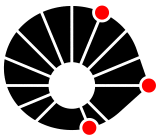

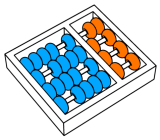

Aplicação Integrada de STPA e CoFI para Derivação de Casos de Teste em Sistemas de Robótica Móvel Colaborativa

Luiz Henrique Marques Gonçalves *Eliane Martins*

Relatório Técnico - IC-PFG-26-31
Projeto Final de Graduação
2026 - Julho

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Aplicação Integrada de STPA e CoFI para Derivação de Casos de Teste em Sistemas de Robótica Móvel Colaborativa

Luiz Henrique Marques Gonçalves* Eliane Martins†

Resumo

Robôs móveis autônomos que operam em ambientes compartilhados com pessoas são sistemas críticos de segurança: falhas de controle têm consequência física direta. Na prática, a análise de segurança e a geração de casos de teste costumam ser conduzidas de forma desarticulada, sem rastreabilidade explícita entre as restrições de segurança levantadas e os testes efetivamente executados. Este trabalho investiga a aplicação integrada de duas técnicas complementares — a *System-Theoretic Process Analysis* (STPA), para a análise de segurança, e a *Conformance and Fault Injection* (CoFI), para a modelagem comportamental e derivação de casos de teste — estendendo uma combinação já proposta na literatura com os passos finais de concretização: a geração e execução automatizada dos testes, por meio da ferramenta AltWalker, contra um simulador independente do sistema, e uma validação preliminar da eficácia da suíte por análise de mutação. A abordagem é aplicada a um cenário concreto — um robô autônomo de entrega de bandejas em um restaurante — e resulta em um *pipeline* rastreável de ponta a ponta, que liga cada perda e perigo identificados na análise a uma restrição de segurança, a uma transição do modelo comportamental e, por fim, a um caso de teste executável.

1 Introdução

Robôs móveis autônomos deixaram de ser uma promessa de laboratório e passaram a operar em ambientes cotidianos compartilhados com pessoas — restaurantes, hospitais, armazéns e residências. Diferentemente de um sistema puramente computacional, um robô que se desloca fisicamente nesses espaços é um sistema crítico de segurança: uma decisão de controle inadequada não produz apenas uma saída errada, mas pode resultar em colisão, queda de carga ou dano a pessoas e ao ambiente. Garantir que o comportamento desses sistemas seja seguro, previsível e controlado é, portanto, um requisito de primeira ordem, e não um detalhe de acabamento.

Duas atividades são centrais para essa garantia: a análise de segurança, que identifica de que formas o sistema pode levar a um acidente ou dano, e o teste, que verifica se a implementação de fato se comporta como esperado. Na prática, porém, essas duas atividades costumam ser conduzidas de forma desarticulada, por equipes e ferramentas distintas. O

*Instituto de Computação, Universidade Estadual de Campinas, 13083-852 Campinas, SP. RA 183218.
†Instituto de Computação, Universidade Estadual de Campinas. Orientadora.

resultado é uma lacuna de rastreabilidade: não se garante que cada restrição de segurança levantada na análise seja efetivamente exercitada por um caso de teste, nem que uma falha observada em teste possa ser atribuída a uma restrição de segurança concreta. E, mesmo quando análise e teste são combinados, a avaliação pode encerrar-se na geração de casos de teste abstratos, sem que estes sejam executados contra uma implementação ou tenham sua capacidade de detecção de defeitos avaliada.

Este trabalho parte da combinação de duas técnicas complementares. A *System-Theoretic Process Analysis* (STPA) [2] é uma técnica de análise de segurança que trata acidentes como um problema de controle inadequado — e não apenas como cadeias de falhas de componentes —, derivando sistematicamente perigos e restrições de segurança a partir da estrutura de controle do sistema. A *Conformance and Fault Injection* (CoFI) [3] é uma metodologia de teste baseada em modelos que sistematiza a construção de máquinas de estados finitos (FSMs) representando o comportamento do sistema — normal e anormal — e delas deriva casos de teste. Hirata e Ambrosio [4] mostraram que STPA e CoFI podem ser combinadas, pois a estrutura de controle produzida pela STPA já contém a informação necessária para construir os modelos da CoFI; contudo, sua avaliação encerra-se na geração de casos de teste abstratos, sem execução contra uma implementação nem medição de eficácia.

O objetivo deste trabalho é usar essa combinação — estendida com os passos finais que a concretizam — para gerar e executar casos de teste para um robô autônomo de entrega de bandejas em um restaurante; a avaliação da própria combinação, de forma integrada e rastreável, é consequência desse exercício. O foco funcional é o controlador do robô — o componente automatizado que decide as ações de navegação (mover, parar) a partir dos sensores e dos comandos recebidos. As duas técnicas se complementam nesse propósito: a STPA fornece as restrições de segurança que o controlador deve respeitar, e a CoFI as transforma em modelos comportamentais executáveis. O *pipeline* resultante, ilustrado na Figura 1, encadeia quatro fases — análise (STPA), modelagem (CoFI), geração e execução de testes (AltWalker) e validação (análise de mutação) — de modo que cada elemento de uma fase é derivado e rastreável ao da fase anterior.

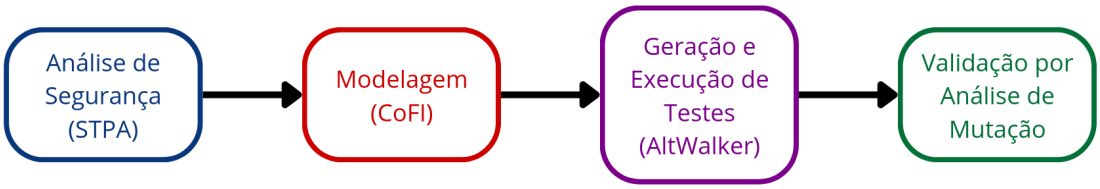

Figura 1: Visão geral do *pipeline* integrado e rastreável adotado.

As principais contribuições deste trabalho são: (i) a aplicação do método combinado STPA + CoFI a um novo cenário concreto de robótica móvel colaborativa, distinto dos casos de estudo do trabalho de referência; (ii) a concretização dos passos finais de geração — a execução automatizada dos testes contra um simulador independente do sistema sob

teste e uma validação preliminar de sua eficácia por análise de mutação — que o trabalho de referência deixa em aberto; e (iii) o estabelecimento de uma rastreabilidade fim-a-fim, que liga os riscos identificados na análise de segurança aos casos de teste que os exercitam, tornando cada resultado de teste atribuível a uma restrição de segurança específica.

O restante deste texto está organizado da seguinte forma. A Seção 2 apresenta a fundamentação teórica das técnicas empregadas (STPA, CoFI e análise de mutação). A Seção 3 descreve o sistema exemplo. A Seção 4 detalha a metodologia, isto é, como as técnicas se combinam no *pipeline* adotado. A Seção 5 apresenta a aplicação passo a passo ao robô de entrega. A Seção 6 discute os resultados e a avaliação. Por fim, a Seção 7 traz as conclusões e os trabalhos futuros.

2 Fundamentação Teórica

Esta seção apresenta as três técnicas que sustentam o trabalho: a STPA, para a análise de segurança (Seção 2.1); a CoFI, para a modelagem comportamental e a derivação de casos de teste (Seção 2.2); e a análise de mutação, empregada na avaliação da suíte de testes (Seção 2.3). A forma como STPA e CoFI se articulam é tratada adiante, na Seção 4.

2.1 System-Theoretic Process Analysis (STPA)

A STPA é uma técnica de análise de segurança baseada na teoria STAMP (*System-Theoretic Accident Model and Processes*), proposta por Nancy Leveson [1]. Sua ideia central é que acidentes não resultam apenas de componentes que falham, mas de interações que levam o sistema a uma decisão insegura: o sensor pode estar funcionando, o *software* pode estar funcionando e o atuador pode estar funcionando, e ainda assim ocorrer um acidente porque a coordenação entre eles produziu um comando inadequado. O foco desloca-se, assim, de "falhas de componentes" para o controle inadequado do sistema. Essa é a diferença essencial em relação a técnicas clássicas como a FMEA (*Failure Mode and Effects Analysis*): enquanto a FMEA pergunta "o que acontece se este componente falhar?", a STPA pergunta "quais ações de controle inadequadas podem levar o sistema a um estado perigoso?". Uma falha de componente é, aqui, apenas uma das causas possíveis de controle inadequado — a STPA a considera, mas não se limita a ela.

Como método de análise, a STPA segue quatro passos [2], ilustrados na Figura 2:

1. **Definir o propósito da análise** — delimitar a fronteira do sistema e identificar as perdas (*losses*) inaceitáveis, os perigos (*hazards*) e as restrições de segurança de nível de sistema. Uma perda envolve algo de valor para as partes interessadas (vida, saúde, propriedade, missão); um perigo é um estado do sistema que, sob condições ambientais de pior caso, levará a uma perda.

2. **Modelar a estrutura de controle** — representar os controladores, os atuadores, os sensores e o processo controlado, com as ações de controle e os *feedbacks* entre eles, incluindo o *process model* de cada controlador (as variáveis com que ele decide suas ações).

3. **Identificar Ações de Controle Inseguras (UCAs)** — para cada ação de controle, determinar os contextos em que ela é insegura, segundo quatro formas canônicas: a ação não é fornecida quando deveria; é fornecida quando não deveria; é fornecida no momento ou ordem errados; ou tem duração errada. Cada UCA dá origem a uma restrição de segurança derivada.

4. **Identificar cenários de perda** — para cada UCA, descobrir os fatores causais que a explicam e por que uma ação de controle, uma vez emitida, poderia ser mal executada; cada cenário fundamenta uma recomendação de projeto.

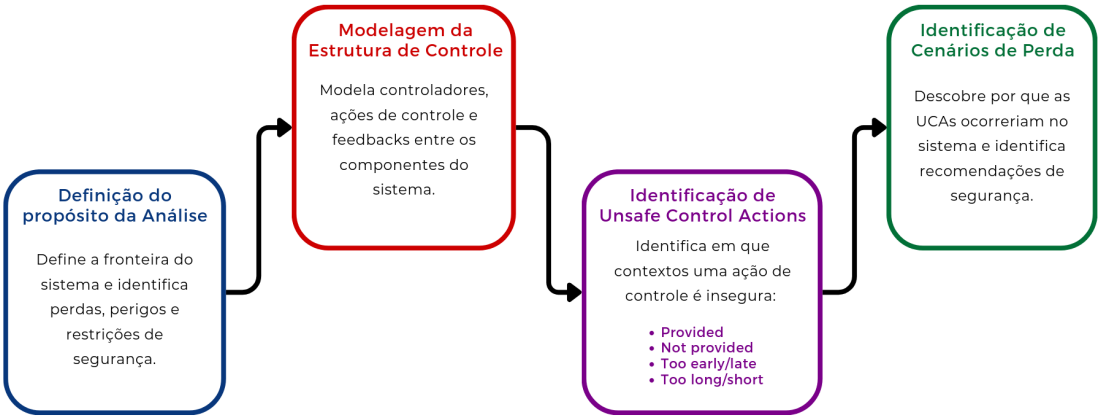

Figura 2: Os quatro passos da STPA.

Os passos 3 e 4 constituem uma análise exaustiva: no passo 3, cada ação de controle é examinada em cada contexto do *process model* e segundo cada tipo de UCA; no passo 4, cada UCA é percorrida em busca de seus fatores causais. O número de combinações a considerar cresce rapidamente e pode facilmente tornar-se grande demais para ser tratado manualmente sem que algo passe despercebido. Por isso, recorre-se a ferramentas de apoio como o AppSTPA [5] — empregado na Seção 5 —, que organizam essa varredura e ajudam a garantir que nenhuma combinação fique sem análise.

2.2 Conformance and Fault Injection (CoFI)

A CoFI (*Conformance and Fault Injection*) [3] é uma metodologia de teste baseado em modelos (*Model-Based Testing*, MBT) [8], de caráter caixa-preta. No MBT, os casos de teste são derivados automaticamente de um modelo comportamental do sistema, comumente uma máquina de estados finitos (FSM), percorrido para produzir sequências de eventos com as saídas esperadas correspondentes. Na CoFI, essa FSM é do tipo Mealy, em que cada transição associa um evento de entrada a uma saída esperada. A metodologia combina, ainda, dois conceitos de teste complementares, a conformidade (verificar se o comportamento observado corresponde ao especificado) e a injeção de falhas (verificar o comportamento sob

condições anormais), e sistematiza a própria construção das FSMs, a partir das quais os casos de teste são gerados.

Essa construção é organizada em oito passos: os dois primeiros caracterizam o sistema, os quatro seguintes constroem as FSMs de cada classe de comportamento e os dois últimos preparam e geram os casos de teste.

1. **Identificar os serviços do sistema** — as principais funções que o sistema oferece ao seu ambiente, que delimitam o escopo do que será modelado e testado.

2. **Identificar os PCOs, os eventos e as ações** — os PCOs (*Points of Control and Observation*) são as interfaces do sistema associadas aos seus eventos e ações: os pontos de controle, por onde os eventos de entrada são fornecidos ao sistema, e os pontos de observação, por onde suas ações de saída são observadas (podendo ser endereços físicos ou lógicos, chamadas em *software*, etc.). Coerentemente com o caráter caixa-preta da metodologia, são a única interface do teste: o sistema é estimulado e observado apenas por meio deles, sem acesso ao seu interior. Define-se aqui também o vocabulário de eventos de entrada e de ações de saída que rotula as transições das FSMs.

3. **Construir a FSM de comportamento normal** — modela os cenários de uso rotineiro, correspondentes ao funcionamento esperado do sistema.

4. **Construir as FSMs de exceções especificadas** — modelam as condições excepcionais previstas na especificação do sistema, como sinais de falha ou respostas a entradas inválidas.

5. **Construir as FSMs de *sneak paths*** — modelam o comportamento do sistema perante eventos que ocorrem em momentos inoportunos, não previstos pelo fluxo normal (entradas "fora de hora"); esses eventos são identificados sistematicamente pela completação da tabela de transição de estados.

6. **Construir as FSMs de tolerância a falhas** — modelam o comportamento esperado sob falhas de *hardware* (sensores, atuadores, processamento), tipicamente exercitado por um injetor de falhas dedicado. Esta classe não entrou no escopo deste trabalho.

7. **Verificar e completar os requisitos** — conferir que os requisitos do sistema estão representados nas FSMs construídas e complementá-las quando necessário.

8. **Gerar os casos de teste** — percorrer as FSMs para produzir as sequências de eventos, com as saídas esperadas correspondentes a cada transição.

O último passo apoia-se na estrutura de Mealy das FSMs: como cada transição associa um evento de entrada a uma saída esperada, percorrer a máquina de estados produz uma sequência de eventos com as saídas que devem ser conferidas a cada passo — isto é, um caso de teste, em que a saída esperada de cada transição funciona como oráculo (Figura 3).

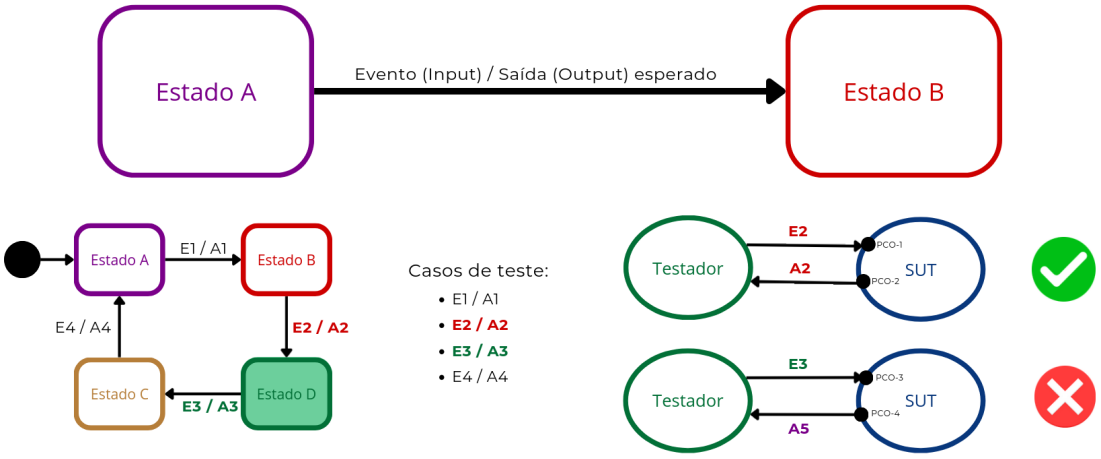

Figura 3: Como uma transição do modelo CoFI atua como oráculo de teste e como o percurso da FSM produz uma sequência de casos de teste.

2.3 Análise de Mutação

A análise de mutação é uma técnica para avaliar a eficácia de uma suíte de testes [9, 10]. A pergunta que a motiva é direta: quando todos os testes passam, isso significa que o sistema está de fato correto, ou apenas que os testes não são capazes de detectar seus defeitos? Para respondê-la, introduzem-se alterações pontuais e deliberadas (mutantes) no comportamento do sistema — por exemplo, trocar ou omitir a ação de saída de uma transição — e verifica-se se a suíte é capaz de detectá-las. A lógica tem duas pontas: sem nenhum mutante, executada contra o sistema fiel, a suíte deve passar integralmente (um controle negativo, que garante que ela não rejeita o comportamento correto); com um mutante injetado, a suíte deve detectá-lo — diz-se então que o mutante foi morto. Um mutante que não é detectado é um sobrevivente, indício de uma lacuna na suíte.

O *mutation score* — a proporção de mutantes mortos sobre o total de mutantes gerados — é uma medida direta da capacidade de detecção da suíte, distinta e complementar a métricas de cobertura estrutural (como a cobertura de arestas de um modelo), que medem apenas o que foi exercitado, e não o que foi efetivamente detectável.

3 Descrição do Sistema Exemplo

O sistema exemplo é um robô autônomo de entrega de bandejas em um restaurante, cuja função é transportar bandejas com pedidos entre a cozinha e as mesas dos clientes, navegando de forma autônoma em um ambiente compartilhado com pessoas e obstáculos (Figura 4). Esta seção descreve o sistema apenas do ponto de vista funcional; a análise de segurança e a modelagem comportamental são tratadas nas Seções 5.

O robô opera em três zonas — a cozinha (onde a bandeja é posicionada e para onde o robô sempre retorna), a área de mesas (onde ocorre a entrega) e a área de circulação (corredores de deslocamento) —, num ambiente que pode conter obstáculos estáticos (paredes, mesas,

Figura 4: Imagem ilustrativa do sistema exemplo, gerada por inteligência artificial apenas para visualização da proposta.

cadeiras) e dinâmicos (clientes e funcionários em movimento). Ele possui conhecimento prévio do mapa, localização contínua e uma lista configurada de mesas com posições fixas.

O ciclo de operação padrão é o seguinte: (i) o robô permanece em *standby* na cozinha; (ii) o atendente posiciona a bandeja, seleciona a mesa de destino e aciona o início da missão; (iii) confirmada a presença da carga, o robô navega até a mesa, monitorando obstáculos e parando ou retomando o movimento conforme necessário; (iv) ao chegar, aguarda a retirada da bandeja pelo cliente; (v) detectada a retirada, retorna automaticamente à cozinha, novamente monitorando obstáculos; e (vi) ao chegar, volta ao estado de *standby*. Além do fluxo normal, o sistema trata condições excepcionais — recusa do pedido pelo cliente, bateria baixa e falha de navegação —, que levam o robô a um estado de intervenção humana.

Três atores interagem com o robô: o atendente, responsável por posicionar a bandeja, selecionar a mesa e iniciar a missão; o cliente, responsável por retirar a bandeja da mesa (podendo recusar o pedido pela interface); e o controlador do robô, responsável por decidir as ações de movimento com base nos sensores (detector de obstáculos, odometria, sensor de carga e sensor de bateria).

Entre as capacidades funcionais do robô destacam-se a navegação autônoma com desvio de obstáculos e a gestão de bateria. Na navegação, diante de um obstáculo no caminho o robô tenta replanejar a rota para contorná-lo; se algum replanejamento permite prosseguir, retoma o movimento, e, se nenhum é bem-sucedido dentro de um tempo limite (*timeout*), declara falha e passa a aguardar intervenção humana. Na gestão de bateria, abaixo de um limite inferior o robô conclui a missão em andamento, retorna à cozinha e recusa novas missões, só voltando à operação normal acima de um limite superior. O sistema transporta uma bandeja por missão e não realiza coleta de pratos sujos, manipulação ativa nem

identificação individual de clientes.

No contexto deste projeto não foi desenvolvida nem uma implementação física do robô nem aplicação em simulador realista: o sistema sob teste (SUT) utilizado na Seção 5 é apenas um sistema computacional que emula a interface de entradas e saídas do sistema descrito (os eventos que recebe e as ações que produz), construído especificamente para este trabalho a fim de permitir a execução automatizada dos casos de teste derivados.

4 Metodologia

Este trabalho adota o método combinado de STPA e CoFI proposto por Hirata e Ambrosio [4], aplicando-o a um novo cenário concreto — o robô autônomo de entrega, em vez do sistema de bomba de insulina do artigo original — e estendendo-o com os passos finais de concretização que aquele trabalho deixa em aberto: a execução automatizada dos testes gerados contra uma implementação e a validação quantitativa de sua eficácia de detecção. O *pipeline* resultante (Figura 1) segue quatro fases — análise (STPA), modelagem (CoFI), geração e execução de testes (AltWalker) e validação (análise de mutação) —, detalhadas a seguir e aplicadas ao sistema exemplo na Seção 5.

4.1 Da análise à modelagem: combinando STPA e CoFI

A articulação entre as duas técnicas é o ponto central do método. Hirata e Ambrosio [4] argumentam que a combinação funciona porque a estrutura de controle funcional produzida pela STPA já contém a informação necessária para construir as FSMs da CoFI. Essa conexão tem duas pontas, ilustradas na Figura 5:

- os eventos que ocorrem na estrutura de controle — ações de controle e *feedbacks* — tornam-se os PCOs, os eventos e as ações das FSMs da CoFI;
- as restrições de segurança derivadas das UCAs orientam a verificação e a complementação das FSMs.

Essa verificação é um ponto crucial do método e merece detalhe. Cada UCA descreve uma ação de controle que, em um dado contexto (o estado do *process model*) e segundo um tipo (fornecida, não fornecida, cedo/tarde demais ou com duração errada), leva a um perigo; a restrição de segurança correspondente proíbe exatamente esse comportamento. Como a FSM é o comportamento de referência — o oráculo, o comportamento ideal a ser verificado —, garantir uma restrição significa conferir que nenhum caminho do modelo permite que a UCA aconteça: que a ação insegura nunca ocorre sem que a sequência de eventos que a tornaria segura a tenha precedido ou, no caso de uma ação omitida, que a ação exigida sempre ocorre no contexto que a requer. Onde algum caminho ainda a permitisse, a FSM é complementada para eliminá-lo. Esse mapeamento fica mais concreto no exemplo (Seção 5).

Como o controlador do robô é o único controlador automatizado da estrutura, é a FSM desse controlador que é construída e testada — o sistema sob teste. Os demais atores (atendente e cliente) entram como fontes de eventos, inclusive os que decorrem de seu comportamento inseguro e que o controlador deve tratar com segurança.

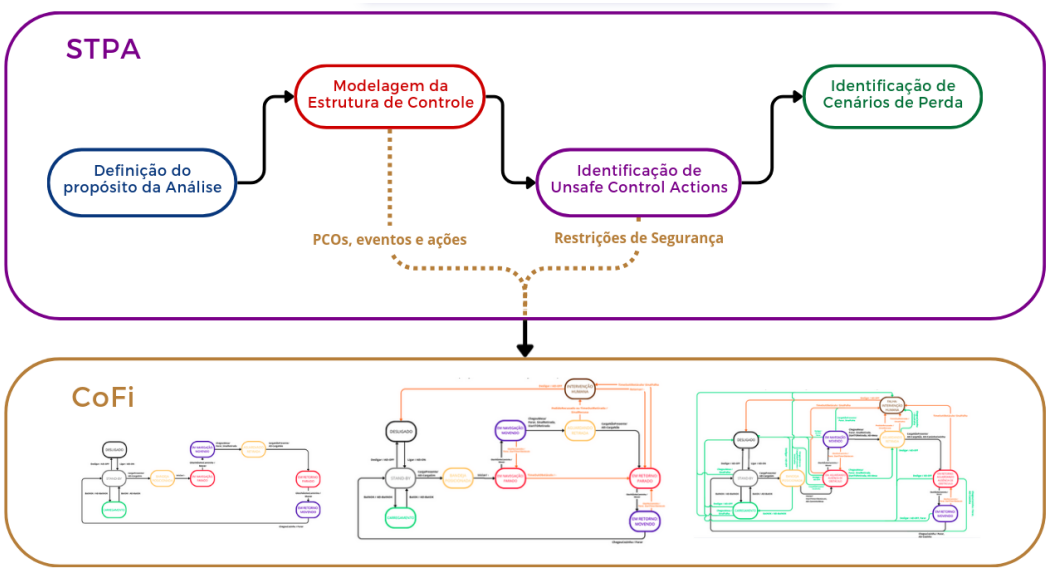

Figura 5: Articulação entre STPA e CoFI. A estrutura de controle e as restrições de segurança da STPA alimentam, respectivamente, o vocabulário e a verificação das FSMs da CoFI.

4.2 Geração e execução de testes

A execução dos testes envolve três partes. O AltWalker [11], *framework* de teste baseado em modelos, percorre uma das FSMs da CoFI e gera um caminho — uma sequência de arestas e vértices a percorrer —, conduzindo a execução passo a passo. O código testador reúne a lógica de teste: a cada aresta percorrida, (i) aplica ao simulador o evento associado à aresta, (ii) lê as ações de saída que este produz e (iii) verifica-as contra as que a FSM prescreve para aquela transição — o oráculo —, acusando falha se houver divergência. O simulador (o SUT) é uma implementação independente do sistema sob teste: a cada evento recebido, atualiza seu estado interno e produz as ações de saída correspondentes. A interação entre testador e simulador ocorre exclusivamente pelos PCOs — as interfaces contratuais que definem por onde os eventos são injetados e por onde as ações são observadas.

O critério de geração adotado é a cobertura de 100% das arestas do modelo: exige-se que cada transição seja exercitada ao menos uma vez, de modo que nenhuma transição especificada deixe de ser testada. Esse critério garante exercitar cada transição isoladamente, mas não toda combinação mais longa de transições — distinção que reaparece na análise de mutação (Seção 4.3).

Duas precauções evitam que a conformidade observada seja trivial. Primeiro, a verificação é estritamente caixa-preta: o oráculo compara apenas as ações de saída observáveis do simulador, nunca o seu estado interno. Segundo, o simulador é codificado de forma independente, a partir da estrutura de controle da STPA e sem reaproveitar a tabela de saídas usada como oráculo, para que a conformidade não seja o próprio artefato conferindo a si mesmo. Faltas de transferência (transição para um estado incorreto) são detectadas indire-

tamente, por propagação, quando uma transição posterior produz uma saída divergente.

4.3 Validação da suíte por análise de mutação

Como exploração complementar, avalia-se se os testes gerados são de fato capazes de detectar defeitos, e não apenas se foram gerados e executados. A partir da especificação de saídas esperadas, gera-se um catálogo de mutantes de primeira ordem — cada um com um único desvio comportamental —, o que mantém o número de mutantes linear no número de transições (evitando a explosão combinatória dos mutantes de ordem superior) e assegura que uma eventual detecção seja atribuível a um defeito isolado. Três operadores são empregados, cada um modelando uma classe de defeito de controle: OMIT, que remove uma ação de saída esperada de uma transição (falha em executar uma resposta prevista); INSERT, que acrescenta uma ação de saída espúria (resposta indevida); e REDIRECT, que altera o estado de destino de uma transição para um estado incorreto (corrupção do fluxo de controle) — e o único operador que pode, em princípio, produzir um mutante equivalente, quando o destino errado é observacionalmente indistinguível do correto. Cada mutante é injetado isoladamente no simulador, contra o qual a suíte de testes é executada; conforme o resultado dessa execução, o mutante é classificado como morto (detectado), sobrevivente (não detectado, apesar de a transição correspondente ter sido exercitada) ou não-coberto (a transição não foi exercitada pelo caminho gerado).

Um mutante sobrevivente, contudo, é ambíguo: sobreviver significa apenas que ele não foi detectado naquela execução, e isso pode ter duas causas distintas. Ele pode ser um equivalente real — indistinguível do sistema original por qualquer sequência de entrada e, portanto, impossível de matar por qualquer suíte — ou um falso equivalente — matável em princípio, que sobreviveu apenas porque o caminho gerado não exercitou a sequência capaz de revelá-lo. Uma verificação de equivalência de estados da FSM separa os dois casos, e a distinção é essencial para interpretar o *mutation score*: um equivalente real não representa um problema para o sistema modelado — por produzir exatamente as mesmas saídas do original, a alteração é imperceptível e não constitui defeito observável, de modo que ele deve ser descontado do cálculo; já um falso equivalente aponta uma insuficiência da própria suíte — do critério de geração dos caminhos —, e não do oráculo nem da metodologia.

Como controle de validade, a mesma suíte é executada, sem qualquer mutante, contra o simulador fiel: esse cenário-base de conformidade deve ser aprovado integralmente, funcionando como controle negativo que torna o *mutation score* posterior significativo.

5 Aplicação ao Sistema Exemplo

Esta seção aplica o *pipeline* da Seção 4 ao robô de entrega, passo a passo: da análise STPA (Seção 5.1) à modelagem CoFI (Seção 5.2) e à execução e campanha de mutação (Seção 5.3).

Embora a apresentação a seguir seja sequencial, o desenvolvimento não foi puramente linear: os artefatos não surgiram prontos na primeira passada, mas por revisões e reanálises iterativas ao longo da construção do *pipeline*, em que etapas posteriores frequentemente evidenciaram a necessidade de ajustar etapas anteriores. O que se reporta aqui é o resultado consolidado desse processo.

5.1 Análise STPA

Passo 1 — Propósito da análise. A análise identificou, para o robô de entrega, três metas, três perdas e três perigos de nível de sistema.

As metas do sistema são:

- G-1: Realizar o serviço de entrega de bandejas;
- G-2: Navegar de forma autônoma, segura e previsível;
- G-3: Evitar danos a pessoas e propriedades.

As perdas inaceitáveis são:

- L-1: Lesão a pessoas por colisão com o robô;
- L-2: Lesão a pessoas por queda de carga;
- L-3: Dano ao robô ou a elementos do ambiente.

Os perigos de nível de sistema, com as perdas que cada um pode causar, são:

- H-1: O robô se movimenta sem manter distância segura de pessoas ou obstáculos (L-1, L-2, L-3);
- H-2: O robô transporta carga instável (L-2, L-3);
- H-3: Ocorre interação externa com o robô em momento não oportuno (L-1, L-2).

Passo 2 — Estrutura de controle. A estrutura de controle modelada (Figura 6) compreende três controladores — cliente, atendente e controlador do robô —, o atuador de motores, quatro sensores (LIDAR, odometria, sensor de carga e sensor de bateria), o *input* externo Cozinha e o processo controlado (os itens em transporte e o ambiente externo). O *process model* do controlador do robô contém cinco variáveis: estado de navegação (Movendo/Parado), bateria (OK/NOK), obstáculo próximo (Sim/Não), carga presente (Sim/Não) e posição (Mesa/Cozinha/CaminhoMesa/CaminhoCozinha).

Passo 3 — Ações de controle inseguras. A identificação das UCAs seguiu a varredura sistemática e exaustiva da STPA, conduzida com o apoio da ferramenta AppSTPA [5]: para cada controlador, cada ação de controle e cada contexto do *process model*, avaliou-se se a ação é insegura segundo as quatro formas canônicas. Cada célula da matriz em que a ação leva a um perigo constitui uma UCA, rastreada a um ou mais dos perigos H-1 a H-3. A varredura produziu sete UCAs e as respectivas restrições de segurança derivadas (SC-1 a SC-7), listadas na Tabela 1.

Todas as sete restrições fundamentam recomendações de segurança no Passo 4, mas alcançam o controlador — o componente sob teste — de formas distintas. SC-5, SC-6 e SC-7 restringem diretamente o controlador e ficam embutidas na estrutura da FSM (Seção 5.2). As demais (SC-1 e SC-2, sobre o atendente; SC-3 e SC-4, sobre o cliente) traduzem-se em diretrizes de comportamento humano, mas sua violação reflete-se no controlador, que deve reagir a ela com segurança.

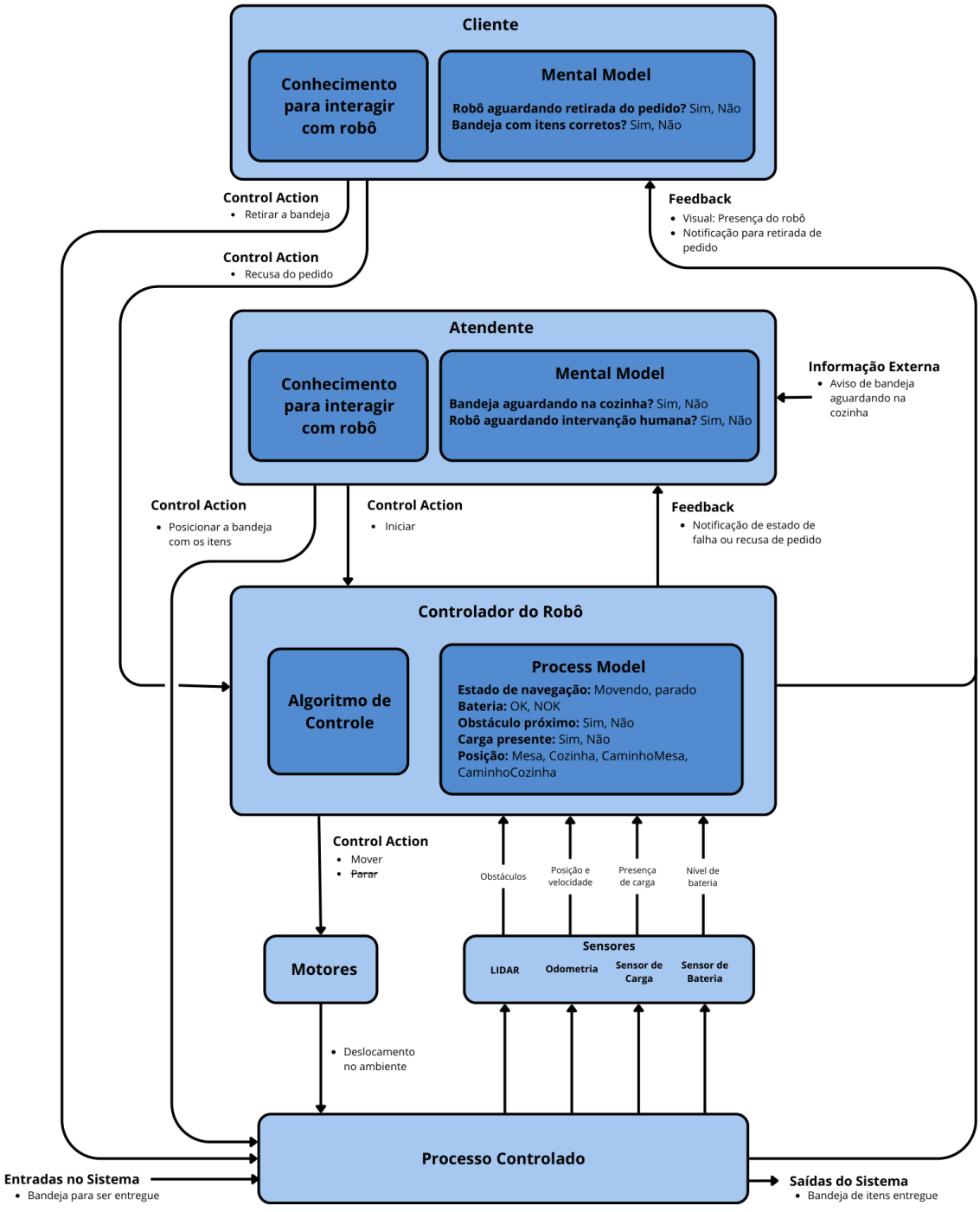

Figura 6: Estrutura de controle do robô de entrega, modelada no Passo 2 da STPA.

Tabela 1: Ações de controle inseguras (UCAs) e restrições de segurança derivadas (SC), com os perigos rastreados.

	Ação de controle insegura (UCA)	**Restrição de segurança derivada (SC)**	**Perigo**
1	O atendente posiciona a bandeja tarde demais.	O atendente não deve posicionar a bandeja tarde demais.	H-2, H-3
2	O atendente interrompe cedo demais o posicionamento da bandeja (resultando em mau encaixe).	O atendente não deve interromper cedo demais o posicionamento da bandeja.	H-2, H-3
3	O cliente retira a bandeja quando não há notificação para retirada do pedido.	O cliente não deve retirar a bandeja quando não há notificação para retirada do pedido.	H-2, H-3
4	O cliente demora demais para retirar a bandeja quando há notificação para retirada (possivelmente robô reinicia movimento).	O cliente não deve demorar demais para retirar a bandeja quando há notificação para retirada.	H-2, H-3
5	O controlador do robô fornece o comando Mover quando há obstáculo próximo.	O controlador do robô não deve fornecer o comando Mover quando há obstáculo próximo.	H-1, H-2
6	O controlador do robô fornece o comando Parar tarde demais quando o estado de navegação é "Movendo" e há obstáculo próximo.	O controlador do robô não deve fornecer o comando Parar tarde demais quando o estado de navegação é "Movendo" e há obstáculo próximo.	H-1, H-2
7	O controlador do robô não fornece o comando Parar quando o estado de navegação é "Movendo" e há obstáculo próximo.	O controlador do robô deve fornecer o comando Parar quando o estado de navegação é "Movendo" e há obstáculo próximo.	H-1, H-2

Passo 4 — Cenários de perda e recomendações. Para cada UCA, a análise identificou os cenários de perda — os fatores causais que explicam por que a UCA ocorreria e por que uma ação, uma vez emitida, seria mal executada. Foram gerados vinte e um cenários, cujas causas se distribuem por todo o sistema, e não apenas no *software* de controle: comportamento inseguro do controlador (algoritmo incorreto, *process model* desatualizado); *feedback* inadequado, tanto do sensor LIDAR (leitura atrasada, perdida, imprecisa ou ausente) quanto das notificações ao cliente e ao atendente; e falhas no caminho de controle (o cliente emitir uma ação inadequada; os motores atrasarem a execução ou executarem uma ação não emitida). Cada cenário fundamenta uma recomendação de projeto — por exemplo, revisar e testar o algoritmo de controle a cada atualização, ou prover redundância e manutenção periódica ao LIDAR e aos motores. Essas recomendações estão fora do escopo de geração de testes deste trabalho (não alimentam a modelagem CoFI), mas evidenciam a riqueza analítica do método.

5.2 Modelagem comportamental CoFI

PCOs, eventos e ações. Seguindo os dois primeiros passos da CoFI — anteriores à construção das FSMs —, foram identificados, a partir da estrutura de controle, os doze PCOs do sistema e o vocabulário de eventos de entrada e ações de saída, resumidos na Tabela 2. Os PCOs cobrem os controladores humanos (cliente e atendente), o atuador (motores), os quatro sensores, os temporizadores internos e a atualização de estados internos. Os eventos

de entrada são os estímulos que o AltWalker aplica ao sistema durante a execução; as ações de saída são o que se observa e constituem a base do oráculo. A coluna *Origem na STPA* explicita essa derivação: cada evento ou ação externo provém diretamente de uma ação de controle ou de um *feedback* identificado na estrutura de controle, ao passo que os PCOs internos (temporizadores e atualização de estados) são acrescentados pela modelagem, sem correspondente na STPA.

Tabela 2: PCOs do controlador do robô, com os eventos de entrada (estímulos) e as ações de saída (base do oráculo), derivados da estrutura de controle da STPA.

PCO	**Interface**	**Direção**	**Origem na STPA**	**Eventos / ações**
PCO1	Cliente	entrada	ação de controle Recusar pedido	`PedidoRecusado`
PCO2	Atendente	entrada	ação de controle Iniciar	`Iniciar`, `Ligar`, `Desligar`
PCO3	Cliente	saída	*feedback* Notificação para retirada	`SinalRetirada`
PCO4	Atendente	saída	*feedback* Notificação de falha/recusa	`SinalRecusa`, `SinalFalha`
PCO5	Motores	saída	ações de controle Mover e Parar	`Mover`, `Parar`
PCO6	Detector de obstáculos	entrada	*feedback* Obstáculo próximo	`ObstNoCaminho`, `ObstNãoNoCaminho`
PCO7	Odometria	entrada	*feedback* Posição e velocidade	`ChegouMesa`, `ChegouCozinha`
PCO8	Sensor de carga	entrada	*feedback* Carga presente	`CargaPresente`, `CargaNãoPresente`
PCO9	Sensor de bateria	entrada	*feedback* Nível de bateria	`BatOK`, `BatNOK`
PCO10	*Start timer*	saída	—	`StartTORetirada`, `StartTOObstáculo`
PCO11	*Timeout*	entrada	—	`TimeOutRetirada`, `TimeOutObstáculo`
PCO12	Atualização de estados	saída	—	`AEI-*`: uma para cada valor possível das variáveis do *process model* — Status, Bateria, Obstáculo, Carga e Posição (doze ao todo)

As máquinas de estado. A partir desse vocabulário, foram construídos três modelos CoFI, cada um uma máquina de Mealy correspondente a uma classe de comportamento (Figuras 7, 8 e 9): o normal (9 estados, 11 transições), que cobre o cenário sem pedidos recusados e sem obstrução de caminhos; o de exceções especificadas (10 estados, 18 transições), que acrescenta os pedidos recusados e a obstrução de caminhos; e o de *sneak paths* (10 estados, 140 transições), que acrescenta ainda os eventos que ocorrem em momentos não previstos inicialmente. Como é completo, o modelo de *sneak paths* contém os demais como subconjuntos e é o utilizado na campanha de mutação (Seção 5.3), por representar exaustivamente o espaço de comportamento observável.

A construção das FSMs foi guiada pela etapa de verificação apresentada na Seção 4.1,

que é o elo mais delicado da junção entre STPA e CoFI. Como a FSM é o comportamento de referência — o oráculo contra o qual o controlador é testado —, garantir uma restrição de segurança é assegurar que *nenhum* caminho do modelo permita a UCA correspondente. Além de preencher toda a funcionalidade do sistema (ciclo normal, exceções especificadas e entradas fora de hora), verificou-se essa condição para cada uma das três UCAs do controlador (Mover e Parar diante de obstáculo, Tabela 1):

- **UCA-5** (mover com obstáculo próximo) — a ação `Mover` só é saída esperada após um evento `ObstNãoNoCaminho`; assim, não existe caminho em que o robô se mova sem que a ausência de obstáculo o tenha precedido, o que embute a restrição SC-5;
- **UCA-7** (não parar com obstáculo) — a ação `Parar` é sempre a saída esperada de um evento `ObstNoCaminho` durante o movimento; a ação exigida ocorre em todo contexto que a requer, sem caminho que a omita, embutindo a SC-7;
- **UCA-6** (parar tarde demais) — o `Parar` é emitido na própria transição do evento `ObstNoCaminho`, sem estado intermediário que o adie, embutindo a SC-6.

Dessa forma, as três restrições do controlador ficam embutidas na estrutura do modelo, e cada uma fica mapeada às transições que a exercitam — estabelecendo a rastreabilidade direta entre a análise STPA e os casos de teste gerados.

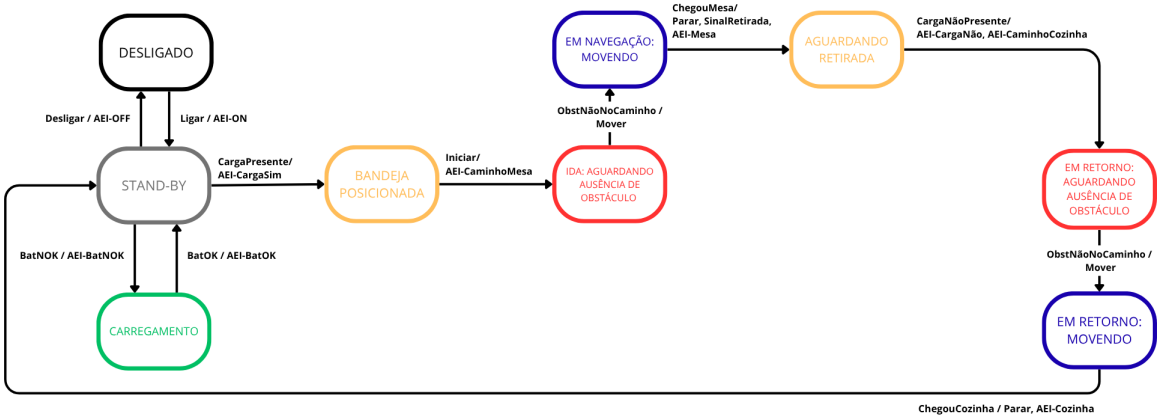

Figura 7: Modelo CoFI de comportamento normal do controlador.

Para simplificação da análise, a classe de tolerância a falhas, prevista pela CoFI para modelar a resiliência a falhas de *hardware*, não foi construída neste trabalho; a classe permanece como extensão natural para trabalhos futuros (Seção 7).

5.3 Execução e campanha de mutação

Os quatro modelos foram validados estruturalmente — sintaxe e percorribilidade, e correspondência entre elementos do modelo e métodos do código de teste — com sucesso em todos os casos.

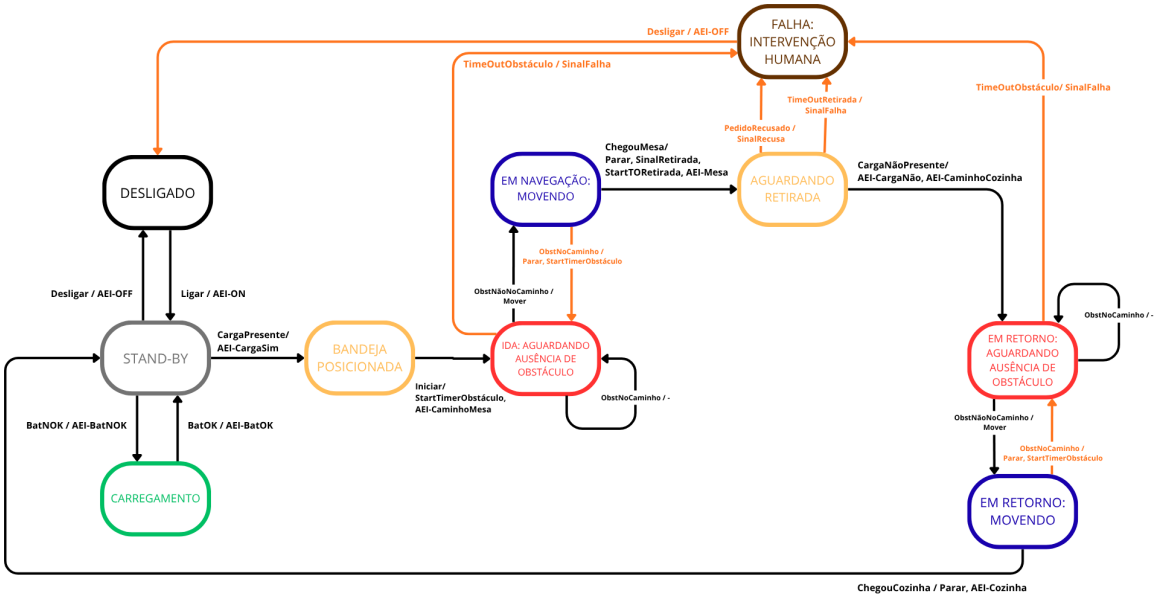

Figura 8: Modelo CoFI de exceções especificadas.

Aplicando ao robô a arquitetura de execução e o processo de validação descritos nas Seções 4.2 e 4.3, a campanha de mutação incide sobre o modelo completo de *sneak paths* (10 estados, 140 transições). Os operadores são aplicados, em primeiro lugar, às 35 transições *ativas* — as que produzem alguma ação de saída. As demais 105 são auto-laços de saída vazia: eventos que, em cada estado, a especificação manda ignorar. Mutá-los não é inócuo — fazer o robô agir, ou mudar de estado, onde deveria permanecer inerte é um defeito real —; cobrir todos os 105, porém, inflaria o catálogo com muitos casos afastados de qualquer restrição de segurança. Aplicou-se, então, uma redução: das transições de saída vazia, incluíram-se apenas 4, de maior potencial de segurança, em que reagir seria inseguro. Sobre esses dois grupos, os três operadores (Seção 4.3) respondem cada um por uma parcela do catálogo:

- **OMIT** (48) — um mutante por ação de saída removida (as 35 transições ativas somam 48 ações); por exemplo, deixar de emitir `Parar` diante de um obstáculo.
- **INSERT** (69) — inserção de `Mover` ou `Parar` onde ainda não são saída, por exemplo mover-se em um estado no qual o robô deveria permanecer parado: nas 35 transições ativas, 2 candidatas cada menos as 9 em que uma delas já é saída ($35 \times 2 - 9 = 61$), mais as 4 transições etiquetadas de saída vazia (2 cada, 8), totalizando 69.
- **REDIRECT** (78) — 2 alvos errados por transição, amostrados com semente fixa para reprodutibilidade: 35 transições ativas (70) mais 4 etiquetadas de saída vazia (8).

Os três tipos somam 195 mutantes de primeira ordem. Cada mutante herda as etiquetas

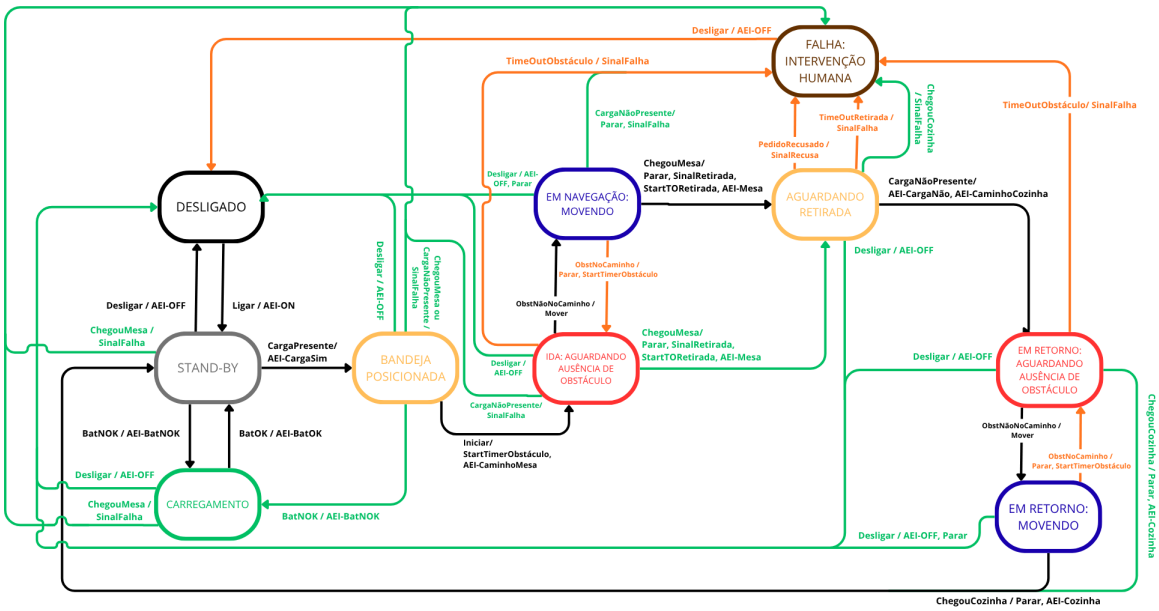

Figura 9: Modelo CoFI com *sneak paths* (*auto-loops* omitidos)

de restrição de segurança da transição que altera, o que permite agregar a detecção por SC. A campanha injeta um mutante por vez, sempre sob o critério de 100% de cobertura de arestas — de modo que a transição mutada é sempre percorrida, e a classe não-coberto não deve ocorrer; como o gerador de caminhos do AltWalker não é semeado, cada execução exercita uma sequência possivelmente distinta. O cenário-base sem mutante e os resultados da campanha são apresentados na Seção 6.

6 Resultados e Avaliação

A avaliação parte da evidência empírica da execução — a conformidade do simulador e a análise de mutação —, que dá continuidade direta à campanha montada ao fim da Seção 5. Sobre essa base, discute-se então o resultado principal do trabalho, de natureza qualitativa: o que a combinação STPA+CoFI permitiu enxergar.

6.1 Conformidade: o controle negativo

A execução do simulador fiel contra o modelo completo de *sneak paths* (que subsume os demais), sem qualquer mutante, resultou em aprovação integral, com 100% de cobertura de vértices e arestas. Esse resultado estabelece que o oráculo aceita corretamente o comportamento válido — condição sem a qual a exploração de mutação não teria significado, pois um oráculo que rejeitasse tudo indiscriminadamente atingiria detecção total de forma espúria.

6.2 Eficácia preliminar da suíte (análise de mutação)

Os resultados a seguir constituem uma exploração preliminar, ainda em consolidação: o gerador de caminhos não é determinístico nem *seedado*, e a campanha não foi formalmente encerrada (Seção 7). Reportam-se por já indicarem a viabilidade da abordagem.

Da campanha de 195 mutantes de primeira ordem (Tabela 3), 179 foram mortos e 16 sobreviveram, sem mutantes não-cobertos, um *mutation score* observado de 92%. Uma verificação de causalidade confirmou que as mortes por omissão e inserção de ação ocorreram exatamente na transição mutada (nenhuma detecção fora do alvo), e que as mortes por redirecionamento de estado ocorreram por propagação a uma transição posterior — evidência de que o oráculo detecta não só saídas erradas imediatas, mas também erros de lógica de estado. Além disso, como cada mutante herda as etiquetas de restrição de segurança da transição que altera, cada um dos 179 mutantes mortos aponta a restrição de segurança associada à transição em que foi detectado — de modo que a detecção é reportável por SC, e não apenas como uma contagem global.

Tabela 3: Resultados da campanha de mutação (exploração preliminar).

Métrica	**Valor**
Mutantes (1ª ordem)	195
Mortos	179
Sobreviventes	16
Não-cobertos / Erros	0 / 0
Mutation score (observado)	92%
Cenário-base sem mutante	100% (0 falsos positivos)

Os 16 sobreviventes, porém, ainda precisam ser interpretados. Uma verificação de equivalência de estados mostrou que os dez estados do modelo formam dez classes distintas — nenhum par é observacionalmente equivalente —, de modo que nenhum deles é um equivalente real: todos são falsos equivalentes, matáveis por uma entrada distinguidora que o caminho gerado não exercitou. Disso decorre que os 92% observados medem a adequação da suíte efetivamente gerada, sob o critério de cobertura de arestas, e não a capacidade de detecção do oráculo ou da abordagem — cujo *mutation score* teoricamente alcançável é 100%. Nesse sentido, a suíte gerada foi insuficiente para matar aqueles 16, uma lacuna atribuível ao critério de geração dos caminhos e corrigível com um critério mais forte (Seção 7), não a uma limitação do método.

Vistas em conjunto, as duas pontas da validação se completam: sem mutante, o controle negativo (Seção 6.1) mostrou que a suíte passa integralmente contra o sistema fiel; com mutante, ela detecta a grande maioria dos desvios.

O principal resultado deste trabalho, porém, não é um número de execução isolado, mas o que a rastreabilidade contínua STPA → CoFI → teste permitiu enxergar e formalizar — e para o qual a execução e a mutação anteriores servem de evidência, não como um fim em si.

6.3 O que a combinação revelou

Antes mesmo de serem combinadas, a STPA e a CoFI já contribuem, cada uma, com uma força para a análise.

A força da STPA é enxergar o que um método clássico centrado em falhas de componente não veria. A partir da estrutura de controle, ela derivou sistematicamente as ações de controle inseguras do controlador e as restrições correspondentes: mover com obstáculo próximo (SC-5), parar tarde demais (SC-6) e não parar (SC-7), todas ligadas ao perigos de colisão e queda de carga (H-1 e H-2). São problemas de coordenação e de *timing*, não necessariamente de um componente que quebra. O limite da STPA é parar aí: as restrições ficam como requisitos textuais, sem meio de serem exercitadas.

A força da CoFI é complementar: dar método à construção de um modelo comportamental executável. Sem um processo sistemático, tende-se a modelar apenas o fluxo esperado — os cenários de uso "felizes" que vêm naturalmente à mente —, resultando em uma FSM enxuta. A CoFI vai além ao separar o comportamento em normal, exceções especificadas e *sneak paths*, exigindo do modelo uma resposta definida para cada evento em cada estado, inclusive as entradas "fora de hora" que o fluxo esperado ignoraria. É essa completação que faz o modelo crescer: 11 transições no fluxo esperado, 18 com as exceções especificadas e 140 no modelo completo, que cobre exaustivamente cada par (estado, evento). Como cada transição associa um evento a uma saída esperada, todo esse comportamento — e não apenas o feliz — torna-se um oráculo executável. O limite da CoFI, isolada, é não distinguir, entre tantas transições, quais carregam significado de segurança.

A combinação fecha exatamente essa lacuna, na etapa de verificação (Seção 4.1): cada restrição do controlador é convertida em uma propriedade estrutural do modelo, garantindo que nenhum caminho habilite a UCA correspondente. No robô, `Mover` só é saída esperada após um evento `ObstNãoNoCaminho` (SC-5) e `Parar` é sempre a saída de um `ObstNoCaminho` durante o movimento (SC-6 e SC-7). Com isso, a restrição de segurança deixa de ser um requisito textual e torna-se uma propriedade verificável e testável: embutida na estrutura do modelo (garantida por construção), mapeada a transições concretas e efetivamente executada contra o simulador. É esse encadeamento — o que é crítico, dado pela STPA, sobre um modelo executável, dado pela CoFI, com a garantia de que o crítico é respeitado em todo caminho e exercitado em teste — que nenhuma das técnicas entrega isoladamente.

6.4 Rastreabilidade fim-a-fim

Como cada transição do modelo carrega a etiqueta da restrição de segurança que exercita, cada resultado de teste é atribuível a uma restrição específica — e, por ela, a um perigo e a uma perda. Vale percorrer essa cadeia fase a fase do *pipeline*, tomando a SC-5 (não mover com obstáculo próximo) como fio condutor:

- **Análise (STPA)** — a perda L-1 (lesão por colisão) leva ao perigo H-1 (o robô se movimenta sem distância segura), do qual se deriva a ação de controle insegura UCA-5 (fornecer `Mover` com obstáculo próximo) e, dela, a restrição SC-5.
- **Modelagem (CoFI)** — SC-5 é embutida na estrutura da FSM: `Mover` só é saída es-

perada após um evento `ObstNãoNoCaminho`, e a etiqueta SC-5 acompanha as transições que a exercitam.

- **Geração e execução (AltWalker)** — essas transições são percorridas e a saída do simulador é conferida contra a FSM; uma divergência ali é reportada como violação da SC-5.
- **Validação (mutação)** — um mutante que insira `Mover` indevidamente nessas transições é morto, e a morte herda a etiqueta SC-5, ligando o defeito detectado de volta à restrição.

O elo é navegável nos dois sentidos: da perda até o caso de teste que a protege e, de volta, de uma falha de teste até a perda que ela evita. Numa abordagem MBT clássica, essa mesma falha seria apenas "saída divergente na transição X", sem caminho de volta ao seu significado de segurança. É essa cadeia — perda → perigo → UCA → SC → transição → caso de teste → mutante —, e não um número de cobertura, o principal produto de avaliação do trabalho.

6.5 Discussão frente ao trabalho de referência

Hirata e Ambrosio [4], cujo método este trabalho adota e estende, aplicaram-no a um sistema de bomba de insulina e geram os casos de teste de forma abstrata, com a ferramenta ConData, encerrando a avaliação nesse ponto — sem executá-los contra uma implementação nem medir sua capacidade de detecção. Este trabalho difere em três aspectos: aplica o método a um novo cenário (o robô de entrega); usa o AltWalker para gerar e *executar* os testes contra um simulador independente; e inicia uma validação quantitativa de sua eficácia por análise de mutação. O que essas diferenças acrescentam é tornar a rastreabilidade STPA+CoFI operacional, e não apenas documental: no trabalho de referência, a ligação entre restrição de segurança e caso de teste existe no papel, mas não chega a ser exercida; aqui, cada restrição do controlador é executada contra uma implementação e sua capacidade de detecção começa a ser medida. As restrições do controlador são, isoladamente, simples (não mover com obstáculo, parar a tempo); o ganho não está, portanto, na sofisticação de cada uma, mas em fechar — de ponta a ponta e de forma verificável — o percurso da análise de segurança ao teste executado.

7 Conclusão e Trabalhos Futuros

Este trabalho investigou a aplicação integrada de STPA e CoFI para a derivação de casos de teste voltados ao controlador de um robô autônomo de entrega, um sistema de robótica móvel colaborativa. Partindo do método combinado de Hirata e Ambrosio [4], aplicou-o a um novo cenário concreto e o estendeu com os passos finais que aquele trabalho deixa em aberto: a geração e a execução automatizada dos testes, por meio do AltWalker, contra um simulador independente, e uma validação preliminar de sua eficácia por análise de mutação. O resultado é um *pipeline* completo e executável — STPA → CoFI → AltWalker → mutação

— rastreável de ponta a ponta, em que cada perda e perigo se ligam a uma restrição de segurança, a uma transição do modelo e a um caso de teste.

A principal contribuição não é métrica, mas de integração e rastreabilidade: a modelagem sistemática da CoFI cobre comportamento que um modelo do fluxo esperado ignoraria — as exceções especificadas e as entradas fora de hora —, e cada restrição de segurança do controlador (não mover com obstáculo, parar a tempo) torna-se uma propriedade estrutural e executável do modelo, com cada resultado de teste atribuível a uma restrição específica.

Como trabalhos futuros, destacam-se:

- **Classe de tolerância a falhas da CoFI** — construir a quarta classe de comportamento, modelando a resiliência do controlador a falhas de *hardware* (sensores, atuadores), complementando as três classes já cobertas.
- **Validação da aplicabilidade em sistema real** — aplicar o *pipeline* a uma implementação real do controlador (ou a um simulador de maior fidelidade), avaliando o esforço de adaptação do oráculo e do adaptador.
- **Consolidação da validação por mutação** — encerrar formalmente a campanha e confirmar empiricamente o *mutation score* teoricamente alcançável de 100%.
- **Automação do encadeamento** — reduzir os passos manuais na passagem entre as fases (relatório STPA → construção das FSMs → modelo do AltWalker), aproximando o *pipeline* de uma ferramenta única e diminuindo o risco de erro na tradução entre etapas; inspira essa direção o próprio AppSTPA [5], que já automatiza a etapa de análise STPA.

Referências

[1] N. G. Leveson. *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, Cambridge, MA, 2011.

[2] N. G. Leveson and J. P. Thomas. *STPA Handbook*. MIT, 2018.

[3] A. M. Ambrosio. *CoFI: uma abordagem combinando teste de conformidade e injeção de falhas para validação de software em aplicações espaciais*. Tese de doutorado, Instituto Nacional de Pesquisas Espaciais (INPE), São José dos Campos, 2005.

[4] C. M. Hirata and A. M. Ambrosio. Combining STPA with CoFI to generate requirements and test cases for safety-critical systems. *IEEE Systems Journal*, 16(4):6605–6616, 2022.

[5] A. Carniel, J. de Melo Bezerra, and C. M. Hirata. Workflow for conflict and reinforcement identification based on STPA and STRIDE. *IEEE Access*, 13:5785–5814, 2025. DOI: 10.1109/ACCESS.2024.3525002. Ferramenta AppSTPA disponível em `https://www.appstpa.com.br` e `https://github.com/andrei-carniel/Appstpa_latest_version`.

[6] A. Adriaensen, L. Pintelon, F. Costantino, G. Di Gravio, and R. Patriarca. An STPA safety analysis case study of a collaborative robot application. *IFAC-PapersOnLine*, 54(1):534–539, 2021.

[7] J. Vermaelen and T. Holvoet. Advancing safe robot behavior by formalizing STPA with Tumato. In *2024 8th International Conference on System Reliability and Safety (ICSRS)*, pages 371–377, Sicily, Italy, 2024. DOI: 10.1109/ICSRS63046.2024.10927587.

[8] M. Utting and B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann, 2007.

[9] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. Hints on test data selection: Help for the practicing programmer. *Computer*, 11(4):34–41, 1978.

[10] Y. Jia and M. Harman. An analysis and survey of the development of mutation testing. *IEEE Transactions on Software Engineering*, 37(5):649–678, 2011.

[11] AltWalker. *AltWalker: a test execution tool for model-based testing*, 2024. Disponível em: `https://altwalker.github.io/altwalker/`.