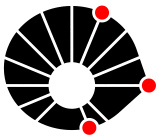

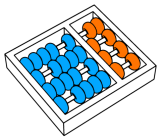

Modelagem e geração de testes de sistemas embarcados críticos

A. C. Bigheti *E. Martins* *A. M. Ambrosio*

Relatório Técnico - IC-PFG-24-36
Projeto Final de Graduação
2024 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Modelagem e geração de testes de sistemas embarcados críticos

Ariadne Bigheti* Eliane Martins† Ana M. Ambrosio ‡

Resumo

Este trabalho, desenvolvido em colaboração com o Instituto Tecnológico de Aeronáutica (ITA) e a Universidade de Tecnologia e Economia de Budapeste (BME), investiga abordagens sistemáticas para a modelagem e geração de casos de teste de um sistema embarcado crítico, denominado Train Door Controller.

A combinação das técnicas STPA e CoFI com a ferramenta GraphWalker foi empregada para criar casos de teste automatizados baseados em modelos de estados que descrevem o sistema.

Os casos de teste, implementados em JUnit, validaram o comportamento do sistema sem a identificação de falhas. Entretanto, alguns requisitos não foram completamente cobertos devido à falta de especificações claras, evidenciando tanto a importância de requisitos bem definidos quanto a capacidade da abordagem de identificar tais lacunas.

Os resultados reforçam a eficácia da integração de técnicas de modelagem com ferramentas automatizadas para garantir a qualidade de sistemas críticos, promovendo maior confiabilidade e segurança no desenvolvimento desses sistemas.

1 Introdução

Sistemas embarcados críticos desempenham um papel essencial em diversas áreas, como aeronáutica, automotiva, dispositivos médicos e equipamentos industriais. Nessas aplicações, o funcionamento correto e seguro do sistema é indispensável, pois qualquer falha pode levar a consequências severas, incluindo prejuízos econômicos, danos ambientais e, em casos mais extremos, risco à vida humana. Por isso, garantir a confiabilidade e a robustez desses sistemas é um desafio de alta prioridade na engenharia de software e sistemas embarcados.

Uma das abordagens mais eficazes para assegurar a qualidade em sistemas embarcados críticos é o uso de modelagem formal e técnicas de geração de testes. A modelagem permite descrever o comportamento do sistema de forma estruturada e precisa, enquanto a geração automatizada de testes a partir desses modelos oferece maior cobertura e eficiência no processo de validação. Essas práticas não apenas aumentam a confiança no sistema, como também podem reduzir significativamente o tempo e os custos de desenvolvimento.

Este trabalho integra um projeto colaborativo entre o Instituto Tecnológico de Aeronáutica (ITA) e a Universidade de Tecnologia e Economia de Budapeste (BME). Nesse projeto, o ITA é responsável pela aplicação da metodologia STPA descrita nas seções 3 e 4 para identificação e análise de requisitos de um sistema, enquanto a BME se encarrega da geração de um modelo executável, que será submetido a testes automatizados. Dentro desse contexto, este trabalho explora a aplicação de abordagens sistemáticas para modelagem e geração de testes de sistemas embarcados críticos, utilizando ferramentas e métodos que potencializam a detecção de falhas e a verificação de requisitos.

*Instituto de Computação, UNICAMP, 13083-852 Campinas, SP
†Instituto de Computação, UNICAMP, 13083-852 Campinas, SP
‡Instituto Nacional de Pesquisas Espaciais, INPE, 12227-010 São José dos Campos, SP

2 Objetivos

O objetivo deste trabalho é modelar e gerar testes de forma sistemática e automatizada para um sistema embarcado crítico denominado Train Door Controller (TDC). Este sistema é o controlador responsável por abrir e fechar automaticamente as portas de um trem, com base nos sinais recebidos de sensores externos.

O ITA aplica a técnica System-Theoretic Process Analysis (STPA) para identificação e análise de riscos, enquanto a BME realiza a aplicação de Model Checking e a implementação de um modelo executável utilizando a ferramenta Gamma [1]. A partir dos requisitos de segurança definidos pelo ITA, este trabalho busca modelar o sistema e gerar testes automatizados que serão aplicados ao modelo executável desenvolvido pela BME.

3 Metodologia

A metodologia adota uma abordagem sistemática que combina dois métodos, conforme descrito inicialmente por C. Hirata e A. Ambrosio em [2]. A técnica System-Theoretic Process Analysis (STPA) [3] é empregada para modelar o sistema crítico e identificar seus requisitos de segurança. Em seguida, a técnica Conformance and Fault Injection (CoFI) [4] é utilizada para gerar casos de teste, combinando conceitos de teste de conformidade e injeção determinística de falhas com base em máquinas de estados que representam o comportamento do sistema em diferentes cenários.

Baseando-se no conceito de Model-Based Testing (MBT) [5], os casos de teste são gerados com a ferramenta Graph Walker a partir das máquinas de estados obtidas pela CoFI. Os casos de teste, posteriormente, são implementados em JUnit para realizar a verificação final do sistema.

4 Fundamentação Teórica

4.1 Testes Baseados em Modelos de Estados

A crescente complexidade dos softwares modernos exige o desenvolvimento de modelos que representem de forma precisa o comportamento do sistema sob teste (System Under Test - SUT). Testes baseados em tentativa e erro ou métodos de força bruta não são viáveis, dada a vasta gama de combinações possíveis. Assim, a abordagem de teste deve ser sistemática, garantindo que, senão todas, a maior parte das combinações de eventos sejam avaliadas. Nesse contexto, os testes baseados em modelos destacam-se como uma solução eficaz, pois permitem a enumeração sistemática de combinações de entradas e estados.

Para que um modelo seja considerado testável, ele deve possibilitar a criação de um algoritmo que gere casos de teste com base apenas nas informações contidas no próprio modelo. Além disso, o modelo deve suportar tanto a geração manual quanto a automatizada de testes. Um modelo testável precisa atender aos seguintes critérios [5]:

1. **Representação completa e precisa**: o modelo deve refletir com fidelidade o tipo de implementação a ser testada, incluindo todos os recursos relevantes.

2. **Abstração eficiente**: deve omitir detalhes que aumentariam desnecessariamente o custo dos testes.

3. **Preservação de detalhes essenciais**: deve manter informações críticas para identificar falhas e verificar a conformidade do sistema.

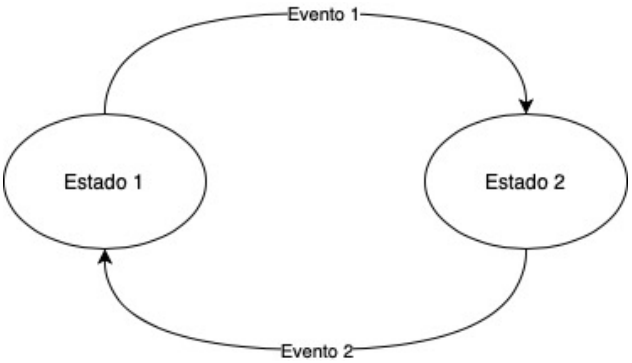

Figura 1: Exemplo de um diagrama de transição de estados

4. **Cobertura de eventos**: deve representar todos os eventos possíveis no modelo de estado, permitindo a geração de eventos como mensagens enviadas ao SUT.
5. **Representação de ações**: deve descrever todas as ações necessárias para avaliar se o sistema responde de forma adequada.
6. **Cobertura de estados**: deve permitir a determinação de quais estados foram atingidos (ou não) durante a execução.

Antes de ser utilizado para a geração de uma suíte de testes, um modelo de estados deve ser consistente e correto. De acordo com R. Binder [5], uma Máquina de Estados Finitos (Finite State Machine - FSM) satisfaz esses requisitos.

Uma Máquina de Estados é um modelo cuja saída é determinada pela entrada atual e pelo conjunto de entradas passadas, representadas por um estado. Uma transição ocorre quando o sistema muda de um estado para outro, sendo caracterizada então por um par de estados: o estado de origem e o estado resultante, que podem ser o mesmo. Uma máquina de estados possui as seguintes características:

- Está em apenas um estado por vez, denominado **estado atual** ou ativo.
- Realiza transições do estado atual para um estado resultante.
- Possui um **estado inicial**, que é o ponto de partida onde o primeiro evento é aceito.
- Pode ter um **estado final**, onde cessa a aceitação de novos eventos.
- É frequentemente representada por diagramas de transição de estados como na Figura 1, que ilustram visualmente os estados e as transições entre eles.

Essa representação sistemática facilita a análise do comportamento do sistema, contribuindo para a geração de testes robustos e eficazes.

4.2 System-Theoretic Process Analysis (STPA)

System-Theoretic Process Analysis (STPA) [3] é uma técnica que busca identificar ações de controle perigosas, cenários de perdas, fatores causais e requisitos de segurança de um sistema. A abordagem pode ser dividida em quatro etapas:

1. Definição do propósito de análise, onde identifica-se os serviços que o sistema deve prover e as suposições que existem sobre ele.
2. Identificação de perdas, hazards e restrições de segurança do sistema.
3. Modelagem da estrutura de controle do sistema. A estrutura de controle é um modelo de sistema hierárquico que compreende os componentes do sistema (controlador, sensor, atuador e processo controlado), fluxos de feedback e ação de controle, perturbações ambientais, entrada do sistema, saída do sistema e ação de controle do controlador de nível superior.
4. Identificação de ações de controle inseguras e requisitos de segurança.

4.3 Conformance and Fault Injection (CoFi)

Conformance and Fault Injection (CoFI) [4] é uma abordagem baseada em modelos que tem por objetivo criar casos de teste forma sistemática, onde o comportamento do sistema a ser testado é representado por Máquinas de Estados Finitos.

Utilizando a combinação STPA-CoFI, o primeiro passo da CoFI, *Identificar serviços do sistema*, é conduzido em conjunto com o primeiro passo da STPA, uma vez que eles compartilham o mesmo objetivo. Após realizar os quatro passos da STPA, os outros passos da CoFI devem ser realizados, de forma que o próximo passo é *Identificar pontos de controle e observação* (PCOs) na estrutura de controle obtida pela STPA.

Na CoFI, considera-se que cada ponto da estrutura de controle que recebe ou envia sinal é um PCO, de forma que:

- Feedbacks e comunicações externas são eventos de entrada;
- Ações de controle recebidas de controladores de nível superior são eventos de entrada;
- Ações de controle emitidas pelo Controlador são eventos de saída;
- Sinais capturados por Sensores são eventos de entrada;
- Ações emitidas por Atuadores são eventos de saída;
- Feedbacks enviados ao Controlador por Sensores são eventos de saída;
- Ações de controle recebidas por Atuadores são eventos de entrada;
- Atuações recebidas pelo Processo Controlado são eventos de entrada;
- Entradas do sistema vindas do ambiente são eventos de entrada;
- Saídas do sistema para o ambiente são eventos de saída;
- Eventos de start timer e timeout são modelados como eventos de saída e entrada, respectivamente.

O próximo passo da CoFI é *Construir as Máquinas de Mealy* que representam o comportamento do sistema.

Uma máquina de Mealy nada mais é do que uma Máquina de Estados Finitos que produz um resultado baseando-se no estado em que se encontra e na entrada de dados. Isto significa que o diagrama de estados irá incluir tanto o sinal de entrada como o de saída para cada transição.

A proposta da abordagem STPA-CoFI é mapear o fluxo da Estrutura de Controle do sistema em uma Máquina de Mealy. O método consiste em identificar o estado inicial, com a condição padrão do sistema, e as variáveis do sistema a partir da estrutura de controle. A mudança de cada variável indica uma transição na Máquina de Estados, e ao identificar as transições, identificamos também os estados da FSM.

A fim de reduzir a complexidade desses modelos de estados, as condições normais e excepcionais do sistema são separadas em diferentes classes de comportamento, gerando modelos dos tipos:

- **Normal**, que representa o comportamento do sistema em um cenário ideal;
- **Exceções Especificadas**, que representa o comportamento do sistema em um cenário de exceção que está especificado nos requisitos (ex.: sistema pode entrar em estado de emergência);
- **Caminhos furtivos**, que representa o comportamento do sistema em cenários inesperados com eventos ocorridos em estados que não deveriam ocorrer normalmente;
- **Tolerância à falhas**, que representa o comportamento do sistema quando há falhas físicas nos componentes do sistema. Para isso é utilizada a técnica de injeção de falhas, onde é simulada a falha desses componentes para verificar se o sistema se comporta como esperado frente às falhas físicas.

A próxima etapa consiste em *Verificar e complementar os diagramas* de máquinas de estados finitos (FSMs) para considerar os requisitos definidos pelo STPA. Essa etapa envolve analisar a cobertura dos requisitos da STPA pelos modelos existentes e, se necessário, expandir as FSMs para garantir que todos os requisitos sejam devidamente cobertos. Após essa revisão, os casos de teste para o sistema deverão ser gerados a partir destes modelos.

5 O estudo de caso

O sistema *Train Door Controller* deve prover as seguintes funções:

- Abrir automaticamente a porta quando o trem estiver parado em uma estação e for detectada a presença de uma pessoa.
- Fechar a porta antes de o trem deixar a estação.
- Garantir que a porta não se feche sobre nenhum passageiro ou funcionário.
- Impedir a abertura da porta enquanto o trem estiver em movimento ou fora de uma estação.
- Abrir a porta quando o trem estiver em estado de emergência.

6 Modelagem do Sistema

Adotando a metodologia descrita na seção 3, foram seguidos os passos da abordagem STPA, identificando o propósito da análise, perdas, hazards e restrições de segurança do sistema, a estrutura de controle, ações de controle inseguras e os requisitos de segurança; em seguida, foram aplicados os passos da CoFI.

A abordagem STPA, aplicada pelo Instituto Tecnológico de Aeronáutica (ITA), serviu como base para a execução deste trabalho e está detalhada na Seção 6.1. Por sua vez, a aplicação da abordagem CoFI, desenvolvida neste trabalho, é apresentada na Seção 6.2.

6.1 Abordagem STPA

6.1.1 Propósito da Análise

Objetivos

- O-1: Prover um meio de controlar a porta
- O-2: Garantir a segurança dos passageiros ao embarcar e desembarcar das estações
- O-3: Evitar qualquer tipo de dano aos passageiros
- O-4: Garantir uma viagem segura entre estações

Suposições

- S-1: O condutor não tem controle sobre a porta do trem

6.1.2 Perdas, hazards e restrições de segurança do sistema

Perdas

- P-1: Dano a uma pessoa ao cair para fora do trem
- P-2: Dano a uma pessoa ao ser atingida por uma porta se fechando
- P-3: Morte ou dano a uma pessoa presa dentro do trem em uma emergência

Hazards

- H-1: Porta se fechar em uma pessoa no caminho [P-2]
- H-2: Porta se abrir quando o trem está se movendo ou fora da estação [P-1]
- H-3: Passageiros/funcionários não conseguem sair durante uma emergência [P-3]

Restrições de segurança do sistema

- RSS-1: A porta não deve ser aberta quando o trem estiver em movimento [H-2]
- RSS-2: Em situação de emergência, a porta deve ser aberta [H-3]
- RSS-3: A porta não deve ser fechada quando houver uma pessoa ou objeto na porta [H-1]

6.1.3 Estrutura de Controle

A Estrutura de Controle do sistema está representada na Figura 2.

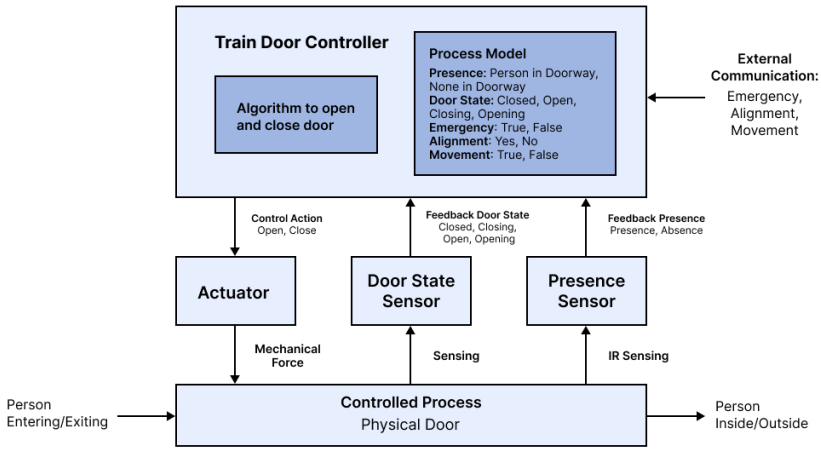

Figura 2: Estrutura de Controle do TDC

6.1.4 Ações de controle inseguras

- UCA-1: Controlador não fornece abertura quando Emergência é Sim. [H-3]
- UCA-2: Controlador não fornece abertura quando Presença é Verdadeiro. [H-1]
- UCA-3: Controlador fornece abertura quando Movimento é Sim. [H-2]
- UCA-4: Controlador fornece abertura quando o Alinhamento é Falso. [H-2]
- UCA-5: Controlador fornece abertura muito cedo quando o Alinhamento é Falso. [H-2]
- UCA-6: Controlador fornece abertura muito tarde quando Emergência é Sim. [H-3]
- UCA-7: Controlador fornece fechamento quando Presença é Verdadeiro. [H-1]
- UCA-8: O controlador fornece fechamento quando a Emergência é Sim. [H-3]
- UCA-9: O controlador fornece fechamento muito tarde quando o Alinhamento é Falso. [H-2]
- UCA-10: O controlador para de fechar muito cedo quando o estado é Fechando. [H-2]

6.1.5 Requisitos de Segurança

A partir das ações de controle inseguras foram identificados os requisitos de segurança do sistema de 1 à 10. Ainda, houve a necessidade de adicionar os requisitos 11 e 12.

- R-1: O Controlador deve fornecer Abertura quando Emergência é Sim.
- R-2: O Controlador deve fornecer Abertura quando Presença é Verdadeiro.
- R-3: O Controlador não deve fornecer Abertura quando Movimento é Sim.

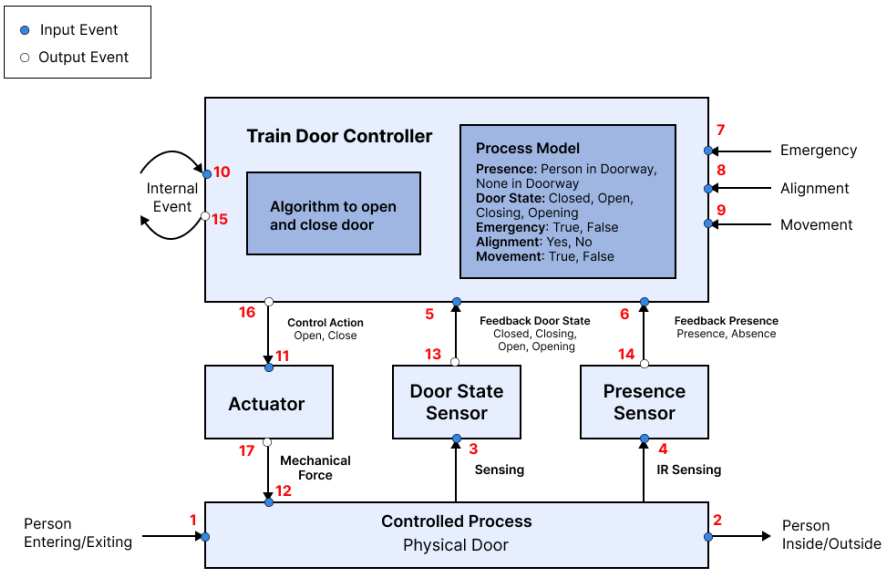

Figura 3: Estrutura de Teste do TDC com PCOs

- R-4: O Controlador não deve fornecer Abertura quando o Alinhamento é Não.
- R-5: O Controlador não deve fornecer Abertura muito cedo quando o Alinhamento é Não.
- R-6: O Controlador não deve fornecer Abertura muito tarde quando Emergência é Sim.
- R-7: O Controlador não deve fornecer Fechamento quando Presença é Verdadeiro.
- R-8: O Controlador não deve fornecer Fechamento quando Emergência é Sim.
- R-9: O Controlador não deve fornecer Fechamento muito tarde quando o Alinhamento é Não.
- R-10: O Controlador não deve fornecer Fechado muito cedo quando o estado for Fechando.
- R-11: O Controlador deve fornecer Fechamento quando o estado for Abrindo ou Aberto, Presença é Falso e Movimento é Sim ou Alinhamento é Não.
- R-12: O Controlador deve fornecer Fechamento quando o estado for Aberto e um timer (por exemplo, de 10s) tiver ocorrido.

6.2 Abordagem CoFI

6.2.1 Pontos de Controle e Observação

A partir da Estrutura de Controle definida na STPA, define-se a Estrutura de Teste (Figura 3) para identificação das entradas e saídas a serem usadas nos modelos da CoFI.

Na Estrutura de teste, as extremidades das setas representam os Pontos de Controle e Observação (PCOs). Em cada PCO são identificados os eventos de entrada e de saída de acordo com

a abordagem CoFI. Os Pontos de Controle (bolinhas fechadas) representam os eventos de entrada (input). Os Pontos de Observação (bolinhas abertas) representam os eventos de saída (output).

Para cada PCO identificado na Figura 3 foram criados os respectivos eventos na Tabela 1, separados entre eventos de entrada e saída.

PCO	**Evento e Descrição**	**Tipo**
1	Pnear: Pessoa se aproxima da porta PNotNear: Pessoa se afasta da porta	Input
3	SnsDetOpening: Sensor detecta início da abertura SnsDetEndOpening: Sensor detecta fim da abertura SnsDetClosing: Sensor detecta início do fechamento SnsDetEndClosing: Sensor detecta fim do fechamento	
4	SnsDetPresence: Sensor detecta presença SnsDetAbsence: Sensor detecta ausência	
5	CtrlRcvEndClosing: Controlador recebe feedback de fim do fechamento CtrlRcvEndOpening: Controlador recebe feedback de fim da abertura CtrlRcvClosing: Controlador recebe feedback de início do fechamento CtrlRcvOpening: Controlador recebe feedback de início da abertura	
6	CtrlRcvPresence: Indicação de presença recebida pelo controlador CtrlRcvAbsence: Indicação de ausência recebida pelo controlador	
7	EmOn: Emergência ativada EmOff: Emergência desativada	
8	AligmOn: Trem alinhado com a estação AligmOff: Trem desalinhado com a estação	
9	MovOn: Trem começa a se mover MovOff: Trem para de se mover	
10	TOut: Ocorreu um timeout de X segundos	
11	ActRcvOpen: Atuador recebe o comando de abertura ActRcvClose: Atuador recebe o comando de fechamento	
12	PhDoorOpening: Porta começa a abrir PhDoorEndOpening: Porta termina de abrir PhDoorClosing: Porta começa a fechar PhDoorEndClosing: Porta termina de fechar	
13	SnsTxOpening: Sensor transmite feedback de abertura SnsTxEndOpening: Sensor transmite feedback de fim da abertura SnsTxClosing: Sensor transmite feedback de fechamento SnsTxEndClosing: Sensor transmite feedback de fim do fechamento	Output
14	SnsTxPresence: Sensor transmite feedback de presença SnsTxAbsence: Sensor transmite feedback de ausência	
15	StTimer: Inicia um timer de X segundos UpDoorToClosing: Atualiza status da porta para fechando UpDoorToClosed: Atualiza status da porta para fechada UpDoorToOpening: Atualiza status da porta para abrindo UpDoorToOpen: Atualiza status da porta para aberta UpPresence: Atualiza status de presença para verdadeiro UpAbsence: Atualiza status de presença para falso UpEm: Atualiza status de emergência UpMov: Atualiza status de movimento UpAligm: Atualiza status de alinhamento	
16	IssOpen: Controlador emite comando de abertura IssClose: Controlador emite comando de fechamento	
17	ActClose: Atuador fechando a porta ActOpen: Atuador abrindo a porta	

Tabela 1: Lista de Pontos de Controle e Observação, com os inputs e outputs do TDC

Neste trabalho, para implementação dos modelos foi considerado apenas os PCOs que são eventos de entrada ou de saída do componente Train Door Controller, pois busca-se validar apenas

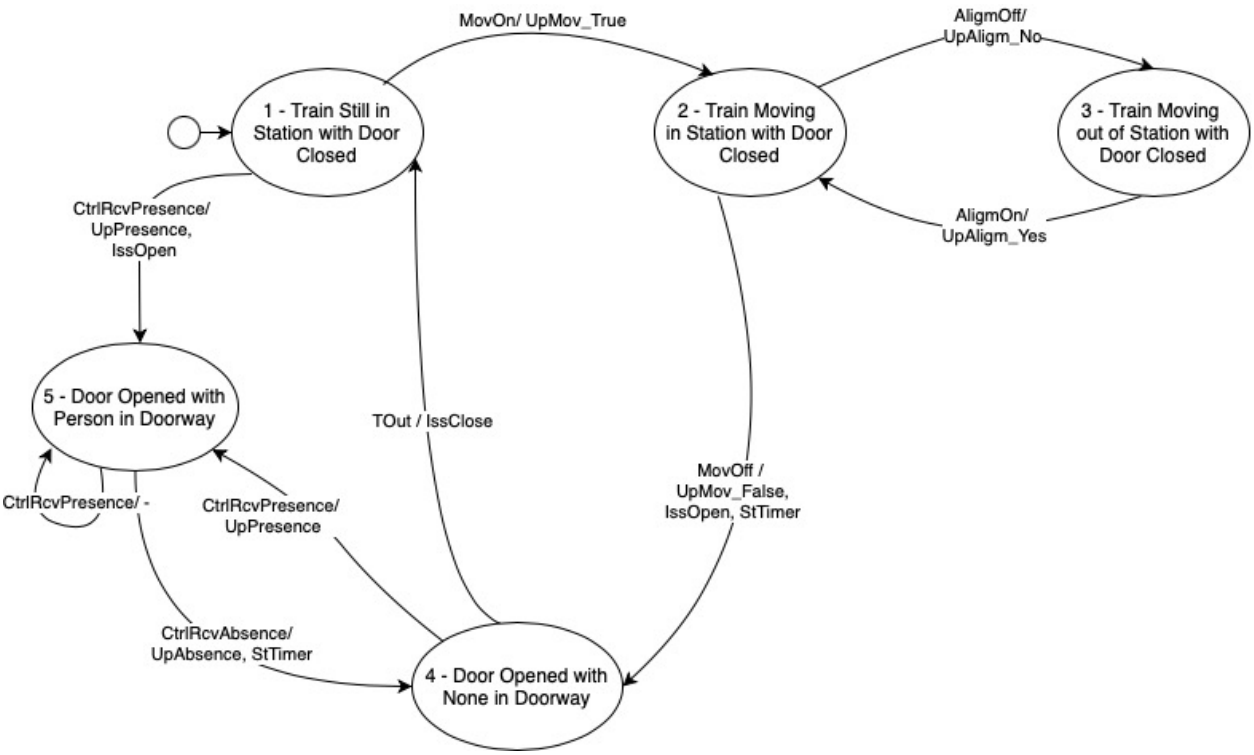

Figura 4: Modelo Normal (N) do Train Door Controller

o comportamento do TDC e não dos atuadores ou sensores. Esses eventos do TDC correspondem aos PCOs de número *5, 6, 7, 8, 9, 10, 15* e *16*.

6.2.2 Modelos de Estados

Os Modelos de Estados utilizados na CoFI são Máquinas de Mealy, onde suas transições representam os eventos de entrada e as saídas esperadas ou ações a serem executadas pelo controlador assim que um evento ocorre [4].

Para o cenário **Normal** (ver Figura 4) foram criados 5 estados:

1. *Trem parado na Estação com Porta Fechada*, representando o estado inicial do sistema onde o trem está parado, alinhado e com a porta fechada.

2. *Trem se movendo na Estação com Porta Fechada*, representando o estado em que o trem começa a se mover mas ainda está alinhado.

3. *Trem se movendo fora da Estação com Porta Fechada*, representando o estado em que o trem está se movendo e está desalinhado.

4. *Porta Aberta com Ninguém no Caminho*, representando o estado em que o trem está parado na estação com a porta aberta e não há ninguém no caminho da porta, esse estado também pode ser considerado como *"apto para fechar a porta"*.

5. *Porta Aberta com Alguém no Caminho*, representando o estado em que o trem está parado na estação com a porta aberta e há alguém no caminho da porta, pode ser considerado também como *"inapto para fechar a porta"*.

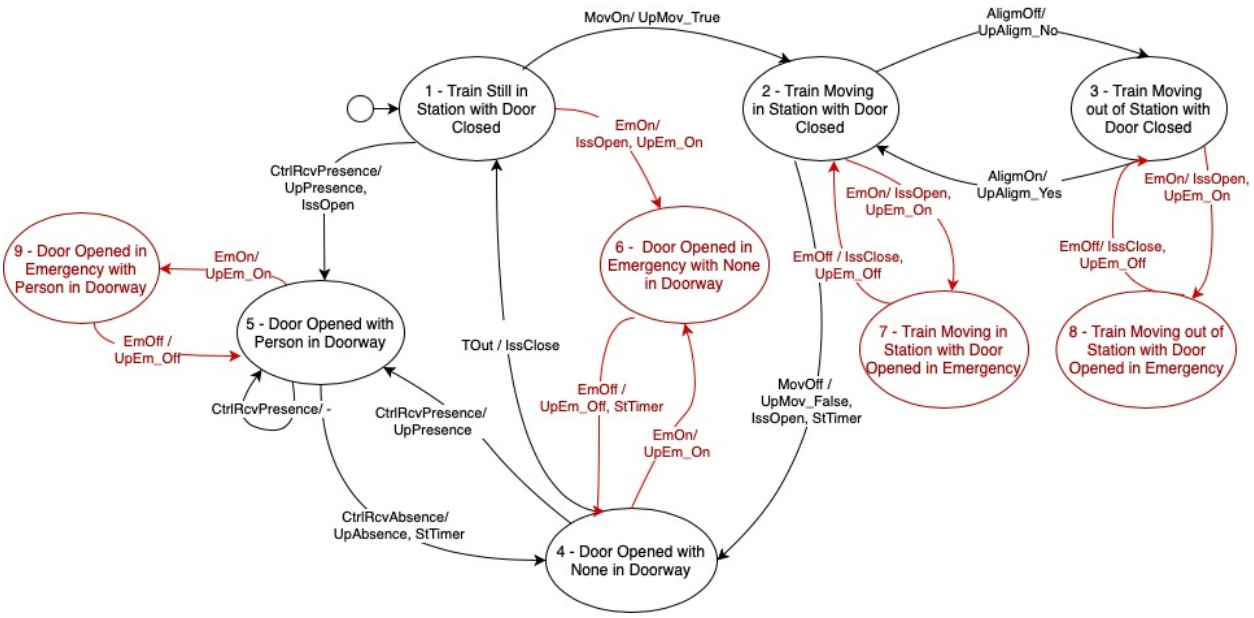

Figura 5: Modelo de Exceções Especificadas (Specified Exceptions - SEx) do Train Door Controller

A definição do estado inicial do sistema foi escolhida para estar em conformidade com o design da implementação da BME, de forma a facilitar os testes. De forma semelhante, a transição do Estado 2 para o Estado 4 foi incluída para atender ao design da implementação. Contudo, **não existe nenhum requisito** que exija que a porta do trem abra automaticamente ao parar em uma estação.

Adicionalmente, não há especificação clara sobre o momento em que o timer (requisito 12) deve ser ativado. Para a modelagem, optou-se por ativá-lo quando o sistema transita para o Estado 4: *Porta Aberta com Ninguém no Caminho*, pois neste estado o sistema está apto a fechar a porta.

Para o cenário de **Exceções Especificadas** (ver Figura 5) foram criados 4 novos estados, que representam todas as possibilidades do sistema em Estado de Emergência:

1. *Porta Aberta em Emergência com Ninguém no Caminho.*

2. *Trem se movendo na Estação com a Porta Aberta em Emergência*, representando o estado em que o trem está se movendo alinhado e ocorre uma emergência que faz o sistema abrir a porta.

3. *Trem se movendo fora da Estação com a Porta Aberta em Emergência*, representando o estado em que o trem está se movendo desalinhado e ocorre uma emergência que faz o sistema abrir a porta.

4. *Porta Aberta em Emergência com Alguém no Caminho.*

Na definição de requisitos, não ficou explicitado se o requisito 1, que determina abrir a porta em *Estado de Emergência*, tem prioridade sobre os requisitos 3 e 4 que proíbem a abertura da porta quando o trem está em movimento ou desalinhado. Para conformidade com o design da implementação, o requisito 1 foi priorizado, permitindo que o trem abra a porta nas transições dos estados 2 para 7 e 3 para 8.

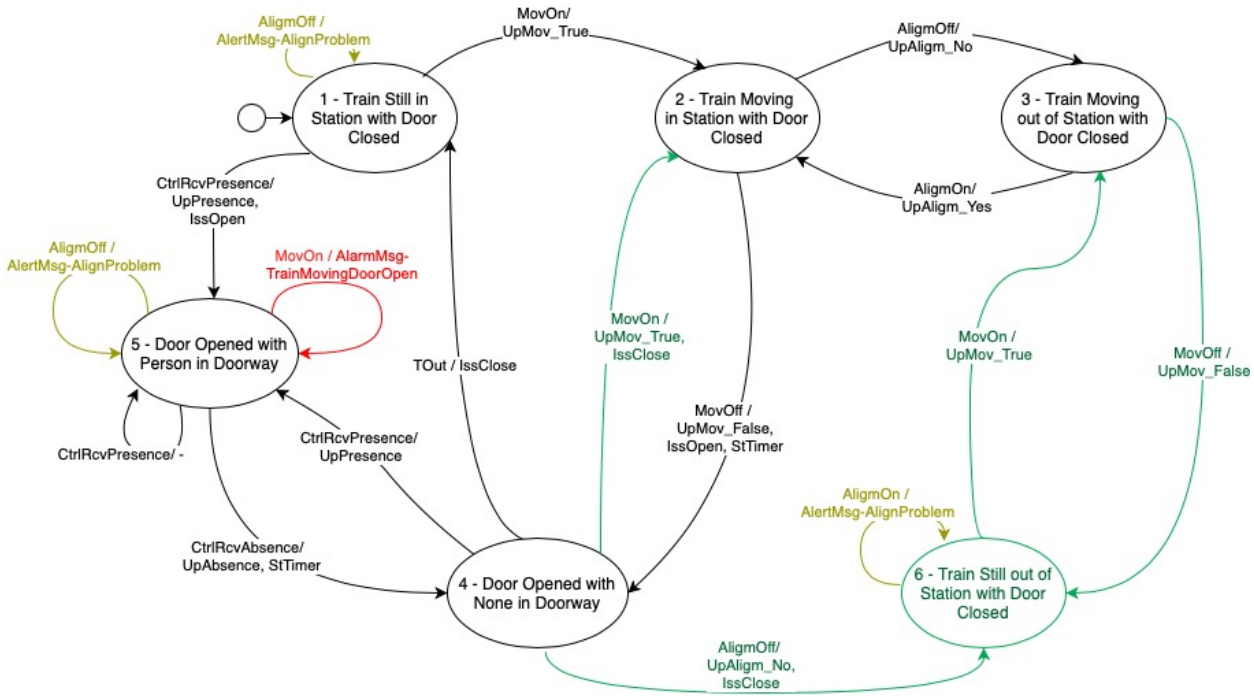

Figura 6: Modelo de Caminhos Furtivos (Sneak Paths - SP) do Train Door Controller

Para o cenário de **Caminhos Furtivos** (ver Figura 6), inicialmente é feita uma Tabela de Transição de Estados (ver Figura 7) na qual observa-se a combinação de todos os eventos possíveis ocorrendo em cada um dos estados normais. As células azuis na tabela representam eventos que podem ser ignorados, as pretas eventos já existentes no modelo, as vermelhas eventos críticos que geram um alarme no sistema, enquanto as amarelas geram um alerta, e as verdes são eventos novos que não representam problema no sistema. Nesta análise, identificou-se a necessidade de criar o Estado 6, além do cenário normal, que corresponde ao trem parado fora da estação com a porta fechada.

Dada a ausência de especificação sobre como o sistema deve reagir quando o trem começa a se mover com a porta aberta e alguém no caminho, foi criado um auto-laço no Estado 5 que gera uma mensagem de **alarme** no sistema, representando um cenário de risco.

De forma semelhante, quando o trem está parado na estação e ocorre um evento de desalinhamento, foi criado um auto-laço nos Estados 1 e 5 para gerar uma mensagem de **alerta** no sistema, indicando um possível erro no sensor, já que o trem está parado. O mesmo comportamento foi adotado para o trem parado fora da estação no Estado 6, caso ele receba um evento de alinhamento estando parado.

Analisando os requisitos de segurança do sistema, foi identificada a necessidade de criar as transições do Estado 4 para o 5 e de 4 para 2 para entrar em conformidade com o Requisito 11: *O Controlador deve fornecer Fechamento quando o estado for Abrindo ou Aberto, Presença é Falso e Movimento é Sim ou Alinhamento é Não.*

Neste trabalho, não foi feita a modelagem de **Tolerância à Falhas**, pois não havia especificação do comportamento do sistema frente a falhas físicas de componentes como sensores e atuadores.

Input Event / States	1-Train Still in Station wit	2- Train Moving in Station	3- Train Moving Out of St	4- Door Opened with None	5- Door Opened with Person in D	6- Train Still Out of Stati
MovOn	UpMov_True	null	null	UpMov_True, IssClose	AlarmMsg-TrainMovingDoorOpen	UpMov_True
	state 2	state 2	state 3	state 2	state 5	state 3
AligmOff	AlertMsg-Alignproblem	UpAligm_No	null	UpAligm_No, IssClose	AlertMsg-Alignproblem	null
	state 1	state 3	state 3	state 6	state 5	state 6
AligmOn	null	null	UpAligm_Yes	null	null	AlertMsg-Alignproblem
	state 1	state 2	state 2	state 4	state 5	state 6
MovOff	null	UpMov_False, IssOpen, StTimer	UpMov_False	null	null	null
	state 1	state 4	state 6	state 4	state 5	state 6
CtrlRcvPresence	UpPresence, IssClose	null	null	UpPresence	null	null
	state 5	state 2	state 3	state 5	state 5	state 6
CtrlRcvAbsence	null	null	null	null	UpAbsence, StTimer	null
	state 1	state 2	state 3	state 4	state 4	state 6
TOut	not aplicable	not aplicable	not aplicable	IssClose	null	null
	state 1	state 2	state 3	State 1	state 5	state 6

Figura 7: Tabela de Transição de Estados para a geração de Caminhos Furtivos

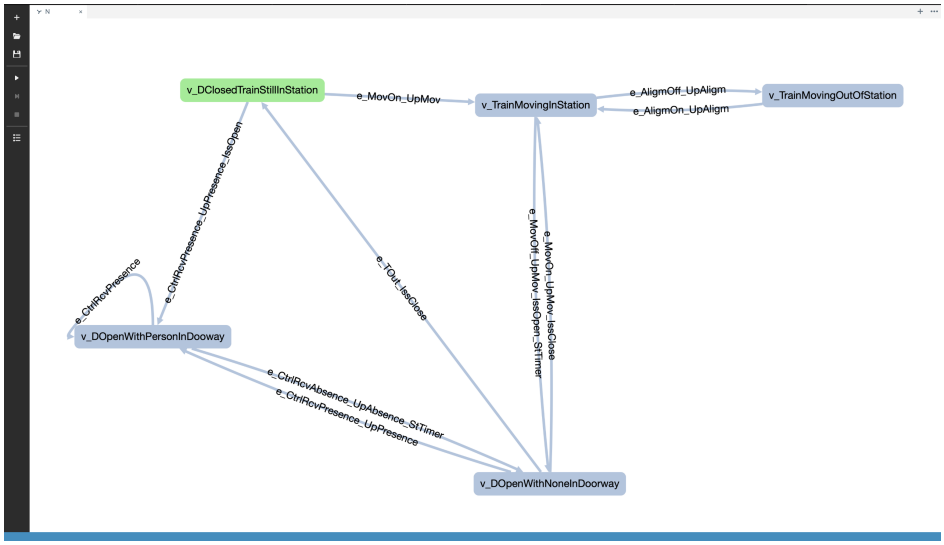

Figura 8: Edição do Model Normal na Graph Walker Studio

7 Geração de Testes

A ferramenta *GraphWalker* [6] é uma solução eficiente para a geração de casos de teste baseados em modelos. Ela adota como base modelos descritos em forma de grafos, como Máquinas de Estados Finitos (FSMs) ou Grafos de Fluxo de Controle (Control Flow Graphs), permitindo sua a criação e edição atráves do *GraphWalker Studio*. Os modelos criados podem depois ser exportados em um arquivo *json*.

A partir do arquivo *json* que descreve o modelo, a *GraphWalker* é capaz de gerar, por meio de sua *Command Line Tool* (CLI) [6] , os casos de teste necessários para atender aos critérios de cobertura especificados.

A ferramenta oferece diferentes estratégias de geração de caminhos de teste, como:

- **Cobertura de Todos os Nós (All Nodes):** Garante que cada estado no grafo seja visitado pelo menos uma vez.

- **Cobertura Todas as Arestas (All Edges):** Assegura que todas as transições sejam percorridas.
- **Cobertura Personalizada:** Permite definir critérios específicos, como priorizar estados críticos ou caminhos mais prováveis de conter falhas.
- **Random Walk:** Percorre o modelo através de uma sequência aleatória de transições, útil para identificar cenários inesperados.
- **Quick Random Walk**: Percorre o modelo de forma mais rápida, por meio de um algoritmo que seleciona uma aresta não visitada aleatoriamente e encontra o caminho mais curto até ela usando o algoritmo de Dijkstra. É eficiente para modelos grandes por gerar casos de testes menores, mas pode falhar em modelos EFSM (Extended Finite State Machine) caso encontre caminhos bloqueados por condições de guarda.

A ferramenta também suporta a execução de casos de testes em tempo real, permitindo verificar o comportamento do sistema enquanto percorre o modelo. Neste trabalho, optou-se por utilizar a geração de casos de testes *offline* por sua maior simplicidade em relação à geração *online* (em tempo real).

Desta forma, a partir dos diagramas de estados obtidos na abordagem CoFI, os modelos foram desenhados na *Graph Walker Studio* (ver Figura 8) e exportados em arquivos do tipo *json*. Os casos de testes foram gerados de forma offline utilizando o algoritmo **quick random**, a fim de gerar casos de testes menores, junto com um critério de parada de **cobertura total de arestas** para cobrir todas as transições dos modelos.

Nesta configuração, foram gerados 4 casos de testes a partir do modelo Normal, 4 casos de testes a partir do modelo Exceções Especificadas e 6 casos de testes a partir do modelo Caminhos Furtivos. O conjunto de casos de teste CoFI conta então com 14 casos de teste.

Abaixo estão apresentados os Casos de Teste número 1 de cada modelo, para fins de exemplificação:

- **Modelo Normal (Caso de Teste 1):**

  v_DClosedTrainStillInStation

  e_MovOn_UpMov

  v_TrainMovingInStation

  e_AligmOff_UpAligm

  v_TrainMovingOutOfStation

  e_AligmOn_UpAligm

  v_TrainMovingInStation

  e_MovOff_UpMov_IssOpen_StTimer

  v_DOpenWithNoneInDoorway

  e_CtrlRcvPresence_UpPresence

  v_DOpenWithPersonInDooway

  e_CtrlRcvPresence

  v_DOpenWithPersonInDooway

  e_CtrlRcvAbsence_UpAbsence_StTimer

v_DOpenWithNoneInDoorway
e_TOut_IssClose

- **Modelo de Exceções Especificadas (Caso de Teste 1):**
  v_DClosedTrainStillInStation
  e_CtrlRcvPresence_UpPresence_IssOpen
  v_DOpenWithPersonInDooway
  e_EmOn_UpEm
  v_DOpenInEmergencyWithPersonInDoorway
  e_EmOff_UpEm
  v_DOpenWithPersonInDooway
  e_CtrlRcvAbsence_UpAbsence_StTimer
  v_DOpenWithNoneInDoorway
  e_TOut_IssClose
- **Modelo de Caminhos Furtivos (Caso de Teste 1)**:
  v_DClosedTrainStillInStation
  e_CtrlRcvPresence_UpPresence_IssOpen
  v_DOpenWithPersonInDooway
  e_CtrlRcvAbsence_UpAbsence_StTimer
  v_DOpenWithNoneInDoorway
  e_CtrlRcvPresence_UpPresence
  v_DOpenWithPersonInDooway
  e_MovOn_AlarmMsgTrainMovingDoorOpen
  v_DOpenWithPersonInDooway
  e_CtrlRcvAbsence_UpAbsence_StTimer
  v_DOpenWithNoneInDoorway
  e_AligmOff_UpAligm_IssClose
  v_TrainStillOutOfStation
  e_MovOn_UpMov
  v_TrainMovingOutOfStation
  e_MovOff_UpMov
  v_TrainStillOutOfStation
  e_MovOn_UpMov
  v_TrainMovingOutOfStation
  e_AligmOn_UpAligm
  v_TrainMovingInStation
  e_MovOff_UpMov_IssOpen_StTimer

v_DOpenWithNoneInDoorway

e_CtrlRcvPresence_UpPresence

v_DOpenWithPersonInDooway

e_CtrlRcvPresence

v_DOpenWithPersonInDooway

e_CtrlRcvAbsence_UpAbsence_StTimer

v_DOpenWithNoneInDoorway

e_AligmOff_UpAligm_IssClose

v_TrainStillOutOfStation

e_AligmOn_AlertMsgAlignProblem

v_TrainStillOutOfStation

e_MovOn_UpMov

v_TrainMovingOutOfStation

e_AligmOn_UpAligm

v_TrainMovingInStation

e_MovOff_UpMov_IssOpen_StTimer

v_DOpenWithNoneInDoorway

e_TOut_IssClose

8 Implementação e Execução dos Testes

Ao utilizar o plugin **generate-sources** [6] da GraphWalker em um projeto Maven, interfaces de teste são geradas automaticamente a partir dos arquivos JSON dos modelos (ver Figura 9). Essas interfaces permitem que os casos de teste façam chamadas dos métodos correspondentes a cada aresta e estado do modelo, abstraindo sua implementação. Essa abordagem permite a reutilização de interfaces e métodos, além de facilitar a manutenção dos casos de teste.

A implementação das interfaces é realizada separadamente. Para cada aresta do modelo, é desenvolvido um método responsável por acionar o evento correspondente no controlador (ver Figura 10). De maneira semelhante, para cada estado, implementa-se um método que executa assertivas específicas para verificar se o controlador realizou corretamente as ações necessárias para a transição do sistema ao estado esperado (ver Figura 11).

Após a geração das interfaces e a implementação dos métodos para cada modelo, os casos de teste descritos na seção anterior foram criados em JUnit. Durante a execução dos testes contra a implementação da BME, nenhum erro foi identificado.

9 Análise dos Resultados

9.1 Resultados Obtidos

Nenhum erro foi detectado durante a execução dos testes contra a implementação do *Train Door Controller* pela BME. Contudo, podemos destacar problemas na definição dos requisitos do sistema que foram identificados durante a fase de modelagem. Os principais pontos são:

```java
@Model(file = "com/company/N1.json")  1 usage  1 implementation
public interface N1 {

    @Edge()  1 implementation
    void e_AligmOn();

    @Edge()  1 implementation
    void e_AligmOff();

    @Vertex()  6 usages  1 implementation
    void v_DOpenWithPersonInDoorway();

    @Vertex()  1 implementation
    void v_DOpenWithNoneInDoorway();
```

Figura 9: Parte da interface de teste gerada a partir do modelo Normal com o plugin generate-sources

```java
@Override
public void e_AligmOn() {
        trainDoorController.getExternalCommunication().raiseAlignment( isAligned: true);
        trainDoorController.schedule();
}

@Override
public void e_AligmOff() {
        trainDoorController.getExternalCommunication().raiseAlignment( isAligned: false);
        trainDoorController.schedule();
}
```

Figura 10: Implementação dos eventos AligmOn e AligmOff recebidos pelo TDC

```java
@Override  6 usages
public void v_DOpenWithPersonInDoorway() {
        assertTrue(trainDoorController.isStateActive( region: "MovementRegion",  state: "Still"));
        assertTrue(trainDoorController.isStateActive( region: "AlignmentRegion",  state: "Aligned"));
        assertTrue(trainDoorController.isStateActive( region: "PresenceRegion",  state: "PersonInDoorway"));
        assertTrue(trainDoorController.isStateActive( region: "DoorStateRegion",  state: "Open"));
}

@Override
public void v_DOpenWithNoneInDoorway() {
        assertTrue(trainDoorController.isStateActive( region: "MovementRegion",  state: "Still"));
        assertTrue(trainDoorController.isStateActive( region: "AlignmentRegion",  state: "Aligned"));
        assertTrue(trainDoorController.isStateActive( region: "PresenceRegion",  state: "NoneInDoorway"));
        assertTrue(trainDoorController.isStateActive( region: "DoorStateRegion",  state: "Open"));
}
```

Figura 11: Implementação das assertivas para os Estados 5 e 4, respectivamente

Requisito	Descrição	Modelos
1	O Controlador deve fornecer Abertura quando Emergência é Sim	SEx
2	O Controlador deve fornecer Abertura quando Presença é Verdadeiro	N, SEx, SP
3	O Controlador não deve fornecer Abertura quando Movimento é Sim.	N, SP
4	O Controlador não deve fornecer Abertura quando o Alinhamento é Não	N, SP
5	O Controlador não deve fornecer Abertura muito cedo quando o Alinhamento é Não	-
6	O Controlador não deve fornecer Abertura muito tarde quando Emergência é Sim	-
7	O Controlador não deve fornecer Fechamento quando Presença é Verdadeiro	N, SEx, SP
8	O Controlador não deve fornecer Fechamento quando Emergência é Sim	SEx
9	O Controlador não deve fornecer Fechamento muito tarde quando o Alinhamento é Não	-
10	O Controlador não deve fornecer Fechado muito cedo quando o estado for Fechando	-
11	O Controlador deve fornecer Fechamento quando o estado for Abrindo ou Aberto, Presença é Falso e Movimento é Sim ou Alinhamento é Não	SP
12	O Controlador deve fornecer Fechamento quando o estado for Aberto e um timer (por exemplo, de 10s) tiver ocorrido	N, SEx, SP

Tabela 2: Cobertura de requisitos por cada modelo

- **Falta de requisitos**: não há especificação para a abertura automática da porta quando o trem para em uma estação.
- **Conflitos entre os requisitos 1 e 3, e 1 e 4**: não está claro qual requisito deve ter precedência quando o trem está em movimento e ocorre uma emergência, ou quando o trem está desalinhado e ocorre uma emergência.
- **Requisitos 5, 6, 9 e 10 incompletos**: não foi definido o que se considera "muito cedo"ou "muito tarde".
- **Requisito 12 incompleto**: não foi especificado quando o timer deve ser ativado, nem o que deve ocorrer caso haja alguém no caminho da porta quando o timer expirar.
- **Falta de especificação sobre falhas físicas**: o comportamento do sistema diante de falhas físicas de componentes, como sensores e atuadores, não foi detalhado.

9.2 Cobertura de Requisitos

Fazendo uma análise dos requisitos cobertos por cada modelo obteve-se a Tabela 2.

Os requisitos *5, 6, 9* e *10* estão relacionados ao intervalo de tempo para abertura e fechamento da porta e não foram cobertos pela modelagem devido à sua incompletude, uma vez que não há especificação clara sobre o que caracteriza "muito cedo"ou "muito tarde". Com exceção destes requisitos, a abordagem adotada foi eficiente para garantir a cobertura dos requisitos nos casos de teste.

9.3 Algoritmos e Critérios de Parada

Para a geração de casos de teste neste trabalho, foi utilizado o algoritmo **quick random** com **cobertura total de arestas**, buscando gerar casos de teste mais enxutos e garantir a cobertura de todas as transições do modelo.

Ao analisar outras opções de geração de casos de teste, destacam-se os seguintes algoritmos e critérios de parada:

- **random(vertex_coverage(100)):** cobre todos os estados do modelo, mas pode não cobrir todas as transições, o que pode deixar passar algum cenário de erro. Tende a gerar mais casos de testes já que percorre o grafo de forma aleatória até visitar todos os vértices.
- **random(edge_coverage(100)):** cobre todas as arestas do modelo, de forma que todas as transições são testadas. Tende a gerar mais casos de testes já que percorre o grafo de forma aleatória até visitar todas as arestas.
- **random(edge_coverage(50)):** cobre metade das arestas do modelo, útil em testes exploratórios ou iniciais, mas pode deixar lacunas em requisitos mais rigorosos.
- **quick_random(vertex_coverage(100)):** cobre todos os estados do modelo, mas pode não cobrir todas as transições, o que pode deixar passar algum cenário de erro. Tende a gerar menos casos de testes já que percorre o grafo de forma mais rápida e eficiente.
- **quick_random(edge_coverage(100)):** cobre todas as arestas do modelo, de forma que todas as transições são testadas. Tende a gerar menos casos de testes já que percorre o grafo de forma mais rápida e eficiente.
- **quick_random(edge_coverage(50)):** cobre metade das arestas do modelo, útil em testes exploratórios ou iniciais, mas pode deixar lacunas em requisitos mais rigorosos.
- Combinar critérios, como **random(edge_coverage(100) and reached_vertex(vértice escolhido)**, aumenta o controle sobre os testes, mas pode resultar em um número maior de casos dependendo da complexidade do modelo.

9.4 Uso de Quick Random vs. Random

Para testes exploratórios, **quick random** com **cobertura parcial** oferece boa relação custo-benefício, devido à sua velocidade e simplicidade.

Para requisitos críticos, **random** com **cobertura total de arestas** garante a maior cobertura, mas pode gerar mais testes e demandar mais tempo e custo de execução, o que faz com que **quick random** com **cobertura total de arestas** seja uma boa substituição.

10 Conclusão

10.1 O que foi feito

Este trabalho apresentou uma abordagem sistemática para a modelagem e geração de testes em sistemas embarcados críticos, integrando técnicas avançadas, como a combinação das abordagens **STPA** e **CoFI**. Essas técnicas foram aplicadas para identificar os requisitos de segurança do sistema **Train Door Controller**, modelar seu comportamento em máquinas de estados e gerar casos de teste automatizados com o auxílio da ferramenta **GraphWalker**.

A principal vantagem dessa abordagem é a detecção precoce de falhas na definição dos requisitos, como ocorreu neste trabalho na fase de modelagem, o que contribui significativamente para a redução de custos de desenvolvimento ao evitar que problemas sejam descobertos em fases mais avançadas.

Outra vantagem significativa é o uso da **GraphWalker**, que permite a geração automatizada de casos de teste e facilita a manutenção deles por meio de interfaces intuitivas, reduzindo o tempo de desenvolvimento e melhorando a eficiência.

Os resultados obtidos confirmam que o uso combinado dessas técnicas foi eficaz para modelar o sistema e validar a conformidade dele com os requisitos de segurança. Além disso, as abordagens propostas têm o potencial de não apenas melhorar a confiabilidade de sistemas críticos, mas também reduzir o tempo e os custos envolvidos no seu desenvolvimento.

10.2 Melhorias e trabalhos futuros

Como sugestão de trabalho futuro recomenda-se a introdução de mutantes nos modelos para verificar se os conjuntos de testes são eficientes. O uso de **mutantes** em testes é uma técnica amplamente utilizada para avaliar a qualidade de casos de teste e sua capacidade de detectar falhas. Essa abordagem, conhecida como **testes de mutação**, envolve a introdução de pequenas alterações deliberadas (mutantes) no código-fonte ou, neste caso, nos modelos do sistema em teste. O objetivo principal é verificar se os casos de teste existentes são capazes de identificar essas alterações e "aniquilar"os mutantes, ou seja, detectar os erros inseridos.

Outra sugestão é a definição de novos requisitos voltados à tolerância a falhas, acompanhada do desenvolvimento de um modelo de injeção de falhas utilizando a abordagem CoFI. Entre as falhas físicas que podem ser consideradas estão: falha no Sensor de Presença, falha no Sensor de Estado da Porta e falha no Atuador. Para aumentar a confiabilidade do sistema, pode-se adicionar um segundo sensor, mas, nesse caso, o design deverá ser modificado para integrar e gerenciar o sensor adicional de forma adequada. Com isso, o sistema ficará robusto e tolerante a falhas, aumentando sua confiabilidade e garantindo uma operação mais segura, mesmo em situações adversas.

Referências

[1] GAMMA STATECHART COMPOSITION FRAMEWORK. Disponível em: `http://inf.mit.bme.hu/en/gamma`

[2] C. M. Hirata and A. M. Ambrosio, "Combining STPA With CoFI to Generate Requirements and Test Cases for Safety-Critical System,"in IEEE Systems Journal, vol. 16, no. 4, pp. 6635-6646, Dec. 2022

[3] N. Leveson and J. Thomas, "STPA Handbook,"2018.

[4] A. M. Ambrosio and E. Martins. "A Conformance Testing Process for Space Applications Software Services", Journal of Aerospace Computing, Information, and Communication (JACIC)/AIAA, vol. 3 no. 4, p. 146–158, abr. 2006.

[5] R. Binder, "Testing Object-oriented Systems: Models, Patterns, and Tools" [s.l: s.n.].

[6] GRAPHWALKER. An open-source model-based testing tool. Disponível em: `http://graphwalker.github.io/`