Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Proposta para a Otimização de Análise de Cenários Implícitos

Filipe Ponte Lima

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof. Dra. Genaína Rodrigues

Brasília
2016

Universidade de Brasília — UnB
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Bacharelado em Ciência da Computação

Coordenador: Prof. Dr. Rodrigo Bonifácio de Almeida

Banca examinadora composta por:

Prof. Dra. Genaína Rodrigues (Orientadora) — CIC/UnB
Prof. Dr. George Luiz Medeiros Teodoro — CIC/UnB
Prof. Ms. Fernando Albuquerque — CIC/UnB

CIP — Catalogação Internacional na Publicação

Lima, Filipe Ponte.

Uma Proposta para a Otimização de Análise de Cenários Implícitos / Filipe Ponte Lima. Brasília : UnB, 2016.
111 p. : il. ; 29,5 cm.

Monografia (Graduação) — Universidade de Brasília, Brasília, 2016.

1. Dependabilidade, 2. Confiabilidade, 3. Cenário Implícito, 4. LTSA, 5. MSC

CDU 004.4

Endereço: Universidade de Brasília
Campus Universitário Darcy Ribeiro — Asa Norte
CEP 70910-900
Brasília–DF — Brasil

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Uma Proposta para a Otimização de Análise de Cenários Implícitos

Filipe Ponte Lima

Monografia apresentada como requisito parcial
para conclusão do Bacharelado em Ciência da Computação

Prof. Dra. Genaína Rodrigues (Orientadora)
CIC/UnB

Prof. Dr. George Luiz Medeiros Teodoro
CIC/UnB

Prof. Ms. Fernando Albuquerque
CIC/UnB

Prof. Dr. Rodrigo Bonifácio de Almeida
Coordenador do Bacharelado em Ciência da Computação

Brasília, 03 de agosto de 2016

Dedicatória

Dedico este trabalho a você, leitor ou leitora, que dedica o seu tempo ao meu trabalho. Espero que este trabalho lhe ensine e lhe ajude a alcançar o que deseja, assim como outros trabalhos me ensinaram e foram vitais para que eu concluísse este. Espero que você encontre o que está procurando aqui e que aprenda o que precisa aprender. E se não for esse o caso, que pelo menos este trabalho lhe ajude a tomar a direção correta. Boa sorte e boa leitura.

Agradecimentos

Aos meus companheiros e companheiras de curso que estiveram ao meu lado, enfrentando os mesmos desafios que enfrentei. Aos meus professores e professoras que me ensinaram, me inspiraram e me motivaram, seja com confiança ou severidade. Aos meus amigos e amigas que me deram motivo aos meus sacrifícios. E à minha família que me deu tudo que eu sempre precisei, não só me mostrando o caminho para me tornar o que sou, mas também garantindo que eu sempre estivesse caminhando.

Resumo

Para se alcançar a confiabilidade de um sistema é necessário se levar em conta todas as fases de desenvolvimento do sistema. A modelagem por cenários tem sido considerada uma forma eficaz de se modelar e analisar o comportamento de um sistema nos estágios iniciais do ciclo de desenvolvimento. Cenários implícitos são comportamentos inesperados, gerados quando componentes de sistemas concorrentes se comunicam de uma forma não descrita na especificação do software. A ferramenta LTSA nos permite identificar cenários implícitos para este fim, porém, a detecção de cenários implícitos muitas vezes se torna onerosa e com resultados repetitivos. Este projeto foi desenvolvido para tornar possível o agrupamento dos cenários implícitos em famílias de maneira automática e identificá-los iterativamente. Isso melhora a interpretação do resultado da análise e torna o processo de análise de confiabilidade mais fácil e eficiente.

Palavras-chave: Dependabilidade, Confiabilidade, Cenário Implícito, LTSA, MSC

Abstract

To reach the reliability of a system it is necessary to take into account all the development stages of the system. Scenario-based specification has been considered an effective way of modeling and analyzing the system's behavior in the initial stages of the development cycle. Implied Scenarios are unexpected behaviors, generated when components of concurrent systems communicate in such a way that was not described in the behavior model of the software. The LTSA tool allow us to identify implied scenarios to this mean, but the detection of implied scenarios many times becomes laborious and with repetitive results. This project was developed to enable the grouping of the implied scenarios in families automatically and identify them iteratively. This improves the interpretation of the analysis result and makes the process of reliability analysis easier and more efficient.

Keywords: Dependability, Reliability, Implied Scenario, LTSA, MSC

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Garantir a qualidade de software sempre foi considerado de alta relevância no desenvolvimento de sistemas. No caso de sistemas concorrentes, o desenvolvimento e arquitetura são consideradas tarefas complexas e passíveis de erros sutis que podem levar a graves consequências [22]. Teste Sistemático é uma das técnicas mais importantes e usadas para validar a qualidade de software. Entretanto, testar costuma ser um trabalho muito oneroso e muitas vezes sem automação eficiente, o que favorece o erro e se torna muito custoso. É estimado que testes consomem de 30% a 50% do custo total de desenvolvimento de software [17]. Por isso, existe a necessidade de se desenvolver sistemas de softwares cada vez mais confiáveis sem extrapolar o orçamento e o cronograma [9].

Cenários descrevem como componentes de um sistema e usuários interagem no funcionamento do mesmo. Um cenário é uma história parcial, que se combinando com outros cenários fornece uma descrição completa do sistema. Cada cenário apresenta as trocas de mensagens entre os componentes do sistema, caracterizando sua arquitetura. [21]

A modelagem por cenários é considerada uma forma eficaz de modelar e analisar o comportamento de um sistema desde os estágios iniciais do ciclo de desenvolvimento [21]. Os cenários podem ser especificados utilizando Diagramas de Sequência da UML (Unified Modeling Language) ou diagramas MSCs (Message Sequence Charts). Uma de suas vantagens é possibilitar uma maior participação dos stackholders, que podem desenvolver descrições de maneira independente, cada um contribuindo com a sua visão do sistema. [21]

Fazendo a análise da modelagem de um sistema em cenários são identificadas muitas propriedades do sistema, uma delas é a presença de cenários implícitos [19]. Cenários implícitos são comportamentos do sistema que surgem a partir da composição paralela dos componentes, que são modelados nos cenários previamente especificados [2], [19].

Idealmente, é esperado que o modelo tenha exatamente os mesmos comportamentos descritos na modelagem de cenários. Entretanto, isso nem sempre acontece. Cenários podem se combinar gerando comportamentos que não foram especificados no documento de especificação de requisitos do software. [19]

A existência de cenários implícitos é um indicativo de comportamento inesperado do sistema e detectá-los pode ser crítico. Um cenário implícito pode significar simplesmente que uma funcionalidade do sistema não foi descrita e a especificação precisa ser complementada. Neste caso, chamamos o cenário implícito de positivo. Por outro lado, o cenário implícito pode indicar um comportamento errôneo, o que implicaria em modificar

a especificação para prevenir que ele ocorra. Este tipo de cenário é chamado de cenário implícito negativo. [19]

Os cenários implícitos podem causar um grande desconforto aos desenvolvedores e testadores de software, visto que, por ser um tipo de falha inconsistente [23], os seus efeitos podem ser de difícil detecção. Podendo resultar em softwares menos confiáveis, mesmo após terem sido submetidos a exaustivos processos de testes.

É ponto pacífico na literatura de sistemas confiáveis que uma análise de confiabilidade mais precisa, ainda na fase de projeto, pode dar suporte a decisões que incorram em menores custos relativos a alterações futuras da arquitetura do sistema. Dessa forma, é possível garantir que a confiabilidade do sistema venha a alcançar um nível adequado antes da implementação propriamente dita, o que é fundamental para o projeto de sistemas críticos. [23]

A confiabilidade desejada ao sistema é um atributo de dependabilidade, que é a habilidade de um sistema computacional executar serviços em que se possa justificadamente confiar. Dependabilidade é um conceito que abrange a confiabilidade, disponibilidade, segurança, integridade e manutenabilidade, principalmente. [3]

Em [23] é apresentado uma metodologia que nos permite analisar de forma quantitativa e qualitativa o impacto da presença dos cenários implícitos na confiabilidade de um sistema de natureza concorrente a partir da sua modelagem em cenários. Esta metodologia se mostra útil ao fornecer informações importantes para se analisar propriedades emergentes na composição dos componentes do sistema. As informações coletadas podem ser utilizadas para orientar importantes ações de refinamento arquitetural, permitindo assim a construção de softwares com maior confiabilidade.

Com essa metodologia, é obtida uma análise que apresenta o impacto da presença dos cenários implícitos em um software. O impacto foi medido pela diferença entre a confiabilidade estimada para o sistema, quando não é avaliada a presença dos cenários implícitos, e a confiabilidade real do sistema, avaliando-se a presença dos cenários implícitos para se compor o resultado. Assim é possível medir a relevância da análise de cenários implícitos na qualidade do sistema.

Como sistemas concorrentes podem conter iterações cíclicas entre seus componentes, a busca por cenários implícitos comumente revela cenários muito similares, que contém o mesmo padrão de mensagens que são responsáveis pela sua formação. Esses cenários são agrupados em uma Família de Cenários Implícitos, que, por possuir o mesmo conjunto de caminhos responsáveis por sua formação, produz um mesmo tipo de falha que pode ser tratado de forma similar. [13]

1.1 O Problema

São conhecidas na literatura várias abordagens para a detecção dos cenários implícitos, como por exemplo a abordagem usada pela ferramenta LTSA [18]. Contudo, essas abordagens possuem limitações com relação à detecção dos cenários implícitos. A detecção dos cenários implícitos, nessa abordagem, é um processo manualmente iterativo, que, na presença de iterações cíclicas entre os componentes de um sistema, pode ocasionar a detecção de um número indefinido de cenários implícitos, tornando mais difícil sua representação e correção.

Portanto, temos como meta deste trabalho solucionar os problemas de onerosidade na identificação de cenários implícitos. Assim como identificar, agrupar e omitir os resultados considerados redundantes, pertencentes a uma mesma família de cenários implícitos.

1.2 Solução Proposta

A solução proposta é aprimorar a detecção dos cenários implícitos pela ferramenta LTSA, que se mostra muito limitada na presença de iterações cíclicas entre os componentes de um sistema.

Nesta abordagem, a busca por cenários implícitos é feita repetidas vezes automaticamente, reduzindo o trabalho do usuário. Também é omitido qualquer resultado considerado da mesma família de cenários implícitos, que representa o agrupamento de cenários implícitos similares, permitindo que a visualização do resultado seja mais clara e com menos redundâncias. Ao final, é feito um relatório detalhado de todas as famílias encontradas, permitindo uma análise mais informativa e completa.

A nova abordagem mostra-se útil ao fornecer informações importantes para se analisar propriedades emergentes na composição dos componentes de um sistema de natureza concorrente. As informações obtidas a partir desta solução podem ser utilizadas para orientar a tomada de ações com relação à correção da arquitetura do sistema, por meio da identificação da extensão das falhas ocasionadas pelos cenários implícitos.

1.3 Trabalhos Relacionados

Uma metodologia para a automação do desenvolvimento de casos de teste foi proposta em [4] por Bertolino et al. Esta metodologia parte da especificação UML com o intuito de gerar os casos de teste antes do desenvolvimento do software. Essa abordagem possui como diferencial um aumento da acurácia e a diminuição do esforço no desenvolvimento dos casos de teste. Para isso eles recebem como entrada diagramas de estado e diagramas de sequência modelados em UML. A partir daí eles verificam a conformidade do diagrama sequência com o diagrama de estado. Contudo, utilizam apenas o diagrama de estado como o modelo de referência, a fim de gerar os futuros casos de teste.

Para realizar a geração de casos de teste, eles pressupõem que a geração dos conjuntos de diagramas de estado e diagramas de sequência pode ser falha, pois os diagramas podem ser inicialmente incompletos e terem somente uma especificação parcial ou inconsistente. Logo, em sua abordagem, é preciso verificar essas inconsistências antes de gerar os casos de teste.

Em seu trabalho, Bertolino et al. utilizam o método UIT (Use Interaction Test) que consiste em um método que constrói e define, de forma sistemática, um conjunto de casos de teste para a fase de testes de integração usando os diagramas UML como único modelo de referência, sem exigir a introdução de quaisquer formalismos adicionais.

Uma das possibilidades para o refinamento arquitetural para tratar cenários implícitos é o uso de contraints. O que pode não ser tão benéfico, pois em fases posteriores à fase de testes de integração ou de componentes fica mais difícil de rastrear os cenários implícitos e verificar a conformidade entre o modelo e a especificação, por não haver, na suite de testes, os testes capazes de identificar os cenários implícitos. Em nossa abordagem propomos a

geração de casos de teste, a partir da identificação dos cenários implícitos, em qualquer momento do desenvolvimento do software, desde a elicitação dos requisitos até a fase de manutenção do software.

Em [1] Al-Azzani et al. afirmam que uma das causas de vulnerabilidades em sistemas é a presença de cenários implícitos. Dessa forma, propuseram então o uso da detecção dos cenários implícitos para a criação de teste de segurança. A abordagem proposta por Al-Azzani et al. consiste em três etapas.

1. Detecção dos cenários implícitos: na primeira fase de sua proposta, Al-Azzani et al. identificam, utilizando a ferramenta LTSA [22], os cenários implícitos presentes em uma determinada especificação de software.

2. Revisão dos cenários implícitos detectados: na segunda fase, eles propõem encontrar as vulnerabilidades de segurança, violações, geradas partir dos cenários implícitos encontrados e catalogá-los. Um cenário implícito pode gerar vários tipos de violações.

3. As vulnerabilidades encontradas são utilizadas por testadores para criar novos casos de teste.

A abordagem proposta por Al-Azzani et al. cria um roteiro para a geração de casos de teste voltados para teste de segurança de forma semi-automatizada, uma vez que o LTSA [22] retorna de forma automatizada somente os cenários implícitos. Embora em sua metodologia seja citada a criação de casos de teste, Al-Azzani et al. não chegam a gerar os casos de teste. Em nossa abordagem propomos uma automatização da geração de casos de teste para a verificação de cenários implícitos.

Felipe Cantal et al. [16] demonstram como o conceito de cenários implícitos, até agora restrita às fases iniciais do ciclo de vida do software, pode ser aplicado para apoiar a compreensão e testes em sistemas já existentes. Eles apresentam um processo de engenharia reversa para apoiar a extração e detecção de cenários implícitos a partir de informações capturadas durante a execução de aplicações concorrentes.

A abordagem proposta em [16] para a detecção dos cenários implícitos se baseia no uso de engenharia reversa para obter uma modelagem parcial do software de forma iterativa, das execuções de suas funcionalidades. Assim, novos cenários são gerados a partir dos dados obtidos com a detecção dos rastros de execução, onde a detecção dos cenários implícitos é então feita na ferramenta LTSA-MSC [22].

Embora essa abordagem promova a detecção dos cenários implícitos a partir do uso de engenharia reversa ela não realiza a geração de casos de teste. Contudo, deixa o ambiente mais favorável a sua geração, visto que promove a identificação dos cenários implícitos em sistemas já implementados, que podem não ter sido documentados corretamente. Em nosso trabalho propomos a geração de forma automática dos casos de teste a partir da detecção dos cenários implícitos. Logo o processo proposto por Cantal et al. pode ser integrado à nossa proposta a fim de gerar casos de teste em sistemas que não possuam um documento de especificação de requisitos.

Em [12], Nogueira et al. promovem a criação de uma ferramenta que gera automaticamente casos de teste orientados a propósitos, a ATG [12]. Esta recebe como entrada o comportamento da aplicação, modelado em CSP (Communicating Sequential Processes)

[5], e o propósito de teste, que é o cenário que se deseja testar. Este cenário é definido manualmente por um testador a partir da avaliação do documento de especificação de requisitos.

A partir do propósito de teste, a ferramenta ATG faz a análise de consistência de tal propósito e verifica se o mesmo de teste está consistente com a implementação. Para realizar essa análise de consistência, a ferramenta ATG encapsula outra ferramenta em seu código fonte, a FDR(Failures and Divergences Refinament), que verifica se, de acordo com os modelos semânticos do CSP, um processo (implementação) é refinamento do outro (especificação). Caso não seja, a FDR gera contra-exemplos que expõe a violação de propriedade do modelo semântico. Utilizando a avaliação promovida pelo FDR e os contra-exemplos gerados, a ferramenta ATG gera os casos de teste em CSP. A ATG verifica também propriedades clássicas de processos, como: ausência de deadlock, ausência de livelock e o comportamento determinístico dos processos. Ela também gera o sistema de transição de estados (do inglês Labelled Transition System - LTS) de um processo CSP.

A ferramenta ATG promove a geração automática de casos de teste, mas não trata especificamente sobre cenários implícitos, pois a ferramenta não promove uma análise da comunicação entre os componentes nos diferentes cenários. Logo, o caminho que configura a presença do cenário implícito seria tratado como um caminho qualquer na modelagem, ficando a cargo do testador descobrir, a partir da avaliação do documento de requisito, se existe ou não a presença do cenário implícito. Isto torna-se uma tarefa muito cara, devido ao tempo despendido, se desenvolvida de forma manual.

A abordagem de [13] tem grandes similaridades a este trabalho. Utilizando a mesma ferramenta LTSA-MSC, [13] identifica um cenário implícito e, a partir dele, avalia todos os casos cenários pertencentes à mesma família de cenários implícitos. Dessa maneira, se pode atualizar o modelo arquitetural a fim de tratar todos os cenários implícitos daquela família, permitindo uma análise de confiabilidade mais rápida e eficiente. No nosso trabalho, ao invés de identificar um cenário implícito e encontrar todos os cenários daquela família, encontramos de uma vez só vários cenários implícito, agrupando-os em famílias quando possível. Por um lado, a nossa abordagem é mais rápida em encontrar várias famílias de cenários implícitos. Por outro lado, a abordagem de [13] trata, de uma vez só, todos os possíveis cenários implícitos de uma família.

1.4 Organização do Trabalho

No Capítulo 2, expomos uma Introdução Teórica que aborda os principais conceitos deste trabalho: Dependabilidade, diferenciando Defeito, Erro e Falha, também mostrando como fazer a Análise de Dependabilidade e categorizando as falhas; Cenários, incluindo Cenários Implícitos e diagramas MSC.

No Capítulo 3, descrevemos detalhadamente a metodologia criada neste trabalho para alcançar a solução proposta.

No Capítulo 4, fizemos a Análise dos Resultados, mostrando passo a passo como ocorre a execução da solução e quais saídas ela mostra. Discutimos todas informações obtidas para os dois sistemas usados de exemplo: Sistema de Caldeiras e Sistema *Cruiser*.

Finalmente, no Capítulo 5, concluímos sobre os resultados deste trabalho e comentamos possíveis trabalhos futuros a serem realizados.

Capítulo 2

Introdução Teórica

Este capítulo apresenta os principais conceitos utilizados nesse trabalho. Primeiro, a Seção 2.1 apresenta uma introdução sobre dependabilidade, incluindo a Seção 2.1.3, onde abordamos o conceito de caracterização da falha, proposto por [3], a fim de descrever o tipo de falha que será tratada pela metodologia. Em seguida, na Seção 2.2, abordamos sobre a modelagem de componentes de um sistema com o uso de cenários, em seguida é introduzido o conceito de cenários implícitos.

2.1 Dependabilidade

Dependabilidade é definida como a confiança de um sistema computacional de modo que se tenha segurança de que o serviço que este sistema desempenha será entregue. Este serviço é o comportamento do sistema como visto por seus usuários. E um usuário é outro sistema, que pode ser humano ou computacional, que interage com o sistema em questão. Resumidamente: a dependabilidade é a habilidade de um sistema computacional executar serviços em que se possa justificadamente confiar. [8]

O conceito de dependabilidade está relacionado à habilidade do sistema em evitar falhas que são mais frequentes e severas que o aceitável [3]. Tal conceito engloba os seguintes atributos:

- Disponibilidade (Availability) - Prontidão do sistema em se executar o serviço de forma correta.
- Confiabilidade (Reliability) - Continuidade da execução do serviço de um sistema de software de forma correta.
- Segurança (Safety) - A execução do sistema não tem consequências catastróficas para o usuário ou para o ambiente.
- Integridade (Integrity) - A execução do sistema não gera alterações impróprias no mesmo.
- Manutenabilidade (Maintainability) - Facilidade na modificação ou no reparo do sistema.

Dependendo do uso e aplicação do sistema, a avaliação da dependabilidade pelos responsáveis pelo sistema pode dar mais ênfase a um atributo ou outro [8]. Alguns desses atributos podem não ser necessários de maneira alguma para um sistema específico. [3]

2.1.1 Defeito, Erro e Falha

Problemas de confiabilidade e disponibilidade do sistema são geralmente causados por falhas de sistema. Algumas dessas falhas são uma consequência de erros de especificação ou falhas em outros sistemas relacionados, como sistemas de comunicações. No entanto, muitas falhas são uma consequência de comportamentos errados do sistema, resultante de defeitos nele. Quando se discute a confiabilidade, é útil usar a terminologia precisa e distinguir entre os termos "defeito", "erro" e "falha". [15]

- Defeito: Uma característica de um sistema de software que pode levar a um erro de sistema [15]. Esta característica pode ser criada por um programador, por exemplo, que inseriu um código errado no programa.
- Erro: Um estado errôneo de sistema que pode levar a um comportamento do sistema inesperado por seus usuários. [15] Caso o usuário tente utilizar esta parte do sistema, pode se deparar com uma falha.
- Falha: Um evento que ocorre em algum momento em que o sistema não fornece um serviço como esperado por seus usuários. [15] Ocorre quando um usuário tenta acessar um serviço do sistema que contém um erro.

Defeitos nem sempre resultam em erros, e erros não resultam necessariamente em falhas. As razões para isso são as seguintes [15]:

1. Nem todos os códigos de um programa são executados. O código que inclui um defeito (por exemplo, a falha para iniciar uma variável) pode nunca ser executado em virtude da maneira como o software é usado.
2. Os erros são transitórios. A variável de estado pode ter um valor incorreto, causado pela execução de um código defeituoso. Entretanto, antes disso, ela pode ser acessada e uma falha de sistema pode ser provocada; alguma outra entrada de sistema pode ser processada e o estado para um valor válido pode ser redefinido.
3. O sistema pode incluir a detecção de defeitos e mecanismos de proteção. Isso assegura que o comportamento errôneo seja descoberto e corrigido antes que os serviços do sistema sejam afetados.
4. Na prática, os usuários adaptam seu comportamento para evitar entradas que eles sabem que causam falhas no programa. Os usuários experientes "contornam" os recursos do software que eles sabem que não são confiáveis.

2.1.2 Análise de Dependabilidade

Falhas costumam ocorrer não devido a pequenos erros no código, mas principalmente durante o processo de projetar e definir a arquitetura o software. Por isso, a importância da análise de dependabilidade nas fases iniciais de desenvolvimento do software tornou-se cada vez mais evidente [7]. Uma vez que a qualidade do software é analisada nas fases iniciais do seu ciclo de vida, é possível antecipar a identificação de problemas relacionados ao desenvolvimento de software e evitar despesas imprevistas de custo, tempo e esforço [14].

Ao longo de mais de cinquenta anos foram desenvolvidos vários meios para se atingir os atributos de dependabilidade, os quais podem ser agrupados em quatro categorias principais [3]:

- Prevenção de Defeitos - Evitar a ocorrência ou a introdução de defeitos;
- Tolerância a Defeitos - Evitar falhas de serviço na ocorrência de defeitos;
- Remoção de Defeitos - Reduzir o número e a gravidade de defeitos; e
- Previsão de Defeitos - Meios para se estimar o número atual, a incidência futura e as prováveis consequências dos defeitos.

Prevenção de defeitos e tolerância a defeitos visa prover a habilidade de entregar um serviço que pode ser confiável. Enquanto remoção de defeitos e previsão de defeitos visam alcançar confiança nessa habilidade, justificando que a especificação funcional e especificação de dependabilidade estão adequadas e que o sistema muito provavelmente as alcança. [3]

Neste trabalho é apresentada uma nova forma para a elicitação de casos de teste, em sistemas concorrentes, que atua nas fases iniciais do desenvolvimento do software, mas não restrita a essas fases, a fim de se proporcionar uma maior qualidade aos casos de teste desenvolvidos. Com esta metodologia tem-se como objetivo evitar que ocorram falhas no sistema, geradas a partir da ocorrência de cenários implícitos, proporcionando uma maior confiabilidade ao sistema por meio de uma melhor previsão e estimativa dos defeitos.

2.1.3 Taxonomia da Falha

Uma falha ocorre quando o serviço entregue diverge daquele que foi especificado [3]. Para a análise dos cenários implícitos é importante a caracterização do tipo de falha que o sistema pode gerar. Em [3] foi proposto uma taxonomia para a classificação de falhas caracterizando-as em quatro pontos de vista: Domínio (Tempo ou Conteúdo), Detectabilidade (Detectável ou Indetectável), Consistência (Consistente ou Inconsistente) e Impacto (Mínima ou Catastrófica).

A categoria de falhas de Domínio é usada para distinguir as falhas de Conteúdo de falhas de Tempo:

- Falhas de Conteúdo: Quando o conteúdo da informação entregue por algum serviço desvia de sua especificação original.

- Falha de Tempo: Quando o tempo de chegada ou duração da informação entregue por algum serviço se desvia de sua especificação.

A Detectabilidade da falha nos permite saber sua real extensão.

- Falhas Detectáveis: As consequências das falhas podem ser detectadas por meio de técnicas de instrumentação, as quais detectam informações por meio de assertivas.
- Falhas Indetectáveis: As consequências das falhas não são detectadas até que a mesma ocorra.

A Consistência da falha se dá quanto à percepção da mesma pelos usuários.

- Falhas Consistentes: o serviço incorreto é percebido de modo idêntico a todos os usuários;
- Falhas Inconsistentes: o serviço incorreto é percebido de modo diferente pelos usuários.

O Impacto da falha permite a definição de sua severidade aos usuários e/ou ambiente.

- Falha Mínima: As consequências danosas da falha são limitadas ou no máximo similares aos benefícios providos pela operação correta do sistema.
- Falha Catastrófica: As consequências danosas das falhas são imensuravelmente maiores que os benefícios providos pela operação correta do sistema.

2.2 Cenários e Diagramas MSC

Um cenário descreve como um ou mais componentes de um sistema se relacionam para oferecer um conjunto de funcionalidades [16]. Cada cenário representa uma visão parcial do comportamento global do sistema, o qual, para ser entendido por completo, depende da combinação dessas diferentes especificações [21].

Especificações baseadas em cenários são geralmente expressas utilizando notações como Message Sequence Charts (MSCs) [12] e Diagramas de Sequência [10], bastante empregadas na descrição de requisitos de software. Eles representam, de uma forma intuitiva e flexível, os diferentes fluxos de mensagens entre componentes que caracterizam a execução de um sistema concorrente [2]. Por outro lado, esse tipo de especificação sofre de uma séria limitação, na qual nem sempre um modelo de comportamento de um sistema é a soma exata do conjunto de comportamentos expressos nas especificações e seus cenários individuais. [21]

Essas combinações inesperadas no modo como os componentes interagem podem forçar o aparecimento de comportamentos que não foram especificados nos cenários originais. Tais comportamentos são chamados de cenários implícitos [19], que surgem principalmente porque os componentes têm, em geral, uma visão local do que está acontecendo no sistema, e não uma visão do comportamento global esperado. [20]

Em um diagrama MSC (descrito na Figura 2.1), componentes do sistema, ambiente e usuários são representados como linhas verticais e descrevem a participação de um

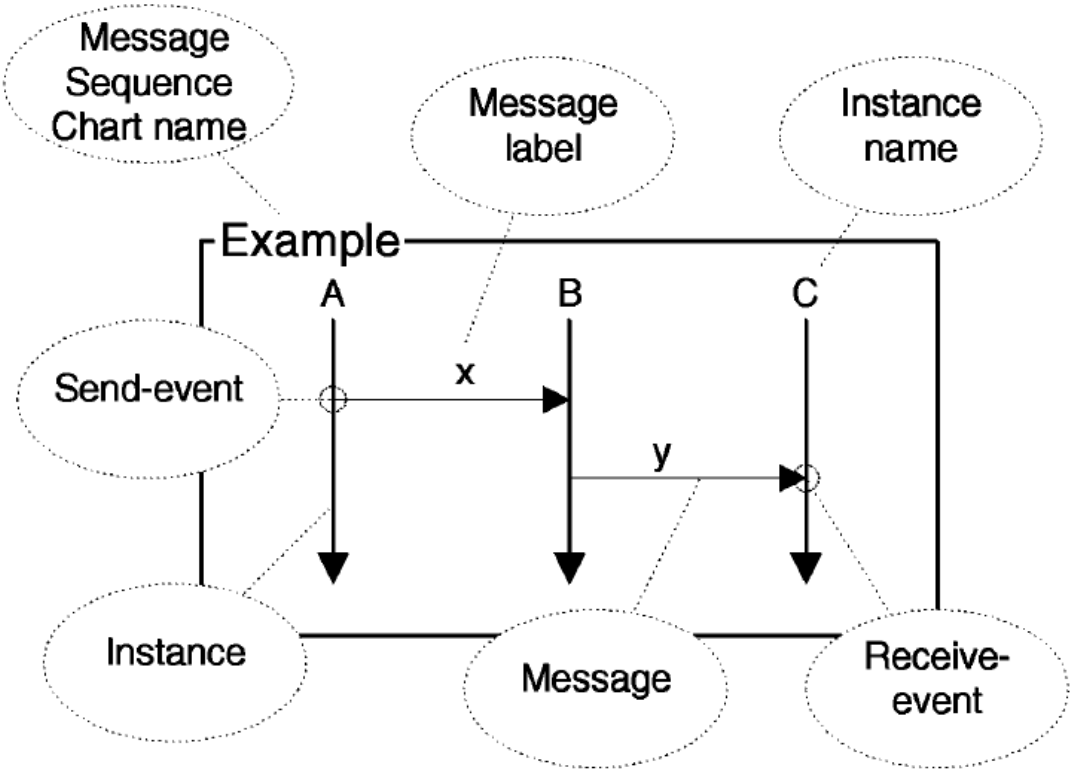

Figura 2.1: Elementos de um MSC [21]

componente no cenário. Interações entre componentes são as linhas horizontais, chamadas mensagens. A direção de uma mensagem indica qual componente inicia a interação. Diagramas MSC (ou simplesmente MSCs) tem representação temporal, de cima para baixo, ou seja: uma mensagem ocorre antes de todas as mensagens abaixo dela. [21]

2.2.1 Cenários Implícitos

Os cenários implícitos capturam comportamentos inusitados que não foram especificados explicitamente, mas que podem ocorrer devido à natureza parcial e concorrente das especificações baseadas em cenários [20]. Por essa razão os cenários implícitos precisam ser detectados, validados e categorizados como cenários implícitos positivos ou negativos. [18]

Cenários implícitos positivos são cenários que não foram especificados originalmente, mas o resultado proporcionado pelo mesmo é benéfico ao sistema, podendo ser integrado ao modelo. Cenários implícitos negativos são cenários que também não foram especificados e evidenciam um desvio do comportamento esperado para o sistema, ocasionando erros.

Em [13] é definido o conceito de Família de Cenários Implícitos como um conjunto de cenários implícitos negativos que contém o mesmo padrão de mensagens que são responsáveis pela sua formação. Essa família, por possuir o mesmo conjunto de caminhos responsáveis por sua formação, produz um mesmo tipo de falha que pode ser tratado de forma similar.

Para esse trabalho, foi necessário adaptar este conceito, para englobar também cenários implícitos positivos, pois nesta implementação, os cenários implícitos são agrupados em famílias antes da identificação de cenário implícito positivo ou negativo. Esta mudança de conceito não influencia na metodologia de maneira significativa, pois tratar um cenário implícito como positivo ou negativo é uma decisão do usuário e não das ferramentas de busca de cenários implícitos.

A ferramenta LTSA-MSC [18] oferece um suporte automatizado para a detecção, incremental, dos cenários implícitos. Ela foi desenvolvida como uma extensão da ferramenta de verificação de modelos concorrentes LTSA, Labelled Transition Systems Analyzer. [2]

O LTSA-MSC possui uma linguagem baseada em diagramas MSCs que se utiliza da composição paralela de cenários básicos. Eles são representados por MSCs básicos ou bMSCs (basic Message Sequence Chart), e são agrupados para criar uma descrição de cenários em alto nível, representados por MSCs de alto nível ou hMSCs (high level Message Sequence Chart) [22]. Uma especificação formada por um cenário de alto nível, produzida a partir de um conjunto de cenários básicos, é o necessário para que a LTSA modele todas as possíveis iterações entre os cenários e detecte a presença dos cenários implícitos na especificação.

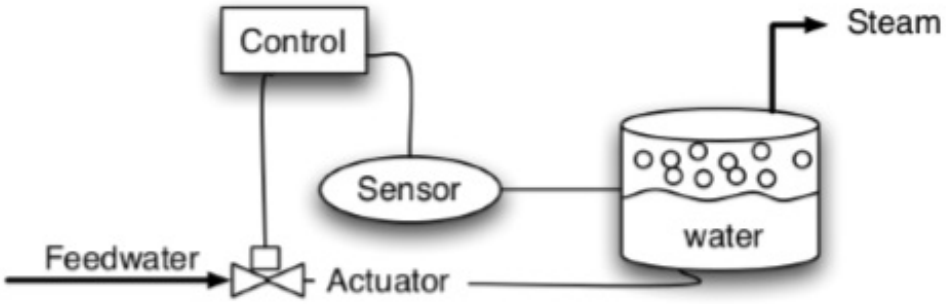

Figura 2.2: Ilustração do sitema de caldeiras, usado para apresentar a metodologia [19]

Na Figura 2.2 temos a representação de um sistema de caldeira [11] que possui cenários implícitos. Este sistema representa uma caldeira com sensores que medem a pressão interna da caldeira, e o processo de análise dos dados coletados pelos sensores resultam no aumento ou na diminuição da temperatura dentro da caldeira.

Na Figura 2.3 temos o sistema de caldeira expresso em MSCs. No bMSC Initialize, a mensagem do componente Control para o componente Sensor representa a ativação do sensor. O bMSC Register representa o registro dos dados, referentes à pressão que foi coletada pelo sensor, no componente Database. O bMSC Analysis representa uma sequência de mensagens entre os componentes Control, Database e Actuator, essas mensagens correspondem ao processo de análise do sistema, que pode resultar no aumento ou na diminuição da temperatura da caldeira, dependendo dos valores de pressão coletados pelo sensor e armazenados no banco de dados. Por fim, o bMSC Terminate representa a ação de desativação do sensor pelo componente de controle.

O cenário de alto nível (hMSC) mostrado na Figura 2.4 define as possíveis relações de continuidade entre os quatro cenários básicos do sistema. Note que, do ponto de vista individual de cada cenário, todos os componentes apresentam comportamentos válidos.

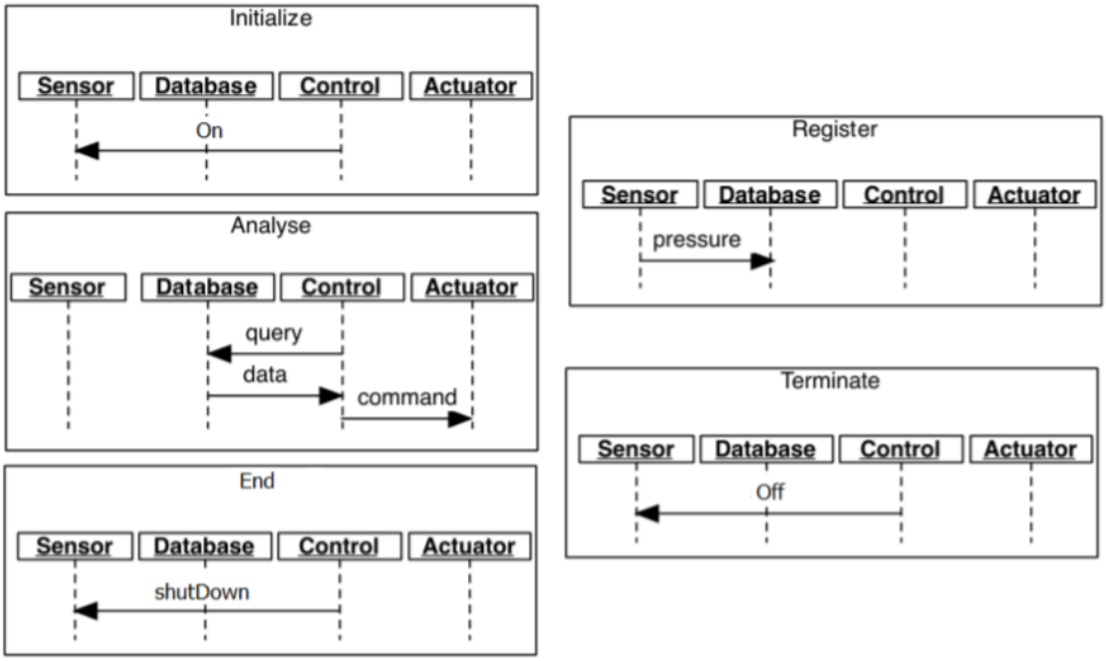

Figura 2.3: Modelagem em cenários do sistema de caldeiras [19]

No exemplo em questão, o cenário implícito retornado pela ferramenta LTSA-MSC, apresentado na Figura 2.5, se caracteriza pela seguinte sequência de mensagens iniciadas pelo controlador: $On \rightarrow Pressure \rightarrow Off$, representando um caminho de mensagens trocadas entre sensor e banco de dados. Posteriormente, o sensor é ligado novamente por uma nova mensagem On do controlador ao sensor seguido imediatamente de uma mensagem Query partindo do controlador ao banco de dados e consultando uma temperatura que pode estar defasada, gerando uma tomada de ação, pelo sistema, que pode ser errada naquele momento.

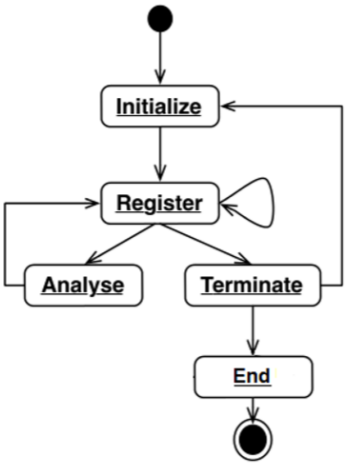

Figura 2.4: Diagrama hMSC do sistema de caldeiras [19]

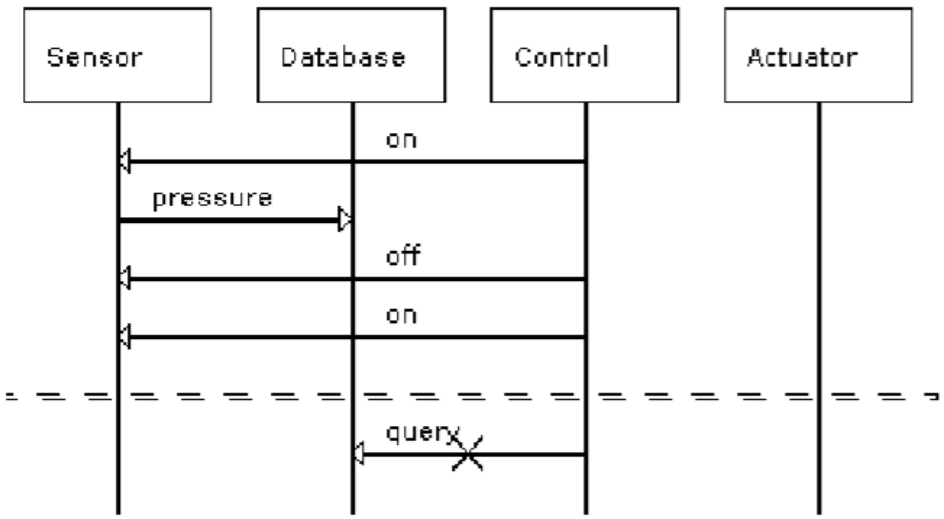

Figura 2.5: Cenário implícito negativo identificado pela ferramenta LTSA-MSC para o sistema de caldeiras

Capítulo 3

Metodologia

Neste capítulo descrevemos detalhadamente a metodologia criada neste trabalho para alcançar a solução proposta, explicando cada etapa e justificando cada decisão que foi tomada para a realização desta implementação.

3.1 Metodologia

A Figura 3.1 apresenta a metodologia desenvolvida. Todas as etapas são automáticas, realizadas em uma única execução. O único trabalho do usuário é carregar o arquivo da modelagem do sistema e solicitar a detecção de cenários implícitos, clicando no botão para iniciar a execução. Com isso, todas as etapas, repetições e decisões são realizadas automaticamente, cabendo ao usuário apenas analisar o resultado final gerado.

Para as etapas I, II e VI fazemos uso da metodologia desenvolvida nos trabalhos de Uchitel et al., já abordadas nas Seções 2.2 e 2.2.1. Após a etapa de identificação do cenário implícito, sucedem-se as etapas para a classificação deste cenário em uma família de cenários implícitos já existente ou não.

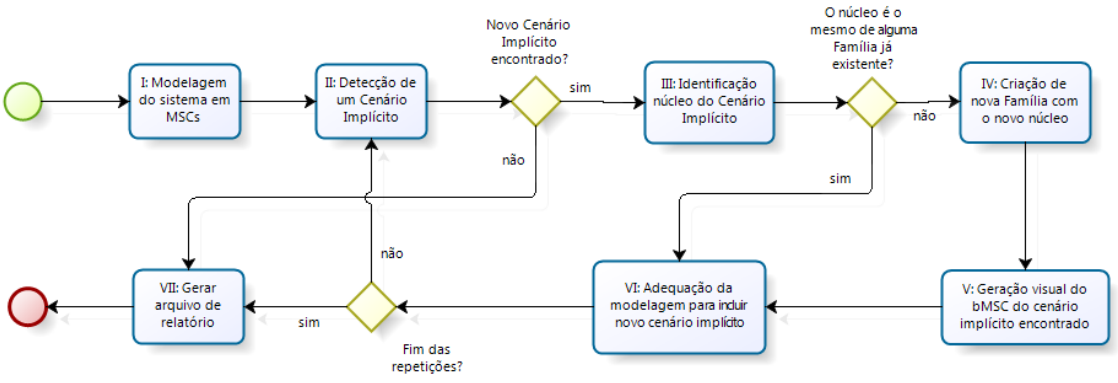

Figura 3.1: Visão geral da Metodologia

As etapas III e IV constituem o fundamental da nossa metodologia. É nelas que se identifica e agrupa os cenários implícitos pertencentes à mesma família. Para isso, é necessário levar em consideração a sequência de mensagens causadoras do cenário implícito detectado. Chamamos esta sequência de mensagens de "núcleo de cenários implícitos".

O processo se repete até que não sejam mais encontrados cenários implícitos ou se alcance o limite de repetições definido para a busca por cenários implícitos. Ao final do processo, temos informações de todas as famílias de cenários implícitos encontradas e quantos cenários implícitos foram encontrados de cada família. Para este trabalho, foi necessário redefinir o conceito de família de cenário implícito, definido em [13], e definir o conceito de núcleo de cenário implícito.

Dos Reis [13] define Família de Cenário Implícito como:

Definição 1 *Família de um Cenário Implícito - Dado um caminho (trace), que caracteriza uma troca de mensagens em um cenário implícito negativo, uma família de um cenário implícito é composta por todos os caminhos finitos que contêm a troca de mensagens que caracterizam o cenário implícito.*

Neste trabalho, redefinimos Família de Cenário Implícito para que considere também cenários implícitos positivos. Esta redefinição é importante, pois nesta implementação o agrupamento de cenários implícitos em famílias ocorre antes que o usuário classifique o cenário implícito como positivo ou negativo. Esta classificação não pode ser automatizada [18], mas o diferencial desta implementação é que sua execução é completamente automatizada, permitindo uma avaliação rápida e em grande escala. Por isso, cabe ao usuário classificar os cenários implícitos encontrados como positivos ou negativos posteriormente à execução.

Esta redefinição não causa prejuízos, pois um cenário implícito precisa ser tratado pelos responsáveis pelo sistema, seja o cenário implícito positivo ou negativo [18]. A diferenciação entre cenário implícito positivo ou negativo é relevante apenas durante a execução do sistema, sendo o cenário implícito positivo uma situação válida, porém não prevista; e o cenário implícito negativo uma situação que não deveria ocorrer, podendo levar a uma falha. [18]

Portanto, segue a definição de "Família de Cenário Implícito" que deve ser considerada para o restante deste trabalho:

Definição 2 *Família de Cenário Implícito - Dado um caminho (trace), que caracteriza uma troca de mensagens em um cenário implícito, uma família de cenário implícito é composta por todos os caminhos finitos que contém a troca de mensagens que caracterizam o cenário implícito (núcleo).*

Para este trabalho, também foi necessário definir um novo conceito, o de "Núcleo de Cenário Implícito", também podendo se referir ao "Núcleo de Família de Cenário Implícito".

Definição 3 *Núcleo de Cenário Implícito (ou simplesmente Núcleo) – Dado um cenário implícito, o núcleo deste cenário é a troca de mensagens que caracterizam o cenário implícito.*

Assim, temos que cada família contém exatamente um núcleo que a define. Este núcleo é o que relaciona todos os cenários implícitos daquela família. Por definição, todos os cenários implícitos de uma mesma família têm o mesmo núcleo. E famílias diferentes têm núcleos diferentes.

As mensagens que caracterizam o cenário implícito (núcleo) incluem N mensagens: a mensagem final do cenário implícito mais $N-1$ mensagens anteriores, Sendo $N \geq 2$. Para este trabalho, consideramos sempre $N = 2$, esta escolha é justificada na Seção 3.2, onde também são discutidas alternativas para o valor de N e suas possíveis motivações. A mensagem final do cenário implícito e a anterior a esta sempre estarão inclusas no núcleo do cenário implícito porque é a parte que não compõe os modelos originais. Ou seja, neste trabalho, o núcleo de um cenário implícito contém exatamente duas mensagens, que são as duas últimas mensagens de um cenário implícito. No caso do cenário implícito ilustrado na Figura 2.5, o núcleo deste cenário implícito é:

$on \rightarrow query$.

Seguem os sete passos definidos na visão geral da metodologia, ilustrada na Figura 3.1:

- Passo 1 - Modelagem do sistema em cenários utilizando os diagramas MSC a fim de se inserir o modelo na ferramenta LTSA - A partir do documento de especificação de requisitos do diagrama de casos de uso o sistema pode ser modelado em cenários;
- Passo 2 – (Primeira etapa da repetição) Detecção de um cenário implícito, realizada pelo LTSA.
- Passo 3 – Identificação do núcleo do cenário implícito encontrado.
- Passo 4 – Caso o núcleo encontrado seja novo, uma nova família de cenários implícitos é criada definida por este novo núcleo.
- Passo 5 – Também no caso de um novo núcleo, é desenvolvida a parte visual do bMSC do cenário implícito encontrado. Este bMSC representará toda a família do núcleo deste cenário implícito, pois foi o primeiro desta família a ser encontrado.
- Passo 6 – (Última etapa da repetição) Para que futuras buscas por cenários implícitos não encontrem o mesmo cenário, é feita a adequação da modelagem levando em consideração o novo cenário implícito encontrado.
- Passo 7 – Geração de um arquivo de relatório com informações sobre a execução e as famílias encontradas

É importante notar que ao final do Passo 6, retornamos ao Passo 2 para fazer uma nova busca por cenários implícitos. Este loop se repete por um número de vezes pré-definido pelo usuário ou até que nenhum cenário implícito seja encontrado. É necessário que se exista um número máximo de repetições realizadas, pois em muitos casos podem existir infinitos cenários implícitos, o que deixaria o programa executando por tempo indeterminado.

Na descrição da metodologia, os passos 1 e 2 são realizados utilizando a ferramenta LTSA-MSC, assim como a parte principal dos passos 5 e 6, nestes foram feitas pequenas adaptações.

3.1.1 Repetições para Agrupamento dos Cenários Implícitos em Famílias – Passos de 2 a 6

Como mencionado, os passos de 2 a 6 são repetidos até que não sejam encontrados cenários implícitos ou até atingir o limite de repetições.

Após a modelagem do sistema em cenários com os diagramas MSC, a ferramenta LTSA faz a busca por cenários implícitos nesta modelagem. Caso nenhum cenário implícito seja encontrado nesta busca, a execução é finalizada, avaliando o sistema como livre de cenários implícitos.

Quando a ferramenta LTSA encontra um cenário implícito em sua busca, é informada a sequência de mensagens que compõem este cenário implícito. No caso do cenário implícito da Figura 2.5, por exemplo, a sequência de mensagens informada é:

1. On
2. Pressure
3. Off
4. On
5. Query

Sabemos que a última mensagem dessa lista é a mensagem que não compõe os modelos originais da arquitetura, o que levou à identificação deste cenário implícito. Por isso, coletamos as duas últimas mensagens desta sequência para definir o núcleo deste cenário implícito. Justificaremos logo mais na Seção 3.2 sobre o tamanho das mensagens a ser considerado para definir o núcleo do cenário implícito. Sendo assim, neste caso, o núcleo do cenário implícito é:

1. On
2. Query

Com o núcleo do cenário implícito identificado (Passo 3), passamos para a próxima etapa que é buscar nas famílias já encontradas se existe alguma família com o mesmo núcleo. Para isso, é comparado o núcleo de cada família já encontrada com o núcleo do cenário implícito, necessitando que os núcleos sejam idênticos, tanto a primeira mensagem quanto a segunda.

Se não houver nenhuma família existente com o mesmo núcleo do cenário implícito, é criada uma nova família com este núcleo, contabilizando um cenário encontrado para esta família. Se já houver uma família existente com o mesmo núcleo, então apenas é incrementado em 1 o número de cenários encontrados desta família.

Para realizar o registro das famílias e a comparação dos núcleos, foi criada uma classe com a estrutura a seguir:

```
public class Family {
    private String FirstMessage;
    private String SecondMessage;
    private int Count;

    public Family(String first, String second);
```

```
    public boolean Compare(String first, String second);
    public void Add();
    public String Report();
}
```

No caso do cenário implícito do sistema de caldeira, o atributo FirstMessage teria o valor "On"e o atributo SecondMessage teria o valor "Query". Enquanto o atributo Count representa o número de cenários implícitos encontrados pertencentes a esta família, ou seja: começaria com o valor 1, e a cada novo cenário implícito encontrado pertencente a esta família, este valor é incrementado em 1.

Assim, fica claro que o agrupamento dos cenários implícitos encontrados se baseia em comparar as strings de cada uma das duas mensagens do núcleo do cenário implícito com as mensagens FirstMessage e SecondMessage de cada família já registrada. Se o resultado da comparação for idêntico para as duas mensagens, então o novo cenário implícito pertence a esta família.

Como evidenciado na estrutura acima, a classe Family também contém métodos de construtor, comparação de mensagens, incrementação e geração de informações para o relatório.

Se foi criada uma nova família, então o atual cenário implícito encontrado é o primeiro cenário implícito encontrado desta família, portanto, é preciso evidenciá-lo para o usuário. Para isso, a ferramenta LTSA-MSC cria um bMSC relativo ao cenário implícito encontrado para expor de maneira visual esta família. A Figura 2.5 é um exemplo deste bMSC criado. Esta será a representação visual da família que contém este cenário implícito, por ter sido o primeiro cenário implícito encontrado desta família. Em termos gerais, os próximos cenários implícitos encontrados que pertençam a esta família serão mais complexos que o primeiro, contendo mais mensagens, devido ao fato de serem mais demorados para se encontrar. Assim, a escolha desse cenário como representação da família busca simplicidade, visando facilitar o entendimento e interpretação dos resultados pelo usuário.

Finalmente, é necessário incluir o cenário implícito na modelagem do sistema para que ele não seja encontrado novamente em futuras buscas. Esta etapa é essencial para a execução apropriada da solução, porém com ela surge um problema. À medida que mais e mais buscas são realizadas repetidamente, a modelagem do sistema se torna cada vez maior, deixando as próximas buscas cada vez mais lentas. Para o contexto dos casos de teste deste trabalho isso ficou evidente, porém não foi tão relevante já que eram exemplos de sistemas simples e limitamos o número de repetições a 100. Entretanto, para sistemas grandes e o interesse em um número maior de repetições pode prejudicar severamente o desempenho da execução desta solução. Ainda assim, buscar por centenas de cenários implícitos sem antes tratar os cenários implícitos já encontrados se torna uma busca sem resultados muito significativos ou relevantes para a análise de confiabilidade do sistema. Para tanto, recomendamos que caso essa situação ocorra, seja aplicada a solução em [23], onde será feito o refinamento da arquitetura com base nos cenários implícitos detectados.

A não ser que o número de buscas pré-definido tenha sido concluído, após incluir o cenário implícito na modelagem do sistema, voltamos ao passo 2, recomeçando o processo de busca por cenários implícitos. Com o decorrer das repetições, cada vez menos famílias novas serão encontradas, levando eventualmente a uma longa sequência de cenários implícitos pertencentes a famílias já encontradas.

Como já mencionado, caso não houvesse um número pré-definido limitado de iterações, a busca poderia se repetir indefinidamente. Por isso, para os nossos casos de teste, definimos um máximo de 100 buscas. Porém, outras condições de parada que não definissem um limite máximo de buscas poderiam ser interessantes, por exemplo limitando o tempo de execução. Alternativas para condições de parada são mencionadas na Seção de Trabalhos Futuros 5.2.4.

Abaixo segue o algoritmo referente ao loop dos passos de 2 a 6:

Algoritmo 1

```
1  implicito = BuscaCenarioImplicito();
2  while (implicito != null && buscas < 100){
3    familiaExiste = false;
4    foreach (familia in Familias) {
5      if (familia.temMesmoNucleo(implicito)) {
6        familiaExiste = true;
7        familia.Add();
8      }
9    }
10
11   if (!familiaExiste) {
12     Familias.Add(new familia(implicito));
13     MostrabMSC();
14   }
15
16   AtualizaModelagem();
17   buscas++;
18   implicito = BuscaCenarioImplicito();
19 }
```

Como podemos ver, o Algoritmo 3.1.1, que representa a metodologia e solução deste projeto, é simples. As linhas 1 e 18 se referem ao Passo 2 da metodologia: busca pelo cenário implícito. Na linha 2 verificamos se nenhum cenário implícito foi encontrado ou se foram terminadas as buscas, para finalizar a execução em um desses casos. Da linha 4 a 9 verificamos se o cenário implícito encontrado pertence a alguma família já registrada. Se não pertence a nenhuma família existente (linha 11), criamos a nova família e evidenciamos o bMSC desse cenário implícito para o usuário. Por fim, atualizamos a modelagem (linha 16), incrementamos a quantidade de buscas realizadas (linha 17) e repetimos o processo.

A maior parte da execução é apenas uma variação da antiga solução já existente da ferramenta LTSA-MSC. Juntamente com os casos de teste, isso mostra como essa execução é rápida e eficiente, simplesmente automatizando as partes essenciais para ter uma análise de cenários implícitos muito abrangente e sem redundâncias, poupando o usuário de um trabalho custoso e impreciso.

3.1.2 Geração de um Arquivo de Relatório – Passo 7

Ao final da execução, é gerado um arquivo relatando detalhes da execução e das famílias e cenários implícitos encontrados.

Primeiramente, é indicado o número de buscas (repetições) realizadas; o número de famílias encontradas; a quantidade de cenários implícitos encontrados que pertenciam a famílias já encontradas; o tempo decorrido da execução; e se mais cenários implícitos podem ser encontrados ou se a execução foi interrompida por não encontrar mais cenários implícitos.

Após esses dados iniciais, é mostrado no arquivo um relatório de todas as famílias encontradas, indicando o núcleo de cada família e a quantidade de cenários implícitos encontrados pertencentes àquela família.

3.2 Número de Mensagens de um Núcleo de Cenário Implícito

Como mencionado logo após a Definição 3 de Núcleo de Cenário Implícito, as mensagens que caracterizam o cenário implícito podem variar em quantidade. Essas N mensagens incluem a mensagem final do cenário implícito mais $N-1$ mensagens anteriores. Sendo $N \geq 2$. A Figura 3.2 mostra quais mensagens fazem parte do núcleo do cenário implícito da Figura 2.5, quando $N = 2$, $N = 3$ ou $N = 4$.

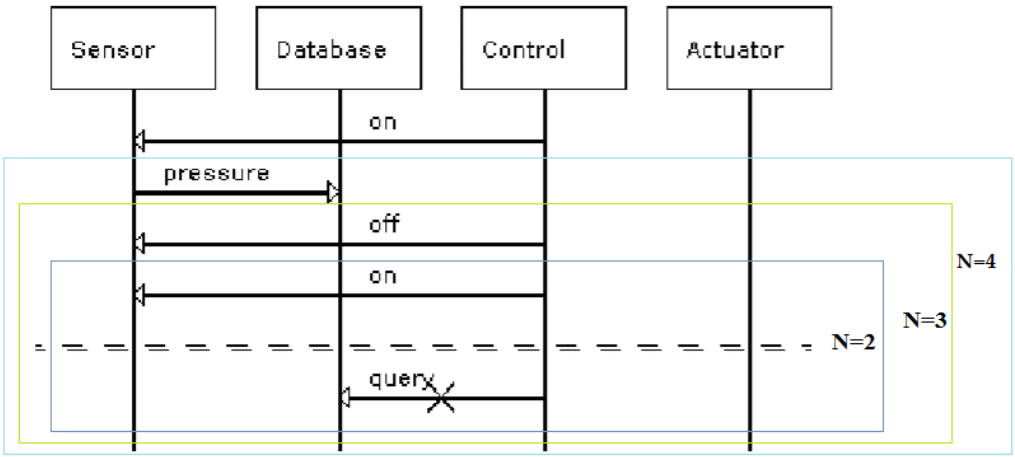

Figura 3.2: Indicação dos elementos do núcleo, para cada valor de N

Para o caso da Figura 3.2, quando $N = 2$, o núcleo é $on \rightarrow query$; quando $N = 3$ o núcleo é $off \rightarrow on \rightarrow query$; e quando $N = 4$ o núcleo é $pressure \rightarrow off \rightarrow on \rightarrow query$.

Quanto maior o valor de N, possivelmente mais famílias diferentes serão definidas para possíveis cenários implícitos similares. Isso é intuitivo, pois quanto maior o valor de N, mais específica é a definição da família de cenários implícitos e por isso menos cenários implícitos irão fazer parte da mesma família.

Para ilustrar isso, digamos que temos dois cenários implícitos:

1. $message0 \rightarrow message1 \rightarrow messageA \rightarrow messageB$
2. $message0 \rightarrow message2 \rightarrow messageA \rightarrow messageB$

Para $N = 2$, os dois cenários fariam parte da mesma família, que teria como núcleo $messageA \rightarrow messageB$. Já para $N = 3$, existiria uma família para cada cenário, a primeira com o núcleo: $message1 \rightarrow messageA \rightarrow messageB$ e a segunda com o núcleo: $message2 \rightarrow messageA \rightarrow messageB$

É tido como premissa que se deseja encontrar o mínimo possível de famílias de cenários implícitos, para que assim o máximo possível de cenários implícitos estejam agrupados nas poucas famílias encontradas. Tornando mais fácil o entendimento, análise e tratamento do resultado da busca por cenários implícitos.

Por outro lado, também não é interessante que cenários implícitos de causas e contextos diferentes estejam agrupados em uma mesma família, pois isso faria com que o resultado da busca por cenários implícitos obscureça algum defeito do sistema, já que a família provavelmente vai evidenciar apenas um destes cenários implícitos.

É importante salientar também que de acordo com [18], depois de ser feita a busca por cenários implícitos, a arquitetura do sistema deve ser aprimorada de acordo com os cenários implícitos encontrados e, posteriormente, uma nova busca por cenários implícitos deve ser feita na nova arquitetura. Isso é evidente, já que é natural que uma mudança na arquitetura pode levar ao surgimento de um defeito que não existia anteriormente [15].

Tendo isso em vista, sabemos que é existente o prejuízo de uma família que agrupe cenários implícitos de causas diferentes e obscureça um dos defeitos. Entretanto, este prejuízo não é tão grave, pois após a correção do defeito encontrado na arquitetura, uma nova busca por cenários implícitos evidenciaria o defeito que na primeira execução não estava aparente, já que ele não foi tratado.

Com esses argumentos como base, optamos por definir $N = 2$, mostrando assim o mínimo possível de famílias de cenários implícitos e favorecendo o agrupamento dos cenários. Esta decisão foi tomada porque consideramos que o prejuízo de possivelmente omitir defeitos era menor que o benefício de possivelmente simplificar os resultados e a análise destes.

Esta decisão se revelou positiva ao analisar os casos de teste, pois mesmo com $N = 2$, muitas famílias foram encontradas, permitindo uma análise eficiente de diversos defeitos encontrados na arquitetura. Na Seção 5.2 discutimos valores alternativos de N.

No próximo capítulo, apresentamos os resultados dos exemplos realizados com base nessa proposta.

Capítulo 4

Análise dos Resultados

Para observar o resultado deste trabalho, foi feita a análise dos resultados de dois casos de teste: o do sistema de caldeiras já mencionado anteriormente e um sistema de navegação (*cruiser*). Para isso, utilizamos a ferramenta LTSA-MSC com a implementação deste trabalho para realizar a análise de cenários implícitos dos sistemas mencionados.

Não é tido como foco desta análise avaliar de fato a qualidade dos sistemas escolhidos, mas sim realizar uma comparação do resultado atual com o antigo resultado da análise da ferramenta LTSA-MSC. E demonstrar com esses casos de teste que a nova implementação fornece resultados mais completos e mais fáceis de analisar do que anteriormente, mantendo a congruência e corretude.

Assim, não iremos discutir se os cenários implícitos encontrados são positivos ou negativos, tampouco a usabilidade dos sistemas dos casos de teste, já que este tema se estenderia para além do escopo deste trabalho e não contribuiria com a demonstração do objetivo: aprimorar uma ferramenta que já tem sua efetividade confirmada.

Para essas análises foram predefinidas execuções com até 100 buscas por cenários implícitos e núcleo de cenário implícito identificado como as duas últimas mensagens do cenário implícito, essas definições são justificadas na Seção 3.1.

É importante observar que devido à natureza concorrente dos sistemas, o resultado das buscas pode variar quando executado duas vezes para um mesmo sistema [22] [13]. Por isso, para nossos casos de teste executamos a solução para o mesmo input 10 vezes. Apesar de que de fato encontramos algumas diferenças em diferentes execuções, nenhuma delas foi significativa, como fica evidenciado nas Seções 4.1.3 e 4.2.3. Em todas as execuções, exatamente as mesmas famílias foram encontradas e aproximadamente ou idêntico número de cenários implícitos pertencentes a cada família foram encontrados. A única diferença entre execuções foi que, eventualmente, a ordem dos cenários implícitos encontrados mudava. Por isso, nenhuma mudança realmente significativa foi encontrada entre diferentes execuções.

Os casos de teste abaixo foram executados em um computador com as seguintes especificações:

- Sistema Operacional: Ubuntu 12.04
- Processador: Intel(R) Core(TM) i3-2350M CPU @ 2.30GHz
- Memória: 4GB

4.1 Análise do Sistema de Caldeira

Para ilustrar a aplicação da metodologia proposta neste trabalho e detalhar suas etapas, utilizaremos um sistema simples, que já foi amplamente explorado em outros trabalhos relacionados com a detecção de cenários implícitos. Trata-se de um sistema de caldeira, um armazenador térmico de água, composto por um sensor de vapor que detecta a presença de vapor e envia as informações para um módulo de controle que aciona um mecanismo para ativar ou desativar o sistema de aquecimento de água. Um importante requisito para o sistema é a necessidade do nível de água desse reservatório se manter em uma determinada faixa de valores, de modo que não fique com níveis altos ou baixos. Se esse requisito não for cumprido, o sistema da caldeira pode sofrer sérios danos [19]. A Figura 2.2 ilustra o referido sistema.

4.1.1 Passo a Passo da Execução

Modelagem do Sistema em Cenários

Para modelar o sistema em cenários, utilizaremos a linguagem MSC, já explicada anteriormente. Baseado na descrição apresentada acima, temos 4 componentes que serão utilizados para a modelar esse sistema. São eles: Controlador (Control), Sensor, Atuador (Actuator) e o Banco de Dados (Database), para armazenar os dados enviados pelo sensor.

Na Figura 2.3 é apresentada a modelagem em cenários utilizada para o experimento, e na Figura 2.4 é apresentado o hMSC que representa todas as possíveis iterações entre os componentes do sistema.

Iniciando as Repetições: Identificação de um Cenário Implícito

Um dos resultados produzidos pelo LTSA-MSC é a identificação dos cenários implícitos existentes no software. O objetivo desse passo é avaliar os cenários implícitos produzidos pela ferramenta a cada iteração, para posteriormente classifica-los nas devidas famílias de cenários implícitos, ou criar uma nova família para este cenário implícito.

A Figura 2.5 apresenta o primeiro cenário implícito detectado pela ferramenta. Assim, temos a informação da sequência de mensagens que compõe o cenário implícito, que é:

1. On
2. Pressure
3. Off
4. On
5. Query

Tratando a Família do Cenário Implícito Encontrado

Coletamos então as duas últimas mensagens desta sequência para definir o núcleo deste cenário implícito, ou seja o núcleo deste cenário implícito é:

1. On
2. Query

Com o núcleo do cenário implícito identificado (Passo 3), passamos para a próxima etapa, que é buscar nas famílias já encontradas se existe alguma família com o mesmo núcleo. Como este é o primeiro cenário implícito encontrado, ainda não foi registrada nenhuma família de cenários implícitos, então é criada uma família com este núcleo e indicada com 1 cenário implícito encontrado pertencente a ela.

Geração do bMSC do Cenário Implícito

Como este cenário implícito é de uma família nova, ele será exposto para o usuário, representando esta família. Essa representação é feita em bMSC, exatamente como na Figura 2.5. Futuros cenários implícitos que pertençam a esta mesma família não serão evidenciados para o usuário, já que este cenário implícito já os representa. Antes da implementação deste trabalho, novas buscas por cenários implícitos revelariam também os cenários implícitos desta mesma família. A Figura 4.1 mostra dois exemplos desses cenários que pertencem à esta mesma família e seriam representados com a execução original da ferramenta:

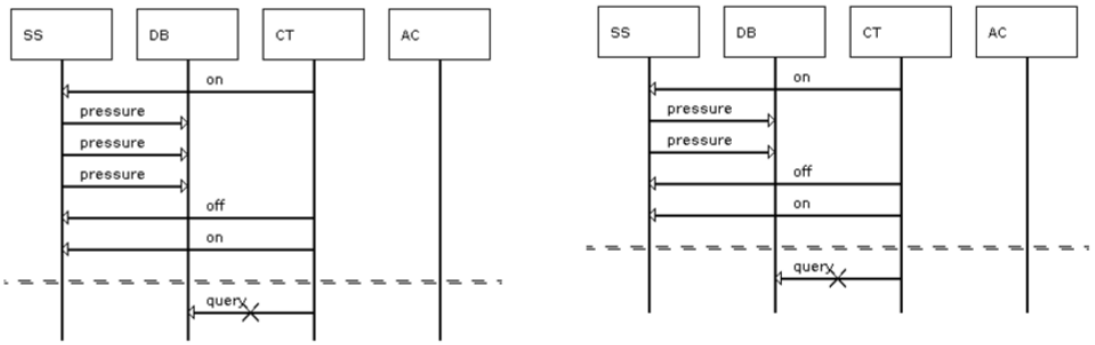

Figura 4.1: Cenários Implícitos pertencentes à mesma família.

Fica claro, observando estes cenários implícitos, que são muito similares ao primeiro cenário implícito encontrado desta família (evidenciado na Figura 2.5). Portanto, fica aqui evidenciada uma das principais funcionalidades propostas por este trabalho: reduzir a redundância de informações obtidas nas buscas por cenários implícitos, facilitando a análise dos resultados.

Em futuras repetições, estes cenários implícitos também serão encontrados, porém como pertencerão à mesma família de um cenário implícito já encontrado, eles não serão expostos ao usuário.

Adequação da modelagem

Além de expor o cenário implícito para o usuário, é necessário atualizar o modelo do sistema com este cenário, para que as próximas buscas não encontrem este mesmo cenário implícito novamente. Isso também é feito pela própria ferramenta LTSA-MSC que já tem esta funcionalidade, mas que precisa ser chamada adequadamente. Esta etapa é realizada em todas as iterações de busca, mesmo em cenários implícitos de famílias já encontradas, pois estes também poderiam ser encontrados novamente pela ferramenta se esta etapa não fosse executada.

Depois da adequação, é verificado se o número de repetições máximo já foi alcançado, esta é a primeira iteração e definimos como 100 repetições, portanto a execução do programa continua.

Segunda Iteração

Voltamos à etapa de Identificação de um Cenário Implícito, porém agora com o modelo do sistema atualizado para considerar o cenário da Figura 2.5, portanto este cenário não será mais encontrado. Agora, também temos registrada a família do cenário implícito encontrado na primeira iteração.

A segunda busca também encontra um cenário implícito. Caso não tivesse encontrado um cenário implícito, avançaríamos para a etapa final para gerar o relatório e finalizaria a execução do programa.

O cenário implícito encontrado é o cenário à direita da Figura 4.1. Como já observado, este cenário implícito pertence à mesma família do primeiro cenário implícito encontrado. Isso é demonstrado olhando as mensagens que compõem o cenário e identificando seu núcleo:

Mensagens do cenário implícito encontrado:

1. On
2. Pressure
3. Pressure
4. Off
5. On
6. Query

Núcleo do cenário implícito encontrado:

1. On
2. Query

Como podemos ver, o núcleo é o mesmo da família já registrada. Isso é identificado na próxima etapa do programa, quando o núcleo deste cenário implícito é comparado com os núcleos das famílias já registradas em busca de um idêntico.

Já que o cenário tem o mesmo núcleo da família registrada, então ele também pertence a esta família. Por isso, não é registrada nova família nem é gerado o bMSC deste cenário implícito para visualização do usuário, pois ele já está sendo representado pelo outro cenário implícito já encontrado da mesma família. A única coisa a ser feita nesta família é incrementar o valor de cenários implícitos pertencentes a ela encontrados, para o relatório final.

Antes de finalizar a segunda iteração, é necessário adequar a modelagem para incluir este cenário implícito, por mais que a família dele já tenha sido encontrada, como já comentado anteriormente, para que este cenário não seja encontrado em buscas futuras.

Terceira Iteração (nova família)

Passando para a nova busca, já que o número de buscas ainda não foi alcançado, encontramos um novo cenário implícito descrito na Figura 4.2. Como se pode ver, a sequência de mensagens que compõe este cenário implícito é:

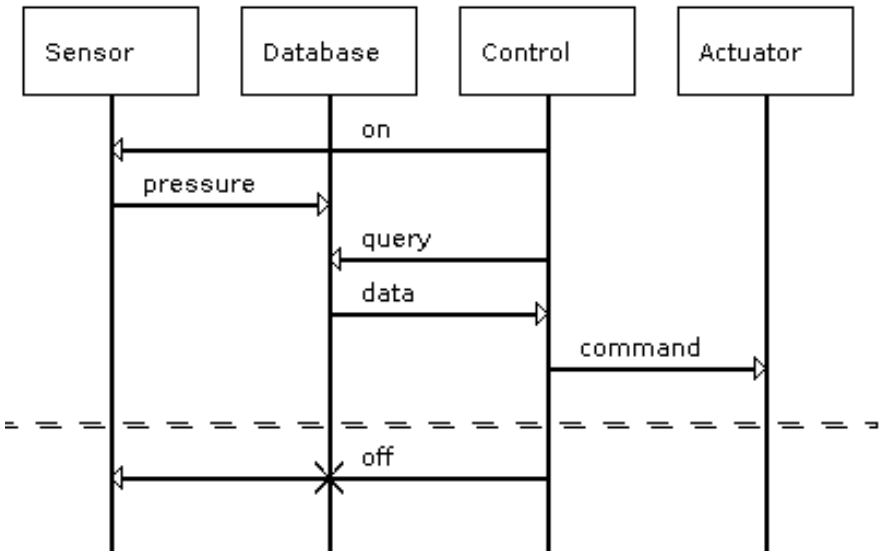

Figura 4.2: Novo Cenário Implícito encontrado.

1. On
2. Pressure
3. Query
4. Data
5. Command
6. Off

Separando as duas últimas mensagens, temos o núcleo:

1. Command
2. Off

Seguindo para a etapa de buscar este núcleo nas famílias já existentes, vemos que este é um núcleo novo, pois a única família já registrada tem o núcleo:

1. On
2. Query

Assim, devemos criar uma nova família de cenários implícitos que inclua este cenário. Criamos uma família com este núcleo indicando que existe até agora 1 cenário implícito encontrado pertencente a ela. Este processo, assim como toda a execução, é feito automaticamente pelo algoritmo desenvolvido. Isso poupa o usuário a realizar a análise cenário a cenário ou família a família, cabe ao usuário analisar apenas o resultado final da execução, já com todas as famílias agrupadas e o relatório gerado.

Como este cenário implícito é de uma família nova, ele será exposto para o usuário, representando esta família. Da mesma maneira que na primeira família, essa representação é feita em bMSC, exatamente como na Figura 4.2. Futuros cenários implícitos que pertençam a esta mesma família não serão evidenciados para o usuário, já que este cenário implícito já os representa.

Como em todas as iterações, adequamos a modelagem do sistema para levar em consideração este cenário implícito e com isso se conclui mais uma iteração.

Iterações Subsequentes

Nas iterações subsequentes, nenhuma nova família é encontrada. Cada novo cenário implícito encontrado é identificado como pertencente a uma das duas famílias mencionadas. Então, para cada iteração, a contagem de cenários implícitos encontrados daquela família é incrementada; não é mostrado nenhum novo bMSC para visualização do usuário; e a modelagem do sistema sempre é atualizada.

Isso se repete até que se completem as 100 iterações previstas, já que todas as buscas sempre encontram um novo cenário implícito.

Geração do Relatório

Ao se concluir as buscas, temos finalmente todos os dados coletados e iremos gerar o relatório final em um arquivo com as informações mencionadas anteriormente.

Esse relatório junto com os dois bMSCs gerados referentes a cada família revelam muitas informações relacionadas aos cenários implícitos encontrados assim como à arquitetura dos sistema avaliado. Cabe agora ao usuário classificar essas famílias de cenários implícitos encontrados como positivas ou negativas e realizar as devidas mudanças na modelagem arquitetural do sistema para contemplar cada cenário implícito se for positivo ou prevenir se for negativo. Observa-se, então, a simplicidade de poder analisar e tratar apenas dois cenários implícitos enquanto cem cenários implícitos foram identificados. E mesmo tratando apenas os dois representados, os cem cenários deverão ser solucionados.

4.1.2 Análise do Resultado

Como já mostrado anteriormente, a execução no sistema de caldeira encontrou apenas duas famílias de cenários implícitos. Os primeiros cenários implícitos encontrados de cada uma destas famílias tiveram seus bMSCs gerados para representar visualmente ao usuário cada família.

Essas famílias foram identificadas nas buscas 1 e 3 (primeira e terceira). Ou seja: A primeira busca encontrou um cenário implícito, e como não existiam famílias registradas ainda, este cenário implícito levou à criação de uma nova família. A segunda busca encontrou um cenário implícito pertencente a essa mesma família, logo não foram criadas famílias novas. A terceira busca encontrou um cenário implícito de núcleo diferente da família já registrada, levando à criação de uma segunda família. A partir da quarta busca e até a centésima (última), todos os cenários implícitos encontrados pertenciam a estas mesmas famílias.

Isso pode ser usado para dar segurança ao usuário de que provavelmente não surgirão novas famílias de cenários implícitos. Anteriormente, ao realizar as buscas após a terceira, o usuário tinha que analisar cada cenário implícito encontrado para confirmar que era ou não pertencente a alguma das famílias encontradas. Atividade que poderia ser muito onerosa e passível de erro, especialmente em sistemas mais complexos.

A Figura 4.3 mostra como fica evidenciado na ferramenta LTSA o resultado após a execução para o sistema de caldeira. As duas famílias encontradas estão identificadas como $Fam1$ e $Fam3$, porque foram encontradas nas buscas 1 e 3. O bMSC de cada família é o bMSC do primeiro cenário implícito encontrado dessas famílias. Observamos

que apesar de terem ocorrido 100 buscas, apenas dois bMSCs são mostrados, evidenciando como a análise do resultado é simplificada para o usuário.

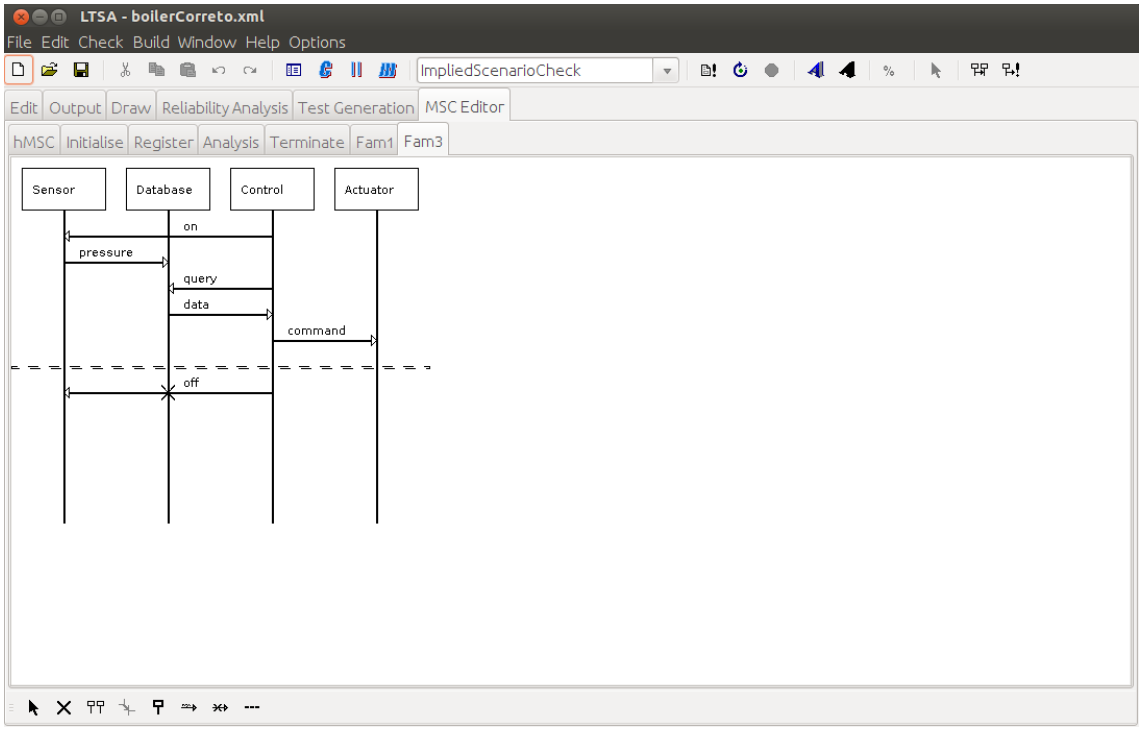

Figura 4.3: Exemplo de saída da ferramenta LTSA após a execução da solução para o sistema de caldeira

Segue abaixo o relatório final gerado para o sistema de caldeira:

```
 100 buscas por cenários implícitos realizadas.
 2 famílias encontradas.
 98 cenários encontrados de famílias já existentes.
 Tempo decorrido: 331,641 segundos.
  Obs.:Mais cenários implícitos possivelmente ainda podem ser encontrados.

Relatório das famílias com a quantidade de cenários implícitos encontrados:
Núcleo da Família:
 on -> query
 Quantidade: 62

Núcleo da Família:
 command -> off
 Quantidade: 38
```

Como já mencionado, o relatório indica que foram feitas 100 buscas: 2 famílias foram encontradas e 98 cenários implícitos foram encontrados de famílias já existentes.

O tempo de execução foi de aproximadamente 5 minutos e meio. Como veremos com a análise do próximo sistema, para sistemas mais complexos, esta execução é mais

demorada. Apesar de não terem sido feitos testes com sistemas grandes, não parece que o tempo de execução poderia inviabilizar a análise. Primeiro porque em nenhum caso de teste o tempo de execução atingiu valores longos demais (nenhum chegou a 10 minutos). Segundo porque o processador utilizado nos testes está abaixo em desempenho de processadores comuns no mercado atualmente. E terceiro porque a quantidade máxima de buscas pode ser diminuída para viabilizar a execução em casos extremos.

O relatório também indica que mais cenários implícitos ainda podem ser encontrados. Isso porque a todas as buscas realizadas encontraram cenários implícitos e a execução só parou porque o limite de buscas pré-definido foi alcançado. Isso sugere que mais famílias poderiam ser encontradas, mas considerando que 97 buscas seguidas não encontraram famílias novas, este caso parece ser muito improvável. A arquitetura do sistema de caldeira abre a possibilidade de infinitos cenários implícitos de uma mesma família, então a busca precisa ser interrompida eventualmente.

Finalmente, o relatório mostra as duas famílias encontradas, indicando seus núcleos e quantos cenários implícitos foram encontrados pertencentes a cada uma delas.

A primeira família encontrada tem o núcleo:

$$on \rightarrow query$$

Foram encontrados 62 cenários implícitos pertencentes a esta família.

A segunda família tem o núcleo:

$$command \rightarrow off$$

Foram encontrados 38 cenários implícitos pertencentes a esta família. A primeira família teve seu primeiro cenário implícito encontrado primeiro e é mais simples que a segunda (é composta de menos mensagens), por isso, não surpreende que mais cenários implícitos foram encontrados da primeira família do que da segunda.

A execução para o sistema de caldeira encontrou com sucesso os dois cenários implícitos relevantes. Em outros trabalhos que analisaram este mesmo sistema [19] [22] [13], os mesmos dois cenários implícitos que formaram as duas famílias foram encontrados e considerados relevantes, enquanto os outros cenários implícitos encontrados eram considerados redundantes e ignorados. Analogamente, neste trabalho, omitimos os outros cenários implícitos, apenas indicando a qual família pertenciam, e mostramos apenas os dois cenários implícitos relevantes. Isso demonstra que para o sistema de caldeira, este trabalho obteve sucesso em sua proposta.

Análise do Tempo de Execução

A título de observação, registramos também o tempo decorrido para cada 10 buscas. Ou seja: o tempo de execução para a realização das buscas de 1 a 10, tempo das buscas de 11 a 20, etc. Como indicado na Figura 4.4, as buscas se tornam cada vez mais lentas com o passar da execução, isso ocorre porque a cada atualização da modelagem de cenários, esta modelagem se torna maior e por isso é mais demorado o processamento nela. Isso evidencia como é importante a discussão já mencionada na Seção 3.1.1 da condição de parada da execução. Para os casos de teste usados, 100 buscas se mostrou um valor desnecessário, já que na terceira busca a última família foi encontrada e foram feitas mais 97 buscas sem encontrar famílias novas.

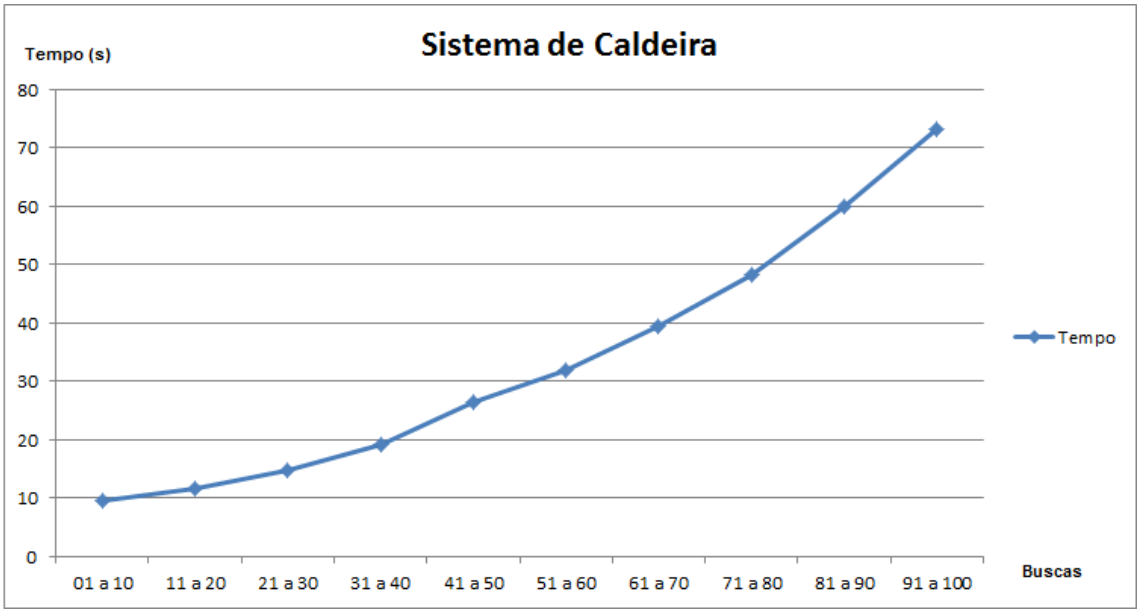

Figura 4.4: Tempo de execução em segundos de cada 10 buscas, das primeiras até as últimas, do sistema de caldeira

Tabela 4.1: Tempo em segundos decorrido nas 10 execuções do sistema de caldeira

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
331,98	332,05	332,59	331,4	332,27	331,44	330,25	331,29	332,14	331,64

4.1.3 Análise de Diferentes Execuções

Como já mencionado, diferentes execuções podem levar a resultados diferentes, por isso, para ter segurança de que os resultados encontrados não foram atípicos ou diferenciados, realizamos a execução da solução 10 vezes para cada caso. As execuções estão enumeradas de #1 a #10, sendo a execução #10 a que foi analisada detalhadamente acima.

A Tabela 4.1 revela que não houve variação significativa no tempo decorrido para cada execução.

A Tabela 4.2 mostra em que iteração foi encontrado o primeiro cenário implícito de cada família. Ou seja: a iteração em que aquele núcleo foi encontrado pela primeira vez, sendo feito o registro da nova família.

Podemos observar que para todas as execuções, os núcleos foram encontrados nas buscas 1 e 3, evidenciando que não houve mudança no resultado de cada execução.

A Tabela 4.3 mostra quantos cenários implícitos foram encontrados de cada família

Tabela 4.2: Número da iteração em que a Família foi registrada pela primeira vez no sistema de caldeira

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
$on \rightarrow query$	1	1	1	1	1	1	1	1	1	1
$command \rightarrow stop$	3	3	3	3	3	3	3	3	3	3

Tabela 4.3: Número de cenários implícitos encontrados por família por execução no sistema de caldeira

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
$on \rightarrow query$	62	62	62	62	62	62	62	62	62	62
$command \rightarrow stop$	38	38	38	38	38	38	38	38	38	38

em cada uma das 10 execuções. Vemos que não houve mudança: em todas as execuções, exatamente os mesmos resultados foram encontrados. Para um sistema simples como o sistema de caldeiras, não foi observada mudança nos resultados de diferentes execuções.

4.2 Análise do Sistema *Cruiser*

O sistema *cruiser control* é um sistema de piloto automático que utilizamos como caso de teste da nossa solução. Ele é um sistema consideravelmente maior e mais complexo que o sistema de caldeira, com mais componentes e mais mensagens nos cenários, o que nos permite fazer uma comparação para avaliar a escalabilidade da solução proposta. Por exemplo, como esperado, a execução foi mais lenta no sistema *cruiser* do que no sistema de caldeira, porém não de maneira tão significativa a ponto de inviabilizar a análise.

O sistema *cruiser* funciona como um piloto automático de carro, em que mantém a velocidade do carro sempre a mesma, verificando a velocidade atual e a ajustando para o valor desejado com o regulador de ingestão (throttle) de combustível no motor do carro. Este sistema é descrito e detalhado em [10].

Esse sistema é controlado por três botões: *resume*, *on* e off. Quando o motor está funcionando e *on* é pressionado, o sistema registra a atual velocidade e mantém o carro nesta velocidade. Quando o acelerador, o freio ou off é pressionado, o sistema é desativado, mas mantém guardado o valor registrado da velocidade. Se *resume* é pressionado, o sistema acelera ou desacelera o carro até que retorne à velocidade registrada anteriormente.

As Figuras 4.5 e 4.6 mostram respectivamente os bMSCs e o hMSC do sistema *cruiser*. Cada bMSC indica os comportamentos esperados do sistema, ligando o motor, realizando operações de detecção de velocidade, regulagem, etc.

4.2.1 Passo a Passo da Execução

Modelagem do Sistema em Cenários

Os cenários foram modelados na ferramenta LTSA-MSC, como representado na Figura 4.5.

Iniciando as Repetições: Identificação de um Cenário Implícito

Começa a primeira busca por cenários implícitos no sistema *cruiser*. O objetivo desse passo é avaliar os cenários implícitos produzidos pela ferramenta a cada iteração, para posteriormente classificá-los nas devidas famílias de cenários implícitos, ou criar uma nova família para este cenário implícito.

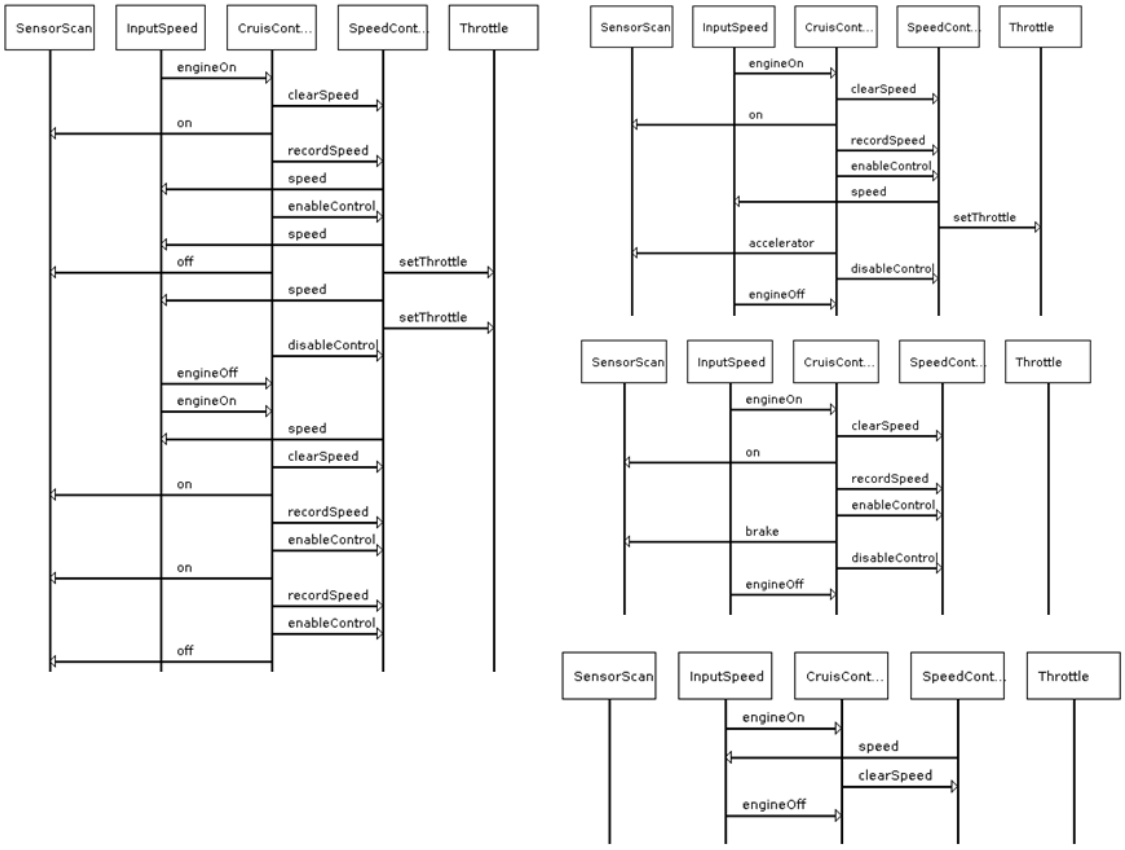

Figura 4.5: bMSCs do sistema *cruiser*. A esquerda: $Scen1$, a direita de cima a baixo: $Scen2$, $Scen3$ e $Scen4$

O primeiro cenário implícito detectado pela ferramenta tem a seguinte da sequência de mensagens que o compõe:

1. engineOn
2. clearSpeed
3. engineOff

Tratando a Família do Cenário Implícito Encontrado

Coletamos então as duas últimas mensagens desta sequência para definir o núcleo deste cenário implícito, ou seja o núcleo deste cenário implícito é:

1. clearSpeed
2. engineOff

Com o núcleo do cenário implícito identificado (Passo 3), passamos para a próxima etapa, que é buscar nas famílias já encontradas se existe alguma família com o mesmo núcleo. Como este é o primeiro cenário implícito encontrado, ainda não foi registrada nenhuma família de cenários implícitos, então é criada uma família com este núcleo e indicada com 1 cenário implícito encontrado pertencente a ela.

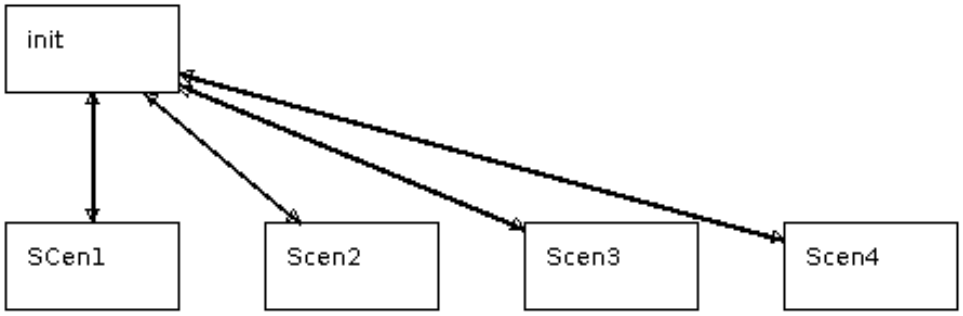

Figura 4.6: hMSC do sistema *cruiser*

Geração do bMSC do Cenário Implícito

Como este cenário implícito é de uma família nova, ele será exposto para o usuário, representando esta família. Essa representação é feita em bMSC, exatamente como na Figura 4.7. Futuros cenários implícitos que pertençam a esta mesma família não serão evidenciados para o usuário, já que este cenário implícito já os representa.

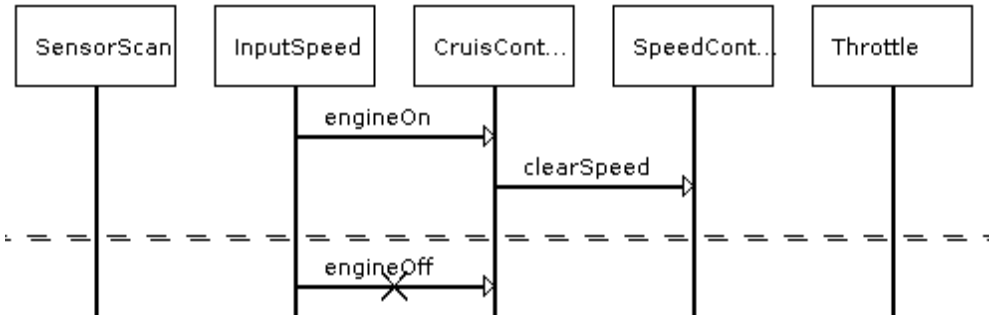

Figura 4.7: bMSC do primeiro cenário implícito encontrado no sistema *cruiser*

Em futuras repetições, cenários implícitos encontrados que pertençam a esta mesma família não serão evidenciados desta maneira, já que este cenário implícito já os representa.

Adequação da modelagem

Além de expor o cenário implícito para o usuário, é necessário atualizar o modelo do sistema com este cenário, para que as próximas buscas não encontrem este mesmo cenário implícito novamente. Esta etapa é realizada em todas as iterações de busca, mesmo em cenários implícitos de famílias já encontradas, pois estes também poderiam ser encontrados novamente pela ferramenta se esta etapa não fosse executada.

Depois da adequação, é verificado se o número de repetições máximo já foi alcançado, esta é a primeira iteração e definimos como 100 repetições, portanto a execução do programa continua.

Segunda Iteração

Voltamos à etapa de Identificação de um Cenário Implícito, porém agora com o modelo do sistema atualizado para considerar o cenário da Figura 4.7, portanto este cenário não

será mais encontrado. Agora, também temos registrada a família do cenário implícito encontrado na primeira iteração.

A segunda busca também encontra um cenário implícito. Caso não tivesse encontrado um cenário implícito, avançaríamos para a etapa final para gerar o relatório e finalizaria a execução do programa.

O cenário implícito encontrado é composto pelas seguintes mensagens:

1. engineOn
2. speed
3. clearSpeed
4. on

Núcleo do cenário implícito encontrado:

1. clearSpeed
2. on

Seguindo para a etapa de buscar este núcleo nas famílias já existentes, vemos que este é um núcleo novo, pois a única família já registrada tem o núcleo:

1. clearSpeed
2. engineOff

Apesar da primeira mensagem do núcleo ser a mesma, a segunda é diferente, portanto são núcleos diferentes. Assim, é criada uma nova família de cenários implícitos que inclui esse cenário.

Como este cenário implícito é de uma família nova, ele será exposto para o usuário, representando esta família. Da mesma maneira que na primeira família, essa representação é feita em bMSC. Futuros cenários implícitos que pertençam a esta mesma família não serão evidenciados para o usuário, já que este cenário implícito já os representa.

Como em todas as iterações, adequamos a modelagem do sistema para levar em consideração este cenário implícito e com isso se conclui mais uma iteração.

Iterações Subsequentes e Geração do Relatório

Na iteração 19, a última família é registrada. A partir da vigésima iteração, todos os cenários implícitos encontrados pertencem a alguma família já registrada. Isso se repete até que se completem as 100 iterações previstas, já que todas as buscas sempre encontram um novo cenário implícito.

Ao se concluir as buscas, temos finalmente todos os dados coletados e iremos gerar o relatório final, temos também os quinze bMSCs gerados referentes a cada família.

4.2.2 Análise do Resultado

A execução para o sistema *cruiser* encontrou quinze famílias de cenários implícitos que tiveram os bMSCs do primeiro cenário encontrado de cada família gerados para visualização da representação de cada família.

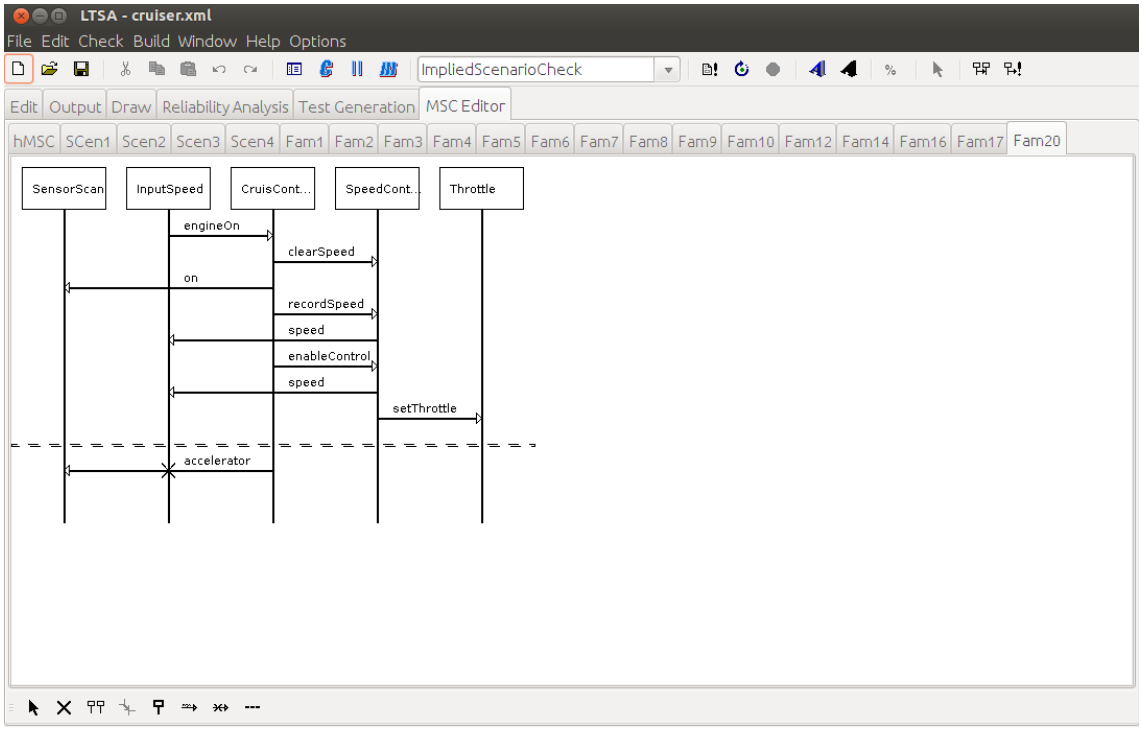

Figura 4.8: Exemplo de saída da ferramenta LTSA após a execução da solução para o sistema *cruiser*

Essas famílias foram identificadas nas buscas 1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 12, 13, 14, 15 e 19. Ou seja: a partir da vigésima busca e até a centésima (última), todos os cenários implícitos encontrados pertenciam a essas mesmas famílias. Este é um resultado interessante, pois mesmo para um sistema com muitas famílias, todas elas foram encontradas relativamente rápido, no começo da execução. Nas 20 primeiras buscas, 15 famílias foram encontradas e apenas 5 cenários implícitos encontrados pertenciam a famílias já encontradas (os cenários das buscas 8, 16, 17, 18 e 20).

A Figura 4.8 mostra com um exemplo de saída (diferente do relatório abaixo) como fica evidenciado na ferramenta LTSA o resultado após a execução para o sistema *cruiser*. As famílias encontradas estão identificadas como $FamX$, sendo X o número da busca que encontrou o primeiro cenário implícito daquela família. O bMSC de cada família é o bMSC do primeiro cenário implícito encontrado dessas famílias. Observamos que apesar de terem ocorrido 100 buscas, apenas 15 bMSCs são mostrados, evidenciando como a análise do resultado é simplificada para o usuário.

Segue abaixo o relatório final gerado para o sistema *cruiser*:

```
100 buscas por cenários implícitos realizadas.
15 famílias encontradas.
85 cenários encontrados de famílias já existentes.
Tempo decorrido: 486,565 segundos.
 Obs.:Mais cenários implícitos possivelmente ainda podem ser encontrados.

Relatório das famílias com a quantidade de cenários implícitos encontrados:
Núcleo da Família:
```

```
 clearSpeed -> engineOff
 Quantidade: 10

Núcleo da Família:
 clearSpeed -> on
 Quantidade: 8

Núcleo da Família:
 clearSpeed -> speed
 Quantidade: 8

Núcleo da Família:
 enableControl -> off
 Quantidade: 5

Núcleo da Família:
 brake -> speed
 Quantidade: 5

Núcleo da Família:
 accelerator -> disableControl
 Quantidade: 5

Núcleo da Família:
 speed -> brake
 Quantidade: 11

Núcleo da Família:
 speed -> off
 Quantidade: 5

Núcleo da Família:
 enableControl -> accelerator
 Quantidade: 5

Núcleo da Família:
 enableControl -> brake
 Quantidade: 5

Núcleo da Família:
 disableControl -> speed
 Quantidade: 7

Núcleo da Família:
 setThrottle -> brake
 Quantidade: 10
```

```
Núcleo da Família:
 setThrottle -> off
 Quantidade: 4

Núcleo da Família:
 speed -> accelerator
 Quantidade: 6

Núcleo da Família:
 setThrottle -> accelerator
 Quantidade: 6
```

Como já mencionado, nas 100 buscas realizadas, 15 famílias foram encontradas e as outras 85 buscas encontraram cenários implícitos pertencentes a alguma dessas famílias.

O tempo de execução foi de pouco mais de 8 minutos. Como esperado, mais demorado que o sistema de caldeira, por ser mais complexo. Como mencionado na Seção 3.1.1, no final de cada busca a modelagem do sistema é atualizada para que o cenário implícito encontrado não seja encontrado novamente, por isso, cada busca é mais demorada que a anterior.

O relatório também indica que mais cenários implícitos ainda podem ser encontrados. Isso porque a todas as buscas realizadas encontraram cenários implícitos e a execução só parou porque o limite de buscas pré-definido foi alcançado. Isso sugere que mais famílias poderiam ser encontradas, mas considerando que a última família encontrada foi na busca 19 e as 81 buscas seguintes não encontraram famílias novas, este caso parece ser muito improvável.

O relatório então descreve as 15 famílias encontradas, indicando seus núcleos e quantos cenários implícitos foram encontrados pertencentes a cada uma delas.

Observando o relatório detalhadamente, podemos notar que existem duas famílias de núcleos similares, com mesmas mensagens, porém ordem invertida. São os núcleos $brake \rightarrow speed$ e $speed \rightarrow brake$. Ao analisar os cenários implícitos que representam essas famílias percebemos que são iguais, com exceção do núcleo. Os cenários são:

1. engineOn
2. clearSpeed
3. on
4. recordSpeed
5. enableControl
6. brake
7. speed

E

1. engineOn
2. clearSpeed
3. on
4. recordSpeed

5. enableControl
6. speed
7. brake

Podemos notar que para este caso seria mais apropriado unificar essas duas famílias. Porém, não consideramos nesta implementação a possibilidade de reordenar o núcleo e, por isso, esta implementação não foi capaz de agrupar essas duas famílias em uma só, apesar de este ser o resultado ideal. O agrupamento em mesma família para casos de reordenação de núcleo não é uma mudança significativa na lógica do programa, porém optamos por manter a implementação desta maneira, já que não conseguimos prever se esta reordenação poderia causar prejuízos, agrupando em famílias cenários implícitos diferenciados e assim omitindo resultados potencialmente relevantes. Na Seção , comentamos a possibilidade de reordenar o núcleo para possível identificação alternativa de famílias.

A família que teve menos cenários implícitos encontrados ainda teve 4 cenários implícitos encontrados pertencentes a ela. Enquanto a família que mais teve cenários implícitos encontrados teve apenas 11 cenários implícitos pertencentes a ela. Considerando que foram encontradas 15 famílias em 100 buscas, essa diferença revela que existe pelo menos parcialmente certa distribuição de famílias encontradas durante a execução. Isso corrobora a possibilidade de que mesmo aumentando muito a quantidade de buscas, nenhuma nova família seria encontrada.

A execução para o sistema *cruiser* encontrou e expôs com sucesso os quinze cenários implícitos relevantes, omitindo todos os outros que foram considerados redundantes. Para o usuário da ferramenta de análise de confiabilidade, caberia agora identificar quais desses cenários implícitos são negativos e quais são positivos, para adaptar a arquitetura evitando-os ou contemplando-os. Após esta adaptação, seria necessário ser realizada uma nova execução da ferramenta para validar a nova arquitetura.

Análise do Tempo de Execução

A título de observação, registramos também o tempo decorrido para cada 10 buscas. Ou seja: o tempo de execução para a realização das buscas de 1 a 10, tempo das buscas de 11 a 20, etc. Como indicado na Figura 4.9, as buscas se tornam cada vez mais lentas com o passar da execução, isso ocorre porque a cada atualização da modelagem de cenários, esta modelagem se torna maior e por isso é mais demorado o processamento nela. Isso evidencia como é importante a discussão já mencionada na Seção 3.1.1 da condição de parada da execução. Para os casos de teste usados, 100 buscas se mostrou um valor desnecessário, já que no primeiro quinto da execução, todas as famílias foram identificadas.

4.2.3 Análise de Diferentes Execuções

Como já mencionado, diferentes execuções podem levar a resultados diferentes, por isso, para ter segurança de que os resultados encontrados não foram atípicos ou diferenciados, realizamos a execução da solução 10 vezes para cada caso. As execuções estão enumeradas de #1 a #10, sendo a execução #10 a que foi analisada detalhadamente acima.

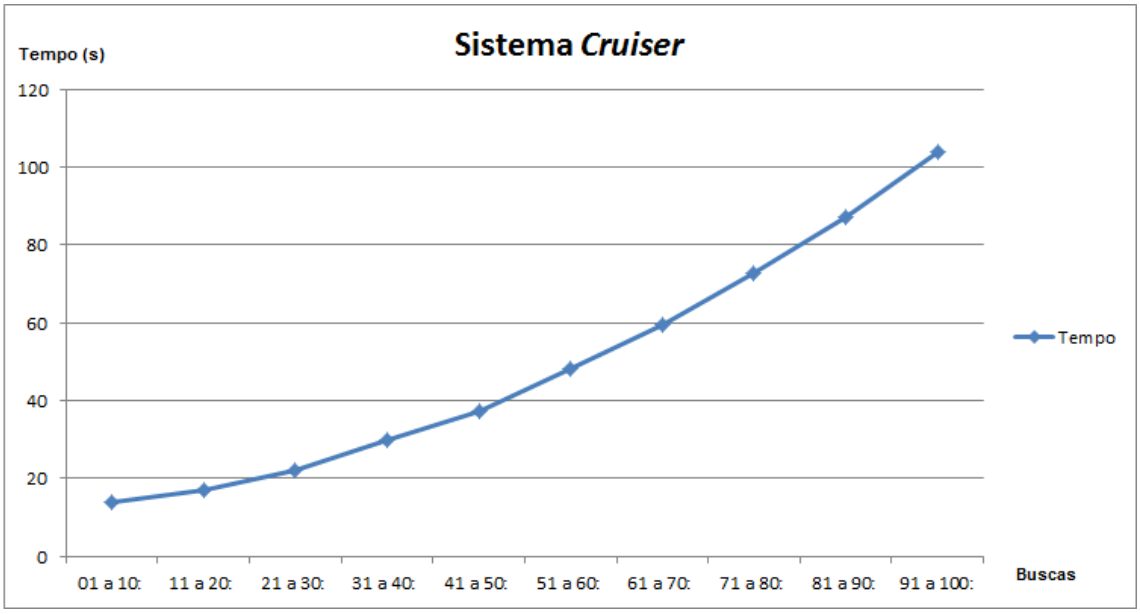

Figura 4.9: Tempo de execução em segundos de cada 10 buscas, das primeiras até as últimas, no sistema *cruiser*

Tabela 4.4: Tempo em segundos decorrido nas 10 execuções do sistema *cruiser*

#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
446,62	490,57	485,07	487,91	486,83	486,51	487,4	487,63	502,94	486,56

A Tabela 4.4 revela que mesmo com uma certa variação no tempo decorrido para cada execução, essa variação não foi tão significativa.

A Tabela 4.5 mostra em que iteração foi encontrado o primeiro cenário implícito de cada família. Ou seja: a iteração em que aquele núcleo foi encontrado pela primeira vez, sendo feito o registro da nova família. Nesta tabela, vemos que apesar de em algumas execuções a ordem ocorrer diferente do que em outras execuções, a última família encontrada sempre foi na execução #19 ou #20, tornando todos os resultados, na prática, muito semelhantes. Vale mencionar também que a ordem em que as famílias são encontradas não é muito relevante, contanto que todas as famílias sejam encontradas, que foi o caso. Portanto, apesar de clara a diferença nas execuções, o resultado final não teve grandes variações.

A Tabela 4.6 mostra quantos cenários implícitos foram encontrados de cada família em cada uma das 10 execuções. Vemos que a diferença foi muito pequena, revelando um resultado coeso.

Tabela 4.5: Número da iteração em que a Família foi registrada pela primeira vez no sistema *cruiser*

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
$clearSpeed \rightarrow engineOff$	1	1	1	1	1	1	1	1	1	1
$clearSpeed \rightarrow on$	2	2	2	2	2	2	2	2	2	2
$clearSpeed \rightarrow speed$	3	3	3	3	3	3	3	3	3	3
$enableControl \rightarrow off$	4	4	4	4	4	4	4	4	4	4
$accelerator \rightarrow disableControl$	9	8	6	5	5	8	9	8	5	6
$brake \rightarrow speed$	8	6	9	6	6	7	5	9	7	5
$speed \rightarrow off$	11	11	11	8	8	11	11	10	8	9
$speed \rightarrow brake$	10	9	10	7	9	10	10	11	9	7
$enableControl \rightarrow brake$	7	7	5	10	10	5	8	5	10	11
$enableControl \rightarrow accelerator$	5	5	7	9	7	6	7	6	11	10
$disableControl \rightarrow speed$	13	13	15	12	13	12	12	12	13	12
$setThrottle \rightarrow off$	15	17	17	14	14	16	14	14	14	14
$setThrottle \rightarrow brake$	14	16	16	18	16	15	16	15	15	13
$speed \rightarrow accelerator$	16	12	13	15	15	14	15	16	16	15
$setThrottle \rightarrow accelerator$	19	20	19	20	20	20	19	19	19	19

Tabela 4.6: Número de cenários implícitos encontrados por família por execução no sistema *cruiser*

	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10
$clearSpeed \rightarrow engineOff$	10	10	10	10	10	10	10	10	10	10
$clearSpeed \rightarrow on$	8	8	8	8	8	8	8	8	8	8
$clearSpeed \rightarrow speed$	8	8	8	8	8	8	8	8	8	8
$enableControl \rightarrow off$	6	5	5	5	5	5	5	5	6	5
$accelerator \rightarrow disableControl$	5	5	5	5	5	5	5	5	5	5
$brake \rightarrow speed$	5	5	5	5	5	5	5	5	5	5
$speed \rightarrow off$	5	5	5	5	5	5	5	5	5	5
$speed \rightarrow brake$	11	11	11	11	11	11	11	11	11	11
$enableControl \rightarrow brake$	5	5	5	5	5	5	5	5	5	5
$enableControl \rightarrow accelerator$	5	5	5	5	5	5	5	5	5	5
$disableControl \rightarrow speed$	6	7	7	7	7	6	7	7	6	7
$setThrottle \rightarrow off$	4	4	4	4	4	4	4	4	4	4
$setThrottle \rightarrow brake$	10	10	10	10	10	10	10	10	10	10
$speed \rightarrow accelerator$	6	6	6	6	6	7	6	6	6	6
$setThrottle \rightarrow accelerator$	6	6	6	6	6	6	6	6	6	6

Capítulo 5

Conclusão e Trabalhos Futuros

5.1 Conclusão

O objetivo deste trabalho foi aprimorar o processo de detecção de cenários implícitos na ferramenta LTSA-MSC. A análise realizada antes da implementação deste trabalho era onerosa, além de expor resultados redundantes. Isso tornava o trabalho do usuário demorado e muito passível de erro, uma vez que o usuário precisaria realizar a detecção manual de cada cenário implícito, sem um prévio diagnóstico de correlação entre os cenários identificados.

Para realizar este aprimoramento, a análise foi adaptada e automatizada para que diversos cenários implícitos sejam encontrados em uma única execução. Além disso, foi feita a identificação dos cenários implícitos de mesma família, removendo assim a redundância dos resultados, porém mantendo sua relevância para a análise de confiabilidade. Dessa maneira, o usuário pode analisar diversos cenários implícitos encontrados de uma só vez, agilizando seu trabalho e tornando este mais preciso. Todo o processo é feito automaticamente, cabendo ao usuário apenas carregar o arquivo, iniciar a execução e analisar a saída final e o relatório gerado.

A solução foi desenvolvida adaptando o plugin MSC já existente da ferramenta LTSA. Foi necessário um estudo detalhado do funcionamento do plugin e da ferramenta para usufruir de suas funcionalidades e utilizá-las corretamente. Foi automatizada a repetição de buscas por cenários implícitos; foi implementado o conceito de família de cenário implícito e feita a análise de cada cenário implícito encontrado para agrupá-los corretamente; também foi feita a geração de um relatório detalhado que informa todos os dados obtidos na execução da solução.

A análise dos casos de teste feitos com os sistemas de caldeira e *cruiser* revelaram o funcionamento esperado de indicar os cenários implícitos agrupados em famílias, de maneira eficiente e precisa, permitindo uma análise de confiabilidade clara, completa, sem redundâncias e rápida. Assim, o objetivo proposto foi alcançado com sucesso, possibilitando melhores análises de confiabilidade a serem realizadas futuramente.

5.2 Trabalhos Futuros

5.2.1 Agrupamentos Alternativos de Cenários Implícitos em Famílias

Neste trabalho, definimos o núcleo como sendo as duas últimas mensagens do cenário implícito e família também depende desta definição. Consideramos essas mensagens como ordenadas, assim como não aceitamos mensagens que ocorressem entre essas duas. Entretanto, essas definições poderiam ser mudadas, levando a agrupamentos diferenciados de cenários implícitos em famílias. Diferentes sistemas podem ter diferentes modos de agrupamentos que sejam mais apropriados, considerando não só o número de mensagens do núcleo, assim como ordenando elas ou reposicionando-as. Para avaliar qual agrupamento é benéfico para a análise dos cenários implícitos, é necessário realizar diversos testes com vários sistemas diferentes, comparando os resultados e analisando quais definições conseguiram agrupar os cenários implícitos de maneira que reduza as redundâncias, mas não omita resultados relevantes.

5.2.2 Adaptar Modo de Visualização

A visualização das famílias, neste trabalho, está limitada ao primeiro cenário implícito encontrado da família, não sendo possível visualizar nem comparar os diferentes cenários implícitos de uma mesma família. Isso se deve a limitações na própria ferramenta LTSA-MSC utilizada, porém, adaptações poderiam ser feitas na interface com o usuário para viabilizar esta funcionalidade. Este tipo de visualização mais completa é interessante para realizar uma análise mais detalhada das famílias e comparar os cenários implícitos agrupados na família, o que também poderia contribuir na decisão de como definir uma família de cenários implícitos.

5.2.3 Aumentar Número de Mensagens do Núcleo

Na Seção 3.2 discutimos nossa motivação de considerar o núcleo do cenário implícito com apenas duas mensagens, ou seja, trantando $N = 2$.

É importante mencionar o caso de sistemas concorrentes em que mensagens podem ter a ordem trocada, pois ocorrem de forma assíncrona. Para este tipo de sistema, aumentar o valor de N e reordenando as mensagens pode levar a uma redução no número de famílias encontradas de maneira benéfica, ou seja: cenários implícitos semelhantes seriam agrupados em uma mesma família, coisa que não aconteceria para N de valor menor.

Isso é possível porque um mesmo defeito pode gerar cenários implícitos diferentes, pois apenas a ordenação das mensagens mudou. Para explicar isso com mais clareza, vamos utilizar o seguinte exemplo:

Digamos que foram encontrados dois cenários implícitos:

1. $message0 \rightarrow message1 \rightarrow message2 \rightarrow messageA$
2. $message0 \rightarrow message2 \rightarrow message1 \rightarrow messageA$

Observe que a diferença entre os dois cenários implícitos é apenas a ordem das mensagens $message1$ e $message2$. Se neste sistema as mensagens message1 e message2 ocorrem de forma assíncrona, então os dois cenários implícitos na verdade evidenciam um mesmo defeito, já que a diferença entre eles não é relevante.

Neste caso, se $N = 2$ então os núcleos de cada cenário encontrado seriam $message2 \rightarrow messageA$ e $message1 \rightarrow messageA$. Dois núcleos diferentes e, sendo assim, duas famílias diferentes, que não teríamos como saber que se referem ao mesmo defeito.

Porem, se definirmos $N = 3$, os núcleos de cada cenário encontrado seriam $message1 \rightarrow message2 \rightarrow messageA$ e $message2 \rightarrow message1 \rightarrow messageA$. Neste caso, poderíamos reordenar as mensagens anteriores à última e unificar as duas famílias, tratando como uma única família com núcleo que pode ser reordenado, mantendo apenas a mensagem final sempre na mesma posição.

Assim, com o cuidado de poder reordenar a parte inicial do núcleo da família, podemos aumentar o valor de N e reduzir o número de famílias encontradas, resultando em uma análise mais precisa dos cenários implícitos.

É preciso ter cuidado nesta abordagem, pois não é qualquer sequência de mensagens de um sistema que é assíncrona, podendo ser reordenada. Realizar esta ordenação e união de famílias quando não é previsto na arquitetura, pode levar ao ocultamento de defeitos do sistema.

Também pode ser difícil definir o valor ideal de N para esta abordagem, pois o mais apropriado pode variar de sistema para sistema, ou de cenário implícito para cenário implícito.

Este trabalho não tomou esta abordagem, simplesmente simplificamos para $N = 2$ sempre, levando a possível redundância de famílias de cenários implícitos.

Uma implementação que levasse isso em consideração poderia obter uma análise ainda mais sucinta, facilitando o entendimento e uso do usuário. Para isso, seria necessário identificar os casos em que esta reordenação se aplica, o que pode ser bem complexo, pois vai depender de cada sistema e cenário. Também seria necessário registrar na família a parte do núcleo reordenável, para que na comparação com futuros núcleos de novos cenários implícitos encontrados esta regra seja aplicada corretamente.

5.2.4 Condição de Parada das Repetições

Como observado na metodologia e análise de resultados deste trabalho, pré-definir o número de buscas por cenários implícitos provavelmente não é a maneira mais eficiente de se obter um bom resultado. É importante que se exista uma definição mais maleável da condição de parada das buscas por cenários implícitos, tal que se adapte automaticamente à complexidade do sistema que está sendo analisado. Realizando mais buscas em sistemas mais complexos e com mais famílias encontradas e menos buscas em sistemas mais simples.

Uma interessante condição de parada para as repetições seria indicar o número máximo de buscas seguidas que não encontrem uma nova família, por exemplo: Caso a execução encontre 50 cenários implícitos seguidos que pertençam a famílias já existentes, interrompa a execução. Esta condição tem a qualidade de que vai se adaptar naturalmente à complexidade dos sistemas, realizando mais buscas em sistemas mais complexos ou com mais famílias.

Essas diferentes condições de parada estão citadas apenas como sugestões para trabalhos futuros já que para a pouca quantidade de casos de teste utilizada neste trabalho não é possível medir qual delas seria mais eficaz.

Neste trabalho sugerimos que possíveis condições de parada mais eficientes dependeriam do número de famílias já encontradas ou do número de buscas seguidas sem encontrar

novo cenário implícito. Um possível estudo de caso seria realizar diversos testes em vários sistemas grandes e comparar as diversas alternativas de condição de parada que torne o resultado mais eficiente possível, mantendo a corretude.

Referências

[1] Sarah Al-Azzani and Rami Bahsoon. Using implied scenarios in security testing. In *Proceedings of the 2010 ICSE Workshop on Software Engineering for Secure Systems*, SESS '10, pages 15–21, New York, NY, USA, 2010. ACM. 4

[2] Rajeev Alur, Kousha Etessami, and Mihalis Yannakakis. Inference of message sequence charts. In *Proceedings of the 22Nd International Conference on Software Engineering*, ICSE '00, pages 304–313, New York, NY, USA, 2000. ACM. 1, 9, 11

[3] Algirdas Avizienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Secur. Comput.*, 1(1):11–33, January 2004. 2, 6, 7, 8

[4] A. Bertolino, E. Marchetti, and H. Muccini. Introducing a reasonably complete and coherent approach for model-based testing. *Electron. Notes Theor. Comput. Sci.*, 116:85–97, January 2005. 3

[5] S. D. Brookes, C. A. R. Hoare, and A. W. Roscoe. A theory of communicating sequential processes. *J. ACM*, 31(3):560–599, June 1984. 5

[6] Edsger W. Dijkstra. Structured programming. chapter Chapter I: Notes on Structured Programming, pages 1–82. Academic Press Ltd., London, UK, UK, 1972.

[7] Leah Hoffman. In search of dependable design. *Commun. ACM*, 51(7):14–16, July 2008. 8

[8] J.C. C. Laprie, A. Avizienis, and H. Kopetz, editors. *Dependability: Basic Concepts and Terminology*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1992. 6, 7

[9] Michael R. Lyu. Software reliability engineering: A roadmap. In *2007 Future of Software Engineering*, FOSE '07, pages 153–170, Washington, DC, USA, 2007. IEEE Computer Society. 1

[10] Jeff Magee and Jeff Kramer. *Concurrency: State Models &Amp; Java Programs*. John Wiley & Sons, Inc., New York, NY, USA, 1999. 9, 31

[11] Glenford J. Myers and Corey Sandler. *The Art of Software Testing*. John Wiley & Sons, 2004. 11

[12] Sidney Nogueira, Augusto Sampaio, and Alexandre Mota. Guided test generation from CSP models. In *Theoretical Aspects of Computing - ICTAC 2008, 5th International Colloquium, Istanbul, Turkey, September 1-3, 2008. Proceedings*, pages 258–273, 2008. 4, 9

[13] Thiago Peixoto dos Reis. Uma abordagem dirigida a modelo para a geração de casos de teste baseada na detecção de cenários implícitos. 2015. 2, 5, 10, 15, 22, 29

[14] Genaína Nunes Rodrigues, Vander Alves, Renato Silveira, and Luiz A. Laranjeira. Dependability analysis in the ambient assisted living domain: An exploratory case study. *J. Syst. Softw.*, 85(1):112–131, January 2012. 8

[15] Ian. Sommerville, Selma Shin Shimizu Melnikoff, Reginaldo Arakaki, Edilson de Andrade Barbosa, and Kechi Hirama. *Engenharia de software*. Pearson Prentice Hall, São Paulo, 2008. 7, 21

[16] Felipe Cantal de Sousa, Nabor C. Mendonca, Sebastian Uchitel, and Jeff Kramer. Detecting implied scenarios from execution traces. In *Proceedings of the 14th Working Conference on Reverse Engineering*, WCRE '07, pages 50–59, Washington, DC, USA, 2007. IEEE Computer Society. 4, 9

[17] Jan Tretmans. Formal methods and testing. chapter Model Based Testing with Labelled Transition Systems, pages 1–38. Springer-Verlag, Berlin, Heidelberg, 2008. 1

[18] Sebastian Uchitel, Robert Chatley, Jeff Kramer, and Jeff Magee. Ltsa-msc: Tool support for behaviour model elaboration using implied scenarios. In *Ninth International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS) at the European Joint Conferences on Theory and Practice of Software (ETAPS),*, 2003. 2, 10, 11, 15, 21

[19] Sebastian Uchitel, Jeff Kramer, and Jeff Magee. Detecting implied scenarios in message sequence chart specifications. *SIGSOFT Softw. Eng. Notes*, 26(5):74–82, September 2001. vii, 1, 2, 9, 11, 12, 13, 23, 29

[20] Sebastian Uchitel, Jeff Kramer, and Jeff Magee. Negative scenarios for implied scenario elicitation. *SIGSOFT Softw. Eng. Notes*, 27(6):109–118, November 2002. 9, 10

[21] Sebastián Uchitel, Jeff Kramer, and Jeff Magee. Synthesis of behavioral models from scenarios. *IEEE Trans. Software Eng.*, 29(2):99–115, 2003. vii, 1, 9, 10

[22] Sebastian Uchitel, Jeff Kramer, and Jeff Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Trans. Softw. Eng. Methodol.*, 13(1):37–85, January 2004. 1, 4, 11, 22, 29

[23] A. Vaz Roriz, G. Nunes Rodrigues, and Laranjeira L.A. Analysis of the impact of implied scenarios on the reliability of computational concurrent systems. 2014. 2, 18