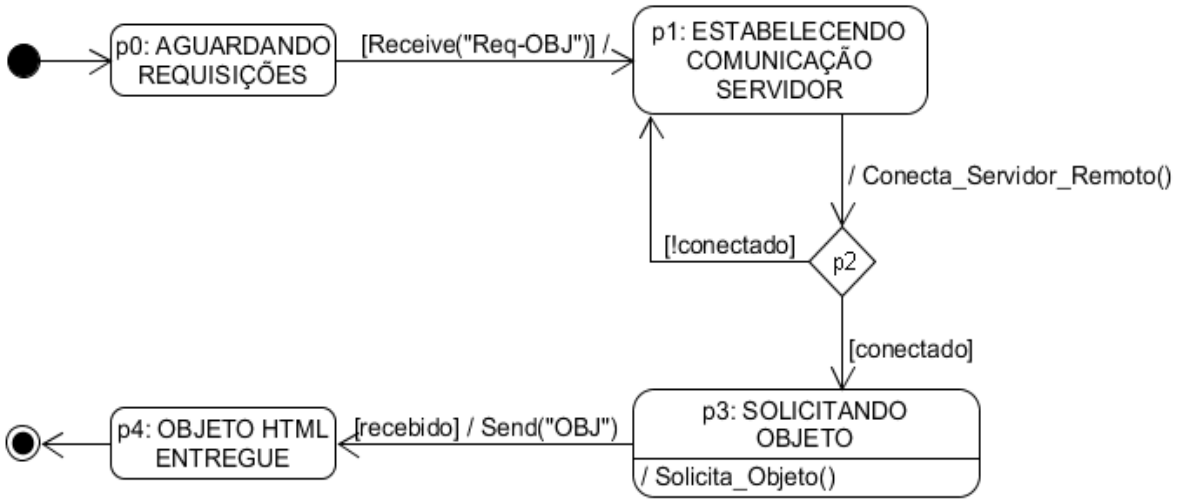
Assim, assume-se dois modelos de máquinas sintéticas desenvolvidos para ilustrar, de forma mais didática, os conceitos empregados pela técnica proposta. Nas Figuras 3.1 e 3.2 abaixo, estão modeladas as CHMs C e P que simulam a interação de requisições WEB de um navegador com memória *cache*.

Figura 3.1: CHM (C, N, α) que representa o módulo de gerenciamento de memória cache de um navegador WEB.



Fonte: O Autor.

Figura 3.2: CHM (P, N, β) que representa o módulo do navegador WEB responsável por executar o protocolo de comunicação com servidores remotos.



Fonte: O Autor.

Definição das CFSMs:

- Seja **C** uma CFSM que representa o módulo simplificado de um navegador WEB responsável por verificar e retornar objetos HTML da memória *cache* local de um computador.
- Seja **P** uma CFSM que representa o módulo simplificado, do mesmo navegador WEB de C, responsável por executar o protocolo de comunicação com servidores WEB remotos.
- **M : {Req-OBJ, OBJ, E}** é o conjunto de mensagens compartilhadas e interpretadas por C e P onde E é a mensagem vazia.
- O par (C, P) é uma **rede de C e P** com estado inicial **[c0, s, E, E]** onde **s** é o estado vazio – aqui empregado até que a máquina P inicie, no estado p0, hierarquicamente a partir de C.

Definição das CHMs:

- Considere **N : {C, P, Ø}** o conjunto de CFSMs C, P e Ø onde **Ø é a máquina vazia**.
- **D : {c0, c1, c2, c2.1, c2.2, c3, c4, c5}** é o conjunto de estados, pseudoestados, estados compostos, superestados e subestados de C.
- **Q : {p0, p1, p2, p3, p4}** é o conjunto de estados, pseudoestados, estados compostos, superestados e subestados de P.
- **α : D→N** é a função que associa os estados de C às máquinas de N definida como α : (c0→Ø, c1→Ø, c2→Ø, c2.0→Ø, c2.1→P, c3→Ø, c4→Ø, c5→Ø) onde c2.0 e c2.1 são subestados do estado composto c2.
- **β : Q→N** é a função que associa os estados de P às máquinas de N definida como β : (p0→Ø, p1→Ø, p2→Ø, p3→Ø, p4→Ø).
- Então as triplas **(C, N, α)** e **(P, N, β)** são CHMs.

Na modelagem utilizada neste trabalho, eventos sem guardas são considerados eventos automáticos do software e são executados transparentemente sem ação do usuário.

3.1.2 Tabela Estado-Transição

Nomeada tabela estado-evento na técnica STT, chamamos tabela estado-transição uma tabela que, além de associar estados a eventos, também permite associar estados a guardas quando uma transição não possui eventos vinculados.

Essa modificação permite que pseudoestados de decisão, descritos na Seção 2.7, sejam tratados e adicionados à tabela como estados simples. Assim podem-se mapear os caminhos de teste derivados dos pseudoestados de decisão, que não possuem eventos associados às suas transições de saída. As transições de saída de um pseudoestado de decisão são definidas unicamente pelas suas guardas (Figura 2.13).

As Tabelas 3.1 e 3.2 apresentam, respectivamente, as tabelas estado-transição das CHMs (C, N, α) (Figura 3.1) e (P, N, β) (Figura 3.2).

Tabela 3.1: Tabela estado-transição da CHM (C, N, α).

	c0: OBJETO HTML SOLICITADO	c1: PSEUDOESTADO DE DECISÃO	c2.0: OBJETO NÃO ENCONTRADO	c2.1: <P INICIADA>	c3: OBJETO ENCONTRADO	c4: BUSCANDO OBJETO	c5: OBJETO RETORNADO
Verif_Cache_Local()	1 - c1	●	●	●	●	●	●
Send()	●	●	4 - c4	●	●	●	●
Retorna_Obj()	●	●	●	●	5 - c5	●	●
Armaz_Cache_Local()	●	●	●	●	●	6 - c3	●
[hit]	●	2 - c3	●	●	●	●	●
[miss]	●	3 - c2	●	●	●	●	●

● Combinações ilegais

Fonte: O Autor.

Tabela 3.2: Tabela estado-transição da CHM (P, N, β).

	p0: AGUARDANDO REQUISIÇÕES	p1: ESTABALECENDO COMUNICAÇÃO SERVIDOR	p2: PSEUDOESTADO DE DECISÃO	p3: SOLICITANDO OBJETO	p4: OBJETO HTML ENTREGUE
Conecta_Servidor_Remoto()	●	2 - p2	●	●	●
Send()	●	●	●	5 - p4	●
[Receive()]	1 - p1	●	●	●	●
[conectado]	●	●	3 - p3	●	●
[!conectado]	●	●	4 - p1	●	●

● Combinações ilegais

Fonte: O Autor.

Adicionalmente, estados simples, subestados e superestados também podem apresentar transições sem eventos, isto é, também definidas exclusivamente por suas guardas. Esta situação é exemplificada na transição [c2, **p0**, E, E] → [c4, **p1**, E, Req-OBJ] da CHM (P, N, β) (Figura 3.2). A semântica dessa representação refere-se a trocas de estados automáticas no software (sem a necessidade de executar-se um evento). Neste caso, como na regra utilizada para pseudoestados de decisão, ao compor a tabela os estados associam-se às guardas das transições.

Um estado composto, utilizado para mapear concorrência em CHMs, deve ser decomposto em subestados, que serão adicionados à tabela. Por exemplo, o estado composto *c2* da Figura 3.1 contendo dois subestados tem os subestados *c2.0* e *c2.1* adicionados à tabela.

Quando um superestado estiver presente no modelo, este será adicionado à tabela como um estado simples. Como superestados denotam CHMs implícitas no modelo, o identificador do superestado deve apresentar uma notação diferenciada para destacar sua semântica – deve ser adicionado à tabela com o seu identificador entre “<” e “>” (como demonstrado através do subestado *c2.1* na Tabela 3.1). Desta forma, é possível identificar mais facilmente os caminhos que atravessam a CHM implícita no superestado.

Na notação UML *Statechart*, eventos internos aos estados são interpretados como auto transições cujo estado resultante é o estado atual. Entretanto, para a técnica STT, os eventos internos são interpretados como ações executadas automaticamente ao atingir-se o estado. Na técnica proposta, as duas interpretações são contempladas de formas distintas:

- Quando um evento interno possuir guarda, será considerado uma auto transição do estado e deve ser adicionado à tabela evento-transição como uma transição para o próprio estado – situação observável no estado *i2* da máquina apresentada na Figura 4.4.
- Quando um evento interno não estiver associado a uma guarda, será considerado uma “ação” automática do estado – exemplo no estado *p3* da CHM (P, N, β) (Figura 3.2). Ações não são incluídas na tabela estado-transição, mas são tratadas na geração do roteiro de testes legais posteriormente na Seção 3.1.4.

Todo estado associado à notação UML de estado inicial (Figura 2.8) é considerado um estado inicial. Assim como todo estado associado à notação UML de estado final (Figura 2.14), também é considerado um estado final.

Em adição às variações apresentadas nesta seção, a tabela estado-transição segue os demais passos de construção da tabela estado-evento da STT, descritos na Seção 2.6.1.1.

3.1.3 Árvore de Transições

Nesta seção, ajustes de notação são propostos para o algoritmo de geração da árvore de transição original da STT (Seção 2.6.1.2) com o objetivo de adequar as novas notações empregadas na modelagem das CHMs e na tabela estado-transição.

Primeiramente, estados compostos são representados na árvore através de seus subestados e da notação de concorrência adotada em UML, como mostrado na Figura 3.3.

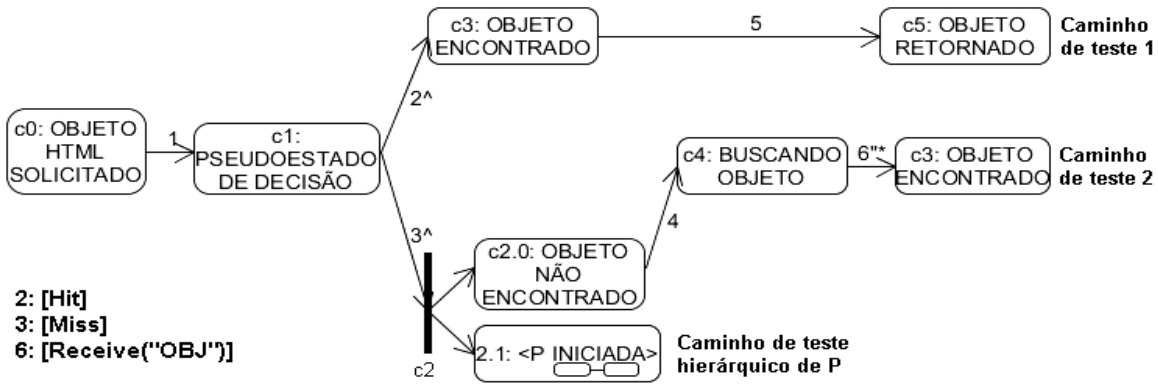
Superestados também devem ser representados usando a notação tradicional UML (Figura 2.12).

As guardas que contenham as primitivas *Send* ou *Receive* das CFSMs serão chamadas "guardas comunicantes". Guardas comunicantes são requisitos que dependem do sincronismo e da comunicação entre a rede de CFSMs. Na árvore de transição, guardas comunicantes serão representadas através da notação "**″**".

Por fim, caminhos terminados em superestados (outras CHMs) são chamados "casos de teste hierárquicos". O roteiro de casos de teste hierárquicos é tratado na Seção 3.1.6.

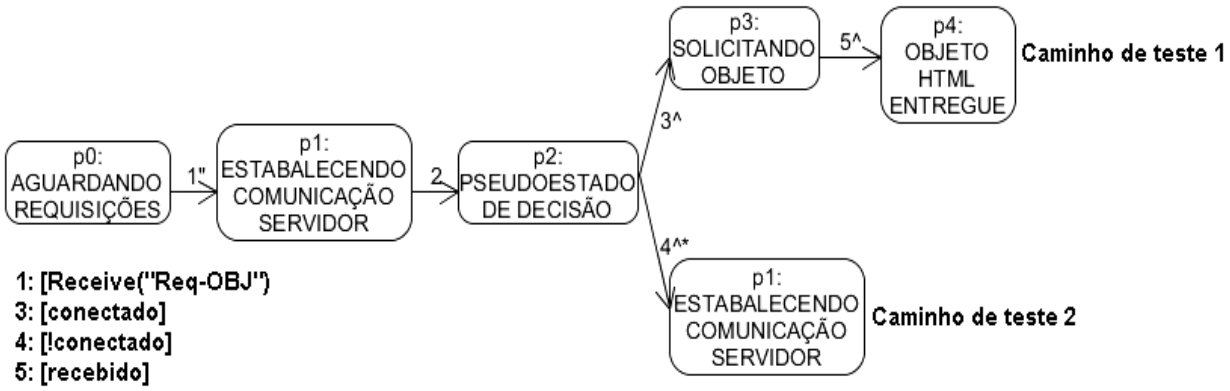
As árvores de transição das CHMs (C, N, α) e (P, N, β) são mostradas nas Figuras 3.3 e 3.4 respectivamente.

Figura 3.3: Árvore de transição da CHM (C, N, α).



Fonte: O Autor.

Figura 3.4: Árvore de transição da CHM (P, N, β).



Fonte: O Autor.

3.1.4 Casos de Teste Legais

Dispondo da tabela estado-transição e da árvore de transições, é possível determinar o roteiro de casos de teste legais conforme o algoritmo original da STT (Seção 2.6.1.3).

As Tabelas 3.3 e 3.4 listam os casos de teste legais gerados para as CHMs (C, N, α) e (P, N, β).

Tabela 3.3: Roteiro de casos de teste legais gerados para a CHM (C, N, α).

	Entrada			Resultado Esperado
ID	**Evento**	**Guarda**	**Ação**	**Estado**
L1.1	Verif_Cache_Local()			c1: PSEUDOESTADO DE DECISÃO
L1.2		Encontrou objeto na memória *cache*		c3: OBJETO ENCONTRADO
L1.3	Retorna_Obj()			c5: OBJETO RETORNADO
L2.1	Verif_Cache_Local()			c1: PSEUDOESTADO DE DECISÃO
L2.2		Objeto não encontrado na memória *cache*	Inicia_P()	c2.0: OBJETO NÃO ENCONTRADO; c2.1: <P INICIADA>
L2.3	Send()			c4: BUSCANDO OBJETO
L2.4	Armaz_Cache_Local()	Objeto recebido da máquina P		c3: OBJETO ENCONTRADO
L2.5	Retorna_Obj()			c5: OBJETO RETORNADO

Fonte: O Autor.

Quando um caso de teste atingir um superestado, ou seja, outra CHM, inclui-se uma ação no caso de teste para destacar o início implícito da CHM. Por exemplo, a ação "Inicia_P()" indica o início da CHM (P, N, β) quando o superestado "c2.1: <P INICIADA>" é atingido no teste L2.2 da Tabela 3.3

Tabela 3.4: Roteiro de casos de teste legais gerados para a CHM (P, N, β).

Entrada			Resultado Esperado	
ID	Evento	Guarda	Ação	Estado
L1.1		Requisição de objeto recebida da máquina C		p1: ESTABELECENDO COMUNICAÇÃO SERVIDOR
L1.2	Conecta_Servidor_Remoto()			p2: PSEUDOESTADO DE DECISÃO
L1.3		Conexão estabelecida com o servidor remoto	Solicita_Objeto()	p3: SOLICITANDO OBJETO
L1.4	Send()	Objeto recebido do servidor remoto		p4: OBJETO HTML ENTREGUE
L2.1		Requisição de objeto recebida da máquina C		p1: ESTABELECENDO COMUNICAÇÃO SERVIDOR
L2.2	Conecta_Servidor_Remoto()			p2: PSEUDOESTADO DE DECISÃO
L2.3		Tentativa de conectar ao servidor remoto falhou		p1: ESTABELECENDO COMUNICAÇÃO SERVIDOR

Fonte: O Autor.

3.1.5 Casos de Teste Ilegais

Tabela 3.5: Roteiro de casos de teste ilegais da CHM (C, N, α).

ID	Setup	Estado	Evento / Guarda	Resultado
I1		c0: OBJETO HTML SOLICITADO	Send()	
I2		c0: OBJETO HTML SOLICITADO	Retorna_Obj()	
I3		c0: OBJETO HTML SOLICITADO	Armaz_Cache_Local()	
I4		c0: OBJETO HTML SOLICITADO	[hit]	
I5		c0: OBJETO HTML SOLICITADO	[miss]	
I6	L1.1	c1: PSEUDOESTADO DE DECISÃO	Verif_Cache_Local()	
I7	L1.1	c1: PSEUDOESTADO DE DECISÃO	Send()	
I8	L1.1	c1: PSEUDOESTADO DE DECISÃO	Retorna_Obj()	
I9	L1.1	c1: PSEUDOESTADO DE DECISÃO	Armaz_Cache_Local()	
I10	L2.2	c2.0: OBJETO NÃO ENCONTRADO	Verif_Cache_Local()	
I11	L2.2	c2.0: OBJETO NÃO ENCONTRADO	Retorna_Obj()	
I12	L2.2	c2.0: OBJETO NÃO ENCONTRADO	Armaz_Cache_Local()	
I13	L2.2	c2.0: OBJETO NÃO ENCONTRADO	[hit]	
I14	L2.2	c2.0: OBJETO NÃO ENCONTRADO	[miss]	
I15	L2.2	c2.1: <P INICIADA>	Verif_Cache_Local()	
I16	L2.2	c2.1: <P INICIADA>	Send()	
I17	L2.2	c2.1: <P INICIADA>	Retorna_Obj()	
I18	L2.2	c2.1: <P INICIADA>	Armaz_Cache_Local()	
I19	L2.2	c2.1: <P INICIADA>	[hit]	
I20	L2.2	c2.1: <P INICIADA>	[miss]	
I21	L1.2	c3: OBJETO ENCONTRADO	Verif_Cache_Local()	
I22	L1.2	c3: OBJETO ENCONTRADO	Send()	
I23	L1.2	c3: OBJETO ENCONTRADO	Armaz_Cache_Local()	
I24	L1.2	c3: OBJETO ENCONTRADO	[hit]	
I25	L1.2	c3: OBJETO ENCONTRADO	[miss]	
I26	L2.3	c4: BUSCANDO OBJETO	Verif_Cache_Local()	
I27	L2.3	c4: BUSCANDO OBJETO	Send()	
I28	L2.3	c4: BUSCANDO OBJETO	Retorna_Obj()	
I29	L2.3	c4: BUSCANDO OBJETO	[hit]	
I30	L2.3	c4: BUSCANDO OBJETO	[miss]	
I31	L1.3	c5: OBJETO RETORNADO	Verif_Cache_Local()	
I32	L1.3	c5: OBJETO RETORNADO	Send()	
I33	L1.3	c5: OBJETO RETORNADO	Retorna_Obj()	
I34	L1.3	c5: OBJETO RETORNADO	Armaz_Cache_Local()	
I35	L1.3	c5: OBJETO RETORNADO	[hit]	
I36	L1.3	c5: OBJETO RETORNADO	[miss]	

Fonte: O Autor.

Tabela 3.6: Roteiro de casos de teste ilegais da CHM (P, N, β).

ID	Setup	Estado	Evento / Guarda	Resultado
I1		p0: AGUARDANDO REQUISIÇÕES	Conecta_Servidor_Remoto()	
I2		p0: AGUARDANDO REQUISIÇÕES	Send()	
I3		p0: AGUARDANDO REQUISIÇÕES	[conectado]	
I4		p0: AGUARDANDO REQUISIÇÕES	[!conectado]	
I5	L1.1	p1: ESTABELECENDO COMUNICAÇÃO SERVIDOR	Send()	
I6	L1.1	p1: ESTABELECENDO COMUNICAÇÃO SERVIDOR	[Receive()]	
I7	L1.1	p1: ESTABELECENDO COMUNICAÇÃO SERVIDOR	[conectado]	
I8	L1.1	p1: ESTABELECENDO COMUNICAÇÃO SERVIDOR	[!conectado]	
I9	L1.2	p2: PSEUDOESTADO DE DECISÃO	Conecta_Servidor_Remoto()	
I10	L1.2	p2: PSEUDOESTADO DE DECISÃO	Send()	
I11	L1.2	p2: PSEUDOESTADO DE DECISÃO	[Receive()]	
I12	L1.3	p3: SOLICITANDO OBJETO	Conecta_Servidor_Remoto()	
I13	L1.3	p3: SOLICITANDO OBJETO	[Receive()]	
I14	L1.3	p3: SOLICITANDO OBJETO	[conectado]	
I15	L1.3	p3: SOLICITANDO OBJETO	[!conectado]	
I16	L1.4	p4: OBJETO HTML ENTREGUE	Conecta_Servidor_Remoto()	
I17	L1.4	p4: OBJETO HTML ENTREGUE	Send()	
I18	L1.4	p4: OBJETO HTML ENTREGUE	[Receive()]	
I19	L1.4	p4: OBJETO HTML ENTREGUE	[conectado]	
I20	L1.4	p4: OBJETO HTML ENTREGUE	[!conectado]	

Fonte: O Autor.

Os casos de teste ilegais mapeiam as ocasiões onde o software não foi especificado para responder a um determinado evento, ou condição, a partir de um estado definido.

A partir das combinações ilegais da tabela estado-transição "●" (Seção 3.1.2), extraímos os casos de teste ilegais conforme a técnica STT (Seção 2.6.1.4).

Nas Tabelas 3.5 e 3.6, estão listados os casos de teste ilegais gerados a partir das tabelas estado-transição das CHMs (C, N, α) e (P, N, β) respectivamente.

3.1.6 Casos de Teste Hierárquicos

Nesta etapa, objetiva-se gerar um roteiro contendo todos os caminhos da árvore de transições da CHM que atinjam um superestado, ou seja, outra CHM.

A descrição dos casos de teste hierárquicos é construída a partir da tabela de casos de teste legais (Seção 3.1.4). Isto é, o caso de teste hierárquico é o *setup* – caminho percorrido – do caso de teste legal até o ponto onde o superestado é alcançado. Quando o superestado é alcançado, o caso de teste é encerrado.

O roteiro de casos de teste hierárquicos para a CHM (C, N, α) é apresentado na Tabela 3.7.

Tabela 3.7: Roteiro de casos de teste hierárquicos da CHM (C, N, α).

	Entrada				Resultado Esperado
ID	**Setup**	**Evento**	**Guarda**	**Ação**	**Estado**
	L2.1	Verif_Cache_Local()			c1: PSEUDOESTADO DE DECISÃO
H1	L2.2		Objeto não encontrado na memória *cache*	Inicia_P()	c2.0: OBJETO NÃO ENCONTRADO; c2.1: <P INICIADA>

Fonte: O Autor.

A execução de cada caso de teste hierárquico deve ser feita de forma recursiva aplicando a técnica proposta sobre a CHM implícita no superestado, quando este é alcançado. Este processo é executado sucessivamente até que todos os roteiros de casos de teste hierárquicos de todas as CHMs acionadas sejam testados.

No caso da CHM (P, N, β), o roteiro de casos de teste hierárquicos é vazio visto que não há superestados no modelo.

4 EXPERIMENTO

Nesta seção é apresentado um experimento real através da aplicação da técnica proposta sobre um projeto de software executado na empresa de tecnologia da informação DECISION IT SUPORTE EMPRESARIAL S.A.

4.1 Desafio

No processo de desenvolvimento de software, um fornecedor terceirizado foi contratado para atender às necessidades de sistemas da empresa. Contudo, as etapas de levantamento de requisitos, análise de negócios, especificação, testes e validação, implantação, documentação e treinamento permaneceram sob responsabilidade da DECISION IT.

Analisando o processo de especificação, verificou-se que a empresa gera um documento não estruturado basicamente contendo requisitos técnicos que se assemelham a casos de uso. Posteriormente, os requisitos listados na especificação são utilizados para efetuar os testes funcionais, de maneira manual e intuitiva, sobre o software entregue pelo fornecedor.

Dentro deste modelo, alguns problemas capitais para a efetividade do processo foram verificados. São eles:

- Geração dos casos de testes de forma intuitiva sobre um documento de especificação não estruturado;
- Difícil padronização do processo de geração do roteiro de testes;
- Dificuldade para propor automações sobre o processo de geração de testes;
- A aplicação de métricas para avaliação da qualidade do processo torna-se inviável;
- Pouca confiança no roteiro de testes gerado;
- Casos de testes imprecisos;
- Sem garantias de cobertura mínima de testes sobre o escopo do projeto;
- Dificuldade em treinar-se novos colaboradores para assumirem o processo de testes.

Considerando o cenário acima, um *sprint* de desenvolvimento de software da DECISION IT foi isolado e submetido à técnica proposta neste trabalho a fim de se avaliar os possíveis benefícios alcançáveis em relação ao processo original.

4.2 Aplicação da Técnica Proposta

O objeto deste experimento é um software para cadastro e cobrança de inscrições de eventos, palestras e treinamentos corporativos.

O software é compartimentalizado em dois módulos: um módulo de cadastro de inscrições e um módulo voltado à cobrança e à geração de notas fiscais. Esses módulos são operados de forma concorrente e sincronizada pelos setores de marketing e financeiro da empresa. A comunicação entre os módulos se dá através da troca de mensagens via rede local ou internet.

Na Seção 4.2.1, os módulos são modelados como CHMs e mais detalhes sobre a especificação do software são apresentados.

4.2.1 Modelagem CHM

A partir dos requisitos descritos na especificação fornecida pela DECISION IT, o método de modelagem CHM apresentado na Seção 3.1.1 foi utilizado para a definição de duas máquinas de estados hierárquicas comunicantes. Nas Figuras 4.1, 4.2 e 4.3 são mostrados, a fim de exemplo, requisitos extraídos do documento de especificação do projeto.

Figura 4.1: Requisito sobre a atualização da inscrição.

Requisito D16: Deve ser possível editar/alterar uma inscrição mesmo que a NFS-E já tenha sido gerada. Desejamos poder substituir um INSCRITO por outro dentro de uma INSCRIÇÃO.

Fonte: DECISION IT SUPORTE EMPRESARIAL S.A

Figura 4.2: Requisito da especificação dos estados da inscrição.

Requisito D5: Campo *STATUS INSCRIÇÃO*: Possuirá 5 status. São eles:

EM DIGITAÇÃO: A inscrição ainda não está completa. Dados estão sedo digitados, preenchidos. Ao solicitar-se o faturamento da inscrição, a inscrição deve mudar para o status CRIADA e o processo de Nota Fiscal deve iniciar.
CRIADA: Primeiro status da inscrição completa.
CONFIRMADA: O sistema deverá atualizar o status da inscrição de CRIADA para CONFIRMADA quando o título/boleto vinculados à inscrição for liquidado pelo retorno do banco.
AGUARDANDO CANCELAMENTO: Este status é atribuído à inscrição quando o usuário de marketing solicita seu cancelamento. A ideia é que o usuário marque inscrições que não se concretizarão com este status. Assim o sistema deve notificar o módulo financeiro para iniciar o cancelamento da NFS-e vinculada ao título/boleto desta inscrição.
CANCELADA: Quando a NFS-e vinculada ao título/boleto da inscrição for cancelada pelo sistema, o ERP deve atualizar o status da inscrição automaticamente para CANCELADA.

Fonte: DECISION IT SUPORTE EMPRESARIAL S.A.

Figura 4.3: Requisito da especificação dos estados da nota fiscal.

Requisito D8: Campo *STATUS NFS-e*: Possuirá 4 status. São eles:

EMITIDA: O ERP iniciar o status da nota como EMITIDA quando uma inscrição for criada.
AUTORIZADA: A Nota Fiscal (NFS-e) deve assumir este status quando a prefeitura confirmar a sua autorização.
AGUARDANDO CANCELAMENTO: A partir de uma solicitação de cancelamento advinda do módulo de inscrições, o sistema deve solicitar o cancelamento da nota à prefeitura. A nota permanece neste status até que a confirmação de cancelamento retorne.
CANCELADA: O ERP deve alterar o status da nota para CANCELADA quando a NFS-E vinculada ao titulo da inscrição for cancelada pela prefeitura. O sistema deve alterar o status da inscrição para CANCELADA também.

Fonte: DECISION IT SUPORTE EMPRESARIAL S.A.

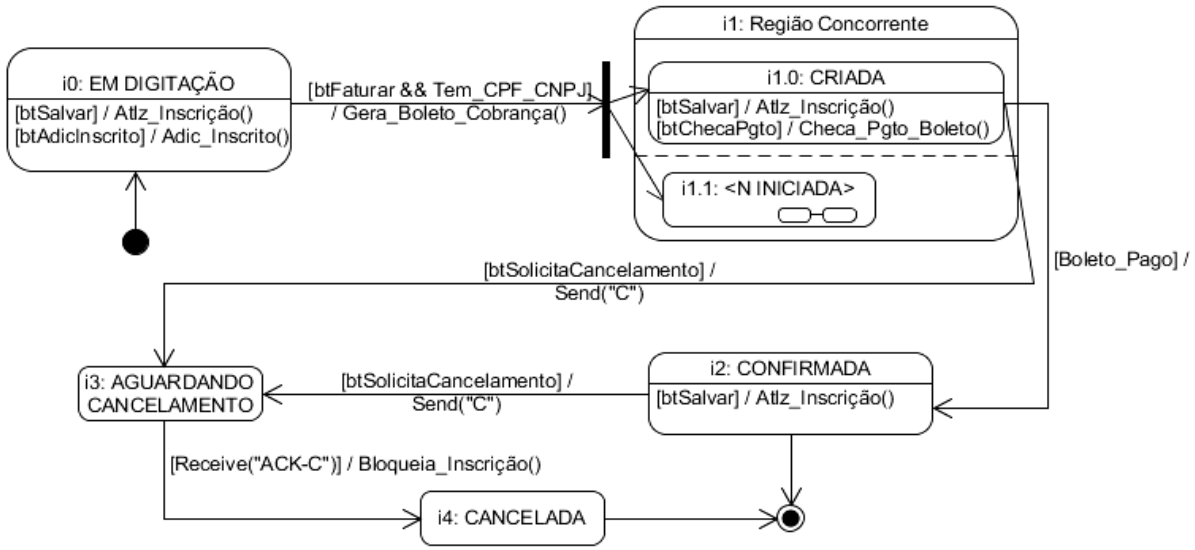
Através da especificação fornecida, as CHMs de cadastro de inscrições de emissão de notas-fiscais foram definidas abaixo:

- Seja **I** uma CFSM que representa o módulo de cadastro de inscrições de eventos corporativos.
- Seja **N** uma CFSM que representa o módulo de geração e emissão de notas fiscais eletrônicas para faturamento das inscrições cadastradas em I.

- **M : {C, ACK-C, E}** é o conjunto de mensagens compartilhadas e interpretadas por I e N onde C é a mensagem de solicitação de cancelamento, ACK-C é a mensagem de confirmação de cancelamento e E é a mensagem vazia.
- O par (I, N) é uma **rede de I e N** com estado inicial **[i0, s, E, E]** onde **s** é o estado vazio – aqui empregado até que a máquina N inicie, no estado n0, hierarquicamente a partir de I.
- Seja **M : {I, N, Ø}** o conjunto de CFSMs I, N e Ø onde **Ø é a máquina vazia**.
- **J : {i0, i1, i1.0, i1.1, i2, i3, i4}** é o conjunto de estados, pseudoestados, estados compostos, superestados e subestados de I.
- **O : {n0, n1, n2, n3, n4, n5}** é o conjunto de estados, pseudoestados, estados compostos, superestados e subestados de N.
- **α : J→M** é a função que associa os estados de I às máquinas de M definida como α : (i0→Ø, i1→Ø, i1.0→Ø, i1.1→N, i2→Ø, i3→Ø, i4→Ø) onde i1.0 e i1.1 são subestados do estado composto i1.
- **β : O→M** é a função que associa os estados de N às máquinas de M definida como β : (n0→Ø, n1→Ø, n2→Ø, n3→Ø, n4→Ø, n5→Ø).
- Então as triplas **(I, M, α)** e **(N, M, β)** são CHMs.

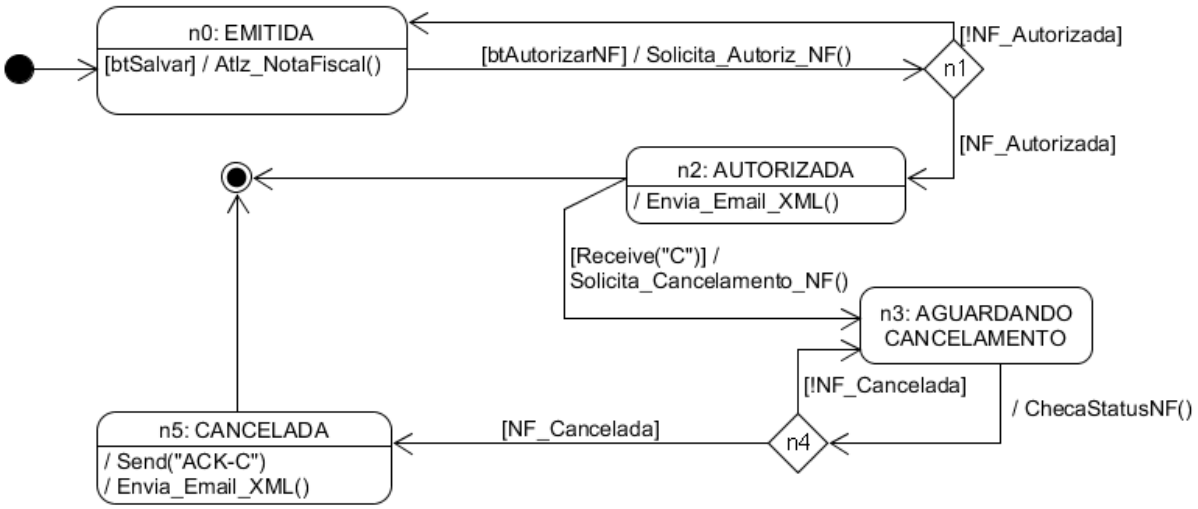
Nas Figuras 4.4 e 4.5 são apresentados os modelos das CHMs (I, M, α) e (N, M, β), respectivamente.

Figura 4.4: CHM (I, M, α) do módulo de inscrições para eventos corporativos.



Fonte: O Autor.

Figura 4.5: CHM (N, M, β) que representa o módulo de emissão de notas fiscais eletrônicas.



Fonte: O Autor.

Por definição, as CHMs (I, M, α) e (N, M, β) também são CFSMs relacionadas pela rede de I e N. A comunicação na rede de I e N pode ser verificada através do processo de cancelamento sincronizado das inscrições e das notas fiscais nas trocas de mensagens ocorridas nas transições de estados abaixo:

[i1.0, n2, E, E] → [i3, n3, E, C]: Solicitação de cancelamento partindo de I para N;
[i2, n2, E, E] → [i3, n3, E, C]: Solicitação de cancelamento partindo de I para N;
[i3, n4, E, E] → [i4, n5, ACK-C, E]: Confirmação de cancelamento partindo de N para I.

A hierarquia empregada no modelo para iniciar a CHM (N, M, β) a partir da CHM (I, M, α) é definida por **α : J→M** e ocorre no momento descrito abaixo:

[i0, s, E, E] → [i1, n0, E, E]: CHM (N, M, β) acionada a partir da CHM (I, M, α).

4.2.2 Tabela Estado-Transição

Sobre os modelos apresentados na Seção 4.2.1, as tabelas estado-transição das CHMs (I, M, α) e (N, M, β) geradas a partir das regras listadas na Seção 3.1.2 são mostradas nas Tabelas 4.1 e 4.2 respectivamente.

Tabela 4.1: Tabela estado-transição da CHM (I, M, α).

	i0: EM DIGITAÇÃO	i1.0 CRIADA	i1.1 <N INICIADA>	i2: CONFIRMADA	i3: AGUARDANDO CANCELAMENTO	i4: CANCELADA
Atlz_Inscrição()	1 - i0	4 - i1.0	●	8 - i2	●	●
Adic_Inscrito()	2 - i0	●	●	●	●	●
Gera_Boleto_Cobrança()	3 - i1	●	●	●	●	●
Checa_Pgto_Boleto()	●	5 - i1.0	●	●	●	●
Send()	●	6 - i3	●	9 - i3	●	●
Bloqueia_Inscrição()	●	●	●	●	10 - i4	●
[Boleto_Pago]	●	7 - i2	●	●	●	●

● Combinações ilegais

Fonte: O Autor.

Tabela 4.2: Tabela estado-transição da CHM (N, M, β).

	n0: EMITIDA	n1: PSEUDOESTADO DE DECISÃO	n2: AUTORIZADA	n3: AGUARDANDO CANCELAMENTO	n4: PSEUDOESTADO DE DECISÃO	n5: CANCELADA
Atlz_NotaFiscal()	1 - n0	●	●	●	●	●
Solicita_Autoriz_NF()	2 - n1	●	●	●	●	●
Solicita_Cancelamento_NF()	●	●	5 - n3	●	●	●
Checa_Status_NF()	●	●	●	6 - n4	●	●
[NF_Autorizada]	●	3 - n2	●	●	●	●
[!NF_Autorizada]	●	4 - n0	●	●	●	●
[NF_Cancelada]	●	●	●	●	7 - n5	●
[!NF_Cancelada]	●	●	●	●	8 - n3	●

● Combinações ilegais

Fonte: O Autor.

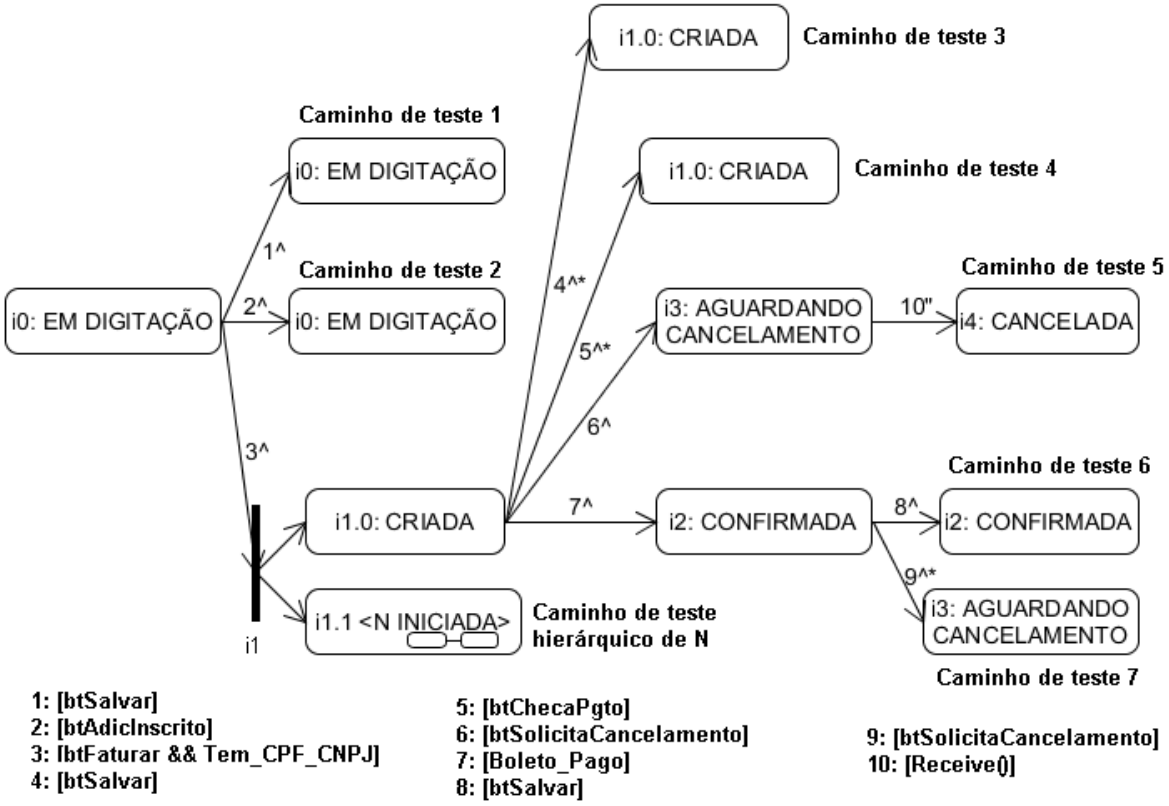
4.2.3 Árvore de Transições

Na etapa de modelagem CHM – Seção 4.2.1 – os modelos desenvolvidos tornam-se documentos acessórios do projeto visto que traduzem a especificação do software de uma forma mais visual.

Enquanto o modelo CHM contribui com a visão funcional do projeto, o documento que descreve a árvore de transições traz um benefício adicional ao projeto de desenvolvimento porquanto acrescenta a visão de navegação e de alcançabilidade dentro do software. Caminhos e fluxos possíveis ficam mais visíveis e perceptíveis facilitando a gestão dos requisitos de usabilidade do sistema.

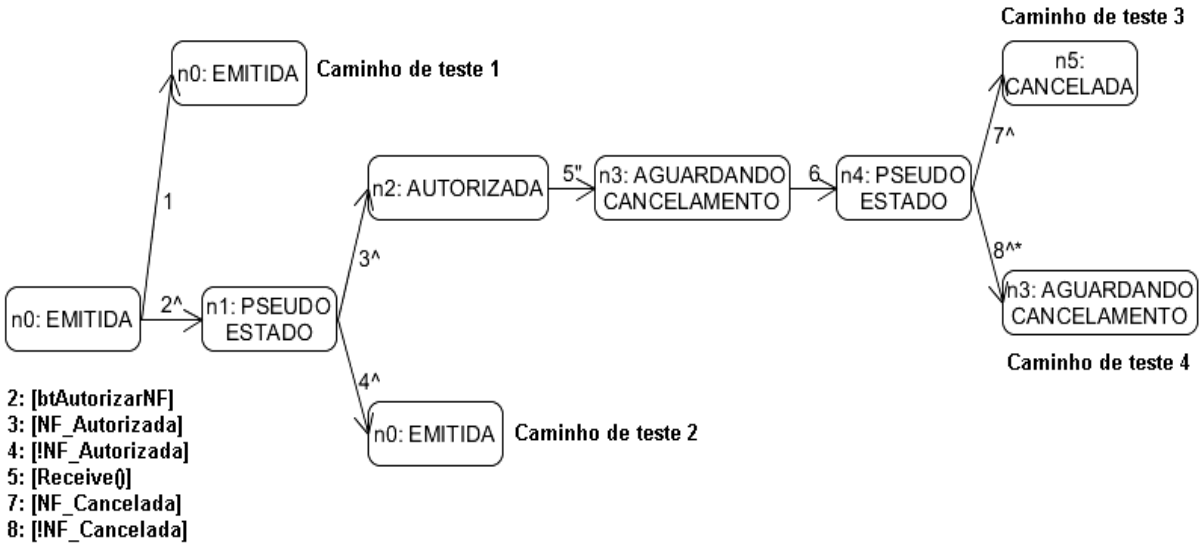
Através das regras apresentadas na Seção 3.1.3, as árvores de transições geradas para as CHMs (I, M, α) e (N, M, β) são apresentadas nas Figuras 4.6 e 4.7 respectivamente.

Figura 4.6: Árvore de transições da CHM (I, M, α).



Fonte: O Autor.

Figura 4.7: Árvore de transições da CHM (N, M, β).



Fonte: O Autor.

4.2.4 Casos de Teste Legais

Conforme a Seção 3.1.4, os roteiros de casos de teste legais para as CHMs (I, M, α) e (N, M, β) são apresentados nas Tabelas 4.3 e 4.4 respectivamente.

Tabela 4.3: Roteiro de casos de teste legais para a CHM (I, M, α).

	Entrada		Resultado Esperado	
ID	Evento	Guarda	Ação	Estado
L1.1	Atlz_Inscrição()	Usuário clicou no botão *Salvar*.		i0: EM DIGITAÇÃO
L2.1	Adic_Inscrito()	Usuário clicou no botão *Adicionar Inscrito*		i0: EM DIGITAÇÃO
L3.1	Gera_Boleto_Cobrança()	Usuário clicou no botão *Faturar*; Cadastro da inscrição possui CPF ou CNPJ.		i1.0: CRIADA; i1.1: <N INICIADA>
L3.2	Atlz_Inscrição()	Usuário clicou no botão *Salvar*.		i1.0: CRIADA
L4.1	Gera_Boleto_Cobrança()	Usuário clicou no botão *Faturar*; Cadastro da inscrição possui CPF ou CNPJ.		i1.0: CRIADA; i1.1: <N INICIADA>
L4.2	Checa_Pgto_Boleto()	Usuário rodou a rotina de sincronização com a instituição financeira.		i1.0: CRIADA
L5.1	Gera_Boleto_Cobrança()	Usuário clicou no botão *Faturar*; Cadastro da inscrição possui CPF ou CNPJ.		i1.0: CRIADA; i1.1: <N INICIADA>
L5.2	Send()	Usuário clicou no botão *Solicitar Cancelamento*.		i3: AGUARDANDO CANCELAMENTO
L5.3	Bloqueia_Inscrição()	O software recebeu a mensagem de *cancelamento confirmado* do módulo financeiro.		i4: CANCELADA
L6.1	Gera_Boleto_Cobrança()	Usuário clicou no botão *Faturar*; Cadastro da inscrição possui CPF ou CNPJ.		i1.0: CRIADA; i1.1: <N INICIADA>
L6.2		O boleto foi pago.		i2: CONFIRMADA
L6.3	Atlz_Inscrição()	Usuário clicou no botão *Salvar*.		i2: CONFIRMADA
L7.1	Gera_Boleto_Cobrança()	Usuário clicou no botão *Faturar*; Cadastro da inscrição possui CPF ou CNPJ.		i1.0: CRIADA; i1.1: <N INICIADA>
L7.2		O boleto foi pago.		i2: CONFIRMADA
L7.3	Send()	Usuário clicou no botão Solicitar Cancelamento.		i3: AGUARDANDO CANCELAMENTO

Fonte: O Autor.

Tabela 4.4: Roteiro de casos de teste legais para a CHM (N, M, β).

Entrada			Resultado Esperado	
ID	Evento	Guarda	Ação	Estado
L1.1	Atlz_NotaFiscal()	Usuário clicou no botão *Salvar*.		n0: EMITIDA
L2.1	Solicita_Autoriz_NF()	Usuário clicou no botão *Autorizar Nota Fiscal*.		n1: PSEUDOESTADO DE DECISÃO
L2.2		A nota fiscal não foi autorizada pela prefeitura.		n0: EMITIDA
L3.1	Solicita_Autoriz_NF()	Usuário clicou no botão *Autorizar Nota Fiscal*.		n1: PSEUDOESTADO DE DECISÃO
L3.2		A nota fiscal foi autorizada com sucesso pela prefeitura.	Envia_Email_XML()	n2: AUTORIZADA
L3.3	Solicita_Cancelamento_NF()	Requisição de cancelamento recebida da máquina I		n3: AGUARDANDO CANCELAMENTO
L3.4	Checa_Status_NF()			n4: PSEUDOESTADO DE DECISÃO
L3.5		A Nota Fiscal foi cancelada pela prefeitura.	Send(); Envia_Email_XML()	n5: CANCELADA
L4.1	Solicita_Autoriz_NF()	Usuário clicou no botão *Autorizar Nota Fiscal*.		n1: PSEUDOESTADO DE DECISÃO
L4.2		A nota fiscal foi autorizada com sucesso pela prefeitura.	Envia_Email_XML()	n2: AUTORIZADA
L4.3	Solicita_Cancelamento_NF()	Requisição de cancelamento recebida da máquina I		n3: AGUARDANDO CANCELAMENTO
L4.4	Checa_Status_NF()			n4: PSEUDOESTADO DE DECISÃO
L4.5		A Nota Fiscal não foi cancelada pela prefeitura.		n3: AGUARDANDO CANCELAMENTO

Fonte: O Autor.

4.2.5 Casos de Teste Ilegais

Consoante os passos descritos na Seção 3.1.5, os roteiros de casos de teste ilegais gerados para as CHMs (I, M, α) e (N, M, β) são apresentados nas Tabelas 4.5 e 4.6 respectivamente.

Tabela 4.5: Roteiro de casos de teste ilegais para a CHM (I, M, α).

ID	Setup	Estado	Evento / Guarda	Resultado
I1		i0: EM DIGITAÇÃO	Checa_Pgto_Boleto()	
I2		i0: EM DIGITAÇÃO	Send()	
I3		i0: EM DIGITAÇÃO	Bloqueia_Inscrição()	
I4		i0: EM DIGITAÇÃO	[Boleto_Pago]	
I5	L3.1	i1.0: CRIADA	Adic_Inscrito()	
I6	L3.1	i1.0: CRIADA	Gera_Boleto_Cobrança()	
I7	L3.1	i1.0: CRIADA	Bloqueia_Inscrição()	
I8	L3.1	i1.1: <N INICIADA>	Atlz_Inscrição()	
I9	L3.1	i1.1: <N INICIADA>	Adic_Inscrito()	
I10	L3.1	i1.1: <N INICIADA>	Gera_Boleto_Cobrança()	
I11	L3.1	i1.1: <N INICIADA>	Checa_Pgto_Boleto()	
I12	L3.1	i1.1: <N INICIADA>	Send()	
I13	L3.1	i1.1: <N INICIADA>	Bloqueia_Inscrição()	
I14	L3.1	i1.1: <N INICIADA>	[Boleto_Pago]	
I15	L6.2	i2: CONFIRMADA	Adic_Inscrito()	
I16	L6.2	i2: CONFIRMADA	Gera_Boleto_Cobrança()	
I17	L6.2	i2: CONFIRMADA	Checa_Pgto_Boleto()	
I18	L6.2	i2: CONFIRMADA	Bloqueia_Inscrição()	
I19	L6.2	i2: CONFIRMADA	[Boleto_Pago]	
I20	L5.2	i3: AGUARDANDO CANCELAMENTO	Atlz_Inscrição()	
I21	L5.2	i3: AGUARDANDO CANCELAMENTO	Adic_Inscrito()	
I22	L5.2	i3: AGUARDANDO CANCELAMENTO	Gera_Boleto_Cobrança()	
I23	L5.2	i3: AGUARDANDO CANCELAMENTO	Checa_Pgto_Boleto()	
I24	L5.2	i3: AGUARDANDO CANCELAMENTO	Send()	
I25	L5.2	i3: AGUARDANDO CANCELAMENTO	[Boleto_Pago]	
I26	L5.3	i4: CANCELADA	Atlz_Inscrição()	
I27	L5.3	i4: CANCELADA	Adic_Inscrito()	
I28	L5.3	i4: CANCELADA	Gera_Boleto_Cobrança()	
I29	L5.3	i4: CANCELADA	Checa_Pgto_Boleto()	
I30	L5.3	i4: CANCELADA	Send()	
I31	L5.3	i4: CANCELADA	Bloqueia_Inscrição()	
I32	L5.3	i4: CANCELADA	[Boleto_Pago]	

Fonte: O Autor.

Tabela 4.6: Roteiro de casos de teste ilegais para a CHM (N, M, β).

ID	Setup	Estado	Evento / Guarda	Resultado
I1		n0: EMITIDA	Solicita_Cancelamento_NF()	
I2		n0: EMITIDA	Checa_Status_NF()	
I3		n0: EMITIDA	[NF_Autorizada]	
I4		n0: EMITIDA	[!NF_Autorizada]	
I5		n0: EMITIDA	[NF_Cancelada]	
I6		n0: EMITIDA	[!NF_Cancelada]	
I7	L2.1	n1: PSEUDOESTADO DE DECISÃO	Atlz_NotaFiscal()	
I8	L2.1	n1: PSEUDOESTADO DE DECISÃO	Solicita_Autoriz_NF()	
I9	L2.1	n1: PSEUDOESTADO DE DECISÃO	Solicita_Cancelamento_NF()	
I10	L2.1	n1: PSEUDOESTADO DE DECISÃO	Checa_Status_NF()	
I11	L2.1	n1: PSEUDOESTADO DE DECISÃO	[NF_Cancelada]	
I12	L2.1	n1: PSEUDOESTADO DE DECISÃO	[!NF_Cancelada]	
I13	L3.2	n2: AUTORIZADA	Atlz_NotaFiscal()	
I14	L3.2	n2: AUTORIZADA	Solicita_Autoriz_NF()	
I15	L3.2	n2: AUTORIZADA	Checa_Status_NF()	
I16	L3.2	n2: AUTORIZADA	[NF_Autorizada]	
I17	L3.2	n2: AUTORIZADA	[!NF_Autorizada]	
I18	L3.2	n2: AUTORIZADA	[NF_Cancelada]	
I19	L3.2	n2: AUTORIZADA	[!NF_Cancelada]	
I20	L3.3	n3: AGUARDANDO CANCELAMENTO	Atlz_NotaFiscal()	
I21	L3.3	n3: AGUARDANDO CANCELAMENTO	Solicita_Autoriz_NF()	
I22	L3.3	n3: AGUARDANDO CANCELAMENTO	Solicita_Cancelamento_NF()	
I23	L3.3	n3: AGUARDANDO CANCELAMENTO	[NF_Autorizada]	
I24	L3.3	n3: AGUARDANDO CANCELAMENTO	[!NF_Autorizada]	
I25	L3.3	n3: AGUARDANDO CANCELAMENTO	[NF_Cancelada]	
I26	L3.3	n3: AGUARDANDO CANCELAMENTO	[!NF_Cancelada]	
I27	L3.4	n4: PSEUDOESTADO DE DECISÃO	Atlz_NotaFiscal()	
I28	L3.4	n4: PSEUDOESTADO DE DECISÃO	Solicita_Autoriz_NF()	
I29	L3.4	n4: PSEUDOESTADO DE DECISÃO	Solicita_Cancelamento_NF()	
I30	L3.4	n4: PSEUDOESTADO DE DECISÃO	Checa_Status_NF()	
I31	L3.4	n4: PSEUDOESTADO DE DECISÃO	[NF_Autorizada]	
I32	L3.4	n4: PSEUDOESTADO DE DECISÃO	[!NF_Autorizada]	
I33	L3.5	n5: CANCELADA	Atlz_NotaFiscal()	
I34	L3.5	n5: CANCELADA	Solicita_Autoriz_NF()	
I35	L3.5	n5: CANCELADA	Solicita_Cancelamento_NF()	
I36	L3.5	n5: CANCELADA	Checa_Status_NF()	
I37	L3.5	n5: CANCELADA	[NF_Autorizada]	
I38	L3.5	n5: CANCELADA	[!NF_Autorizada]	
I39	L3.5	n5: CANCELADA	[NF_Cancelada]	
I40	L3.5	n5: CANCELADA	[!NF_Cancelada]	

Fonte: O Autor.

4.2.6 Casos de Teste Hierárquicos

Segundo a Seção 3.1.6, foi gerado um roteiro de casos de teste hierárquicos para a CHM (I, M, α). O roteiro é apresentado na Tabela 4.7.

Tabela 4.7: Roteiro de casos de teste hierárquicos para a CHM (I, M, α).

		Entrada		Resultado Esperado	
ID	Setup	Evento	Guarda	Ação	Estado
H1	L3.1	Gera_Boleto_Cobrança()	Usuário clicou no botão Faturar; Cadastro da inscrição possui CPF ou CNPJ.		i1.0: CRIADA; i1.1: <N INICIADA>

Fonte: O Autor.

No caso da CHM (N, M, β), o roteiro de casos de teste hierárquicos não foi gerado visto que não há superestados no modelo.

4.3 Resultado

Através da aplicação da técnica proposta, a execução do experimento se mostrou simples, de baixo investimento e facilmente sustentável para uma empresa de porte pequeno como a DECISION IT.

Com a sua aplicação, a técnica aumentou o nível de confiança sobre os testes visto que todos caminhos de nível de profundidade 1, nas máquinas, foram cobertos. Ainda, a geração de testes ilegais é outro acréscimo valioso para o processo porquanto, no processo antigo, a geração de testes ilegais ficava dificultada quando feita intuitivamente.

Com os modelos, árvores e roteiros gerados, a empresa também obteve uma documentação acessória ao projeto. Esta documentação facilita a visualização e detecção de inconsistências, a manutenção e a atualização da especificação, antes composta apenas por descrição textual.

Com o processo estruturado uma gama de possíveis aprimoramentos tornou-se alcançável à empresa. O treinamento, a formação e a passagem de conhecimento para novos colaboradores foram facilitados através da formalização do método de testes. Por esta mesma razão, a definição e a aplicação de métricas para avaliação de qualidade podem ser realizadas de forma mais precisa e eficiente.

Por fim, visando-se a trabalhar com projetos de software maiores e mais complexos, a automatização da geração das árvores e dos roteiros também se tornou um benefício factível (mediante a expansão da técnica porquanto um passo adicional para avaliação de entradas e saídas será necessário à automação dos casos de teste).

5 CONCLUSÃO

Neste trabalho foi abordado o problema da geração de testes funcionais sobre softwares comunicantes e hierárquicos e, através da adaptação da técnica STT para CHMs, propôs uma nova técnica viável para solucionar este desafio em uma dada classe de sistemas.

Esta técnica foi desenvolvida no intuito de propor um processo de baixo investimento e de fácil execução para gerar casos de teste funcionais sobre especificações de projetos de softwares mais elaborados.

Considerando a importância do teste de software – atualmente presente e necessário em qualquer projeto profissional de desenvolvimento – a técnica mostrou alta aplicabilidade por apresentar-se simples e eficaz na geração de testes sobre softwares modelados como CHMs.

Portanto, acredita-se que a técnica pode ser adotada principalmente no nicho das micro, pequenas e médias empresas – onde processos não estruturados para testar software ainda são empregados.

Quando discutida a sua aplicação sobre projetos maiores, a técnica apresenta potencial para aprimoramento e expansão. Neste aspecto, se prospectam diversos trabalhos futuros no âmbito de continuar evoluindo a técnica. Por exemplo, sua execução no campo de sistemas distribuídos, sua compatibilidade com as demais propriedades de processamento concorrente, a cobertura de caminhos de maior profundidade nas máquinas, a definição de métricas formais para avaliação de seus resultados e a automação total do método.

Por fim, como trabalho futuro, sugere-se a observação da aplicação da técnica para avaliar-se o esforço e a aplicabilidade quando utilizada por uma terceira pessoa que, previamente, desconheça o método.

BIBLIOGRAFIA

MYERS, Glenford J. **The Art of Software Testing**. 2 ed. Nova Jérsei: John Wiley & Sons, 2004.

GILL, Arthur. *Introduction to* **The Theory of Finite-State Machines**. McGraw-Hill: [s.n.], 1962.

WAGNER, Ferdinand. **Modeling Software with Finite State Machines***: A Practical Approach*. [S.l]: CRC Press, 2006.

ALUR, Rajeev; KANNAN, Sampath; YANNAKAKIS, Mihalis. Communicating hierarchical state machines. **International Colloquium on Automata, Languages, and Programming**, Springer Berlin Heidelberg, p. 169-178, 1999.

BROEKMAN, Bart, and Edwin Notenboom. **Testing embedded software.** [s.l.]: Pearson Education, 2003.

BRAND, Daniel; ZAFIROPULO, Pitro. On communicating finite-state machines. **Journal of the ACM (JACM)**, [s.l.], v. 30, n. 2, p. 323-342, 1983.

ROSIER, Louis E.; GOUDA, Mohamed G. **Deciding progress for a class of communicating finite state machines**. Computer Science Department, University of Texas at Austin: [s.n.], 1983.

GOUDA, Mohamed G.; MANNING, Eric G.; YU, Yao-Tin. On the progress of communication between two finite state machines. **Information and control**, [S.l], v. 63, n. 3, p. 200-216, 1984.

AMMANN, Paul; OFFUTT, Jeff. **Introduction to software testing**. Cambridge University Press: [s.n.], 2008.

COHEN, D., LINDVALL, M., & COSTA, P. An introduction to agile methods. In Advances in Computers. **Elsevier Science.** New York, p. 1–66. 2004.

UML Statechart. Disponível em <uml.org>. Acessado em 28 de Setembro de 2016.

The Unified Modeling Language. Disponível em <uml-diagrams.com>. Acessado em 28 de Setembro de 2016.