Lincoln Souza Rocha

CAEH✓: Um Método para Verificação de Modelos do Tratamento de Exceção Sensível ao Contexto em Sistemas Ubíquos

Fortaleza – Ceará – Brasil

2013

Lincoln Souza Rocha

CAEH✓: Um Método para Verificação de Modelos do Tratamento de Exceção Sensível ao Contexto em Sistemas Ubíquos

Tese submetida a coordenação do programa de Mestrado e Doutorado em Ciência da Computação, da Universidade Federal do Ceará como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação.

Orientadora: Rossana M. C. Andrade, PhD.

UNIVERSIDADE FEDERAL DO CEARÁ
CENTRO DE CIÊNCIAS
DEPARTAMENTO DE COMPUTAÇÃO

Fortaleza – Ceará – Brasil

2013

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca de Ciências e Tecnologia

R574c Rochá, Lincoln Souza.

CAEHV: um método para verificação de modelos do tratamento de exceção sensível ao contexto em sistemas ubíquos / Lincoln Souza Rocha. – 2013.

100f. : il. , enc. ; 30 cm.

Tese (doutorado) – Universidade Federal do Ceará, Centro de Ciências, Departamento de Computação, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2013.

Área de Concentração: Ciência da Computação.

Orientação: Profa. PhD. Rossana Maria de Castro Andrade.

1. Software de sistema. 2. Análise de sistema. 3. Software de aplicação. I. Título.

CDD 005

Tese de Doutorado sob o título *"CAEH✓: Um Método para Verificação de Modelos do Tratamento de Exceção Sensível ao Contexto em Sistemas Ubíquos"* defendida por Lincoln Souza Rocha no programa de Mestrado e Doutorado em Ciência da Computação, da Universidade Federal do Ceará como requisito parcial para a obtenção do grau de Doutor em Ciência da Computação e aprovada em 08 de março de 2013, em Fortaleza, Ceará, pela banca examinadora constituída pelos doutores:

Rossana Maria de Castro Andrade, PhD (Orientadora)
Universidade Federal do Ceará

Alessandro Fabricio Garcia, DSc
Pontifícia Universidade Católica do Rio de Janeiro

Cláudia Maria Lima Werner, DSc
Universidade Federal do Rio de Janeiro

Francisco Heron de Carvalho Junior, DSc
Universidade Federal do Ceará

José Antonio Fernandes de Macêdo, DSc
Universidade Federal do Ceará

Aos meus pais Luiz e Tereza e
a minha amada esposa Pollyana.

Agradecimentos

Agradecer é difícil, pois sempre corremos o risco de esquecermo-nos de alguém. Por esse motivo, vou começar agradecendo a todos os que me ajudaram durante estes anos de doutorado, deste modo, aqueles que não forem citados diretamente, sintam-se lembrados.

Em primeiro lugar quero agradecer a Deus por me conceder o dom da inquietude que me desafia e conduz pelos caminhos desconhecidos do conhecimento. Agradeço aos meus pais, Luiz e Tereza, pelos exemplos de carinho, respeito e companheirismo que só o verdadeiro amor é capaz de proporcionar. Às minhas irmãs, Karina e Marina, e ao meu sobrinho, Luiz Eutimio, pelo incentivo constante e pelas poucas, porém prazerosas, horas em família que tivemos ao longo destes anos. À minha amada esposa, Pollyana, pela prova de paciência, renuncia, confiança e amor que me dedicou durante todo este tempo.

De maneira especial, agradeço a professora Rossana, minha orientadora, pelas horas dedicadas às discussões e revisões deste trabalho, por me conduzir nos primeiros passos como pesquisador desde o mestrado, pelo ombro amigo nos momentos difíceis e pelo exemplo de foco, responsabilidade e perseverança. Ao professor Alessandro pela acolhida no período em que passei na PUC-Rio, pelo entusiasmo contagiante e pela sua disponibilidade para as discussões que muito contribuíram para o desfecho desta tese. Aos professores Heron e José Antônio pelas importantes contribuições não só para a realização deste trabalho, mas também para minha formação acadêmica como um todo.

Aos meus colegas do campus da UFC em Quixadá pelo apoio e companheirismo nos momentos mais turbulentos dessa jornada, muito obrigado. A todos os professores e servidores do MDCC e do GREat que, gentilmente, me proporcionaram momentos de discussão, reflexão, alegria e descontração.

Por fim, agradeço a CNPq pelo fomento, imprescindível a realização desta tese.

"Elegance is not a dispensable luxury but a quality that decides between success and failure."

— Edsger W. Dijkstra

Resumo

Os sistemas de software adaptativos sensíveis ao contexto, também conhecidos como auto-adaptativos, representam uma classe particular de sistemas de software complexos. Esse tipo de sistema de software tem como requisito básico observar o ambiente (físico e lógico) em que executa e reagir de forma apropriada às mudanças, quer seja adaptando sua estrutura e comportamento ou executando tarefas de forma automática. Nos ambientes ubíquos, o software adaptativo sensível ao contexto assume um papel fundamental, sendo projetado para interagir com o ambiente físico com intuito de auxiliar as pessoas, de forma transparente, na execução das suas atividades cotidianas, tornando a interação humano-computador mais natural. Devido ao seu amplo domínio de aplicação (e.g., casas inteligentes, guias móveis de visitação, jogos, saúde e missões de resgate), os sistemas de software adaptativo sensível ao contexto precisam ser confiáveis, para cumprir com a sua função e lidar com situações anormais, evitando eventuais falhas. Nesse sentido, o tratamento de exceção sensível ao contexto vem sendo empregado na melhoria dos níveis de robustez e confiabilidade desse tipo de sistema de software. Entretanto, o projeto e implementação do tratamento de exceção sensível ao contexto exige dos projetistas insights e expertise de domínio, sendo uma atividade complexa e propensa a erros. Desse modo, para que o tratamento de exceção sensível ao contexto atinja os objetivos esperados, é necessário que ele seja projetado de forma rigorosa, buscando eliminar ao máximo o número de faltas de projeto (*design faults*). Nesse cenário, esta tese de doutorado propõe um método para verificação de modelos do tratamento de exceção sensível ao contexto, denominado CAEH✓ (*Context-Aware Exception Handling Verification*). O CAEH✓ provê um conjunto de abstrações que permitem aos projetistas modelarem o comportamento excepcional sensível contexto e mapeá-lo para uma estrutura de Kripke. Além disso, um conjunto de propriedades comportamentais é estabelecido com o intuito de auxiliar os projetistas no processo de identificação de determinados tipos de faltas de projeto. Por fim, com o objetivo de avaliar a viabilidade do método, uma ferramenta para a modelagem e verificação do comportamento excepcional sensível ao contexto foi desenvolvida e cenários de injeção de faltas (*fault injection*) foram modelados para analisar a sua efetividade na identificação de faltas de projeto no modelo.

Abstract

The context-aware adaptive software, also known as self-adaptive software, represents a specific class of complex software systems, which has as primary requirement the ability to monitor its execution environment (physical and logical) and reacts upon change, either adapting its internal structure and behavior or performing automatic tasks. In ubiquitous environments, context-aware adaptive software plays a fundamental role, as it must be designed to interact with the physical environment, aiming to transparently supporting users in their daily activities, improving or hiding the interaction between human and computer. In that direction, the application domain in which context-aware adaptive software may be applied is broad (e.g., smart home, mobile visit guide, gaming, health or rescue missions), requiring a high-degree of reliability, not only capable of performing its functional requirements, but also dealing with abnormal conditions, avoiding occasional failures. Although context-aware exception handling has been used to improve robustness and reliability levels of context-aware adaptive software, it requires from the software designers a good understanding of the application domain, turning out to be a complex and error-prone activity. In order to achieve the expected results of using context-aware exception handling, it is mandatory to follow a rigorous design approach, trying to minimize the number of design faults. In this scenario, this doctoral thesis proposes a method for model checking context-aware exception handling, named CAEH✓ (*Context-Aware Exception Handling Verification*). It provides a set of abstractions that permits designers to model the context-aware exceptional behavior and translates it to a Kripke structure. Additionally, a set of behavior properties is established, attempting to aid designers to identify specific types of design faults. To evaluate the feasibility of the proposed method, a tool, called JCAEH✓, for modeling and verification of context-aware exceptional behavior is developed, and fault injection scenarios are modeled to analyze its effectiveness in identifying design faults.

Sumário

Lista de Figuras **x**

Lista de Tabelas **xi**

1 Introdução **1**

1.1 Contextualização e Motivação 1

1.2 Problema, Hipótese e Questões de Pesquisa 5

1.3 Objetivo e Contribuições 7

1.4 *UbiParking*: Sistema Exemplo 8

1.4.1 Funcionalidades 9

1.4.2 Cenários Excepcionais 10

1.5 Organização do Documento 10

2 Fundamentação Teórica **12**

2.1 Software Adaptativo Sensível ao Contexto 12

2.1.1 Definições 13

2.1.2 Propriedades Auto-* 14

2.1.3 Elicitação de Requisitos de Adaptação 16

2.1.4 Modelagem de Dimensões 18

2.1.5 Laço de Adaptação 20

2.2 Tratamento de Exceção 22

2.2.1 Falta, Erro, Falha e Exceção 22

2.2.2 Levantamento, Sinalização e Tratamento 23

2.2.3 Modelos de Tratamento 24
2.3 Verificação de Modelos 25
2.3.1 O Processo de Verificação de Modelos 25
2.3.2 Estruturas de Kripke 26
2.3.3 Lógicas Temporais para Especificação de Propriedades 29
2.4 Programação por Restrições 33
2.4.1 Restrições e suas Propriedades 33
2.4.2 Modelando com Restrições 35
2.5 Considerações Finais 36

3 Trabalhos Relacionados 37

3.1 Tratamento de Exceção Sensível ao Contexto 37
3.1.1 Tipos de Exceções Contextuais 37
3.1.2 O Comportamento Excepcional Sensível ao Contexto 40
3.1.3 Propensão a Faltas de Projeto 43
3.2 Abordagens de Modelagem e Verificação 46
3.2.1 Trabalho de Zhang e Cheng (2006) 46
3.2.2 Trabalho de Cubo et al. (2009) 48
3.2.3 Trabalho de Sama et al. (2010) 49
3.2.4 Trabalho de Liu, Xu e Cheung (2013) 50
3.2.5 Discussão dos Trabalhos 51
3.3 Considerações Finais 53

4 O Método CAEH✓ 54

4.1 Visão Geral do Método 54
4.2 Atividade de Modelagem 55
4.2.1 Abstrações Elementares 56

4.2.2 Estruturando o Comportamento Excepcional 60
4.2.3 Derivando a Estrutura de Kripke 63
4.3 Atividade de Especificação 69
4.3.1 Progresso de Detecção 70
4.3.2 Progresso de Captura 70
4.3.3 Progresso de Tratador 71
4.3.4 Estabilidade de Tratamento 72
4.3.5 Alcançabilidade 72
4.4 Considerações Finais 73

5 Avaliação do CAEH✓ 75

5.1 Introdução 75
5.2 A Ferramenta JCAEH✓ 75
5.3 Cenários de Injeção de Faltas 79
5.3.1 Descrição dos Cenários 80
5.3.2 Ambiente de Execução e Discussão dos Resultados 83
5.4 Considerações Finais 88

6 Conclusões e Trabalhos Futuros 89

6.1 Visão Geral do Trabalho 89
6.2 Resultados Alcançados 90
6.3 Análise da Hipótese de Pesquisa 90
6.4 Trabalhos Futuros 91

Referências Bibliográficas 93

Lista de Figuras

1.1 Planta Baixa do Estacionamento 9

2.1 Laço de Adaptação (adaptado de (SALEHIE; TAHVILDARI, 2009)). 20

2.2 Componente Tolerante a Faltas Ideal (adaptado de Garcia et al. (2001)). . 23

2.3 O Processo de Verificação de Modelos. 26

2.4 Exemplo de Estrutura de Kripke. 27

2.5 Conjunto de Caminhos Infinitos (2.5a) e Árvore de Computação Infinita (2.5b) da Estrutura de Kripke decrita na Figura 2.4. 30

2.6 Operadores Temporais de LTL 31

2.7 Operadores Temporais de CTL 32

5.1 Visão geral do JCAEH✓. 76

Lista de Tabelas

2.1 Sintaxe de LTL. 30

2.2 Intuição para os Operadores Temporais de LTL. 30

2.3 Relação de Satisfação de LTL. 31

2.4 Sintaxe de CTL. 31

2.5 Intuição para os Operadores Temporais de CTL. 32

2.6 Relação de Satisfação de CTL. 33

4.1 Proposições Contextuais do *UbiParking*. 57

5.1 Sumário da Execução da Rodada 1 do Cenário 1 84

5.2 Sumário da Execução da Rodada 2 do Cenário 1 84

5.3 Sumário da Execução da Rodada 1 do Cenário 2 85

5.4 Sumário da Execução da Rodada 2 do Cenário 2 85

5.5 Sumário da Execução da Rodada 3 do Cenário 2 85

5.6 Sumário da Execução da Rodada 1 do Cenário 3 86

5.7 Sumário da Execução da Rodada 2 do Cenário 3 86

5.8 Sumário da Execução da Rodada 3 do Cenário 3 87

5.9 Sumário da Execução da Rodada 1 do Cenário 4 87

5.10 Sumário da Execução da Rodada 2 do Cenário 4 88

5.11 Sumário da Execução da Rodada 3 do Cenário 4 88

1 *Introdução*

Esta tese propõe o CAEH✓[1], um método para verificação de modelos do tratamento de exceção sensível ao contexto em sistemas ubíquos. Na Seção 1.1 deste capítulo são introduzidas a motivação e a caracterização do problema abordado neste trabalho. A definição do problema, a hipótese e as questões de pesquisa que nortearam o desenvolvimento desta tese são declaradas na Seção 1.2. Na Seção 1.3, os objetivos, as metas e as principais contribuições deste trabalho são elencados. Em seguida, na Seção 1.4, é introduzido um sistema exemplo, com cenários de exceções contextuais, que será utilizado ao longo de todo texto desta tese para ajudar na compreensão. A Seção 1.5 finaliza o capítulo apresentado a estrutura organizacional do restante deste documento.

1.1 Contextualização e Motivação

O crescente avanço tecnológico tornou possível a construção de sistemas computacionais cada vez maiores, complexos e onipresentes que colaboram para prover serviços essenciais à sociedade nos mais variados domínios de aplicação (e.g., entretenimento, telecomunicação, financeiro, assistência à saúde e transporte aéreo) (FRANCE; RUMPE, 2007; KNIGTH, 2011). Nesse tipo de sistema computacional, o software assume um papel fundamental devendo ser capaz de: (i) operar em ambientes computacionais distribuídos e embarcados, compostos por diversos tipos de dispositivos computacionais (e.g., computadores pessoais, dispositivos portáteis, sensores especializados e atuadores); (ii) comunicar-se através de uma grande variedade de paradigmas de interação e tecnologias de comunicação; (iii) adaptar-se dinamicamente às mudanças do ambiente (físico ou lógico); e (iv) comportar-se de maneira fidedigna (*dependable*).

Uma classe particular desse tipo de sistema de software complexo são os desenvolvidos segundo os princípios da Computação Ubíqua (WEISER, 1991), onde a computação passa a estar integrada aos objetos (e.g., aparelhos de televisão, condicionadores de ar e dispo-

[1] Acrônimo do inglês ***C**ontext-**A**ware **E**xception **H**andling **V**erification*

sitivos computacionais móveis) e lugares (e.g., casas, escritórios e hospitais) do cotidiano das pessoas, tornando-se onipresente. Nesses ambientes, ditos **ambientes ubíquos**, os objetos (dispositivos) estão conectados através de enlaces de redes sem fio, formando uma rede aberta, discreta, contínua e confiável para a troca de informações, compartilhamento de recursos e prestação de serviços. Na Computação Ubíqua, essa onipresença computacional tem como objetivo a criação de "ambientes inteligentes programáveis" que possam ser monitorados, reconfigurados e controlados dinamicamente por elementos de software. Nessa perspectiva, os sistemas de software desenvolvidos para os ambientes ubíquos, passam a ser projetados para interagir com o ambiente (físico e lógico) com o objetivo de auxiliar as pessoas de forma proativa, transparente e casual na execução das suas atividades cotidianas (e.g., estacionar um veículo, fazer compras ou visitar um museu) (GARLAN et al., 2002; SADRI, 2011; ROCHA et al., 2011).

A sensibilidade ao contexto (DEY, 2001), considerada um dos aspectos centrais no desenvolvimento de sistemas de software para ambientes ubíquos (ABOWD, 1999; SATYANARAYANAN, 2001; KINDBERG; FOX, 2002; COUTAZ et al., 2005; ROCHA et al., 2007; MAIA; ROCHA; ANDRADE, 2009; KULKARNI; TRIPATHI, 2010; LIMA et al., 2011), está relacionada com a habilidade do sistema em perceber o contexto em que está inserido e reagir de forma apropriada, quer seja adaptando sua estrutura e comportamento ou executando tarefas de forma automática (LOKE, 2009). O contexto pode ser entendido como um conjunto de informações sobre o ambiente (físico ou lógico, incluindo os usuários e o próprio sistema), que pode ser usado com o propósito de manter ou melhorar a interação entre o usuário e o sistema (DEY, 2001). Dessa forma, enquanto os usuários, dispositivos computacionais ou elementos de software movem-se pelo ambiente ubíquo (ou permanecem em estado estacionário), o sistema explora o contexto para manter a sua execução de forma correta, estável e otimizada, buscando garantir o atendimento aos seus requisitos (metas) e aos interesses (necessidades e expectativas) dos seus usuários (HUEBSCHER; MCCANN, 2008; SOYLU; CAUSMAECKER; DESMET, 2009). Esse tipo de software é referenciado na literatura como **adaptativo sensível ao contexto** (*context-aware adaptive software*) (SAMA et al., 2010; KULKARNI; TRIPATHI, 2010).

Devido às suas peculiaridades (e.g., acesso à informação e processamento em qualquer lugar e a qualquer momento), o conceito de Computação Ubíqua tem sido explorado em diversos domínios de aplicação, tais como casas (INTILLE, 2002) e cozinhas inteligentes (CHEN et al., 2010), guias móveis de visitação (MARINHO et al., 2010), jogos (MAGERKURTH et al., 2005), saúde (VARSHNEY, 2009), ambientes hospitalares (BARDRAM; CHRISTENSEN, 2007) e missões de resgate (CATARCI et al., 2008).

Nesses domínios de aplicação, o software passa a tomar decisões de forma proativa no lugar das pessoas, buscando proporcionar uma melhor experiência de interação para o usuário. Dessa forma, dependendo do domínio de aplicação, falhas em sistemas de software para ambientes ubíquos podem ter consequências que variam desde leves aborrecimentos aos usuários (e.g., domínio de guias móveis de visitação e jogos) até situações em que suas vidas podem ser postas em risco (e.g., domínio de saúde e missões de resgate) (CHETAN; RANGANATHAN; CAMPBELL, 2005). Por esse motivo, esses sistemas precisam ser confiáveis para cumprir com a sua função e lidar com situações anormais, evitando eventuais falhas (CHETAN; RANGANATHAN; CAMPBELL, 2005; MILNER, 2006; XU et al., 2012).

Segundo Meyer (1997), a **confiabilidade** de um sistema de software pode ser analisada em termos da sua **corretude** e **robustez**. A corretude é a habilidade de um sistema de software executar exatamente as suas tarefas, como descrito em sua **especificação funcional**. Por outro lado, a robustez é a habilidade de um sistema de software reagir de forma apropriada diante de situações anormais. Nessa perspectiva, a robustez complementa a corretude. Enquanto a corretude endereça o comportamento do sistema nos casos cobertos pela sua especificação funcional, a robustez aborda o que acontece fora dessa especificação, sendo denominada de **especificação de robustez** (MEYER, 1997). Contudo, nessa discussão, Meyer (1997) assume que tanto a especificação funcional quanto a especificação de robustez estão livres de faltas[2] de projeto (*design faults*) e, portanto, corretas, existindo apenas a possibilidade de inserção de faltas de software (*software fault*) durante a atividade de codificação. Entretanto, a ocorrência de faltas de projeto são comuns no desenvolvimento de software, em parte devido a falibilidade inerente à natureza humana e ao crescente aumento da complexidade dos sistemas de software. Isso obriga os projetistas a realizarem uma análise rigorosa e adotarem meios apropriados para evitá-las e, se inseridas, encontrá-las e removê-las, evitando que sejam propagadas até a fase de codificação (KNIGTH, 2011).

Na Engenharia de Software, o tratamento de exceção é uma técnica de recuperação de erros por avanço (*forward error recovery*)[3] tipicamente empregada na melhoria da

[2]Este trabalho adota a nomenclatura de Avizienis et al. (2004), a qual diz que uma falta (*fault*) é a causa física ou algorítmica de um erro (*error*), que, se propagado até a interface de serviço do sistema, caracteriza uma falha (*failure*). Essa nomenclatura é explicada mais tarde, no Capítulo 2.

[3]Segundo Campbell e Randell (1986) as principais técnicas utilizadas para a recuperação de erros são: (i) recuperação de erros por retrocesso (*backward error recovery*); e (ii) recuperação de erros por avanço (*forward error recovery*). A primeira técnica atua recuperando o último, e presumidamente correto, estado do sistema antes da detecção do erro, chamado de *checkpoint*. Já a segunda técnica, consiste na condução do sistema a um novo estado consistente (não necessariamente o último estado correto antes da detecção do erro) a partir de medidas apropriadas para corrigir ou isolar a causa do erro ou confiná-lo.

robustez de sistemas de software (GOODENOUGH, 1975; PARNAS; WüRGES, 1976). Uma exceção é um evento que modela uma situação ou estado em que o fluxo normal de execução do sistema não pode continuar (KNUDSEN, 1987; KIENZLE, 2008). Para que o sistema continue executando corretamente, o fluxo de execução deve ser desviado e uma computação adicional deve ser empregada para tratar aquela situação (KNUDSEN, 1987). O mecanismo de tratamento de exceção permite aos desenvolvedores especificarem situações excepcionais e estruturar as atividades de tratamento por meio de elementos chamados de tratadores de exceção (LEE; ANDERSON, 1990; BUHR; MOK, 2000; GARCIA et al., 2001).

O **tratamento de exceção sensível ao contexto** é uma abordagem de tratamento de exceção específica para lidar com situações anormais em ambientes ubíquos, provendo meios para estruturar as atividades de tratamento de erros em sistemas adaptativos sensíveis ao contexto (DAMASCENO et al., 2006; MERCADAL et al., 2010; KULKARNI; TRIPATHI, 2010; CHO; HELAL, 2011). No tratamento de exceção sensível ao contexto, considerado nesse trabalho como parte da especificação de robustez, o contexto e a sensibilidade ao contexto são utilizados pelo mecanismo de tratamento de exceção para definir, detectar e tratar condições excepcionais (anormais) em ambientes ubíquos, chamadas de **exceções contextuais** (DAMASCENO et al., 2006) ou **exceções semânticas** (CHO; HELAL, 2012). Trabalhos existentes na literatura (DAMASCENO et al., 2006; MERCADAL et al., 2010; KULKARNI; TRIPATHI, 2010; CHO; HELAL, 2011) concordam em muitos aspectos sobre como o mecanismo de tratamento de exceção pode se beneficiar do contexto e da sensibilidade ao contexto para detectar exceções contextuais, encontrar tratadores apropriados e executá-los, levando o sistema de volta a um estado seguro (*safe*). Contudo, o projeto do tratamento de exceção sensível ao contexto é uma atividade complexa e propensa a erros (CHO; HELAL, 2012). Por exemplo, faltas de projeto podem ser cometidas por projetistas na especificação do contexto que caracteriza as exceções contextuais ou na definição dos critérios utilizados para selecionar as medidas de tratamento adequadas para um contexto particular.

Um exemplo de exceção contextual é descrito por Damasceno et al. (2006) no cenário de um sistema de aquecimento "inteligente". Nesse sistema, a temperatura do ambiente é ajustada automaticamente de acordo com as preferência dos seus frequentadores. A elevação da temperatura para um patamar acima do limite estabelecido pelas preferências do usuário, pode indicar uma falha no sistema que controla o equipamento de aquecimento ou uma falha no próprio equipamento. Essa situação anormal é detectada de forma indireta, observando as informações de contexto sobre a temperatura do ambiente e as preferências

do usuário. Entretanto, se o contexto que caracteriza essa situação excepcional for mal projetado, a exceção contextual pode nunca ser detectada pelo sistema ou ser detectada em um momento impróprio (i.e., diferente daquele da intensão do projetista). Esse tipo de falta de projeto compromete a robustez e a confiabilidade do sistema, podendo levá-lo a reagir de forma indesejada e colocar em risco a saúde das pessoas.

Nesse cenário, com o intuito de reduzir o número de faltas de projeto no tratamento de exceção, estudos na área evidenciam que a análise formal do fluxo de controle excepcional (i.e., do comportamento excepcional) fornece meios para validar aspectos comportamentais do projeto do tratamento de exceção, melhorando assim a corretude da especificação de robustez e, consequentemente, a confiabilidade do sistema (FILHO; BRITO; RUBIRA, 2006; BRITO et al., 2009). Entretanto, mesmo com essas evidências e a existência de abordagens formais para verificação de corretude do comportamento adaptativo de sistemas adaptativos sensíveis ao contexto (ZHANG; CHENG, 2006; RANGANATHAN; CAMPBELL, 2008; CUBO et al., 2009; KRAMER; MAGEE, 2009; SAMA et al., 2010; SIEWE; ZEDAN; CAU, 2011; LIU; XU; CHEUNG, 2013), não foi encontrado no escopo da revisão bibliográfica realizada neste trabalho, abordagens formais para especificação e verificação da corretude do comportamento excepcional sensível ao contexto, sendo esta a motivação principal desta tese de doutorado.

1.2 Problema, Hipótese e Questões de Pesquisa

Em linhas gerais, a problemática abordada nessa tese consiste em:

> *Como representar formalmente o modelo de comportamento do tratamento de exceção sensível ao contexto e identificar, de forma automática, a ocorrência de faltas de projeto nesse modelo?*

Esse tipo de problema pode ser reduzido ao problema clássico de Verificação de Modelos (*Model Checking*) (CLARKE JR.; GRUMBERG; PELED, 1999). A Verificação de Modelos é um método formal empregado na verificação automática de modelos de sistemas reativos com número finito de estados (CLARKE JR.; GRUMBERG; PELED, 1999). Nessa abordagem, tipicamente o modelo de comportamento do sistema é representando por meio de uma estrutura de Kripke, formalismo baseado em estados e transições, e as propriedades a serem verificadas são especificadas usando lógicas temporais (e.g., LTL – *Linear Temporal Logic* e CTL – *Computation Tree Logic*). As propriedades capturam a

semântica de determinados padrões de comportamento que se deseja verificar a existência ou ausência no modelo de comportamento do sistema em questão. A verificação das propriedades comportamentais é dada através de uma exaustiva enumeração (implícita ou explícita) de todos os estados alcançáveis do modelo de comportamento. Dessa forma, para um dado modelo de comportamento e uma propriedade a ser verificada, o verificador de modelos (*model checker*) sempre termina e retorna um dos seguintes valores: (i) "verdadeiro", se a propriedade é satisfeita; ou (ii) "falso", caso a propriedade não seja satisfeita. No segundo caso, dependendo do verificador de modelos utilizado, um contra exemplo é oferecido como evidência da violação da propriedade. Desse modo, esta tese de doutorado busca investigar a seguinte hipótese:

> *É possível representar o modelo de comportamento do tratamento de exceção sensível ao contexto via estrutura de Kripke e utilizar a técnica de Verificação de Modelos para identificar, de forma automática, a ocorrência de faltas de projeto nesse modelo.*

Nesse cenário, é importante mencionar que o fato do problema endereçado nessa tese ser reduzido a um problema de Verificação de Modelos, essa redução traz inúmeros ganhos em termos de embasamento teórico, rigor formal e suporte ferramental, sendo esses ganhos uma das justificativas para a sua escolha. Além disso, como evidenciado em grande parte dos trabalhos apresentados por Weyns et al. (2012) na sua revisão sistemática sobre métodos formais em sistemas de software auto-adaptativo, as abordagens que buscam fazer análise comportamental nesse tipo de sistema, comumente adotam a técnica de Verificação de Modelos de forma direta (i.e., especificando seus modelos na sintaxe da linguagem de um verificador de modelos existente) ou indireta (i.e., provendo abstrações que permitem os projetistas especificarem seus modelos em uma notação específica de domínio e esta, por sua vez, é mapeada para algum verificador de modelos existente). A última abordagem é exatamente a adotada nesta tese pelo fato de tornar a atividade de modelagem mais próxima do domínio do projetista. Entretanto, para que essa redução seja bem sucedida, algumas questões de pesquisa (QP) precisam ser respondidas, a saber:

QP01 *Como representar a projeto do tratamento de exceção sensível ao contexto por meio de uma estrutura de Kripke?*

QP02 *Quais são os tipos mais comuns de faltas de projeto que podem ser cometidas pelos projetistas durante o projeto do tratamento de exceção sensível ao contexto?*

QP03 *Como representar formalmente em lógica temporal as propriedades que capturam a semântica das faltas de projeto?*

1.3 Objetivo e Contribuições

Levando em consideração o que foi apresentado nas seções anteriores, esta pesquisa de doutorado tem como propósito fornecer uma solução que possibilite a verificação formal da corretude do projeto do tratamento de exceção sensível ao contexto. O objetivo foi baseado em (i) nos relatos encontrados na literatura sobre a importância e efetividade do emprego dos métodos formais na identificação e remoção de faltas de projeto em sistemas de software adaptativo sensível ao contexto; e (ii) em não ter sido encontradas abordagens para modelagem e verificação formal do comportamento excepcional sensível ao contexto durante a atividade de revisão de literatura conduzida neste trabalho.

Para atingir este objetivo, ele foi decomposto em 4 (quatro) metas (MET):

MET01 *Identificar e representar formalmente em lógica temporal a semântica dos tipos mais comuns de faltas de projeto*;

MET02 *Investigar uma forma de representar o comportamento excepcional sensível ao contexto por meio de uma estrutura de Kripke*;

MET03 *Projetar uma ferramenta que permita analisar, de forma automática, o modelo de comportamento excepcional sensível ao contexto com o intuito de identificar faltas de projeto*; e

MET04 *Analisar a efetividade*[4] *da abordagem proposta por meio de cenários onde faltas de projeto são injetadas em um modelo correto.*

A partir desse objetivo e dessas metas, as principais contribuições (CTR) desta tese de doutorado são listadas a seguir:

CTR01 *Um método para verificação do comportamento excepcional sensível ao contexto.* O método provê abstrações que permitem modelar o comportamento excepcional sensível ao contexto. Essas abstrações possibilitam que esse modelo de comportamento seja mapeado para um modelo de comportamento compatível com a técnica

[4]Nesta tese, a efetividade é considerada como a medida da capacidade de algo ser bem sucedido em atender de forma adequada a um proposito ou produzir o resultado pretendido ou esperado.

de Verificação de Modelos, tornando possível a especificação e verificação automática de propriedades;

CTR02 *Uma lista de propriedades comportamentais.*
Um conjunto de propriedades descritas em lógica temporal, que capturam a semântica do comportamento excepcional sensível ao contexto, é proposto nesse trabalho. Essas propriedades, quando violadas durante o processo de verificação, indicam que determinados tipos de faltas de projeto estão presentes no projeto do tratamento de exceção sensível ao contexto; e

CTR03 *Uma ferramenta de suporte ao método.*
Com o intuito de automatizar as atividades de modelagem e especificação descritas pelo método proposto, uma ferramenta de suporte foi desenvolvida. Essa ferramenta foi integrada a um verificador de modelos existente, facilitando o processo de verificação.

1.4 *UbiParking*: Sistema Exemplo

Com o intuito de facilitar o entendimento dos conceitos e das proposições apresentadas ao longo desta tese, esta seção descreve um sistema de software adaptativo sensível ao contexto que, embora pequeno e simplificado, apresenta um conjunto de características comportamentais e cenários de operação relevantes no contexto deste trabalho. Este sistema exemplo, chamado de *UbiParking*, surge no cenário do conceito de "Cidades Ubíquas", ou simplesmente *U-City*, como descrito por Jang e Suh (2010). A ideia por trás desse conceito é o provimento de serviços ubíquos relacionados com o cotidiano das cidades e das pessoas que nelas habitam, com o propósito de melhorar a convivência urbana sob diversos aspectos, tais como trânsito, segurança e atendimento aos cidadãos. A escolha desse domínio foi feita com base em um estudo realizado sobre uma aplicação real, chamada *SFPark*[5], que é um sistema implementado pela Agência Municipal de Transportes de São Francisco, nos Estados Unidos, para controlar a ocupação de vagas de estacionamento em vias e estacionamentos públicos em determinadas regiões do centro da cidade.

[5] http://sfpark.org/

1.4.1 Funcionalidades

O *UbiParking* disponibiliza um mapa plotado com todas as vagas de estacionamento disponíveis por região. Este mapa de vagas livres é atualizado com base em informações coletadas por meio de sensores implantados nos acostamentos das vias e nos estacionamentos públicos. Os sensores detectam quando uma vaga de estacionamento está ocupada ou livre, enviando esta informação para o sistema. Desse modo, utilizando o *UbiParking* em seus dispositivos móveis ou no computador de bordo dos seus veículos, os cidadãos podem obter informações sobre a distribuição das vagas livres por região, podendo reservar uma vaga e solicitar ao sistema uma rota mais apropriada com base em algum critério de sua preferência (e.g., menor distância, maior número de vagas livres disponíveis ou menor preço).

Ao chegar ao estacionamento escolhido, o *UbiParking* conduz o motorista até a vaga reservada ou à vaga livre mais próxima, considerando os casos onde a vaga reservada é ocupada de forma imprevisível por outro veículo. Do mesmo modo, quando o motorista retorna ao seu veículo, o *UbiParking* o conduz até a saída mais próxima, poupando-lhe tempo. Os estacionamentos do *UbiParking* possuem uma disposição espacial composta por entradas, pátio de vagas e saídas, conforme ilustrado na Figura 1.1.

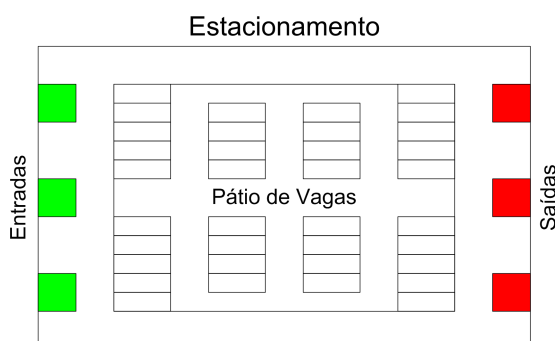


Figura 1.1: Planta Baixa do Estacionamento

Além disso, o estacionamento ubíquo encontra-se equipado com sensores que monitoram a temperatura do estacionamento, de modo que quando a temperatura diminui até um determinado limiar inferior (e.g., 15 °C), o sistema de aquecimento é acionado. Por outro lado, se a temperatura aumentar, chegando a um limiar superior (e.g., 35 °C), o sistema de refrigeração do estacionamento é acionado. Em todo caso, se a temperatura do estacionamento permanece entre os limiares ($15\,°C < \texttt{temperatura} < 35\,°C$), apenas o sistema de ventilação é mantido em funcionamento. Além disso, o estacionamento ubíquo é equipado com detectores de fumaça e aspersores controlados automaticamente, para o

caso de incêndio.

1.4.2 Cenários Excepcionais

Abaixo são descritos dois cenários de exceções contextuais que podem ocorrer dentro do *UbiParking*:

> **Exceção de Incêndio**. *Essa exceção contextual modela uma condição de incêndio dentro do estacionamento. Por meio das informações de contexto coletadas pelos sensores de fumaça e temperatura, o UbiParking consegue detectar a ocorrência desse tipo de exceção contextual dentro do estacionamento. Para lidar com essa exceção, o sistema aciona os aspersores, notifica o corpo de bombeiros e conduz os motoristas que estão em movimento dentro do estacionamento até o lado de fora.*

> **Exceção de Vaga Indisponível**. *Essa exceção contextual modela uma situação em que o veículo está em movimento dentro do pátio de vagas indo em direção à sua vaga reservada. Porém, outro veículo ocupa aquela vaga. Nesse caso, se a vaga for a última vaga livre disponível, fica caracterizado a situação de anormalidade. Essa exceção contextual é detectada pelo sistema através de informações de contexto sobre as reservas de vagas, a localização do veículo e os dados que vem dos sensores de detecção de vaga ocupada. Como forma de tratar essa exceção contextual, o UbiParking conduz o veículo até o lado de fora do estacionamento, onde outra vaga livre em outro estacionamento pode ser localizada para ele.*

Embora simples, estes cenários possuem detalhes que serão explorados no decorrer do texto que ajudam na compreensão dos conceitos sobre o tratamento de exceção sensível ao contexto.

1.5 Organização do Documento

Este capítulo apresentou a problemática que motiva esta tese de doutorado, a hipótese assumida, as questões de pesquisa investigadas, os objetivos, as principais contribuições, além do sistema exemplo, referenciado no decorrer do texto. O restante desse documento esta organizado da seguinte forma:

Capítulo 2 - Este capítulo é dedicado a fundamentação teórica desta tese de doutorado. Nele são apresentados os temas relacionados ao projeto de sistemas de software adaptativo sensível ao contexto, ao tratamento de exceção, a técnica de verificação de modelos e o conceito de programação por restrições.

Capítulo 3 - Este capítulo é dedicado aos trabalhos relacionados a esta tese de doutorado. Nele são investigados trabalhos relacionados que ajudam capturar os principais tipos de exceções contextuais, aspectos relevantes do tratamento de exceção sensível ao contexto para a modelagem de comportamento e pontos do projeto do tratamento de exceção sensível ao contexto propensos a faltas de projeto. Além disso, um estudo sobre trabalhos que fazem verificação de comportamento adaptativo de sistemas adaptativos sensíveis ao contexto é apresentado.

Capítulo 4 - Este capítulo é dedicado a descrição do método para verificação de modelos do tratamento de exceção sensível ao contexto. Nele são apresentadas as abstrações utilizadas para construir o modelo de comportamento e um conjunto de propriedades comportamentais úteis para identificação de determinados tipos de faltas de projeto.

Capítulo 5 - Este capítulo mostra detalhes sobre a implementação da ferramenta de suporte ao método e a forma como os cenários de injeção de faltas foram utilizados para avaliar a efetividade do método. Além disso, uma discussão sobre os resultados é apresentada.

Capítulo 6 - Este capítulo é dedicado às considerações finais da tese e possíveis direcionamentos para pesquisas futuras.

2 Fundamentação Teórica

Este capítulo é dedicado a apresentação dos temas que representam a base teórica desta tese de doutorado. Na Seção 2.1 são introduzidos aspectos de projeto de sistemas de software adaptativo sensível ao contexto. Na Seção 2.2 são introduzidos os principais conceitos de tratamento de exceção. O processo de verificação de modelos e a sua base formal são apresentados na Seção 2.3. A Seção 2.4 é dedicada a apresentação do conceito de programação por restrições, suas propriedades e definição matemática. Por fim, na Seção 2.5 são apresentadas algumas considerações sobre os temas abordados neste capítulo, sua correlação e relevância para o escopo deste trabalho.

2.1 Software Adaptativo Sensível ao Contexto

Por serem um tipo particular de sistema de software auto-adaptativo (*self-adaptive*) ou auto-gerenciado (*self-managed*), os sistemas de software adaptativo sensível ao contexto[1] (*context-aware adaptive*) herdam inúmeros desafios de desenvolvimento, muitos ainda em aberto, que estão relacionados com a engenharia de requisitos, o projeto, a implementação e o suporte adequado à verificação e validação, tanto em tempo de desenvolvimento quanto em tempo de execução (KRAMER; MAGEE, 2007; CHENG et al., 2009a; LEMOS et al., 2012). Neste cenário, esforços na área buscam contribuir para a construção de um corpo de conhecimento[2] e o estabelecimento de uma disciplina específica para a construção desse tipo sistema, chamada de Engenharia de Software para Sistemas Auto-Gerenciados (CHENG et al., 2009b). O software auto-adaptativo tem como objetivo ajustar seus vários artefatos ou atributos em resposta à mudanças internas e/ou externas ao software (SALEHIE; TAHVILDARI, 2009). Tipicamente, os sistemas de software auto-adaptativo são capazes de monitorar e atuar sobre si e sobre o seu ambiente com o

[1] Embora o software auto-adaptativo compreenda uma classe mais geral que a classe dos sistemas adaptativos sensíveis ao contexto, no restante desse documento esses termos serão utilizados de forma intercambiável, uma vez que os tópicos discutidos abordam questões de projeto comuns às duas classes.

[2] http://self-adaptive.org/

propósito de manter a sua execução em conformidade com sua especificação (funcional e de robustez). Isso significa que o comportamento desse tipo de sistema é baseado em **laço de controle fechado com retroalimentação** (*closed control loop with feedback*) sobre si e sobre o seu ambiente (PATIKIRIKORALA et al., 2012). Nas próximas subseções são apresentadas definições, propriedades e questões essenciais sobre a elicitação de requisitos e o projeto (*design*) de sistemas de software auto-adaptativo.

2.1.1 Definições

Os sistemas de software auto-adaptativo surgem como uma alternativa para lidar com o crescente custo em tratar a complexidade dos sistemas de software, fazendo com que estes atinjam suas metas. Algumas dessas metas estão relacionadas com: (i) o gerenciamento da complexidade; (ii) a robustez no tratamento de situações indesejadas ou inesperadas (e.g., exceções e falhas); (iii) a alteração de prioridades e políticas que governam as metas do sistema; e (iv) as diversas condições de mudança (e.g., mobilidade e variações no ambiente operacional) (SALEHIE; TAHVILDARI, 2009).

Tradicionalmente, uma parte significante da pesquisa sobre como tratar a complexidade do software, fazendo com que este atinja suas metas de qualidade, tem sido focada no processo de desenvolvimento e nos atributos internos de qualidade do software, como descrito no modelo de qualidade da primeira parte da norma técnica (ISO9126, 2001). Entretanto, tem havido uma demanda crescente para que essas questões sejam tratadas em tempo de execução, obrigando os projetistas a buscarem meios adequados para lidar com isso (LEMOS et al., 2012). Segundo Salehie e Tahvildari (2009), a causa primária para essa tendência reside (i) no crescente nível de heterogeneidade dos componentes de software; (ii) na maior frequência com que o contexto, os requisitos e as metas mudam em tempo de execução; e (iii) na necessidade de maiores níveis de segurança. Salehie e Tahvildari (2009) argumentam que algumas dessas causas são consequências da alta demanda por aplicações ubíquas, embarcadas e móveis, principalmente no contexto da internet e das redes *ad-hoc*.

Uma das primeiras definições para software auto-adaptativo foi dada pela *Defense Advanced Research Projects Agency* (DARPA), em dezembro de 1997, por meio do *Broad Agency Announcement on Self Adaptive Software* (BAA-98-12), a qual diz que:

> "Um software auto-adaptativo avalia seu próprio comportamento e muda o comportamento quando a avaliação indica que ele não está atendendo ao que se destina a fazer; ou quando um melhor desempenho ou funcionalidade são possíveis" (LADDAGA, 1997).

Poucos anos depois, outra definição para software auto-adaptativo é dada por Oreizy et al. (1999) (declarada abaixo) na qual o fator causador da mudança e o objetivo da adaptação são removidos, tornando-a mais genérica.

> "Um software auto-adaptativo modifica seu próprio comportamento em resposta a mudanças no seu ambiente operacional. Por ambiente operacional entende-se qualquer coisa observável pelo sistema de software, tais como entradas do usuário, dispositivos de hardware e sensores ou instrumentação de programas" (OREIZY et al., 1999).

É importante observar que ambas as definições explicitam a capacidade do software de **observar** a si próprio (*self*) e o seu ambiente (contexto), e **atuar** sobre si e, em muitos casos, sobre o seu ambiente, com base nas observações realizadas, caracterizando um ciclo de decisão bem definido. As definições mais atuais também seguem nesta mesma direção, como a apresentada por Cheng et al. (2009b) que diz que sistemas de software auto-adaptativo são:

> "Sistemas capazes de ajustar o seu comportamento em resposta a sua percepção do ambiente e de si mesmo" (CHENG et al., 2009b).

Particularmente, o foco deste trabalho é dado para os sistemas de software auto-adaptativo projetados para os ambientes ubíquos, cujos domínios de aplicação exploram os conceitos da Computação Ubíqua (WEISER, 1991) (e.g., cidades inteligentes, guias de visitação e saúde). Nestes domínios, o objetivo principal do software é auxiliar as pessoas na execução das suas atividades cotidianas de uma forma simples, casual e não intrusiva, causando a mínima distração possível aos usuários (GARLAN et al., 2002; SADRI, 2011). Nesse cenário, o objetivo principal da auto-adaptação do software é manter a sua execução em um estado correto, estável e otimizado, garantindo que os seus requisitos (metas) e os interesses (necessidades e expectativas) dos seus usuários sejam atendidos (HUEBSCHER; MCCANN, 2008; SOYLU; CAUSMAECKER; DESMET, 2009).

2.1.2 Propriedades Auto-*

Em 2001, a companhia IBM (*International Business Machines*) introduziu o conceito de **computação autônoma** inspirado no sistema nervoso do corpo humano (KEPHART; CHESS, 2003). Na computação autônoma, os sistemas computacionais devem ser capazes de gerenciar a si próprios em função de um conjunto de objetivos de alto nível fornecidos pelo administrador do sistema. A essência dos sistemas computacionais autônomos (ou auto-adaptativos) é a propriedade de autogerenciamento. Para facilitar o entendimento

dessa propriedade, a IBM propôs o seu desmembramento em 4 (quatro) áreas funcionais, também conhecidas como propriedades auto-* (*self-**), são elas:

- **Autoconfiguração** (*self-configuring*): é a capacidade do sistema de reconfigurar-se automaticamente e de forma dinâmica em resposta a mudanças por instalação, atualização, integração e composição/decomposição de entidades de software;
- **Auto-cura** (*self-healing*): é a capacidade do sistema de descobrir, diagnosticar e reagir a situação adversas, podendo agir de forma antecipada para detectar potenciais problemas e, consequentemente, tomar ações apropriadas para evitar uma possível falha. A auto-cura está diretamente relacionada com outras duas propriedades: (i) o autodiagnóstico (*self-diagnosing*), que refere-se a diagnosticar erros, defeitos e falhas; e (ii) a auto-reparação (*self-repairing*), que se concentra na recuperação de erros, defeitos e falhas;
- **Auto-otimização** (*self-otimizing*): é a capacidade do sistema de gerenciar seu desempenho e alocação de recursos com o objetivo de satisfazer os requisitos de diferentes usuários. Essa propriedade também é referenciada na literatura como sinônimo de auto-ajuste (*self-tuning*) e auto-regulamento (*self-adjusting*);
- **Autoproteção** (*self-protecting*): é a capacidade do sistema de detectar e se recuperar de falhas de segurança e dos seus efeitos. Essa propriedade possui dois aspectos (i) defender o sistema contra ataques maliciosos; e (ii) antecipar-se aos problemas, tomando medidas para evitá-los ou para mitigar seus efeitos.

Salehie e Tahvildari (2009) argumentam que duas outras propriedades são consideradas elementares para os sistemas de software auto-adaptativo, formando uma base sólida para que outras propriedades, incluindo as citadas anteriormente, possam ser alcançadas, são elas:

- **Autoconsciência** (*self-awareness*): significa que o sistema tem consciência do seu próprio estado e comportamento. Essa propriedade é baseada no auto-monitoramento, que reflete o que está sendo monitorado;
- **Sensibilidade ao Contexto** (*context-awareness*): significa que o sistema tem consciência do seu contexto, isto é, do seu ambiente operacional.

Ao longo das duas últimas décadas, uma discussão sobre o que é o contexto, a forma como este é gerenciado (modelado, adquirido, processado, armazenado e difundido), como

as aplicações podem se beneficiar com o seu uso e qual é o limite entre a sua fronteira e a fronteira das aplicações que o utilizam, foi conduzida e relatada por vários pesquisadores (SCHILIT; THEIMER, 1994; PASCOE, 1998; DEY; ABOWD, 1999; DEY, 2001; HENRICKSEN; INDULSKA; RAKOTONIRAINY, 2002; COUTAZ et al., 2005; BALDAUF; DUSTDAR; ROSENBERG, 2007; SOYLU; CAUSMAECKER; DESMET, 2009; BETTINI et al., 2010; BELLAVISTA et al., 2012; RAYCHOUDHURY et al., 2012). Dessa forma, influenciados pela definição de Dey e Abowd (1999) (enunciada a seguir), uma parte significante dos pesquisadores concordam que o sistema e seu estado interno fazem parte do contexto. Logo, a propriedade de sensibilidade ao contexto engloba a propriedade de autoconsciência, sendo este o entendimento adotado neste trabalho. Como consequência, toda adaptação realizada pelo software é consequência da observação do seu contexto. Sendo assim, no escopo deste trabalho, o software auto-adaptativo passa a ser considerado sinônimo de software adaptativo sensível ao contexto.

> "Contexto é qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade pode ser uma pessoa, um lugar, ou um objeto que é considerado relevante para a interação entre um usuário e uma aplicação, incluindo o próprio usuário e a própria aplicação" (DEY; ABOWD, 1999).

2.1.3 Elicitação de Requisitos de Adaptação

Segundo Salehie e Tahvildari (2009), uma forma plausível de capturar os requisitos de sistemas de software auto-adaptativo é fazendo o uso das 6 (seis) questões descritas no poema *Six Honest Men*[3]. Para esses autores, estas seis questões, referenciadas como 5W1H, são extremante importantes para elicitação de requisitos essenciais de software auto-adaptativo, pois ajudam a capturar os aspectos de projeto e execução relacionados com o atendimento às propriedades auto-* descritas na subseção anterior, são elas:

- **Onde** (*where*): este conjunto de questões está preocupado em encontrar a necessidade de adaptação. Qual artefato, em qual camada (e.g., aplicação, middleware ou rede) e em qual nível de granularidade precisa ser adaptado? Para atingir esse objetivo, é necessário coletar informações sobre os atributos do software, as dependências componente-componente/componente-camada e informações sobre o ambiente operacional. Desse modo, o conjunto de questões "onde" está relacionado com a localização de problemas que necessitam ser resolvidos por meio de adaptação;

[3] *Six Honest Men* de R. Kippling, publicado no livro *Just so Stories*. Penguin Books, Londres, 1902.

- **Quando** (*when*): este conjunto de questões endereçam aspectos temporais relacionados à adaptação. Quando uma adaptação necessita ser aplicada e quando é possível (ou viável) isso ocorrer? Por exemplo, a adaptação pode ser aplicada a qualquer momento que é requerida, ou existem restrições que limitam alguns tipos de adaptação? Com qual frequência o sistema precisa ser adaptado? As adaptações são continuas ou ocorrem apenas quando necessário? É suficiente executar adaptações de forma reativa, ou é necessário prever situações de mudança e agir pro-ativamente?

- **O que** (*what*): ajuda a identificar quais atributos ou artefatos do sistema podem ser modificados através de ações de adaptação e o que precisa ser adaptado em cada situação. Estes atributos ou artefatos podem variar desde parâmetros até componentes, estilos arquitetônicos e recursos do sistema. É importante, também, identificar as alternativas disponíveis para o conjunto de ações de adaptação e o intervalo de variação para cada um dos atributos. Além disso, é essencial determinar quais eventos e atributos devem ser monitorados e quais recursos são essenciais para que as ações de adaptação possam ocorrer;

- **Por quê** (*why*): este conjunto de questões lidam com a motivação para construir um software auto-adaptativo. Essas questões estão interessadas nos objetivos do sistema (e.g., segurança, robustez e alta disponibilidade). De forma prática, se uma abordagem orientada a metas (*goal-oriented*) é utilizada durante a engenharia de requisitos, este conjunto de questões identificam as metas do sistema de software auto-adaptativo;

- **Quem** (*who*): este conjunto de questões endereçam o nível de automação e intervenção humana necessário no software auto-adaptativo. Com relação a automação, é esperado que exista uma mínima intervenção humana no processo de adaptação. No entanto, é necessário que haja uma interação efetiva entre os proprietários do sistema e os gerentes para a construção de um sentimento de confiança e a criação de políticas de transferência;

- **Como** (*how*): este conjunto de questões está relacionado com a forma como a adaptação ocorre. Um dos mais importantes requisitos para a adaptação é determinar como os artefatos adaptáveis podem ser modificados e quais ações de adaptação podem ser apropriadamente aplicadas em uma dada condição. Para isso, é necessário estabelecer uma relação de ordem entre as ações de adaptação, seus custos e efeitos para decidir qual será a próxima ação ou plano de ações.

As questões 5W1H precisam ser respondidas em duas fases: (i) na fase de desenvolvimento; e (ii) na fase operacional. Na fase de desenvolvimento, os projetistas elicitam requisitos com base nas questões 5W1H com o objetivo de construir o software auto-adaptativo, bem como estabelecer o mecanismo e as alternativas de adaptação a serem utilizadas na fase operacional. Por outro lado, na fase operacional, o sistema requer ser adaptado com base nas questões 5W1H. Tipicamente, as questões nesta fase perguntam por: **onde** está a fonte da necessidade de adaptação, **o que** necessita ser adaptado, **quando** e **como** é melhor realizar a adaptação. As respostas para estas questões na fase operacional depende da abordagem e do tipo de adaptação que foram estabelecidos na fase de desenvolvimento.

2.1.4 Modelagem de Dimensões

A forma como a auto-adaptação deve ser concebida em um sistema de software depende de vários aspectos (e.g., necessidades dos usuários e características do ambiente). A compreensão do problema e a seleção de soluções adequadas requer modelos precisos para representar importantes aspectos do próprio sistema, dos seus usuários e do seu ambiente (CHENG et al., 2009a). Embora na literatura sejam encontrados diversos relatos sobre sistemas de software auto-adaptativo, Cheng et al. (2009a) argumentam que existe uma falta de consenso entre pesquisadores e práticos em relação aos pontos de variação entre esses sistemas, referenciados como **modelagem de dimensões**. Para lidar com essa questão, Andersson et al. (2009) propõem uma classificação para modelagem de dimensões, onde cada dimensão descreve um aspecto particular do sistema que é relevante para a auto-adaptação. Essas dimensões encontram-se divididas em 4 (quatro) grupos: (i) **metas**, que agrupa as dimensões relacionadas com a auto-adaptabilidade das metas do sistema; (ii) **mudança**, que reúne as dimensões relacionadas com as causas da auto-adaptação; (iii) **mecanismos**, que agrupa as dimensões relacionadas com os mecanismos empregados para atingir a auto-adaptação; e (iv) **efeitos**, que reúne as dimensões associadas com os efeitos da auto-adaptação sobre o sistema.

Metas. As metas representam os objetivos que o sistema deve atingir. Elas podem estar associadas ao ciclo de vida do sistema ou a cenários que estão relacionados com o sistema. Além disso, as metas podem estar relacionadas com aspectos de auto-adaptabilidade do próprio sistema, do sistema de suporte (e.g., plataforma de middleware) ou da infraestrutura de suporte a execução do sistema (e.g., dispositivos computacionais sensores e atuadores). Esse grupo engloba 5 (cinco) dimensões: (i) **evolução**, que está

relacionada com a possibilidade de mudança na quantidade e na semântica das metas durante a execução do sistema; (ii) **flexibilidade**, que se preocupa em capturar o grau de flexibilidade das metas com relação a forma como essas são expressas; (iii) **duração**, que está interessada no período de validade de uma meta durante o ciclo de vida do sistema; (iv) **multiplicidade**, que está relacionada com o número de metas associadas à aspectos de adaptabilidade do sistema; e (v) **dependência**, que, no caso do sistema possuir mais de uma meta, essa dimensão captura como essas metas estão relacionadas entre si.

Mudança. As mudanças são as causas da adaptação. Sempre que o contexto muda, o sistema deve decidir se precisa se adaptar. De forma prática, o contexto é qualquer informação que é computacionalmente acessível e sobre a qual variações comportamentais do sistema são dependentes. Informações sobre os atores (entidades que interagem com o sistema), o ambiente (partes do mundo externo com as quais o sistema interage) e o próprio sistema podem ser vistos como parte do contexto. As variações dependentes dos atores, do sistema ou do ambiente podem ocorrer de forma separada ou em uma ordem particular. Entender os aspectos relacionados às mudanças é importante para identificar a forma como o sistema deve reagir à essas mudanças em tempo de execução. Esse grupo compreende 4 (quatro) dimensões: (i) **fonte**, que busca identificar a origem (interna ou externa) das mudanças; (ii) **tipo**, está relacionada com a natureza da mudança, podendo ser funcional, não funcional ou tecnológica; (iii) **frequência**, que está interessada em identificar com qual frequência uma mudança em particular ocorre; e (iv) **antecipação**, que está interessada em identificar se uma mudança pode ser prevista de forma antecipada.

Mecanismos. Esse grupo de 7 (sete) dimensões busca compreender a reação do sistema face às mudanças. Isso significa que essas dimensões estão relacionadas diretamente com processo de adaptação do sistema, são elas: (i) **tipo**, que identifica se a adaptação está relacionada com parâmetros dos componentes do sistema (adaptação paramétrica) ou com a estrutura do sistema (adaptação estrutural/composicional); (ii) **autonomia**, que busca identificar grau de intervenção externa durante o processo de adaptação, o qual pode ser autônomo (totalmente automático e sem auxílio externo) ou assistido (via intervenção humana ou de outros sistemas); (iii) **organização**, que identifica se a adaptação é conduzida por um único componente (adaptação centralizada) ou por vários componentes de forma distribuída (adaptação descentralizada); (iv) **escopo**, está relacionado com a determinação da abrangência da adaptação, podendo ser classificada como adaptação local, caso ocorra de forma localizada, ou adaptação global, se envolve o sistema como um todo; (v) **duração**, identifica o período de tempo necessário a ser empregado na realização da adaptação; (vi) **pontualidade**, investiga se o período de tempo necessário para executar

a adaptação pode ser assegurado; e (vii) **desencadeamento**, identifica se a mudança que inicia a adaptação é desencadeada por eventos (ou correlação de eventos) ou pelo tempo.

Efeitos. Esse grupo de dimensões identificam o impacto da adaptação sobre o sistema. Enquanto os mecanismos de adaptação são propriedades relacionadas com a adaptação, as dimensões deste grupo são propriedades relacionadas com o sistema onde a adaptação ocorre. As 4 (quatro) dimensões que fazem parte deste grupo são: (i) **criticidade**, responsável por medir o impacto sobre o sistema nos casos em que a auto-adaptação falha; (ii) **previsibilidade**, identifica se as consequências da auto-adaptação podem ser previstas em termos de custo e tempo, assumindo um das duas formas: determinística ou não determinística; (iii) **sobrecarga**, investiga o impacto negativo da adaptação sobre o desempenho do sistema; e (iv) **resiliência**, que está relacionada com a manutenção da entrega de serviço de forma confiável frente a situações de mudança.

2.1.5 Laço de Adaptação

Como mencionado anteriormente, os sistemas de software auto-adaptativo incorporam um mecanismo de adaptação baseado em **laço de controle fechado com retroalimentação** (*closed control loop with feedback*). Esse laço de controle, denominado por Salehie e Tahvildari (2009) de **laço de adaptação**, consiste em um conjunto de processos, sensores e atuadores (Figura 2.1). O laço de adaptação é baseado no laço de controle MAPE-K (*Monitoring, Analyzing, Planning, Executing, and Knowledge*) da computação autônoma (KEPHART; CHESS, 2003). Os sensores, atuadores e os processos do laço de adaptação são detalhados nos parágrafos subsequentes.

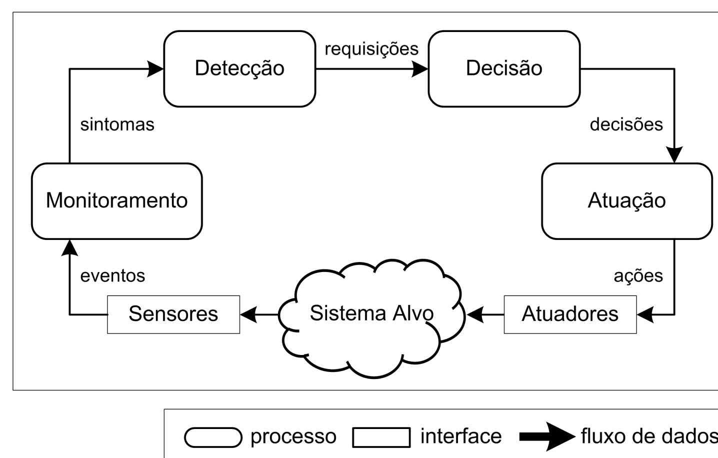


Figura 2.1: Laço de Adaptação (adaptado de (SALEHIE; TAHVILDARI, 2009)).

Os sensores monitoram entidades de software para gerar uma coleção de dados que

refletem o estado do sistema (incluindo o seu contexto), enquanto os atuadores dependem de mecanismos *in vivo* para aplicarem as mudanças. Os sensores e atuadores são elementos essenciais em sistemas de software auto-adaptativo, pois é por meio deles que o sistema consegue perceber o seu estado e modificar suas propriedades e comportamento. Para um melhor entendimento, é sugerido ao leitor a leitura do trabalho de Salehie e Tahvildari (2009) que elenca um conjunto de técnicas e exemplos de sensores e atuadores.

O processo de monitoramento é responsável por coletar e correlacionar dados advindos dos sensores e convertê-los em informação útil sobre o estado e o comportamento do sistema, identificando padrões de comportamento e sintomas de interesse. Essa atividade pode ser realizada por meio de correlação de eventos, verificação de limiares (valores máximos e mínimos), dentre outros. O processo de monitoramento aborda, em parte, as questões **onde**, **quando** e **o que** na fase de operação. Por outro lado, o processo de detecção é responsável por analisar os sintomas identificados pelo processo de monitoramento e o histórico do sistema, com o objetivo de detectar **quando** a mudança é requerida. Este processo também ajuda a identificar **onde** está a fonte da necessidade de transição do sistema para um novo estado.

O processo de decisão é responsável por determinar **o que** necessita ser adaptado e **como** essa adaptação deve ser feita para atingir o melhor resultado possível. Para isso, é necessário a criação de critérios para comparar diferentes formas de aplicar a adaptação, por meio de diferentes sequências de ações. Já o processo de atuação, é responsável por aplicar as ações determinadas pelo processo de decisão. Isso inclui o gerenciamento de ações através de um fluxo de trabalho (*workflow*) pré-definido ou do mapeamento das ações para o que é provido pelos atuadores e pelas técnicas de adaptação disponibilizadas pelo mecanismo de adaptação subjacente. Este processo está relacionado com as questões **como**, **o que** e **quando** adaptar.

Uma questão que deve ser observada com respeito ao laço de adaptação, é a forma como ele pode ser implementado. Essa questão diz respeito a separação entre o mecanismo de adaptação e a lógica da aplicação. Esta separação pode ser classificada em **interna** ou **externa**. Na abordagem interna a lógica da adaptação, e o código fonte responsável por implementá-la, encontra-se entrelaçada com a lógica da aplicação, podendo ocasionar problemas de escalabilidade e manutenção. Por outro lado, a abordagem externa faz uso de um engenho (ou gerente) de adaptação externo, o qual implementa o processo de adaptação. Usando esta abordagem, o sistema de software auto-adaptativo consiste de um engenho de adaptação e um software adaptável (i.e., passível de adaptação). Este

engenho implementa a lógica da adaptação (e.g., por meio de uma camada de middleware e um conjunto de regras e políticas de adaptação) cujo alvo é o software adaptável.

2.2 Tratamento de Exceção

O tratamento de exceção é uma técnica de recuperação de erros por avanço (*forward error recovery*) tipicamente empregada na melhoria da robustez de sistemas de software. Nesta seção são introduzidos os principais conceitos sobre exceções, mecanismo de tratamento de exceções e modelos de tratamento.

2.2.1 Falta, Erro, Falha e Exceção

Segundo a terminologia de Avizienis et al. (2004), uma **falha** (*failure*) é um evento que ocorre quando o serviço entregue por um sistema (ou componente) desvia da sua especificação funcional (i.e., está em desacordo com a sua especificação de funcionalidade e desempenho). A falha de um serviço pode ser vista como uma transição entre o estado onde um serviço correto está sendo entregue para um estado onde um serviço incorreto passa a ser entregue. Portanto, uma falha é um evento passível de observação externa por parte de seus usuários (i.e., seres humanos e outros sistemas). Um **erro** (*error*) é definido como uma parte do estado total do sistema que pode levá-lo a uma posterior falha. A causa física ou algorítmica de um erro é chamada de **falta** (*fault*). A falta é qualquer evento, ou sequência de eventos, que pode provocar um erro no estado interno do sistema (ou componente). Consequentemente, por computação, esse estado errôneo pode propagar-se internamente até afetar o comportamento do sistema, resultando na ocorrência de uma falha. É importante mencionar que alguns erros não afetam o comportamento do sistema e, portanto, não causam falhas.

Uma **exceção** (*exception*) é um evento que modela uma situação em que o fluxo normal de execução do sistema não pode continuar (KNUDSEN, 1987; KIENZLE, 2008). Para que o sistema continue executando corretamente, o fluxo de execução deve ser desviado e uma computação adicional deve ser empregada para tratar aquela situação (KNUDSEN, 1987). Em sistemas confiáveis, um erro, por ser considerado um evento que raramente ocorre durante a execução do sistema, pode ser modelado como uma exceção (GOODENOUGH, 1975). O tratamento de exceção provê meios para estruturar as atividades de tolerância a faltas (*fault tolerance*) através da recuperação de erros (CRISTIAN, 1982; LEE; ANDERSON, 1990). Entretanto, exceções podem modelar outro tipo de situações,

tais como as descritas por Miller e Tripathi (1997): (i) desvio - surgimento de um estado invalido, mas que é permitido pelo sistema; (ii) notificação - informação ao invocador da operação que o estado do sistema mudou; e (iii) idiomas - outros usos em que a ocorrência da exceção é rara em vez de anormal (e.g., final de arquivo – `EOF`).

2.2.2 Levantamento, Sinalização e Tratamento

No modelo de Componente Tolerante a Faltas Ideal descrito por Lee e Anderson (1990) (Figura 2.2), cada componente de software pode receber requisições de serviço de outros componentes. Quando as respostas àquelas requisições estão em conformidade com a especificação, diz-se que são respostas normais. Por outro lado, se o componente não é capaz de atender, por algum motivo, aquela requisição de serviço, ele retorna exceções. Dessa forma, componentes podem dar respostas normais ou excepcionais. Nesse modelo, as exceções podem ser classificadas em três categorias (GARCIA et al., 2001): (i) **exceções de interface** que são sinalizadas quando a requisição não está em conformidade com a interface de serviço do componente; (ii) **exceções de falha** que são sinalizadas para indicar que, por alguma razão, o componente não pôde atender a requisição de serviço; e (iii) **exceções internas** que são levantadas internamente ao componente para que este provenha suas próprias medidas de tratamento. Exceções são **levantadas** internamente ao componente e **sinalizadas** entre componentes. As exceções sinalizadas (de interface e serviço) são chamadas de **exceções externas**.

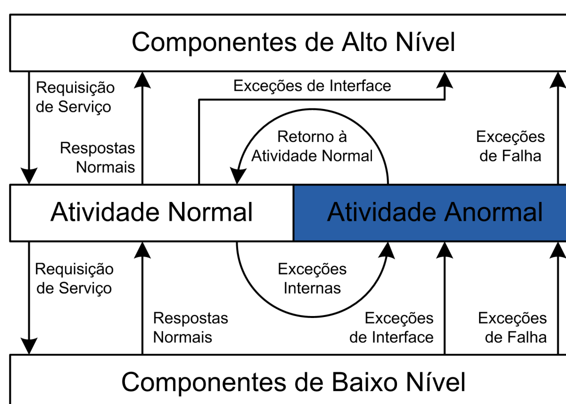


Figura 2.2: Componente Tolerante a Faltas Ideal (adaptado de Garcia et al. (2001)).

A atividade de cada componente do modelo pode ser dividida em duas partes: **atividade normal** e **atividade anormal** (ou excepcional). Na atividade normal, o componente processa as requisições de serviço de acordo com a sua especificação. Por outro lado, na atividade excepcional o componente aplica medidas para tratar as condições de erro detectadas. Idealmente, segundo Garcia et al. (2001), sistemas confiáveis podem ser

construídos a partir de um conjunto de componentes desse modelo (Figura 2.2). Dessa forma, componentes podem tratar exceções levantadas durante sua atividade normal ou exceções sinalizadas por componentes de mais baixo nível (que receberam requisição de serviço). Caso o componente não consiga tratar a exceção levantada, esta é propagada para os componentes de mais alto nível (requisitantes de serviço). Depois que a exceção é tratada, o sistema retorna à sua atividade normal.

O **mecanismo de tratamento de exceções** permite aos desenvolvedores definirem exceções e estruturarem o comportamento da atividade excepcional dos componentes através dos **tratadores de exceção** ou, simplesmente, **tratadores**. Os tratadores representam a parte do código do componente que provê medidas especificas para o tratamento da exceção levantada. Dessa forma, quando uma exceção é levantada durante a atividade normal do sistema (ou componente), o mecanismo de tratamento de exceções desvia o **fluxo de controle normal** para o **fluxo de controle excepcional**. Em tempo de execução, quando uma exceção é levantada, diz-se que houve uma **ocorrência de exceção**. A instrução que estava sendo executada quando uma ocorrência de exceção foi detectada é chamada de **instrução sinalizadora**. O trecho de código cuja instrução sinalizadora faz parte é chamado de **sinalizador**. Quando uma ocorrência de exceção é detectada, o mecanismo de tratamento de exceção procura o tratador adequado para o tratamento daquela exceção e o invoca. Os tratadores estão associados a uma parte particular do código do componente chamada de **região protegida** ou **contexto de tratamento**, delimitando o seu **escopo** de atuação. Idealmente, as instruções sinalizadoras devem ficar dentro da região protegida. Por fim, o mecanismo de tratamento de exceções pode ser parte integrante da linguagem de programação utilizada para construir o sistema, possuindo construtores bem definidos, ou ser tratado como uma funcionalidade inserida através de chamadas a bibliotecas externas (GARCIA et al., 2001).

2.2.3 Modelos de Tratamento

Os modelos de tratamento de exceção são diferenciados pelas suas políticas de gerenciamento do fluxo de controle excepcional. É importante mencionar que um mecanismo de tratamento de exceção pode incorporar mais de um modelo de tratamento. Os principais modelos de tratamento são descrito por Buhr e Mok (2000): (i) modelo de transferência não-local - que faz uso de “variáveis rótulo” como argumento do comando `goto` para redirecionar o fluxo de controle; (ii) modelo baseado em terminação – quando uma exceção é levantada dentro de uma região protegida, o fluxo de controle é transferido para o tra-

tador e a execução das instruções subsequentes à instrução sinalizadora é abortada sem, no entanto, apresentar sintomas de anormalidade; (iii) modelo baseado em retentativa – é uma extensão do modelo (ii) incorporando a possibilidade de reexecução da instrução sinalizadora depois da execução do tratador selecionado para o tratamento da exceção levantada; e (iv) modelo baseado em continuação – quando uma exceção é levantada, o fluxo de controle é transferido do sinalizador para o tratador para tratar a exceção, depois, o fluxo de controle retorna ao ponto onde a exceção foi levantada.

2.3 Verificação de Modelos

Um método formal é uma técnica baseada em modelos matemáticos para descrever sistemas computacionais (WOODCOCK et al., 2009). Na Engenharia de Software, o emprego de métodos formais permite aos engenheiros de software especificar, desenvolver e verificar sistemas de software de forma sistemática (BOWEN; HINCHEY, 1995). A verificação de modelos (*model checking*) é um método formal empregado na verificação automática de sistemas reativos concorrentes com número finito de estados (CLARKE JR.; GRUMBERG; PELED, 1999). Nessa abordagem, o comportamento do sistema é modelado através de algum formalismo baseado em estados e transições e as propriedades a serem verificadas são especificadas usando lógicas temporais (e.g., LTL, CTL e CTL*). A verificação das propriedades comportamentais é dada através de uma exaustiva enumeração (implícita ou explícita) de todos os estados alcançáveis do modelo do sistema. Nas próximas subseções são descritos o processo de verificação de modelos, as estruturas de *Kripke* (modelo formal para representação de comportamento) e uma visão geral sobre as lógicas temporais e como elas são utilizadas para especificar propriedades sobre o modelo do sistema.

2.3.1 O Processo de Verificação de Modelos

Do ponto de vista processual, a verificação de modelos pode ser compreendida em três fases (CLARKE JR.; GRUMBERG; PELED, 1999): modelagem, especificação e verificação. O processo de verificação de modelos é ilustrado na Figura 2.3 e suas atividades descritas a seguir:

- **Modelagem** – compreende a construção do modelo formal do sistema. Ela recebe como entrada as decisões de projeto (*design*) que visam atender aos requisitos do

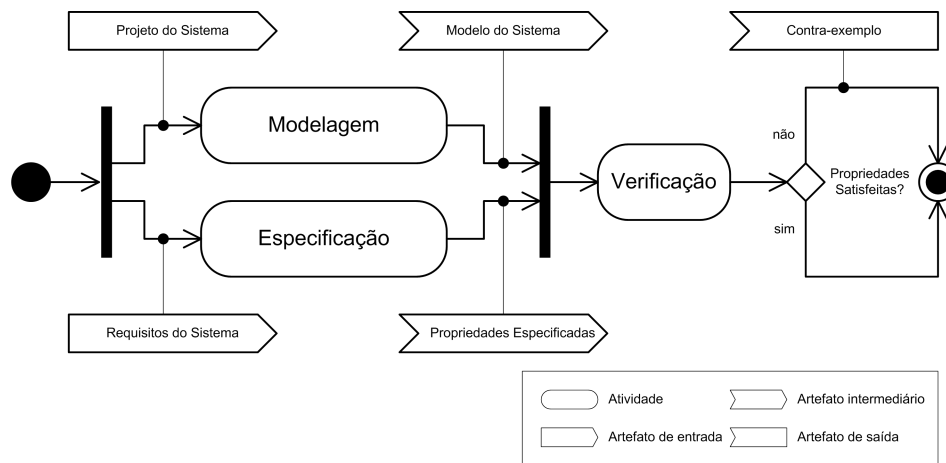


Figura 2.3: O Processo de Verificação de Modelos.

sistema. Essas decisões de projeto são, então, especificadas na linguagem do verificador de modelos (*model checker*) adotado. Dependendo do verificador de modelos adotado e das suas abstrações e construtores de linguagem, essa atividade pode ser mais ou menos complexa e propensa a erros. Ao final dessa atividade, o modelo formal do sistema é construído;

- **Especificação** – compreende a especificação, em lógica temporal, das propriedades do sistema a serem verificadas. A atividade de especificação recebe como entrada os requisitos do sistema e gera como saída a especificação das propriedades em lógica temporal. Um ponto importante a ser considerado durante a especificação de propriedades é a sua **completude**. A verificação de modelos provê meios para verificar se o modelo formal do sistema satisfaz uma dada especificação. Entretanto, é impossível determinar se uma dada especificação cobre todas as propriedades que o sistema deve satisfazer;

- **Verificação** – compreende a execução da ferramenta de verificação propriamente dita. Para um dado modelo e uma propriedade a ser verificada, o verificador de modelos sempre termina e retorna um dos seguintes valores: (i) "verdadeiro", se a propriedade é satisfeita; ou (ii) "falso", caso a propriedade não seja satisfeita. No segundo caso, dependendo do verificador utilizado, um contra exemplo é oferecido como evidência da violação da propriedade.

2.3.2 Estruturas de Kripke

Na verificação de modelos, o comportamento dos sistemas pode ser modelado usando, basicamente, dois tipos de formalismos (SCHNEIDER, 2004): **Estruturas de Kripke** e

Sistemas de Transição Rotulados (*Labelled Transition Systems* - LTS). As estruturas de Kripke são um formalismo baseado em estados, no qual os estados são rotulados ao invés das transições, sendo este último exatamente o caso dos LTS. Os rótulos possuem significados diferentes em cada um dos formalismos. Nas estruturas de Kripke, os rótulos representam um instantâneo (*snapshot*) da execução do sistema, contendo informações sobre cada estado. Já nos LTS, os rótulos caracterizam as ações que ocorrem em cada uma das transições do sistema durante a sua execução. O fato das estruturas de Kripke guardarem informações sobre os estados do sistema, foi preponderante para a sua adoção como formalismo para a modelagem do comportamento dos sistemas auto-adaptativos neste trabalho. A definição formal das estruturas de Kripke é dada a seguir (Definição 1).

Definição 1 (Estruturas de Kripke)**.** *Uma estrutura de Kripke* $\mathcal{K} = \langle S, I, L, \rightarrow \rangle$ *sobre um conjunto finito de proposições atômicas* $\mathcal{AP}$ *é dado por um conjunto finito de estados* S*, um conjunto de estados iniciais* $I \subseteq S$*, uma função de rótulos* $L : S \rightarrow 2^{\mathcal{AP}}$*, a qual mapeia cada estado em um conjunto de proposições atômicas que são verdadeiras naquele estado, e uma relação de transição total* $\rightarrow \subseteq S \times S$*, isto é, que satisfaz a restrição* $\forall s \in S\ \exists s' \in S$ *tal que* $(s, s') \in \rightarrow$*. O tamanho de* $\mathcal{K}$ *é dado por* $\mathtt{max}(|S|, |\rightarrow|)$*.*

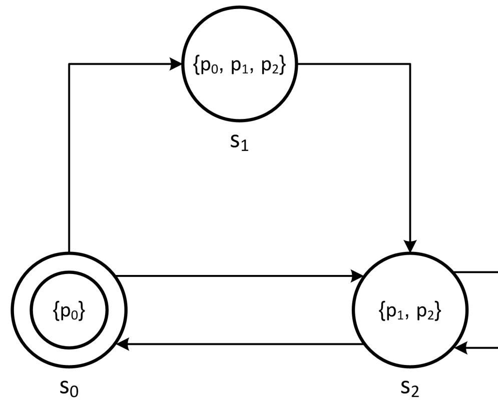


Figura 2.4: Exemplo de Estrutura de Kripke.

As estruturas de Kripke modelam o comportamento do sistema como um grafo de transições onde os nós são estados e as arestas transições. Por exemplo, na estrutura de Kripke descrita na Figura 2.4, o conjunto de estados é $S = \{s_0, s_1, s_2\}$, o conjunto de estados iniciais é $I = \{s_0\}$, a relação de transição é dada por $\rightarrow = \{(s_0, s_1), (s_1, s_2), (s_0, s_2), (s_2, s_0), (s_2, s_2)\}$, o conjunto de propriedades atômicas é $\mathcal{AP} = \{p_0, p_1, p_2\}$ e os rótulos dos estados são $L(s_0) = \{p_0\}$, $L(s_1) = \{p_0, p_1, p_2\}$ e $L(s_2) = \{p_1, p_2\}$.

Em uma estrutura de Kripke, a evolução do sistema é vista como uma sequência de estados e transições (e.g., $s_0 \rightarrow s_1 \rightarrow s_2 \ldots$). A sequência de transição entre um estado

válido ($s_0 \in S$) até outro estado válido ($s_n \in S$) representa uma **execução** do sistema (e.g., $\rho = s_0 \rightarrow s_1 \rightarrow s_2 \dots s_{n-1} \rightarrow s_n$). A sequência de estados em uma execução é chamado de **caminho** (e.g., para ρ temos o caminho $s_0, s_1, s_2, \dots, s_{n-1}, s_n$). A noção de caminho é formalmente apresentada na Definição 2. Para simplificar a notação, usa-se a seguinte convenção: dado um caminho $\pi = s_0, s_1, s_2, s_3, \dots$, a notação $\pi^{(0)}$ refere-se ao primeiro estado do caminho π, isto é, o estado s_0, o $\pi^{(1)}$ refere-se ao segundo estado, e assim por diante. Além disso, a notação $\pi^{(0,k)}$ é utilizada para denotar o prefixo do estado s_k no caminho π (i.e., $s_0, s_1, s_2, \dots, s_k$) e a notação $\pi^{(k,\infty)}$ o sufixo s_k no caminho π (i.e., $s_k, s_{k+1}, s_{k+2}, \dots$).

Definição 2 (Caminhos em Estruturas de Kripke)**.** *Seja $\mathcal{K} = \langle S, I, L, \rightarrow \rangle$ uma estrutura de Kripke definida sobre um conjunto finito de proposições atômicas $\mathcal{AP}$. Um caminho em $\mathcal{K}$ a partir de uma estado $s \in S$ é uma sequência infinita de estados $\pi = s_0, s_1, s_2, s_3, \dots$ tal que $s_0 = s$ e $\forall i \in \mathbb{N}$, $(s_i, s_{i+1}) \in \rightarrow$.*

As estruturas de Kripke definem, apenas, estados e execuções do sistema, sem prover nenhuma forma de **causalidade** (SCHNEIDER, 2004). Isto significa dizer que as estruturas de Kripke não "explicam" porque o sistema está em um determinado estado, ou porque se moveu para um outro estado. Elas, apenas, coletam possíveis valores de diferentes variáveis que podem ocorrer nas execuções do sistema. Seja $V = \{v_1, v_2, \dots, v_n\}$ o conjunto de variáveis do sistema cujo domínio é o conjunto finito D (domínio de interpretação). A valoração de V é dada pela função que associa valores em D para cada $v \in V$. Dessa forma, o estado do sistema é descrito pela atribuição de valores a todas variáveis em V (i.e., o estado é uma função de valoração $s : V \rightarrow D$).

As proposições atômicas $p \in \mathcal{AP}$ tipicamente assumem a forma $v \circ d$, com $v \in V$ e $d \in D$, onde $\circ$ denota um operador relacional (e.g., $=$, $\leq$, $\geq$, e $\neq$). Em uma estrutura de Kripke, todo estado s é rotulado por um subconjunto de proposições atômicas $T \subseteq \mathcal{AP}$ que são verdadeiras naquele estado (i.e., $L(s) = T$). Dessa forma, dada uma valoração para V, uma fórmula lógica, que é verdadeira para essa valoração, pode ser escrita. Por exemplo, dados um conjunto de variáveis $V = \{v_1, v_2, v_3\}$, uma valoração $\langle v_1 \leftarrow 5, v_2 \leftarrow 6, v_3 \leftarrow 7 \rangle$ no estado $s_1 \in S$ em $\mathcal{K}$ (exemplo da Figura 2.4) e um conjunto de proposições atômicas $\mathcal{AP} = \{p_0, p_1, p_2\}$, tal que $p_0 \equiv v_1 \leq 5$, $p_1 \equiv v_2 = 6$ e $p_2 \equiv v_3 \geq 2$, é possível derivar uma fórmula lógica que assume valor verdade nesse estado, qual seja: $p_0 \vee (p_1 \wedge p_2)$. Nesse caso, como todas as proposições atômicas assumem valor verdade verdadeiro no estado s_1, o rótulo de s_1 é $L(s_1) = \{p_1, p_2, p_3\}$. É importante notar que essa fórmula lógica pode assumir valor verdade para outras valorações de V em outros estados (e.g., $s_0, s_2 \in S$),

logo, é possível utilizar fórmulas lógicas para descrever/caracterizar conjuntos de estados.

2.3.3 Lógicas Temporais para Especificação de Propriedades

As lógicas temporais foram desenvolvidas por filósofos com o objetivo de estudar a forma como o tempo é utilizado nos argumentos da linguagem natural (CLARKE JR.; GRUMBERG; PELED, 1999). Essas lógicas possuem operadores temporais que permitem expressar a noção de passado e futuro. Tipicamente, a interpretação de lógicas temporais se dá sobre estruturas de Kripke, as quais capturam a essência do comportamento de um sistema reativo. Dessa forma, dada uma estrutura de Kripke $\mathcal{K}$ e uma fórmula temporal φ, uma formulação geral para o problema de verificação de modelos consiste em verificar se φ é satisfeita ($\models$) na estrutura $\mathcal{K}$, como descrito em (2.1).

$$\mathcal{K} \models \varphi \tag{2.1}$$

As lógicas temporais divergem quanto ao modo como representam o tempo, existindo dois modelos básicos: **lógicas de tempo linear** e **lógicas de tempo ramificado**. Nas lógicas de tempo linear, o comportamento do sistema consiste num **conjunto de caminhos infinitos** (Figura 2.5a) que iniciam em um estado inicial. Por outro lado, nas lógicas de tempo ramificado, o comportamento do sistema é capturado por uma **árvore de computação infinita** (Figura 2.5b) de profundidade ilimitada que tem como raiz um estado inicial. Entretanto, é importante mencionar que essas lógicas se referem a sequência de estados que descrevem execuções do sistema (caminhos), não a intervalos ou instantes de tempo de forma explícita. Essa é uma característica importante, pois se deseja tratar comportamentos não determinísticos, os quais envolvem diversos caminhos. Nessa perspectiva, existem duas possibilidades: (i) o comportamento é dado por um conjunto de caminhos onde cada estado possui apenas um único sucessor; ou (ii) cada estado possui vários sucessores em termos de ramificação. Nas lógicas de tempo linear, a multiplicidade de comportamentos é expressa de forma implícita (i.e., através do conjunto de todos os caminhos lineares da estrutura). Por outro lado, nas lógicas de tempo ramificado, a multiplicidade de comportamentos pode ser especificada explicitamente por uma propriedade de todos os próximos estados ou por uma propriedade do próximo estado. Nas próximas subseções são apresentados dois exemplos de lógicas temporais: LTL (*Linear Temporal Logic*), de tempo linear; e CTL (*Computation Tree Logic*), de tempo ramificado. Para obter mais detalhes sobre LTL e CTL e sobre seu uso na verificação de modelos, é sugerido ao leitor uma consulta às seguintes referências (CLARKE JR.; GRUMBERG; PELED,

1999), (SCHNEIDER, 2004) e (BAIER; KATOEN, 2008).

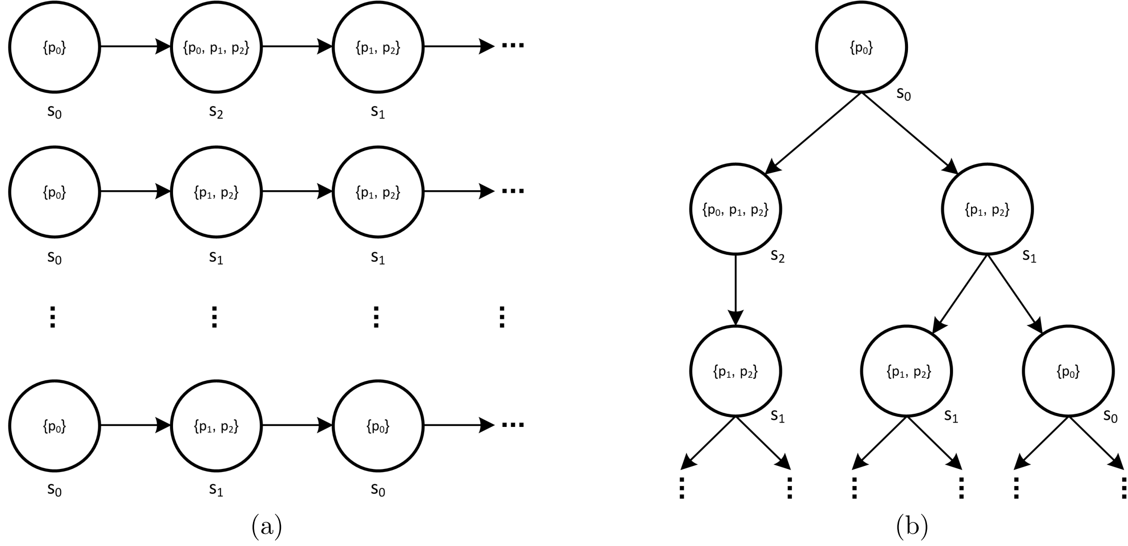


Figura 2.5: Conjunto de Caminhos Infinitos (2.5a) e Árvore de Computação Infinita (2.5b) da Estrutura de Kripke decrita na Figura 2.4.

A Lógica LTL

Tabela 2.1: Sintaxe de LTL.

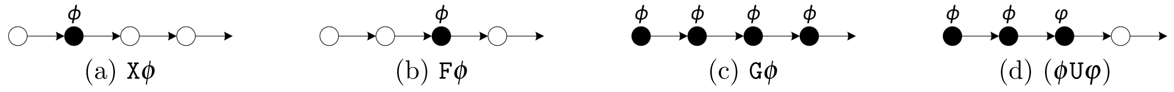
ϕ,φ	$::=$	$\texttt{true} \mid \texttt{false} \mid p \mid \neg\phi \mid \phi\wedge\varphi \mid \phi\vee\varphi \mid \phi\rightarrow\varphi \mid \phi\leftrightarrow\varphi \mid \texttt{X}\phi \mid \texttt{G}\phi \mid \texttt{F}\phi \mid (\phi\texttt{U}\varphi)$

A sintaxe de LTL é dada na Tabela 2.1, onde ϕ e φ são fórmulas, p uma proposição atômica, `true` e `false` são, respectivamente, os valores verdade verdadeiro e falso e os símbolos $\neg$ (negação), $\wedge$ (conjunção), $\vee$ (disjunção), $\rightarrow$ (implicação) e $\leftrightarrow$ (dupla implicação) tem o mesmo significado que os seus similares da lógica proposicional. Os operadores temporais `X` (*next time*), `F` (*in the future*), `G` (*globally*) e `U` (*until*) podem ser entendidos, intuitivamente, como descrito na Tabela 2.2 e ilustrados na Figura 2.6.

Tabela 2.2: Intuição para os Operadores Temporais de LTL.

$\texttt{X}\phi$	"no próximo estado do caminho, ϕ é verdadeira."
$\texttt{F}\phi$	"em algum estado do caminho, ϕ será verdadeira. "
$\texttt{G}\phi$	"em todo estado do caminho, ϕ é sempre verdadeira."
$(\phi\texttt{U}\varphi)$	"no caminho, ϕ é verdadeira até que φ passe a ser."

Seja $\mathcal{K} = \langle S, I, L, \rightarrow\rangle$ uma estrutura Kripke definida sobre um conjunto finito de proposições atômicas $\mathcal{AP}$, tal que $p \in \mathcal{AP}$ e φ fórmulas LTL. A notação $\mathcal{K} \models_{\pi} \phi$ é utilizada



(a) $\mathtt{X}\phi$ (b) $\mathtt{F}\phi$ (c) $\mathtt{G}\phi$ (d) $(\phi\mathtt{U}\varphi)$

Figura 2.6: Operadores Temporais de LTL

para denotar que a fórmula ϕ é satisfeita no caminho π da estrutura $\mathcal{K}$. A semântica formal de LTL é dada pela relação de satisfação $\models$ descrita na Tabela 2.3.

Tabela 2.3: Relação de Satisfação de LTL.

$\mathcal{K} \models_\pi p$	$\iff$	$p \in L(\pi^{(0)})$.
$\mathcal{K} \models_\pi \neg\phi$	$\iff$	NÃO $\mathcal{K} \models_\pi \phi$.
$\mathcal{K} \models_\pi \phi \wedge \varphi$	$\iff$	$\mathcal{K} \models_\pi \phi$ E $\mathcal{K} \models_\pi \varphi$.
$\mathcal{K} \models_\pi \phi \vee \varphi$	$\iff$	$\mathcal{K} \models_\pi \phi$ OU $\mathcal{K} \models_\pi \varphi$.
$\mathcal{K} \models_\pi \phi \rightarrow \varphi$	$\iff$	SE $\mathcal{K} \models_\pi \phi$ ENTÃO $\mathcal{K} \models_\pi \varphi$.
$\mathcal{K} \models_\pi \phi \leftrightarrow \varphi$	$\iff$	$\mathcal{K} \models_\pi \phi \rightarrow \varphi$ E $\mathcal{K} \models_\pi \varphi \rightarrow \phi$.
$\mathcal{K} \models_\pi \mathtt{X}\phi$	$\iff$	$\mathcal{K} \models_{\pi^{(1,\infty)}} \phi$ é verdadeiro.
$\mathcal{K} \models_\pi \mathtt{F}\phi$	$\iff$	Existir um $k \geq 0$ tal que $\mathcal{K} \models_{\pi^{(k,\infty)}} \phi$.
$\mathcal{K} \models_\pi \mathtt{G}\phi$	$\iff$	Para todo $k \geq 0$ é verdade que $\mathcal{K} \models_{\pi^{(k,\infty)}} \phi$.
$\mathcal{K} \models_\pi (\phi\mathtt{U}\varphi)$	$\iff$	Existir um $k \geq 0$ tal que $\mathcal{K} \models_{\pi^{(k,\infty)}} \varphi$ e para todo $0 \leq l < k$, $\mathcal{K} \models_{\pi^{(0,l)}} \phi$ é verdadeiro.

A Lógica CTL

CTL é uma lógica temporal de tempo ramificado que permite expressar propriedades sobre estados. A sintaxe de CTL é dada na Tabela 2.4, onde ϕ e φ são fórmulas, p uma proposição atômica, `true` e `false` são, respectivamente, os valores verdade verdadeiro e falso e os símbolos $\neg$ (negação), $\wedge$ (conjunção), $\vee$ (disjunção), $\rightarrow$ (implicação) e $\leftrightarrow$ (dupla implicação) tem o mesmo significado que os seus similares da lógica proposicional.

Tabela 2.4: Sintaxe de CTL.

$$\begin{aligned} \phi, \varphi \quad ::= \quad & \mathtt{true} \mid \mathtt{false} \mid p \mid \neg\phi \mid \phi \wedge \varphi \mid \phi \vee \varphi \mid \phi \rightarrow \varphi \mid \phi \leftrightarrow \varphi \\ & \mid \mathtt{EX}\phi \mid \mathtt{EG}\phi \mid \mathtt{EF}\phi \mid \mathtt{E}(\phi\mathtt{U}\varphi) \mid \mathtt{AX}\phi \mid \mathtt{AG}\phi \mid \mathtt{AF}\phi \mid \mathtt{A}(\phi\mathtt{U}\varphi) \end{aligned}$$

Em CTL, cada operador temporal é composto por um quantificador de caminho (`E`, para algum caminho, ou `A`, para todos os caminhos) seguido por um quantificador de estado (`X`, `F`, `G` e `U`). Uma intuição para o significado de cada um dos operadores temporais de CTL é dada na Tabela 2.5 e ilustrada na Figura 2.7.

Tabela 2.5: Intuição para os Operadores Temporais de CTL.

$\texttt{EX}\phi$	"existe um caminho tal que no próximo estado ϕ é verdadeira."
$\texttt{EF}\phi$	"existe um caminho tal que no futuro ϕ será verdadeira."
$\texttt{EG}\phi$	"existe um caminho tal que ϕ é sempre verdadeira."
$\texttt{E}(\phi\texttt{U}\varphi)$	"existe um caminho tal que ϕ é verdadeira até que φ passe a ser."
$\texttt{AX}\phi$	"para todo caminho, no próximo estado ϕ é verdadeira."
$\texttt{AF}\phi$	"para todo caminho, ϕ é verdadeira no futuro."
$\texttt{AG}\phi$	"para todo caminho, ϕ é sempre verdadeira."
$\texttt{A}(\phi\texttt{U}\varphi)$	"para todo caminho, ϕ é verdadeira até que φ passe a ser."

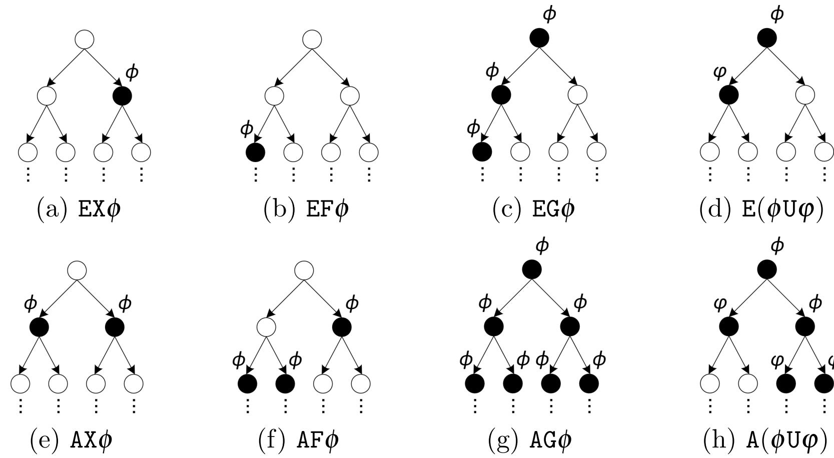


Figura 2.7: Operadores Temporais de CTL

Seja $\mathcal{K} = \langle S, I, L, \rightarrow \rangle$ uma estrutura Kripke definida sobre um conjunto finito de proposições atômicas $\mathcal{AP}$, tal que $p \in \mathcal{AP}$, $s \in S$ e ϕ e φ fórmulas CTL. A notação $\mathcal{K} \models_s \phi$ indica que a fórmula ϕ é satisfeita no estado s da estrutura $\mathcal{K}$. Dessa forma, a semântica formal de CTL é dada pela relação de satisfação $\models$ descrita na Tabela 2.6.

Usualmente, as propriedades dos sistemas reativos que se deseja verificar são divididas em dois tipos: (i) propriedades de segurança (*safety*); e (ii) propriedades de progresso (*liveness*). As propriedades de segurança buscam expressar que "nada ruim acontecerá" durante a execução do sistema (e.g., o sistema nunca entrará em *deadlock* e nunca dois processos entrarão na região crítica ao mesmo tempo). Por outro lado, as propriedades de progresso buscam expressar que, eventualmente, "algo bom acontecerá" durante a execução do sistema (e.g., cada processo entrará na região crítica e toda requisição será atendida).

Tabela 2.6: Relação de Satisfação de CTL.

$\mathcal{K} \models_s p$	$\iff$	$p \in L(s)$.
$\mathcal{K} \models_s \neg\phi$	$\iff$	NÃO $\mathcal{K} \models_s \phi$.
$\mathcal{K} \models_s \phi \wedge \varphi$	$\iff$	$\mathcal{K} \models_s \phi$ E $\mathcal{K} \models_s \varphi$.
$\mathcal{K} \models_s \phi \vee \varphi$	$\iff$	$\mathcal{K} \models_s \phi$ OU $\mathcal{K} \models_s \varphi$.
$\mathcal{K} \models_s \phi \rightarrow \varphi$	$\iff$	SE $\mathcal{K} \models_s \phi$ ENTÃO $\mathcal{K} \models_s \varphi$.
$\mathcal{K} \models_s \phi \leftrightarrow \varphi$	$\iff$	$\mathcal{K} \models_s \phi \rightarrow \varphi$ E $\mathcal{K} \models_s \varphi \rightarrow \phi$.
$\mathcal{K} \models_s \texttt{EX}\phi$	$\iff$	Existir um caminho π a partir de s tal que $\mathcal{K} \models_{\pi(1)} \phi$.
$\mathcal{K} \models_s \texttt{EF}\phi$	$\iff$	Existir um caminho π a partir de s tal que $\mathcal{K} \models_{\pi(k)} \phi$ para algum $k \geq 0$.
$\mathcal{K} \models_s \texttt{EG}\phi$	$\iff$	Existir um caminho π a partir de s tal que $\mathcal{K} \models_{\pi(k)} \phi$ para todo $k \geq 0$.
$\mathcal{K} \models_s \texttt{E}(\phi \texttt{U} \varphi)$	$\iff$	Existir um caminho π a partir de s tal que $\mathcal{K} \models_{\pi(k)} \varphi$ para algum $k \geq 0$ e $\mathcal{K} \models_{\pi(j)} \phi$ para todo $0 \leq j < k$.
$\mathcal{K} \models_s \texttt{AX}\phi$	$\iff$	Para todo caminho π a partir de s, $\mathcal{K} \models_{\pi(1)} \phi$ é verdadeiro.
$\mathcal{K} \models_s \texttt{AF}\phi$	$\iff$	Para todo caminho π a partir de s, $\mathcal{K} \models_{\pi(k)} \phi$ é verdadeiro para algum $k \geq 0$.
$\mathcal{K} \models_s \texttt{AG}\phi$	$\iff$	Para todo caminho π a partir de s, $\mathcal{K} \models_{\pi(k)} \phi$ é verdadeiro para todo $k \geq 0$.
$\mathcal{K} \models_s \texttt{A}(\phi \texttt{U} \varphi)$	$\iff$	Para todo caminho π a partir de s, $\mathcal{K} \models_{\pi(k)} \varphi$ é verdadeiro para algum $k \geq 0$ e $\mathcal{K} \models_{\pi(j)} \phi$ é verdadeiro para todo $0 \leq j < k$.

2.4 Programação por Restrições

Hentenryck e Saraswat (1996) definem programação por restrições como sendo o estudo dos sistemas computacionais baseados em **restrição** (*constraint*). A ideia por trás da programação por restrições é resolver problemas através da especificação de restrições (e.g., condições e propriedades) que devem ser satisfeitas pelas soluções. Uma restrição pode ser entendida, intuitivamente, como uma **condição a ser atendida** sobre um **espaço de possibilidades** (HENTENRYCK; SARASWAT, 1996). Nas próximas subseções são apresentadas as principais propriedades das restrições e como a programação por restrições é utilizada para modelar e resolver problemas combinatórios.

2.4.1 Restrições e suas Propriedades

Restrições matemáticas são, precisamente, relações especificáveis entre várias variáveis, onde cada variável possui uma valoração em um dado domínio. Restrições limitam a possibilidade de valores que as variáveis podem assumir, representando alguma informação (parcial) sobre as variáveis de interesse. Por exemplo, a condição de existência de um

triângulo[4] restringe a relação entre três variáveis (as medida dos lados do triângulo), sem informar, no entanto, os valores que elas podem assumir. Dessa forma, pode-se pensar nessa restrição como um conjunto de valores (possivelmente infinito) que satisfazem a condição especificada, no exemplo dos triângulos, esse conjunto é $\{ \langle 1,1,1 \rangle, \langle 1,2,2 \rangle, \ldots \}$.

As restrições surgem naturalmente na maioria das áreas da atividade humana, tais como: computação gráfica (para expressar coerência geométrica no caso de análise de cenas); processamento de linguagem natural (na construção de analisadores sintáticos eficientes); sistemas de banco de dados (para garantir e/ou restaurar a consistência dos dados); operações em problemas de busca (e.g., escalonamento e roteamento); biologia molecular (sequenciamento de DNA); aplicações de negócio (*option trading*); engenharia elétrica (para localização de falhas); e projeto de circuitos (para computar *layouts*) (HENTENRYCK; SARASWAT, 1996). Para Hentenryck e Saraswat (1996) as restrições são um meio natural para expressar as regularidades formais que dão sustentação aos mundos computacional e físico (natural ou projetado), incluindo suas abstrações matemáticas. Eles ainda argumentam que as restrições atendem, de forma natural, a um conjunto interessante de propriedades que são levadas em conta na resolução de vários problemas computacionais e do mundo físico:

- Restrições podem especificar informação parcial, elas não necessitam especificar os valores de suas variáveis;
- Restrições são não direcionais. Tipicamente, uma restrição em, por exemplo, três variáveis X, Y, Z pode ser usada para inferir uma restrição sobre X, dada as restrições sobre Y e Z, ou uma restrição sobre Y, dada as restrições sobre X e Z, e assim por diante;
- Restrições são declarativas, elas especificam quais relacionamentos devem ser atendidos, sem especificar qual procedimento computacional será usado para garantir esse atendimento;
- Restrições são aditivas, a ordem de imposição de restrições não importa, pois todas deverão ser atendidas no final (i.e., o resultado final é dado pela conjunção de todas as restrições);
- Restrições são, raramente, independentes, pois tipicamente restrições compartilham variáveis.

[4] A medida do lado de um triângulo deve ser menor que a soma das medidas dos outros dois lados e maior que o valor absoluto da diferença entre essas medidas.

2.4.2 Modelando com Restrições

A formulação e resolução de problemas combinatórios são os dois maiores objetivos do domínio de programação por restrições. Esse é o caminho essencial para resolver vários problemas industriais, tais como escalonamento, planejamento e projeto de calendários (HENTENRYCK; SARASWAT, 1996). O principal interesse da programação por restrições é propor ao usuário modelar o seu problema sem estar, obrigatoriamente, interessado na forma pela qual o problema será resolvido. A programação por restrições possibilita a resolução de problemas combinatórios modelados na forma de um Problema de Satisfação de Restrições – PSR (Definição 3).

Definição 3 (Problema de Satisfação de Restrições). *O Problema de Satisfação de Restrições é definido como uma tripla* $\langle X, D, C\rangle$*, onde:*

- $X = \{x_1, x_2, \ldots, x_n\}$ *é o conjunto de variáveis do problema;*
- D *é a função que associa cada* $x_i \in X$ *ao seu domínio* $D(x_i)$*, isto é, o conjunto de valores que pode ser atribuído à* x_i*;*
- $C = \{c_1, c_2, \ldots, c_m\}$ *é o conjunto de restrições. A restrição* $c_j \in C$ *é a relação definida sobre o subconjunto* $X^j = \{x^j_1, x^j_2, \ldots, x^j_{n^j}\} \subseteq X$ *de variáveis que restringe quais as possíveis tuplas de valores* $(v_1, v_2, \ldots, v_{n^j})$ *terão essas variáveis:*

$$(v_1, v_2, \ldots, v_{n^j}) \in c_j \cap (D(x^j_1) \times D(x^j_2) \times \cdots \times D(x^j_{n^j}))$$

Precisamente, solucionar um PSR é encontrar a tupla $v = (v_1, v_2, \ldots, v_n) \in D(X)$ sobre o conjunto de variáveis V que satisfaz a todas as restrições de C:

$$(v_1, v_2, \ldots, v_{n^j}) \in c_j \cap (D(x^j_1) \times D(x^j_2) \times \cdots \times D(x^j_{n^j})) \quad \forall j \in \{1, \ldots, m\}.$$

Para problemas de otimização, é necessário definir uma **função objetivo** $f : D(X) \rightarrow \mathbb{R}$. Nesse caso, uma solução ótima para o problema é dada pela tupla do PSR que minimiza (ou maximiza) a função f. Existe um gama de linguagens de programação que incorporam o paradigma de programação por restrições de forma nativa (e.g., Oz, SICStus, IF/Prolog e ECL^{i}PS^e) (FERNáNDEZ; HILL, 2000) e outras que permitem o uso dos conceitos a partir de bibliotecas, como é o caso da linguagem Java que possui bibliotecas de programação por restrição maduras com extenso uso na indústria (e.g., JaCoP[5] e Choco Solver[6]).

[5]http://www.jacop.eu/
[6]http://www.emn.fr/z-info/choco-solver

2.5 Considerações Finais

Neste capítulo foram apresentados os assuntos que servem como base para o entendimento do problema investigado nesta tese bem como para a solução proposta. Os assuntos abordados neste capítulo que estão relacionados ao problema investigado são os sistemas de software adaptativo sensível ao contexto (Seção 2.1) e o tratamento de exceção (Seção 2.2). Por outro lado, os assuntos relacionados à solução proposta compreendem a verificação de modelos (Seção 2.3) e a programação por restrições (Seção 2.4).

O enfoque dado aos sistemas de software adaptativo sensível ao contexto neste capítulo diz respeito a importantes questões de projeto que devem ser analisadas durante o desenvolvimento do sistema como suporte à tomada de decisão por parte dos projetistas. Particularmente, os aspectos de projeto descritos na modelagem de dimensões (Seção 2.1.4), caracterizam pontos chave que, se não forem bem compreendidos, podem levar a uma especificação de adaptação errônea. Além disso, embora o foco do tratamento de exceção abordado nessa tese seja o tratamento de exceção sensível ao contexto, o entendimento dos conceitos elementares dessa técnica consolidada da Engenharia de Software faz-se necessário. Contudo, os conceitos pertinentes ao tratamento de exceção sensível ao contexto são abordados no Capítulo 3.

Com relação a verificação de modelos, apenas uma visão geral sobre a técnica e o seu arcabouço formal foi apresentado, uma discussão sobre as vantagens de adotá-la, em detrimento de outras possíveis abordagens, é conduzida na próximo capítulo (Capítulo 3). Do mesmo modo, aspectos conceituais e matemáticos sobre a programação por restrições foram apresentados. Entretanto, a justificativa para a sua escolha como ferramenta de geração do espaço de estados a ser investigado durante o processo de verificação de corretude do comportamento excepcional sensível ao contexto, é apresentada no Capítulo 4.

3 Trabalhos Relacionados

Neste capítulo, os trabalhos relacionados com esta tese de doutorado são investigados em 2 (duas) categorias. A Seção 3.1 apresenta a primeira categoria que busca identificar os principais tipos de exceções contextuais, entender a forma como o comportamento excepcional funciona e identificar pontos do projeto do tratamento de exceção que são propensos a faltas de projeto. A Seção 3.2 cobre a segunda categoria, onde um levantamento entre os principais trabalhos sobre verificação formal de sistemas software adaptativos sensíveis que embasaram as decisões tomadas com respeito a solução proposta nesta tese é conduzido e uma discussão é oferecida. Por fim, na Seção 3.3 são apresentadas algumas considerações sobre os temas abordados neste capítulo e a sua relação com o método proposto nesta tese.

3.1 Tratamento de Exceção Sensível ao Contexto

Nesta seção são relatados, com base no levantamento bibliográfico conduzido durante esta tese, os principais tipos de exceções contextuais, uma discussão sobre os aspectos do comportamento excepcional sensível ao contexto que são relevantes para este trabalho e onde faltas de projeto podem ser cometidas.

3.1.1 Tipos de Exceções Contextuais

As exceções contextuais representam situações baseadas em contexto que são consideradas anormais em sistemas adaptativos sensíveis ao contexto para ambientes ubíquos. Essa classe de exceções são de natureza assíncrona e caracterizam possíveis situações de contexto cuja origem pode ser o ambiente do usuário, o hardware ou o próprio software. Nesta tese, as exceções contextuais foram agrupadas em 3 (três) categorias, são elas: exceções contextuais de **infraestrutura**, **invalidação de contexto** e **segurança**. Cada uma dessas categorias é apresentada nas seções subsequentes.

Exceções Contextuais de Infraestrutura

Esse tipo de exceção contextual está relacionada com a detecção situações de contexto que indicam que alguma falha de hardware ou software ocorreu no ambiente ubíquo. Damasceno et al. (2006) descrevem uma exceção desse tipo no cenário de um sistema de aquecimento "inteligente" onde a temperatura é ajustada automaticamente de acordo com as preferência das pessoas que o frequentam. Nesse cenário, a elevação da temperatura para um valor acima do limite superior estabelecido pelas preferências do usuário, pode indicar uma falha no sistema que controla o equipamento de aquecimento (falta de software) ou uma falha do próprio equipamento (falta de hardware). Para Damasceno et al. (2006), esse tipo de situação é excepcional porque pode colocar em risco a saúde de pessoas.

Outro exemplo desse tipo exceção é descrita por Mercadal et al. (2010). Trata-se de um sistema de gerenciamento de incêndio "inteligente". Este sistema analisa dados coletados por sensores de temperatura e fumaça e detecta a ocorrência de incêndios dentro de um edifício. Quando algum incêndio é detectado, o sistema, automaticamente, liga os aspersores, soa o alarme e abre as portas de incêndio. Entretanto, com base em um modelo de detecção de falhas proposto pelos autores, o sistema consegue identificar se os sensores em funcionamento conseguem cobrir toda a área física do prédio. O não atendimento desse critério de cobertura, significa que o sistema não consegue oferecer a garantia de que é possível detectar a existência de incêndio dentro da área física edifício. Para Mercadal et al. (2010), essa situação é considerada excepcional, pois pode colocar em risco a vida de pessoas.

Exceções Contextuais de Invalidação de Contexto

Esse tipo de exceção contextual está relacionada com a violação de determinadas condições de contexto durante a execução de alguma tarefa do sistema. Essas condições de contexto funcionam como invariantes da tarefa e quando violadas indicam uma situação de anormalidade. Por exemplo, Kulkarni e Tripathi (2010) descrevem esse tipo de exceção em uma aplicação de leitor de música sensível ao contexto. O leitor de música executa no dispositivo móvel do usuário, enviando um fluxo continuo de som para a saída de áudio do dispositivo. Entretanto, quando o usuário entra em uma sala vazia, o aplicativo busca por algum dispositivo de áudio disponível no ambiente e transfere o fluxo de som para esse dispositivo. Nessa aplicação, é estabelecido como contexto invariante a necessidade do usuário estar sozinho dentro da sala. Para Kulkarni e Tripathi (2010), a violação desse

invariante, chamada de invalidação do contexto, é considerado uma situação excepcional. Note que a detecção dessa exceção contextual depende de informações de contexto sobre a localização do usuário e o número de pessoal que estão na sala.

Outro exemplo de exceção contextual de invalidação de contexto é a exceção de vaga indisponível do *UbiParking*. Nessa exceção, o invariante contextual durante a tarefa de estacionar é caracterizado pela seguinte condição: a vaga reservada deve permanecer livre até que o veículo a ocupe. Quando o veículo está dentro do estacionamento e esse invariante contextual é violado, a exceção é, então, detectada e as medidas de tratamento adequadas são executadas.

Exceções Contextuais de Segurança

Esse tipo de exceção esta relacionada com situações de contexto que ajudam a identificar a violação de políticas de segurança (e.g., autenticação, autorização e privacidade) e demais situações que podem colocar em risco a integridade física e financeira dos usuários do sistema. Por exemplo, Kulkarni e Tripathi (2010) descreve esse tipo de exceção dentro de um sistema de informação sensível ao contexto de registro médico. Nessa aplicação, existem três usuários envolvidos: pacientes, enfermeiros e médicos. Os médicos podem fazer registros sobre seus pacientes e os enfermeiros podem ler e atualizar esses registros enquanto assistem aos pacientes. Entretanto, os enfermeiros só podem ter acesso aos registros se estiverem dentro da enfermaria em que o paciente se encontra e o médico responsável estiver presente. Nessa aplicação descrita por Kulkarni e Tripathi (2010), quando um enfermeiro tenta acessar os registros do paciente, porém não se encontra na mesma enfermaria que este paciente ou encontra-se na enfermaria, mas o médico responsável não está presente, caracteriza-se uma situação excepcional. Perceba que a detecção desse tipo de exceção depende das informações de contexto sobre a localização do paciente, do enfermeiro e do médico.

Outro exemplo de exceção contextual de segurança é a exceção de incêndio do *UbiParking*. A ocorrência desse tipo de exceção pode colocar em risco tanto a integridade física como financeira das pessoas. Perceba que o efeito da exceção contextual do sistema de aquecimento "inteligente", classificado como exceção contextual de infraestrutura, pode, também, colocar em risco a integridade física das pessoas. Porém, naquela situação, é possível tomar medidas por meio da intervenção humana para corrigir a causa da exceção, no caso uma falha de software ou de hardware, fazendo o sistema voltar a funcionar corretamente depois de algum período de tempo, eliminando a possibilidade de riscos

para as pessoas. Por outro lado, na exceção de incêndio do *UbiParking*, dificilmente será possível identificar automaticamente a causa real da exceção, cabendo ao sistema, apenas, tomar medidas compensatórias para reduzir os seus efeitos.

De uma maneira geral, do ponto de vista de severidade, as exceções contextuais de segurança tendem a ser mais severas que as de infraestrutura que, por sua vez, são mais severas que as de invalidação de contexto. Contudo, esse grau de severidade pode variar dependendo do contexto excepcional modelado por cada exceção contextual em um domínio particular, cabendo ao projetista estabelecer essa classificação.

3.1.2 O Comportamento Excepcional Sensível ao Contexto

A natureza distribuída e móvel dos sistemas ubíquos impõe limites aos modelos clássicos de tratamento de exceção. Isso é evidenciado de forma particular, para os sistemas ubíquos que são construídos sobre infraestruturas de comunicação desacoplada onde a identidade dos elementos que interagem nem sempre é conhecida pelas partes comunicantes. Tratar exceções contextuais requer um fluxo de controle excepcional sensível ao contexto, consistindo de reações baseadas em notificações envolvendo diversos elementos de software (DAMASCENO et al., 2006; MERCADAL et al., 2010). No restante desta seção, é descrito como trabalhos existentes na literatura abordam aspectos relacionados a definição, detecção, captura e tratamento de exceções contextuais.

Cho e Helal (2011) argumentam que a detecção de exceções contextuais está fortemente relacionada com a habilidade do sistema em observar e raciocinar sobre o seu contexto. Nessa perspectiva, os trabalhos de Damasceno et al. (2006), Kulkarni e Tripathi (2010) propõem o uso de expressões lógicas sobre variáveis de contexto (ou, simplesmente, expressões de contexto) para definir e detectar exceções contextuais. Damasceno et al. (2006) apresenta em seu trabalho um relato de experiência sobre a implementação de um mecanismo de tratamento de exceção em aplicações móveis colaborativas sensíveis ao contexto sobre a plataforma MoCA (*Mobile Collaboration Architecture*). MoCA é um sistema de middleware *publish-subscribe* que da suporte a desenvolvimento de aplicações colaborativas móveis, provendo serviços relacionados com o gerenciamento do contexto (i.e., aquisição, armazenamento, inferência e difusão). Nesse trabalho, Damasceno et al. (2006) introduzem o conceito de contexto excepcional. Para eles, contexto excepcional significa que uma ou mais situações contextuais denotam uma falha no ambiente físico ou lógico da aplicação. Já Kulkarni e Tripathi (2010) usam o conceito de invalidação de contexto para caracterizar a ocorrência de exceções contextuais. Eles propõem que

uma expressão de contexto seja definida como invariante para execução de determinadas atividades da aplicação. Dessa forma, se essa condição é violada durante a execução da atividade relacionada, uma exceção especial, chamada de `ContextInvalidationException`, é levantada.

Uma observação relevante sobre a importância do contexto para a caracterização de exceções contextuais é feita por Beder e Araujo (2011). Para eles, a interpretação sobre a excepcionalidade da situação é dada observando o contexto do sistema no momento em que a situação é detectada. Desse modo, para uma mesma situação, dependendo do contexto do sistema, esta pode ser considerada excepcional ou não. Além disso, mesmo quando a situação é reconhecida como excepcional, a seleção das medidas de tratamento apropriadas devem levar em consideração o contexto da aplicação. Para lidar com essa questão, Damasceno et al. (2006), Kulkarni e Tripathi (2010) propõem a criação de abstrações próprias que permitem associar, de forma explícita, medidas de tratamento (tratadores de exceção) à exceções contextuais. Tornando assim, não só a detecção, mas também a seleção de tratadores, sensível ao contexto.

Para endereçar esse problema, Damasceno et al. (2006) redefinem a noção de escopo de tratamento. Segundo eles, existem situações em que o tratamento de exceções requer que diversos dispositivos sejam envolvidos dependendo da sua região física ou do contexto. Dessa forma, para dar suporte ao tratamento sensível ao contexto, as exceções podem ser propagadas através de quatro níveis de escopos: um dispositivo, um grupo de dispositivos, um servidor e uma região. A busca pelo tratador adequado é feita pelos diferentes escopos de tratamento seguindo uma ordem pré-definida pela aplicação. Além disso, os tratadores também são sensíveis ao contexto. Dessa forma, um tratador só pode ser selecionado se (i) estiver dentro do escopo de propagação, (ii) for capaz de tratar aquele tipo de exceção e (iii) o seu contexto o permitir. Já Kulkarni e Tripathi (2010) propõem a associação de tratadores aos tipos de exceções contextuais que eles tem condição de tratar. Isso é feito por meio de uma abstração chamada de papel. No framework de Kulkarni e Tripathi (2010), os usuário são associados à papéis e cada papel está associado a um conjunto de obrigações. Para eles, em tempo de execução, a ocorrência de uma exceção contextual representa uma violação semântica de alguma obrigação estabelecida para um papel. Embora os tratadores estejam associados aos papéis, estes são selecionados, somente, se determinadas condições de contexto forem satisfeitas.

Ainda sobre a detecção de exceções contextuais, um aspecto importante observado por Lopes, Cacho e Batista (2007) está relacionado ao tratamento de exceções contex-

tuais concorrentes (i.e., exceções que são levantadas concomitantemente). Esse tipo de problema é endereçado nas abordagens de tratamento de exceção clássicas por meio de mecanismos de resolução de exceção (CAMPBELL; RANDELL, 1986), o qual avalia quais exceções foram levantadas concomitantemente e seleciona as medidas de tratamento mais apropriadas para realizar o tratamento. Tipicamente isso é feito pode meio de uma **função de resolução**. Campbell e Randell (1986), citam duas técnicas conhecidas para resolução de exceções concorrentes baseadas na existência de uma exceção universal, são elas: **hierarquia de exceções** e **árvore de exceções**. A primeira estratégia associa a cada exceção uma prioridade e, por conseguinte, considera a exceção universal como sendo aquela que possuir a maior prioridade. Quando várias exceções são levantadas, o mecanismo notifica apenas aquela que possui a maior prioridade dentre as levantadas. Por outro lado, a árvore de exceções representa na raiz da árvore a exceção universal e nas folhas as demais exceções. A árvore de exceções impõem um disciplina de ordem parcial às exceções. Com base nessa ordem, quando várias exceções são levantadas simultaneamente, o mecanismo notifica apenas a exceção que está na raiz da menor sub-árvore que engloba todas as exceções levantadas. Em seu trabalho, Lopes, Cacho e Batista (2007) estendem a técnica de árvore de exceções adicionando um mecanismo de composição de eventos contextuais que permite utilizar informações de contexto na resolução de exceções em uma árvore de exceção.

Uma questão importante sobre o tratamento de exceção sensível ao contexto está relacionada com a concepção da retomada do fluxo de controle após a atividade de tratamento. De um modo geral, as exceções contextuais são levantadas assincronamente e tratadas de forma colaborativa por elementos de software que podem estar executando sobre plataformas móveis distribuídas e interagindo de forma desacoplada. Cada elemento engajado na atividade de tratamento, possui seu próprio fluxo de controle excepcional interno. Além disso, algumas partes do tratamento podem envolver a participação de seres humanos, complicando ainda mais o cenário. Desse modo, não é trivial obter uma visão global sobre a retomada do fluxo de controle normal em um mecanismo de tratamento de exceção sensível ao contexto. Por exemplo, Damasceno et al. (2006) assumem que após a propagação das exceções para os escopos estabelecidos e o envio de notificações especificas pelos tratadores, o fluxo de controle excepcional termina. Já Kulkarni e Tripathi (2010), esperam que as ações de tratamento reestabeleçam a condição de contexto violada (i.e., a invariante), caso contrário a atividade é interrompida e outra é iniciada ou a sessão do usuário é desativada, resultando no encerramento do fluxo de controle excepcional.

De todo modo, se os detalhes sobre a reação do sistema forem abstraídos e for assumido

que o sistema é posto de volta em um estado seguro, é possível pensar na retomada do fluxo de controle em termos de alcançabilidade desse **estado do contexto seguro**. Nesse caso, o estado do contexto seguro diz respeito ao conjunto de informações contextuais que indicam que as medidas esperadas para o tratamento da exceção foram tomadas. Essas informações constituem uma espécie de "testemunha" de que o sistema encontra-se em um estado seguro. Dependendo do nível de confiabilidade requerido pelo sistema, essas propriedades de segurança podem ser mais ou menos rígidas para cada tipo de exceção contextual.

3.1.3 Propensão a Faltas de Projeto

Os trabalhos de Damasceno et al. (2006), Lopes, Cacho e Batista (2007), Kulkarni e Tripathi (2010), Mercadal et al. (2010), Beder e Araujo (2011), Cho e Helal (2011, 2012), Rocha e Andrade (2012) concordam que exceções contextuais podem ser definidas utilizando informações de contexto e que as condições de seleção dos tratadores também são determinadas com base na observação do contexto. Adicionalmente, embora a maneira como esses trabalhos abordam a questão da retomada do fluxo de controle seja ligeiramente diferente, o entendimento de que ações de contingência devem ser executadas para tratar uma exceção contextual é um ponto comum a todos os trabalhos, independente do sistema retomar o seu fluxo de controle normal ou interromper sua execução após o tratamento. Desse modo, faltas de projeto podem ser cometidas em três momentos do projeto do tratamento de exceção sensível ao contexto: (i) na especificação do contexto excepcional; (ii) na especificação do contexto que ajuda a selecionar as medidas de tratamento; e (iii) na especificação das ações de tratamento a serem executadas.

Na atividade (i), os projetistas especificam as condições de contexto que caracterizam as situações de anormalidade identificadas no sistema. Dessa forma, em tempo de execução, quando uma dessas situações são detectadas pelo mecanismo de TESC, diz-se que uma ocorrência da exceção contextual associada foi identificada. Por outro lado, na atividade (ii), os projetistas especificam o contexto que desencadeiam as ações de tratamento a serem executadas para mitigar a situação de excepcionalidade detectada. Nesse caso, dependendo do contexto corrente do sistema quando uma ocorrência excepcional é detectada, um conjunto de ações de tratamento podem ser mais apropriado do que outro para tratar aquela determinada ocorrência excepcional. Desse modo, faz parte do trabalho do projetista na atividade (ii), agrupar as ações de tratamento e estabelecer condições de contexto que ajudem o mecanismo de TESC, em tempo de execução, a selecionar o con-

junto de ações de tratamento mais apropriado para lidar com uma ocorrência excepcional em função do contexto corrente do sistema. Já a atividade (iii) diz respeito a especificação das medidas de tratamento que serão executadas para mitigar o problema relacionado a ocorrência excepcional e levar o sistema de volta a um estado seguro.

A falibilidade dos projetistas, o conhecimento parcial sobre a forma como o contexto do sistema evolui em tempo de execução, a inexistência de uma notação apropriada e a falta de suporte ferramental, tornam o projeto do TESC uma atividade extremamente propensa a faltas de projeto. Por exemplo, devido a negligência ou lapsos de atenção, contradições podem ser facilmente inseridas pelos projetistas durante a especificação das condições de contexto construídas nas atividades (i) e (ii) do projeto do TESC. Além disso, mesmo que os projetistas criem especificações livres de contradições, essas podem representar situações de contexto que nunca ocorrerão em tempo de execução devido a forma como o sistema e o seu contexto evoluem. Faltas de projeto como estas podem fazer com que o mecanismo de TESC seja mal configurado, comprometendo a sua confiabilidade em detectar as situações de anormalidade desejadas e selecionar as ações de tratamento adequadas para lidar com ocorrências excepcionais específicas.

Desse modo, considerando a atividade (i), se o contexto que caracteriza a situação excepcional for mal projetado, a exceção contextual pode nunca ser detectada pelo sistema em tempo de execução ou ser detectada em um momento impróprio (i.e., diferente daquele da intensão do projetista). Se a primeira situação acontecer, todo o esforço e trabalho de projeto e implementação do TESC terá sido em vão. Por outro lado, se a segunda situação ocorrer, os desdobramentos são ainda mais graves, pois o mecanismo de TESC passará a detectar ocorrências excepcionais de forma equivocada, levando o sistema a reagir de forma indesejada ou imprópria e, consequentemente, trazendo riscos aos seus usuários. Por exemplo, se o contexto excepcional da exceção de incêndio do *UbiParking* for mau projetado, para ambas as situações, as pessoas que estão no estacionamento passam a correr risco de vida.

Com respeito a atividade (ii), se o contexto de seleção das medidas de tratamento for mal projetado, pode ocorrer de uma exceção ser detectada, mas nunca ser tratada ou, dependendo do contexto do sistema no momento da detecção, o conjunto de medidas selecionadas não serem àquelas que o projetista realmente tinha em mente para lidar com aquela situação excepcional. Por exemplo, na exceção de incêndio do *UbiParking*, existem conjuntos de medidas destinadas a tratar a exceção de incêndio em função da localização do veículo dentro do estacionamento. Se, erroneamente, o conjunto de medidas destinadas

a tratar a exceção de incêndio quando o veículo encontra-se na entrada do estacionamento for selecionado para tratar a exceção que ocorre quando o veículo encontra-se dentro do pátio de vagas, pode levar o sistema a reagir de forma imprópria colocando em risco a vida do motorista.

Adicionalmente, existe outro tipo de falta de projeto que pode ser facilmente cometida por projetistas. Por exemplo, considere o projeto do TESC para uma exceção contextual em que as especificações das condições de contexto construídas nas atividades (i) e (ii) estejam livres de faltas de projeto. Observe que, mesmo nesse caso, pode ocorrer do projetista especificar a condição de contexto que caracteriza a situação de anormalidade e as condições de seleção das ações de tratamento de tal forma que estas nunca sejam satisfeitas, simultaneamente, em tempo de execução. Isso pode acontecer nos casos em que essas condições de contexto sejam contraditórias entre si ou que não seja possível para o sistema atingir um estado em que seu contexto satisfaça a ambas ao mesmo tempo.

Com respeito a atividade (iii), se as medidas de tratamento especificadas para tratar aquela situação excepcional forem mal projetadas, mesmo que a especificação de contexto excepcional e o contexto de seleção sejam bem especificados, as medidas de tratamento podem não ser executadas no momento esperado ou serem executadas equivocadamente, levando o sistema a uma situação imprópria. Por exemplo, na exceção de incêndio do *UbiParking*, se no conjunto de medidas destinadas a tratar a exceção de incêndio nos casos em que o veículo encontra-se na entrada do estacionamento existir uma medida de tratamento cuja ação seja algo do tipo “conduza o veículo até o pátio de vagas”, caracteriza uma situação em que a vida do usuário é posta em risco.

Desse modo, face a propensão à falta de projetos, uma abordagem rigorosa deve ser empregada pelos projetistas para que faltas de projeto sejam identificadas e removidas, antes que sejam propagadas até a fase de codificação. Contudo, não foi encontrado no escopo da revisão de literatura conduzido nesta tese trabalhos que busquem endereçar esse tipo de problema no projeto do TESC. Adicionalmente, essa tese considera que um modelo formal para a verificação do comportamento excepcional sensível ao contexto deve ser capaz de prover aos projetistas meios para identificar esses tipo de faltas de projeto logo nas fases iniciais do desenvolvimento, evitando ao máximo a necessidade de retrabalho.

3.2 Abordagens de Modelagem e Verificação

Tipicamente, em tempo de projeto, a especificação do comportamento adaptativo sensível ao contexto é dado por meio de regras de adaptação (ou regras de adaptação desencadeadas por contexto), uma variante do modelo ECA (*Event-Condition-Action*) (BALDAUF; DUSTDAR; ROSENBERG, 2007). Essas regras de adaptação são compostas por uma condição de guarda (antecedente) e um conjunto de ações associadas (consequente) (XU et al., 2012), algo semanticamente parecido com o descrito em (3.1). A condição de guarda descreve situações de contexto às quais o sistema deve reagir e, usualmente, são especificadas como expressões lógicas sobre variáveis de contexto (i.e., variáveis que guardam o estado das informações de contexto observadas). Já as ações, caracterizam a reação do sistema ao contexto detectado. Assim, quando uma condição de guarda de uma regra é satisfeita, o sistema reage executando as ações descritas no consequente daquela regra. Essa reação pode implicar na mudança de configuração do próprio software (sua estrutura e comportamento) e/ou do ambiente (físico ou lógico) em que este executa.

$$\texttt{Se}\ \langle contexto \rangle\ \texttt{Então}\, \{a_1, a_2, \ldots, a_n\} \tag{3.1}$$

Neste cenário, o projeto errôneo da especificação de adaptação pode levar o sistema (incluindo o seu ambiente), em tempo de execução, a uma configuração indesejada ou imprópria e, consequentemente, por computação, a uma posterior falha. Esse tipo de problema vem sendo tratado na literatura por diversos pesquisadores (ZHANG; CHENG, 2006; CUBO et al., 2009; HUANG et al., 2009; SAMA et al., 2010; LIU; XU; CHEUNG, 2013), os quais propõem o emprego de métodos formais para a realização de uma análise rigorosa sobre o comportamento adaptativo do sistema, permitindo a identificação e remoção de falas de projeto em estágios iniciais do desenvolvimento. Esse tipo de problema é bem próximo ao problema tratado nessa tese. Portanto, nas próximas seções são descritos trabalhos encontrados na literatura que buscam aplicar métodos formais para identificar faltas de projeto no comportamento adaptativo de sistemas adaptativos sensíveis ao contexto.

3.2.1 Trabalho de Zhang e Cheng (2006)

Zhang e Cheng (2006) apresentam em seu trabalho um estudo que ajuda a compreender a forma como os sistemas auto-adaptativos se comportam. Nesse estudo, eles discutem um conjunto de trabalhos existentes na literatura e capturam aspectos impor-

tantes do comportamento adaptativo. Estes aspectos comportamentais estão relacionados com as diferentes formas como o processo de adaptação pode ser realizado. Os autores chamam essas diferentes formas de adaptação de **semânticas de adaptação**. Com o intuito de facilitar o entendimento e a formalização dessas semânticas, Zhang e Cheng (2006) assuem que o modelo de comportamento dos sistemas auto-adaptativos é formado pela composição de um conjunto finito de comportamentos estacionários não adaptativos (i.e., uma sequência de estados e transições que levam a um estado estacionário) e adaptações entre esses comportamentos (i.e., transições entre comportamentos estacionários diferentes). Dessa forma, uma adaptação no modelo de comportamento do sistema pode ser compreendida como uma mudança que leva o sistema de um comportamento estacionário (**comportamento fonte**) para outro comportamento estacionário (**comportamento alvo**).

Com o objetivo de especificar propriedades sobre o comportamento adaptativo, Zhang e Cheng (2006) estendem a lógica temporal LTL (*Linear Temporal Logic*) adicionando o operador *adapt* ($\stackrel{\Omega}{\rightharpoonup}$), chamando esta extensão de A-LTL. A ideia por trás do operador é expressar restrições sobre sequências de estados conectadas (i.e., comportamentos estacionários compostos). Essa sequência de estados conectadas é representada por um caminho π, onde o prefixo (i.e., $\pi^{(0,i)}$) representa o comportamento fonte e o sufixo (i.e., $\pi^{(i+1,\infty)}$) o comportamento alvo. Desse modo, informalmente, um sistema que satisfaz a fórmula $\phi \stackrel{\Omega}{\rightharpoonup} \varphi$, com ϕ, Ω e φ fórmulas temporais, significa que inicialmente o sistema satisfaz ϕ (comportamento fonte). Em um certo estado, $s_i \in \pi$, ele deixa de ser restringido por ϕ e, no próximo estado $s_{i+1} \in \pi$, começa a satisfazer φ (comportamento alvo). A fórmula Ω é utilizada para restringir os estados que conectam os comportamentos fonte e alvo (i.e., Ω deve ser satisfeita em s_i e s_{i+1}).

Zhang e Cheng (2006) dividem o comportamento adaptativo em duas partes: (i) invariantes de adaptação (*adaptation invariants*) – que definem propriedades que devem ser válidas durante toda a execução do sistema; e (ii) variantes de adaptação (*adaptation variants*) – que definem propriedades que podem mudar durante a execução do sistema. As invariantes de adaptação são expressas como propriedades LTL convencionais (i.e., que são verificadas globalmente no modelo de comportamento do sistema). Por outro lado, as variantes de adaptação são definidas em função das semânticas de adaptação e buscam responder questões de projeto específicas sobre o comportamento adaptativo, tais como: Qual é o comportamento esperado após a adaptação? Quais restrições, se existirem, devem ser atendidas para que a adaptação ocorra? O comportamento fonte e o comportamento alvo podem alguma alguma sobreposição? O comportamento do sistema

(fonte ou alvo) deve ser restringido durante a adaptação? Se sim, qual tipo de restrição?

Com o intuito de prover subsídios para responder essas questões, Zhang e Cheng (2006) propõem a especificação de 3 (três) semânticas de adaptação: (i) **adaptação imediata** (*one point adaptation*) – nessa semântica, a adaptação ocorre quando o sistema atinge uma estado seguro (i.e., um estado onde suas obrigações tenham sido cumpridas) após a requisição de adaptação; (ii) **adaptação guiada** (*guided adaptation*) – nessa semântica não há necessidade do sistema atingir um estado seguro, entretanto, uma condição restritiva deve ser atendida pelo comportamento fonte como garantia de que a adaptação é possível; e (iii) **adaptação por sobreposição** (*overlap adaptation*) – nessa semântica o comportamento alvo inicia antes que o comportamento fonte tenha terminado, nesse caso, uma condição de guarda deve ser satisfeita como garantia de que o comportamento do sistema continue correto.

3.2.2 Trabalho de Cubo et al. (2009)

Um dos trabalhos que utilizam métodos formais no desenvolvimento de aplicações adaptativas sensíveis ao contexto é de Cubo et al. (2009). Eles argumentam que a introdução de clientes móveis e comportamento sensível ao contexto na composição de *Web Services* pode gerar faltas e inconsistências. Para resolver esse problema, eles propõem uma extensão no modelo de composição onde o contexto pode ser explicitado e um conjunto de propriedades podem ser verificadas automaticamente. Cubo et al. (2009) propõem um mapeamento da especificação da aplicação baseada em BPEL (*Business Process Execution Language* - linguagem usada para descrever composições de *Web Services*) para um modelo chamado CA-STS (*Context-Aware Symbolic Transition System*), um tipo de sistema de transição rotulado.

No CA-STS, cada *Web Service* que faz parte da composição possui uma máquina de estados interna. Dependendo do contexto da requisição, uma determinada sequência de transições no *Web Service* pode ser realizada. Dessa forma, é possível especificar todos os possíveis estados para cada *Web Service* e, consequentemente, fazer a análise da composição como um todo. Com o intuito de identificar faltas de projeto nesse modelo, Cubo et al. (2009) propõem um conjunto de propriedades comportamentais, são elas: determinismo, progresso de estado (*state liveness*), progresso de requisição/resposta e estados não bloqueantes (*non-blocking states*).

A propriedade de determinismo diz que em cada estado nos quais a computação pode seguir diferentes caminhos, as condições para a seleção das múltiplas requisições/respostas

devem ser mutuamente exclusivas. Já a propriedade de progresso de estado estabelece que se o contexto do estado é utilizado para realizar alguma transição, pelo menos uma das valorações possíveis das variáveis de contexto nesse estado deve possibilitar a ocorrência da transição. Por outro lado, a propriedade de progresso de requisição/resposta estabelece que se uma requisição/resposta é condicionada por uma condição de contexto, esta deve ser satisfeita. Por fim, a propriedade de estados não bloqueantes estabelece que independentemente de valores das variáveis de contexto, a comunicação deve sempre atingir um estado final. A violação dessas propriedades no modelo indicam a existência de faltas de projeto.

Com o intuito de identificar essas faltas de projeto de forma automática, Cubo et al. (2009) propõem um conjunto de algoritmos simbólicos baseados em OBDD (*Ordered Binary Decision Diagrams*) e disponibilizam uma ferramenta que provê uma implementação para estes e realiza a verificação do modelo de forma automática.

3.2.3 Trabalho de Sama et al. (2010)

O trabalho proposto por Sama et al. (2010) tenta endereçar as faltas causadas por regras de adaptação erradas e inconsistência de contexto. Em seu trabalho, Sama et al. (2010), adotam o modelo de aplicações adaptativas sensíveis ao contexto onde as aplicações são inicializadas em função do contexto. Dessa forma, quando um contexto de interesse está ativo uma determinada tarefa é executada automaticamente. Na sua solução, Sama et al. (2010) propõem a criação de uma máquina de estados finitos, chamada A-FSM (*Adaptation Finite-State Machine*), para modelar o comportamento das aplicações e servir como base para a verificação automática de propriedades e detecção de faltas de projeto.

Na A-FSM, cada estado representa uma configuração do ambiente e as transições entre estados são representadas por regras de adaptação. Cada regra de adaptação está associada a um predicado contextual (i.e., uma expressão de contexto). Quando o predicado é satisfeito (i.e., o contexto está ativo) a regra de adaptação é desencadeada, gerando mudanças no ambiente, o que caracteriza uma transição entre estados. Dessa forma, uma regra só é executada se o contexto estiver ativo e a aplicação estiver no estado esperado para que a adaptação ocorra. A partir desse modelo é possível verificar se as regras de adaptação foram bem especificadas (i.e., se elas são executadas apenas nos contextos certos e nos estados devidos). Contudo, é possível que mais de uma regra de adaptação esteja apta a executar em um dado momento. Para resolver esse problema, a A-FSM permite a definição de prioridade de execução entre as regras de adaptação. Assim, quando mais de

uma regra estiver apta para execução, apenas a regra de maior prioridade é executada.

Além disso, Sama et al. (2010) propõe um conjunto de propriedades comportamentais que ajudam a identificar determinados tipos de faltas de projeto na especificação das regras de adaptação, são elas: determinismo, progresso de regra (*rule liveness*), progresso de estado (*state liveness*), estabilidade e alcançabilidade. A propriedade de determinismo estabelece que para toda valoração das variáveis de contexto em um determinado estado só pode existir no máximo uma regra ativa. A violação dessa regra caracteriza a presença da falta de projeto, denominada de **ativação não determinística**, no modelo. Por outro lado, a propriedade de progresso de regra estabelece que para cada estado, deve existir pelo menos uma valoração para as variáveis de contexto que torne a regra ativa (i.e., satisfaça o seu predicado). A violação dessa propriedade indica a presença da falta de projeto, denominada de **predicado morto**, no modelo. Já a propriedade de progresso de estado, estabelece que para cada estado deve existir pelo menos uma regra ativa. Caso esse propriedade seja violada, indica que todas as regras de adaptação são regras mortas, logo, existe uma falta de projeto, denominada de **estado morto**, no modelo.

A propriedade de estabilidade diz que as regras não sistema não devem ser dependentes da quantidade de tempo que uma variável de contexto mantém o seu valor. Isso significa que não deve ocorrer o caso em que a mesma regra seja ativada e desencadeada no seu estado de origem e no seu estado de destino infinitamente. Caso essa situação ocorra, diz que o modelo possui uma falta de **corrida de adaptação** (*adaptation race*). Caso esse fenômeno ocorra de forma cíclica, diz-se que modelo possui uma falta de projeto de **adaptação cíclica**. Por outro lado, a propriedade de alcançabilidade diz que para cada estado deve existe uma sequencia de adaptações tal que este seja alcançado. A violação dessa propriedade indica a existência de uma falta de projeto particular no modelo denominada de falta de **estado não alcançável**. Por fim, para analisar todas essas propriedades, Sama et al. (2010) propõem um conjunto de algoritmos simbólicos baseados em OBDD implementados em uma ferramenta que realiza a análise do modelo de forma automática em busca de faltas de projeto.

3.2.4 Trabalho de Liu, Xu e Cheung (2013)

Em seu trabalho, Liu, Xu e Cheung (2013) propõem uma abordagem para melhorar a efetividade da A-FSM, proposta por Sama et al. (2010) e apresentada anteriormente neste capítulo. Liu, Xu e Cheung (2013) reportam em seu trabalho que, mesmo conseguindo identificar todas as faltas de projeto, a A-FSM gera muitos alarmes falsos (i.e., falsos

positivos), podendo comprometer a utilidade da técnica. Para endereçar esse problema, Liu, Xu e Cheung (2013) apresentam uma forma de derivar o que eles chamam de **modelo de domínio** e **modelo de ambiente** de aplicações adaptativas sensíveis ao contexto. O modelo de domínio consiste nas relações internas entre as proposições atômicas que constituem as regras de adaptação (i.e., proposições lógicas que caracterizam determinas informações de contexto), as quais capturam a lógica de adaptação interna. Por outro lado, o modelo de ambiente compreende a correlação externa entre as proposições atômicas, a qual captura determinadas características ocultas do ambiente de execução.

Segundo Liu, Xu e Cheung (2013), algumas correlações internas (e.g., inconsistências lógicas entre proposições atômicas) podem ser estaticamente derivadas a partir das regras de adaptação utilizando alguma ferramenta de resolução de problema de satisfação de restrições. Liu, Xu e Cheung (2013) definem esse tipo de restrições como **restrições determinísticas**, que podem ser utilizadas para filtrar faltas de projeto e reduzir o espaço de estados a ser explorado. Entretanto, segundo Liu, Xu e Cheung (2013), a identificação das correlações internas entre proposições atômicas não pode ser feita da mesma forma, uma vez que essas relações envolvem informações que vem do ambiente da aplicação, sendo necessário o uso de **restrições probabilísticas**. Esse tipo de restrição é derivada a partir da análise do histórico de informações coletadas sobre o ambiente de execução da aplicação, utilizando técnicas de mineração de dados e casamento de padrões (*pattern-matching*).

Aplicando a técnica proposta por Liu, Xu e Cheung (2013), os relatórios de faltas passam a conter apenas as verdadeiras faltas de projeto ou, caso o projetista queira, penas as faltas que ocorrem com mais frequência. Com esse tipo de relatório de faltas, os usuários podem identificar e rapidamente corrigir faltas de projeto nas suas regras de adaptação. Com o intuito de realizar a análise de forma automática, Liu, Xu e Cheung (2013) propõem uma ferramenta que implementa a A-FSM e a abordagem proposta, permitindo que os projetista verifiquem o mesmo conjunto de propriedades definidas por Sama et al. (2010): determinismo, progresso de regra (*rule liveness*), progresso de estado (*state liveness*), estabilidade e alcançabilidade.

3.2.5 Discussão dos Trabalhos

De um modo geral, os trabalhos apresentados buscam representar o comportamento adaptativo do sistema por meio de algum formalismo baseado em estados e transições. De posse desse modelo de comportamento, técnicas formais de análise (e.g., algoritmos sim-

bólicos e verificadores de modelos) são empregadas com o intuito de identificar eventuais faltas de projeto ou caracterizar pontos importantes onde faltas de projeto podem ocorrer. Entretanto, é importante mencionar que não foram encontrados, no escopo da revisão bibliográfica realizada, trabalhos que abordam a mesma problemática endereçada nesta tese, porém, os trabalhos descritos nas seções anteriores possuem um relação próxima à solução proposta nesta tese, por esse motivo receberam destaque.

Por exemplo, as propriedades estabelecidas por Cubo et al. (2009) e Sama et al. (2010) serviram de inspiração para a definição das propriedades comportamentais a serem verificadas sobre o comportamento excepcional sensível ao contexto estabelecidas nessa tese. Além disso, o uso de restrições como forma de eliminar problemas de inconsistências entre informações de contexto e construir o espaço de estados a ser explorado na verificação, foi uma das decisões tomadas com base nos trabalhos de Sama et al. (2010) e Liu, Xu e Cheung (2013). Adicionalmente, o entendimento dado por Zhang e Cheng (2006) de como lógicas temporais podem ser empregadas para especificar propriedades sobre o comportamento de sistemas adaptativos sensíveis ao contexto, serviu de guia para a especificação formal das propriedades propostas. Entretanto, é importante mencionar que todos os trabalhos, exceto o de Zhang e Cheng (2006) , são limitados com relação ao tipo de propriedades a serem verificadas. Por proporem seus próprios formalismos e implementarem ferramentas que analisam apenas um conjunto particular de propriedades, a sua extensão acaba sendo limitada. Com o intuito de tornar flexível a inserção de novas propriedades, o método proposto nesta tese optou pela adoção da técnica de verificação de modelos, descrita no Capítulo 2.

Contudo, a revisão bibliográfica sobre a utilização de métodos formais em sistemas adaptativos não se limitou aos trabalhos descritos nas seções anteriores. Trabalhos com os de Milner (2009) que busca estabelecer um modelo formal para representação de sistemas ubíquos foi estudado em detalhes e, mesmo não tendo sido utilizado na solução proposta nesta tese, serviu de embasamento teórico importante para o entendimento dos fundamentos da Computação Ubíqua enquanto paradigma computacional. Além disso, o trabalho de Siewe, Zedan e Cau (2011) que propõe uma álgebra de processo para modelar sistemas adaptativos sensíveis ao contexto, foi estudado e serviu como base para o entendimento de aspectos comportamentais desse tipo de sistema. Adicionalmente, os trabalhos de Sykes et al. (2008), Kramer e Magee (2009) foram úteis para entender como mapear requisitos e metas de aplicações auto-adaptativas para modelos de comportamento como LTS (*Labelled Transition System*). Por fim, a revisão sistemática conduzida por Weyns et al. (2012) sobre o emprego métodos formais em sistemas auto-adaptativos, onde diversos nichos de

aplicação de métodos formais nesse tipo de sistema são apresentados, aponta que a verificação de modelos é o segundo tipo de abordagem formal mais empregada no projeto de sistemas dessa natureza, ficando atrás apenas das abordagens formais utilizadas para fazer inferência e raciocínio dedutivo. Esse resultado, de certo modo, corrobora a escolha da Verificação de Modelos como abordagem formal adotada nesta tese.

3.3 Considerações Finais

Neste capítulo foram apresentados os trabalhos relacionados com esta tese de doutorado divididos em duas categorias. A primeira categoria, apresentada na Seção 3.1, descreve os principais tipos de exceções contextuais, a forma como o comportamento excepcional funciona e identifica pontos do projeto do tratamento de exceção que são propensos a faltas de projeto. Já a segunda categoria, apresentada na Seção 3.2, a discrição dos principais trabalhos sobre verificação de sistemas software adaptativos sensíveis, que dão embasamento às decisões tomadas com respeito a solução proposta nesta tese, são descritos e, ao final, uma discussão é oferecida. O próximo capítulo é dedicado a apresentação do CAEH✓, um método para verificação de modelos do tratamento de exceção sensível ao contexto proposto nesta tese.

4 O Método CAEH✓

Neste capítulo é apresentada a proposta desta tese, o CAEH✓[1], um método para verificação de modelos do tratamento de exceção sensível ao contexto. Na Seção 4.1, uma visão geral do método e das suas atividades é oferecida. A Seção 4.2 é dedicada a descrição da atividade de modelagem do comportamento excepcional sensível ao contexto e a sua representação no formato de uma estrutura de Kripke. Na Seção 4.3, um conjunto de propriedades comportamentais, estabelecidas sobre o comportamento excepcional sensível ao contexto, é apresentado como forma de identificar determinados tipos de faltas de projeto. Por fim, na Seção 4.4, algumas considerações a respeito do método são apresentadas.

4.1 Visão Geral do Método

O CAEH✓é um método para verificação de modelos do tratamento de exceção sensível ao contexto em sistemas de software adaptativo sensível ao contexto. O CAEH✓provê um conjunto de abstrações e convenções que permitem aos projetistas expressarem de forma intuitiva, porém rigorosa, o comportamento excepcional sensível ao contexto e mapeá-lo para uma estrutura de Kripke particular, formalismo apresentado no Capítulo 2 que serve de base para a técnica de verificação de modelos. Além disso, o CAEH✓oferece uma lista de propriedades comportamentais, a serem verificadas sobre o comportamento excepcional sensível ao contexto, com o intuito de descobrir a existência de determinados tipos de faltas de projeto.

Do ponto de vista de fluxo de atividades, o CAEH✓é exatamente igual ao processo de verificação de modelos (Capítulo 2, Seção 2.3.1). A diferença é que o CAEH✓propõe uma forma própria para a realização das atividades de modelagem e especificação. Para a atividade de modelagem, apresentada na Seção 4.2, o comportamento excepcional sensível ao contexto é modelado utilizando um conjunto de construtores próprios, os quais

[1] Acrônimo do inglês ***C**ontext-**A**ware **E**xception **H**andling **V**erification*

são detalhados na próxima seção. Para a atividade de especificação, apresentada na Seção 4.3, um conjunto de propriedades que permitem identificar um conjunto bem definido de faltas de projeto são apresentadas e formalizadas utilizando CTL (*Computation Tree Logic*). Entretanto, o fato do método conseguir representar o modelo de comportamento do tratamento de exceção sensível ao contexto como uma estrutura de Kripke, permite que outros tipos de propriedades comportamentais possam ser definidas pelos projetistas utilizando CTL.

É importante mencionar que está fora do escopo do CAEH✓modelar e verificar o comportamento adaptativo do sistema. Além disso, o CAEH✓assume uma certa independência entre os comportamentos adaptativo e excepcional sensível ao contexto. Particularmente, o método considera que o fluxo de controle excepcional sensível ao contexto tem início em um estado particular do comportamento adaptativo, onde uma exceção contextual é detectada. Além disso, um outro ponto onde os dois comportamentos voltam a se conectar, é após a atividade de tratamento, quando ocorre a retomada do fluxo de controle. Adicionalmente, o CAEH✓leva em consideração algumas decisões de projeto adotadas por trabalhos existentes na literatura (e.g., A-FSM (SAMA et al., 2010) e AFChecker (LIU; XU; CHEUNG, 2013) discutidos no Capítulo 3) que ajudam, por exemplo, a estabelecer o espaço de estados válidos a serem explorados e a evitar/reduzir a ocorrência de inconsistências semânticas entre as informações de contexto.

4.2 Atividade de Modelagem

Durante a modelagem do comportamento do tratamento de exceção sensível ao contexto, existem, basicamente, quatro grandes questões de projeto a serem pensadas: (i) a definição e a detecção de exceções contextuais; (ii) a seleção das medidas de tratamento; (iii) a execução das medidas de tratamento; e (iv) a retomada do fluxo de controle.

A primeira questão diz respeito à forma como o contexto pode ser empregado para definir e detectar uma exceção contextual. Já a segunda está relacionada com a escolha da estratégia de tratamento mais adequada para uma exceção contextual levantada em um determinado estado do sistema (incluindo o seu contexto). Por outro lado, a terceira está relacionada com a sequência de ações de tratamento que fazem tanto o estado do sistema quanto o seu contexto alterarem. Por último, diferente das anteriores, a quarta questão está relacionada com o estado do sistema após o término do tratamento.

A atividade de modelagem do comportamento excepcional sensível ao contexto tem

como objetivo tratar as questões mencionadas anteriormente. Entretanto, antes de apresentar a forma como esta é conduzida no CAEH✓na Seção 4.2.2, algumas abstrações e convenções, necessárias para a realização da modelagem, são apresentadas na Seção 4.2.1. Por fim, na Seção 4.2.3, o mapeamento do modelo de comportamento excepcional sensível ao contexto para uma estrutura de Kripke é formalmente apresentado.

4.2.1 Abstrações Elementares

Embora a modelagem e a verificação do comportamento adaptativo não seja o alvo do CAEH✓, o comportamento excepcional sensível ao contexto será combinado, em tempo de execução, com o comportamento adaptativo para formar o comportamento total do sistema. Por esse motivo, algumas decisões de projeto tomadas para a modelagem do comportamento adaptativo, devem ser observadas durante a atividade de modelagem do comportamento excepcional sensível ao contexto. Essas decisões de projeto incluem: (i) a forma de abstrair detalhes sobre a estrutura das informações de contexto; (ii) eliminar inconsistências semânticas entre as informações de contexto; (iii) estabelecer o espaço de estados válidos a serem explorados; e (iv) evitar que transições inconsistentes entre estados ocorram. Para lidar com essas questões, quatros abstrações elementares adotadas pelo método são apresentadas e formalmente definidas no decorrer desta subseção.

Proposições Contextuais

Como discutido no Capítulo 2, nos estágios iniciais do projeto de sistemas de software adaptativo sensível ao contexto, um dos principais esforços está na identificação das informações de contexto (i.e., informações internas e externas ao sistema) que podem influenciar o comportamento do sistema em desenvolvimento. Nesses estágios, é pertinente abstrair determinados detalhes sobre a forma como essas informações são obtidas ou a estrutura utilizada para representá-las, focando, principalmente, em identificar **como** e **quais** informações são relevantes para delinear o comportamento do sistema. Particularmente, na modelagem do comportamento excepcional sensível ao contexto, a ideia é buscar identificar quais informações são relevantes para a caracterização de contextos excepcionais e para o tratamento de exceções contextuais.

Desse modo, uma forma de encontrar essas informações é através da formulação de perguntas (e.g., "O veículo está em movimento?" e "Há vaga livre no estacionamento?") que ajudam a caracterizar as situações que podem vir à influenciar o comportamento do sistema em tempo de execução. Essas perguntas podem ser vistas como proposições

Tabela 4.1: Proposições Contextuais do *UbiParking*.

Proposições	**Significado**
`inMovement`	"O veículo está em movimento?"
`atParkEntrance`	"O veículo está na entrada do estacionamento?"
`atParkPlace`	"O veículo está no pátio de vagas do estacionamento?"
`atParkExit`	"O veículo está na saída do estacionamento?"
`hasSpace`	"Há vaga livre no estacionamento?"
`tempGEQ35`	"A temperatura no estacionamento está maior do que ou igual a 35 °C?"
`tempLEQ15`	"A temperatura no estacionamento está menor do que ou igual a 15 °C?"
`tempBt15And35`	"A temperatura no estacionamento está entre 15 °C e 35 °C?"
`hasSmoke`	"Há fumaça no estacionamento?"
`isSprinklerOn`	"Os aspersores estão ligados?"
`isHeatingSystemOn`	"O sistema de aquecimento está ligado?"
`isRefrigerationSystemOn`	"O sistema de refrigeração está ligado?"

lógicas, ou proposições atômicas, sobre o contexto do sistema, que recebem uma interpretação, verdadeira ou falsa, de acordo com o estado global do sistema. No CAEH✓, essas proposições são denominadas de **proposições contextuais** e compõem o conjunto $\mathcal{CP}$, que representa a base de conhecimento sobre a qual são realizadas inferências para derivar informações de contexto de mais alto nível ou caracterizar situações contextuais, excepcionais ou não, de interesse do sistema durante a sua execução como, por exemplo, "O veículo está estacionado?" ($\neg$`inMoviment` $\wedge$ `atParkPlace`) ou "O veículo está saindo do estacionamento?" (`inMoviment` $\wedge$ `atParkExit`). A Tabela 4.1 descreve um conjunto de proposições contextuais extraídas do sistema exemplo *UbiParking*.

Restrições Semânticas

Em tempo de execução, as proposições contextuais assumem uma deteminada valoração (verdadeiro ou falso) em função dos valores reais atribuídos às variáveis de contexto observadas pelo sistema. A quantidade de variáveis de contexto observadas, com diferentes domínios e faixas de valoração, podem causar um aumento exponencial no espaço de estados a ser explorado, inviabilizando a aplicação da técnica de verificação de modelos. Por este motivo, o método propõe abstrair o domínio real das variáveis de contexto e trabalhar, apenas, com as proposições contextuais, sem perda de expressividade. Contudo, embora essa abordagem ajude a reduzir o espaço de estados a ser explorado, viabilizando a adoção da técnica de verificação de modelos, a inconsistência semântica entre as proposições contextuais surge como um efeito colateral. No CAEH✓, essas inconsistências

semânticas entre proposições contextuais são resolvidas por meio da formulação de restrições globais sobre o conjunto $\mathcal{CP}$, denominadas de **restrições semânticas** (Definição 4).

Definição 4 (Restrições Semânticas)**.** *Uma restrição semântica é definida como uma fórmula lógica sobre o conjunto $\mathcal{CP}$ de proposições contextuais tal como descrito na gramática em (4.1), onde $p \in \mathcal{CP}$ é uma proposição contextual, ϕ e φ são fórmulas lógicas e $\neg$ (negação), $\wedge$ (conjunção), $\vee$ (disjunção), $\oplus$ (disjunção exclusiva), $\rightarrow$ (implicação) e $\leftrightarrow$ (dupla implicação) operadores lógicos.*

$$\phi, \varphi ::= p \,|\, \neg\phi \,|\, \phi \wedge \varphi \,|\, \phi \vee \varphi \,|\, \phi \oplus \varphi \,|\, \phi \rightarrow \varphi \,|\, \phi \leftrightarrow \varphi \tag{4.1}$$

Por exemplo, considere as seguintes proposições contextuais descritas na Tabela 4.1: `atParkPlace`, `atParkExit` e `atParkEntrance`. Observe que essas proposições possuem uma relação semântica particular. No *UbiParking*, o veículo, do ponto de vista espaço-temporal, só pode estar fora ou dentro do estacionamento em um dado instante. Caso ele esteja fora do estacionamento, as três proposições devem assumir valor verdade falso. Por outro lado, se o veículo estiver no estacionamento, ele só poderá estar em um dos seguintes lugares: na entrada (`atParkEntrance`), no pátio de vagas (`atParkPlace`) ou na saída (`atParkExit`) do estacionamento, mas não em mais de um local simultaneamente. Observe que esse tipo de relação semântica entre proposições contextuais podem ser expressas por meio de restrições, como discutido no Capítulo 2, Seção 2.4. Desse modo, para o exemplo dado, a seguinte restrição semântica pode ser derivada: (`atParkEntrance` ⊕ `atParkPlace` ⊕ `atParkExit`) ∨ (¬`atParkEntrance` ∧ ¬`atParkPlace` ∧ ¬`atParkExit`).

Estados do Contexto

No CAEH✓, o espaço de estados a ser explorado na atividade de verificação é obtido através do emprego da técnica de Programação por Restrições, apresentada no Capítulo 2, Seção 2.4. Desse modo, o problema de encontrar o conjunto $\mathcal{C}$ de estados do sistema pode ser reduzido a um Problema de Satisfação de Restrições (Capítulo 2, Seção 2.4.2, Definição 3). No CAEH✓, esse conjunto de estados é chamado de conjunto de **estados do contexto** (Definição 5).

Definição 5 (Estados do Contexto)**.** *Dados um conjunto finito de proposições contextuais $\mathcal{CP}$, um conjunto finito de restrições semânticas $\boldsymbol{G}$ sobre $\mathcal{CP}$ e uma função de valoração $\boldsymbol{D}$: $\mathcal{CP} \rightarrow \{\texttt{true}, \texttt{false}\}$, o conjunto de estados do contexto $\mathcal{C}$ é definido através da resolução*

do Problema de Satisfação de Restrições descrito pela tupla $\langle \mathcal{CP}, D, G \rangle$, tal que:

$$\mathcal{C} = \{(v_1, v_2, \ldots, v_n) \mid p_i \in \mathcal{CP}, (v_1, v_2, \ldots, v_n) \in \bigwedge_{g \in G} g \cap (D(p_1) \times D(p_2) \times \cdots \times D(p_n))\} \quad (4.2)$$

Pela Definição 5, cada estado $s \in \mathcal{C}$ é representado por uma tupla valorada de tamanho $|\mathcal{CP}|$ que satisfaz todas as restrições semânticas em G, que visa eliminar o problema de inconsistência semântica entre as proposições contextuais. Essa tupla valorada caracteriza o rótulo do estado s, designado por `label`(s). Além disso, como o tamanho do conjunto do domínio de interpretação das proposições contextuais ({`true`,`false`}) é fixo e igual a 2 (dois), o número de estados pode ser dado em função do número de proposições contextuais. Assim, no pior caso, onde nenhuma restrição é definida sobre $\mathcal{CP}$ ou as restrições definidas não restringem o espaço das soluções, o número de estados obtidos é de tamanho $|\mathcal{C}| = 2^{|\mathcal{CP}|}$. Consequentemente, o número de estados é, no máximo, $2^{|\mathcal{CP}|}$.

Restrições de Transição

Durante a execução do sistema, a evolução de algumas proposições contextuais (i.e., a forma como sua valoração muda com o decorrer da execução do sistema) segue uma lei de formação bem definida, a qual deve ser levada em consideração pelo projetista. Por exemplo, o tempo é sempre incrementado, a carga da bateria sempre diminui e as trajetórias devem obedecer às leis do espaço-tempo. Dessa forma, é necessário garantir que as transições entre estados atendam a este tipo de restrição. Com o objetivo de endereçar esse problema, o CAEH✓ propõe o conceito de restrição de transição (Definição 6).

Definição 6 (Restrição de Transição)**.** *Dado um conjunto finito de proposições contextuais $\mathcal{CP}$, uma restrição de transição é definida como uma tupla $\overset{\times}{\beta} = \langle \beta_\circ, \beta_\times \rangle$, onde $\beta_\circ$ e $\beta_\times$ são fórmulas lógicas sobre $\mathcal{CP}$ e caracterizam, respectivamente, a situação de contexto do estado de origem e a restrição que deve ser satisfeita pelo estado de destino de uma transição.*

Por exemplo, considere as seguintes proposições contextuais do *UbiParking* (Tabela 4.1): `atParkEntrance`, `atParkPlace` e `atParkExit`. No *UbiParking*, se o veículo estiver fora do estacionamento ($\neg$`atParkEntrance` $\wedge$ $\neg$`atParkPlace` $\wedge$ $\neg$`atParkExit`) em um dado estado, no próximo estado o veículo só poderá assumir duas possibilidades: (i) permanecer fora; ou (ii) ir para a entrada do estacionamento (`atParkEntrance`). Perceba, não é possível, do ponto de vista espaço-temporal, que o veículo alcance o pátio de vagas (`atParkPlace`) ou a saída (`atParkExit`) do estacionamento, antes de ter passado pela en-

trada do estacionamento (`atParkEntrance`). Essa garantia pode ser dada através de uma restrição de transição como a definida a seguir: $\overset{\times}{\boldsymbol{\beta}} = \langle \boldsymbol{\beta}_{\circ}, \boldsymbol{\beta}_{\times} \rangle$, com $\boldsymbol{\beta}_{\circ} = \neg\texttt{atParkEntrance} \wedge \neg\texttt{atParkPlace} \wedge \neg\texttt{atParkExit}$ e $\boldsymbol{\beta}_{\times} = \neg\texttt{atParkPlace} \wedge \neg\texttt{atParkExit}$.

4.2.2 Estruturando o Comportamento Excepcional

No CAEH✓, a estruturação do tratamento de exceção sensível ao contexto tem como objetivo estabelecer uma forma padrão para representar os aspectos comportamentais relacionados com a definição e a detecção de exceções contextuais, o agrupamento, seleção e execução das medidas de tratamento e a retomada do fluxo de controle. Com o objetivo de estruturar estes aspectos comportamentais, o CAEH✓propõe três abstrações, são elas: **exceções contextuais**, **casos de tratamento** e **escopos de tratamento**. Cada uma dessas abstrações são descritas e formalizadas no decorrer desta subseção.

Exceções Contextuais

No CAEH✓, uma exceção contextual é definida por um nome e uma fórmula lógica utilizada para caracterizar o seu contexto excepcional (Definição 7). Uma exceção contextual é detectada quando a fórmula $\boldsymbol{ecs}$ é satisfeita em algum dado estado do contexto. Nesse momento, diz-se que a exceção contextual foi levantada. Por convenção, dada uma exceção contextual $\boldsymbol{e} = \langle \boldsymbol{name}, \boldsymbol{ecs} \rangle$ as funções $\texttt{name}(\boldsymbol{e})$ e $\texttt{ecs}(\boldsymbol{e})$ são definidas para recuperarem os valores do nome ($\boldsymbol{name} \in \boldsymbol{e}$) e da especificação de contexto excepcional ($\boldsymbol{ecs} \in \boldsymbol{e}$), respectivamente.

Definição 7 (Exceção Contextual)**.** *Dado um conjunto finito de proposições contextuais* $\mathcal{CP}$*, uma exceção contextual é definida pela tupla* $\langle \boldsymbol{name}, \boldsymbol{ecs} \rangle$*, onde* $\boldsymbol{name}$ *é o nome da exceção contextual,* $\boldsymbol{ecs}$ *é uma fórmula lógica definida sobre* $\mathcal{CP}$ *que especifica o contexto excepcional de detecção.*

Por exemplo, considere as seguintes proposições contextuais apresentadas na Tabela 4.1: `hasSmoke` e `tempGEQ35`. Essas proposições contextuais podem ser utilizadas para definir uma exceção contextual (4.3) que caracteriza uma situação de incêndio no estacionamento no sistema exemplo *UbiParking*. Para esse exemplo, $\texttt{name}(\boldsymbol{fire}) =$ "*FireException*" e $\texttt{ecs}(\boldsymbol{fire}) = \texttt{hasSmoke} \wedge \texttt{tempGEQ35}$.

$$\boldsymbol{fire} = \langle \text{"}\textit{FireException}\text{"}, \texttt{hasSmoke} \wedge \texttt{tempGEQ35} \rangle \tag{4.3}$$

Casos de Tratamento

Como discutido no Capítulo 3, uma exceção contextual pode ser tratada de formas diferentes dependendo do contexto em que o sistema se encontra. Os **casos de tratamento** (Definição 8) definem as diferentes estratégias que podem ser empregadas para tratar uma exceção contextual em função do contexto do sistema. Um caso de tratamento é composto por uma condição de seleção e um conjunto de fórmulas lógicas que são utilizadas para descrever a situação de contexto esperada após a execução de cada ação (ou bloco de ações) de tratamento de forma sequencial. Por convenção, os constituintes de um caso de tratamento serão referenciados de agora em diante como **condição de seleção** e conjunto de **medidas de tratamento**, respectivamente.

Definição 8 (Caso de Tratamento)**.** *Dado um conjunto finito de proposições contextuais* $\mathcal{CP}$*, um caso de tratamento é definido como uma tupla* $\texttt{hcase} = \langle \alpha, \mathcal{H} \rangle$*, onde* α *é uma fórmula lógica definida sobre* $\mathcal{CP}$ *e* $\mathcal{H}$ *é um conjunto ordenado de fórmulas lógicas definidas sobre* $\mathcal{CP}$*.*

Por exemplo, considere a exceção contextual *fire* definida em (4.3) e as seguintes proposições contextuais descritas na Tabela 4.1: `inMovement`, `atParkEntrance`, `atParkPlace` e `atParkExit`. Diferentes casos de tratamento podem ser derivados para tratar essa exceção contextual dependendo da situação de contexto do veículo. Para a situação em que o veículo encontra-se na entrada do estacionamento, o seguinte caso de tratamento pode ser formulado: $\texttt{hcase}_0 = \langle \alpha_0, \mathcal{H}_0 \rangle$, onde $\alpha_0 = \texttt{inMovement} \wedge \texttt{atParkEntrance}$ e $\mathcal{H}_0 = \{\texttt{isSprinklerOn} \wedge (\neg\ \texttt{atParkEntrance} \wedge \neg\ \texttt{atParkPlace} \wedge \neg\ \texttt{atParkExit})\}$. O caso de tratamento $\texttt{hcase}_0$ é selecionado quando o veículo encontra-se entrando no estacionamento (α_0). Dessa forma, se ele é selecionado, o efeito esperado após a execução do tratamento ($\mathcal{H}_0$) é que o sistema atinja um estado em que os aspersores estejam ligados e o veículo esteja fora do estacionamento ($\texttt{isSprinklerOn} \wedge (\neg\ \texttt{atParkEntrance} \wedge \neg\ \texttt{atParkPlace} \wedge \neg\ \texttt{atParkExit})$).

Por outro lado, na situação em que o veículo encontra-se dentro do pátio de vagas do estacionamento, um possível caso de tratamento seria: $\texttt{hcase}_1 = \langle \alpha_1, \mathcal{H}_1 \rangle$, onde $\alpha_1 = \texttt{inMovement} \wedge \texttt{atParkPlace}$ e $\mathcal{H}_1 = \{\texttt{isSprinklerOn} \wedge \texttt{atParkExit}, \texttt{isSprinklerOn} \wedge (\neg\ \texttt{atParkEntrance} \wedge \neg\ \texttt{atParkPlace} \wedge \neg\ \texttt{atParkExit})\}$. No $\texttt{hcase}_1$, o veículo encontra-se em movimento dentro do pátio de vagas do estacionamento (α_1). Nesse caso de tratamento, duas medidas de tratamento são esperadas que ocorram sequencialmente ($\mathcal{H}_1$). A primeira consiste em levar o sistema a um estado em que o veículo esteja na saída do estacionamento e os aspersores encontrem-se ligados ($\texttt{isSprinklerOn} \wedge \texttt{atParkExit}$). Já a

segunda, consiste em levar o sistema a um estado no qual os aspersores continuem ligados e o veículo encontre-se fora do estacionamento (`isSprinklerOn` $\wedge$ ($\neg$ `atParkEntrance` $\wedge$ $\neg$ `atParkPlace` $\wedge$ $\neg$ `atParkExit`)). Este último estado é considerado o ponto onde a retomada do fluxo de controle acontece. A partir dele, o comportamento do sistema fica a cargo do modelo de comportamento normal (ou adaptativo) do sistema. Observe que, em ambos os casos, as medidas de tratamento representam, apenas, uma forma de compensação, uma vez que eliminar a causa da exceção, o incêndio, requer outras ações que vão além do escopo do sistema.

Escopos de Tratamento

Como apresentado no Capítulo 2, Seção 2.2, os tratadores de exceção encontram-se vinculados a áreas específicas do código do sistema onde exceções podem ocorrer. Essa estratégia ajuda a delimitar o escopo de atuação de um tratador durante a atividade de tratamento. No CAEH✓, o conceito de **escopos de tratamento** (Definição 9) é criado para delimitar a atuação dos casos de tratamento e estabelecer uma relação de precedência entre eles. Essa relação de precedência é essencial para resolver situações de sobreposição entre condições de seleção de casos de tratamento (i.e., situações em que mais de um caso de tratamento pode ser selecionado num mesmo estado do contexto). Dessa forma, o CAEH✓avalia primeiro o caso de tratamento de maior precedência, se este não tiver a sua condição de seleção satisfeita, o próximo caso de tratamento com maior precedência é avaliado, e assim por diante.

Definição 9 (Escopo de Tratamento)**.** *Dado um conjunto finito de proposições contextuais* $\mathcal{CP}$*, um escopo de tratamento é definido pela tupla* $\langle e, \texttt{HCASE}\rangle$*, onde* e *é uma exceção contextual e* `HCASE` *é um conjunto ordenado de casos de tratamento para a exceção* e*.*

A noção de conjunto ordenado, mencionado na Definição 9, está relacionada com a existência de uma relação de ordem entre os casos de tratamento. Essa relação permite estabelecer a ordem de precedência em que cada caso de tratamento será avaliado quando uma exceção contextual for levantada. No CAEH✓, a ordem de avaliação utilizada leva em consideração a posição ocupada por cada caso de tratamento dentro do conjunto `HCASE`. Portanto, para os casos de tratamento $\texttt{hcase}_i$ e $\texttt{hcase}_j$, se $i < j$, então $\texttt{hcase}_i$ tem precedência sobre $\texttt{hcase}_j$ (i.e., $\texttt{hcase}_i \prec \texttt{hcase}_j$). No entanto, essa relação de ordem não é fixa, porém obrigatória, podendo ser alterada pelo projetista com o propósito de obter algum tipo benefício. Por fim, levando em consideração a exceção contextual *fire* (4.3) e os dois casos de tratamento $\texttt{hcase}_0$ e $\texttt{hcase}_1$, descritos na subseção anterior, o

seguinte escopo de tratamento pode ser derivado: $\langle fire, \{\texttt{hcase}_0, \texttt{hcase}_1\}\rangle$. Observe que, nesse exemplo, a precedência entre os casos de tratamento é dada pela ordem dos índices.

4.2.3 Derivando a Estrutura de Kripke

Como apresentado no Capítulo 2, Seção 2.3.2, uma estrutura de Kripke é uma tupla $\mathcal{K} = \langle S, I, L, \rightarrow\rangle$ definida sobre um conjunto finito de proposições atômicas $\mathcal{AP}$. Desse modo, o processo de derivação de uma estrutura de Kripke consiste em estabelecer os elementos que a constituem, observando todas as restrições impostas pela sua definição (Seção 2.3.2, Definição 1), quais sejam: (i) o conjunto S de estados deve ser finito; e (ii) a relação de transição $\rightarrow$ deve ser total. Ao longo desta seção são descritos os procedimentos adotados pelo CAEH✓para obter cada um dos constituintes da estrutura de Kripke que representa o comportamento excepcional sensível ao contexto, chamada de $\mathcal{EK}$.

Proposições, Estados e a Função de Rótulos

O conjunto $\mathcal{AP}$ de proposições atômicas sobre o qual $\mathcal{EK}$ é definida, é formado pelo conjunto $\mathcal{CP}$ de proposições contextuais (i.e., $\mathcal{AP} = \mathcal{CP}$). Por outro lado, o conjunto S de estados de $\mathcal{EK}$ é obtido a partir dos conjuntos $\mathcal{CP}$ de proposições contextuais, $\mathcal{G}$ de restrições semânticas estabelecidas sobre $\mathcal{CP}$ e Γ de escopos de tratamento definidos para o sistema. A forma como essa derivação ocorre é descrita no pseudocódigo do Algoritmo 1. O objetivo desse algoritmo é adicionar uma nova restrição ao conjunto de restrições semânticas $\mathcal{G}$. Essa restrição adicional leva em consideração a especificação do contexto excepcional de cada exceção contextual, o critério de seleção e as medidas de tratamento de todos os casos de tratamento durante a geração dos estados.

Desse modo, no Algoritmo 1, para cada escopo de tratamento pertencente ao conjunto Γ (linha 2), é selecionado a especificação de contexto excepcional da exceção contextual associada ao escopo de tratamento (linha 3) e operado de forma disjuntiva com a fórmula φ (linha 4). Para cada caso de tratamento (linha 5), φ é operada de forma disjuntiva com o critério de seleção do caso de tratamento (linha 6) e com todas as medidas de tratamento associadas (linha 8). Ao final, φ é adicionada ao conjunto $\mathcal{G}$ de restrições semânticas (linha 12) e então o retorno consiste no conjunto finito de estados de contexto gerados a partir da resolução de um Problema de Satisfação de Restrições (linha 13). Além disso, é importante mencionar que essa restrição adicional não prejudica a construção do modelo, por outro lado, ajuda a reduzir o número de estados a ser explorado no processo de verificação. Isso é evidenciado pelo fato de que essa restrição adicional é uma disjunção ($\vee$) que é composta

Algoritmo 1 Geração do Conjunto S de Estados de $\mathcal{EK}$.

1: **function** EstadosEK($\mathcal{CP}, \mathcal{G}, \Gamma$)
2: $\varphi =$ `false` *//Uma fórmula lógica.*
3: **for all** $\langle e, \texttt{HCASE} \rangle \in \Gamma$ **do**
4: $\varphi = \varphi \vee \texttt{ecs}(e)$
5: **for all** $\langle \alpha, \mathcal{H} \rangle \in \texttt{HCASE}$ **do**
6: $\varphi = \varphi \vee \alpha$
7: **for all** $h \in \mathcal{H}$ **do**
8: $\varphi = \varphi \vee h$
9: **end for**
10: **end for**
11: **end for**
12: $\mathcal{G} = \mathcal{G} \cup \varphi$
13: **return** $PSR\langle \mathcal{CP}, \{\texttt{true}, \texttt{false}\}, \mathcal{G} \rangle$ *//PSR - Problema de Satisfação de Restrições*
14: **end function**

de forma conjuntiva ($\wedge$) com as demais restrições em $\mathcal{G}$. Logo, os estados gerados serão apenas os estados que satisfazem as restrições semânticas (que é mandatório) e que são relevantes para a modelagem do comportamento excepcional sensível ao contexto.

Dados um conjunto $\mathcal{E}$ de exceções contextuais e um conjunto $\mathcal{S}$ de estados do contexto gerado a partir do Algoritmo 1, o conjunto I de estados iniciais de $\mathcal{EK}$ é dado pelo Algoritmo 2. De forma direta, esse algoritmo define todos os estados onde exceções contextuais podem ser levantadas como estados iniciais.

Algoritmo 2 Geração do Conjunto I de Estados Iniciais de $\mathcal{EK}$.

1: **function** EstadosIniciaisEK($\mathcal{E}, \mathcal{S}$)
2: $\mathcal{A} = \emptyset$ *//Conjunto auxiliar de estados.*
3: **for all** $e \in \mathcal{E}$ **do**
4: **for all** $s \in \mathcal{S}$ **do**
5: **if** $\texttt{label}(s) \models \texttt{ecs}(e)$ **then**
6: $\mathcal{A} = \mathcal{A} \cup s$
7: **end if**
8: **end for**
9: **end for**
10: **return** $\mathcal{A}$
11: **end function**

Já a função de rótulos L de $\mathcal{EK}$ é dada a partir da valoração de cada estado do contexto do conjunto S, anteriormente gerado pelo Algoritmo 1. O Algoritmo 3, descreve o processo de construção da função de rótulo L.

Algoritmo 3 Geração dos Rótulos L de $\mathcal{EK}$.

```
1: function RotulosEK(𝒮)
2:     ℒ = ∅  //Conjunto auxiliar de rótulos.
3:     for all s ∈ 𝒮 do
4:         ℒ = ℒ ∪ label(s)
5:     end for
6:     return ℒ
7: end function
```

Relação de Transição

A relação de transição $\rightarrow$ de $\mathcal{EK}$ representa a sequência de ações realizadas durante a atividade de tratamento para cada exceção contextual detectada e tratada. Essas transições entre estados iniciam em um estado excepcional (i.e., onde uma exceção contextual é detectada) e terminam em um estado caracterizado pela última medida de tratamento de algum caso de tratamento selecionado para tratar aquela exceção. O Algoritmo 4 descreve como as transições em $\mathcal{EK}$ são geradas.

No Algoritmo 4, para cada estado s em $\mathcal{S}$ (linha 5) e para cada escopo de tratamento em Γ (linha 6) cuja exceção e associada ao escopo de tratamento pode ser levantada no estado s (i.e., $\texttt{label}(s) \models \texttt{ecs}(e)$) é aplicado o processo de tratamento. Para isso, é selecionado o primeiro caso de tratamento capaz de tratar a exceção levantada no escopo de tratamento corrente (linhas 7 à 12). Após isso, transições entre esse estado (estado excepcional) e outros estados que satisfazem a primeira medida de tratamento do caso de tratamento selecionado ($\mathcal{HAUX}[0]$) é feita por meio de uma chamada ao Algoritmo 5 (linha 13), que leva em consideração o conjunto $\mathcal{TC}$ de restrições de transição. Antes de continuar com o processo de tratamento, um teste é feito para verificar se alguma transição foi realizada (i.e., se foi possível conectar o estado excepcional s aos estados caracterizados por $\mathcal{HAUX}[0]$ sob as restrições de $\mathcal{TC}$). Em caso positivo, o processo se repete para cada par de medidas de tratamento por meio de chamadas ao Algoritmo 6 (linha 17). Por fim, antes de retornar o conjunto de relações de transição (linha 27), é feita uma chamada ao Algoritmo 8, que adiciona uma auto-transição (transição de loop) nos estados terminais (i.e., nos estados que não possuem sucessores) para garantir a restrição de totalidade da relação de transição imposta pela definição de estrutura de Kripke.

Os algoritmos 5 e 6 são responsáveis por gerar relações de transição. O Algoritmo 5 recebe como parâmetros um estado s, uma fórmula lógica α, um conjunto de estados $\mathcal{S}$ e um conjunto de restrições de transição $\mathcal{TC}$. O objetivo desse algoritmo é estabelecer relações de transição que ligam o estado s a todos os estados que satisfazem a fórmula

Algoritmo 4 Geração da Relação $\rightarrow$ de Transição de $\mathcal{EK}$.

1: **function** TRANSICAOEK$(\Gamma, \mathcal{S}, \mathcal{TC})$
2: $\mathcal{TR} = \emptyset$ *//Conjunto auxiliar de relações de transição.*
3: $\mathcal{TAUX} = \emptyset$ *//Conjunto auxiliar de relações de transição.*
4: $\mathcal{HAUX} = \emptyset$ *//Conjunto auxiliar de casos de tratamento.*
5: **for all** $s \in \mathcal{S}$ **do**
6: **for all** $\langle e, \texttt{HCASE} \rangle \in \Gamma \mid \texttt{label}(s) \models \texttt{ecs}(e)$ **do**
7: **for all** $\langle \alpha, \mathcal{H} \rangle \in \texttt{HCASE}$ **do**
8: **if** $\texttt{label}(s) \models \alpha$ **then**
9: $\mathcal{HAUX} = \mathcal{H}$
10: **break**
11: **end if**
12: **end for**
13: $\mathcal{TAUX} =$ GERATRANSICAOI$(s, \mathcal{HAUX}[0], \mathcal{S}, \mathcal{TC})$
14: **if** $\mathcal{TAUX} \neq \emptyset$ **then**
15: $\mathcal{TR} = \mathcal{TR} \cup \mathcal{TAUX}$
16: **for** $i = 1; i < |\mathcal{HAUX}|; i{+}{+}$ **do**
17: $\mathcal{TAUX} =$ GERATRANSICAOII$(\mathcal{HAUX}[i-1], \mathcal{HAUX}[i], \mathcal{S}, \mathcal{TC})$
18: **if** $\mathcal{TAUX} \neq \emptyset$ **then**
19: $\mathcal{TR} = \mathcal{TR} \cup \mathcal{TAUX}$
20: **else**
21: **break**
22: **end if**
23: **end for**
24: **end if**
25: **end for**
26: **end for**
27: **return** ADICIONALOOPEMTERMINAL$(\mathcal{S}, \mathcal{TR})$
28: **end function**

α. Desse modo, para cada estado r em $\mathcal{S}$ (linha 3), se α é satisfeita em r, a relação de transição (s,r) é adicionada num conjunto auxiliar (linha 5). Ao final, antes de retornar o conjunto de relações de transição, uma chamada ao Algoritmo 7 (linha 8), responsável por remover as relações de transição que violam as restrições de transição estabelecidas, é feita.

Algoritmo 5 Geração de Transições a Partir de um Estado e uma Fórmula.

```
1: function GeraTransicaoI(s, α, S, TC)
2:     TR = ∅  //Conjunto auxiliar de relações de transição.
3:     for all r ∈ S do
4:         if label(r) ⊨ α then
5:             TR = TR ∪ (s, r)
6:         end if
7:     end for
8:     if TR ≠ ∅ then
9:         return RemoveTransicaoInvalida(TR, TC)
10:    else
11:        return ∅
12:    end if
13: end function
```

Diferente do anterior, o Algoritmo 6 recebe como parâmetros duas fórmulas α e β, um conjunto de estados $\mathcal{S}$ e um conjunto de restrições de transição $\mathcal{TC}$. Nesse algoritmo, o objetivo é estabelecer relações de transição entre os estados que satisfazem a fórmula α (origem) e os estados que satisfazem a fórmula β (destino). Desse modo, é criado um conjunto com todos os estados que satisfazem α (linhas 5-9) e um conjunto com todos os estados que satisfazem β (linhas 10-14). Depois disso, para cada estado f de origem e cada estado t de destino, a transição (f,t) é guardada em um conjunto auxiliar (linhas 15-19). Por fim, antes de retornar o conjunto de relações de transição, uma chamada ao Algoritmo 7 (linha 8), responsável por remover as relações de transição que violam as restrições de transição estabelecidas.

O Algoritmo 7 recebe como parâmetros um conjunto $\mathcal{TR}$ de relações de transição e um conjunto $\mathcal{TC}$ de restrições de transição. Desse modo, pra cada restrição de transição $\langle\beta_\circ,\beta_\times\rangle$ em $\mathcal{TC}$ (linha 2) e para cada relação de transição (s,t) em $\mathcal{R}$ (linha 3) se o estado de origem da relação de transição s satisfaz a condição de seleção da restrição de transição $\beta_\circ$ e o estado de destino da transição t viola a restrição de chegada $\beta_\times$ da restrição de transição (linha 4), essa transição é removida de $\mathcal{TR}$ (linha 5). Por fim, o novo conjunto $\mathcal{TR}$ é retornado (linha 9).

Por outro lado, o Algoritmo 8 recebe como parâmetros um conjunto $\mathcal{S}$ de estados e

Algoritmo 6 Geração de Transições a Partir de duas Fórmulas.

1: **function** GeraTransicaoII($\alpha, \beta, \mathcal{S}, \mathcal{TC}$)
2: $\mathcal{TR} = \emptyset$ *//Conjunto auxiliar de relações de transição.*
3: $\mathcal{F} = \emptyset$ *//Conjunto auxiliar de estados de origem.*
4: $\mathcal{T} = \emptyset$ *//Conjunto auxiliar de estados de destino.*
5: **for all** $s \in \mathcal{S}$ **do**
6: **if** $\texttt{label}(s) \models \alpha$ **then**
7: $\mathcal{F} = \mathcal{F} \cup s$
8: **end if**
9: **end for**
10: **for all** $s \in \mathcal{S}$ **do**
11: **if** $\texttt{label}(s) \models \beta$ **then**
12: $\mathcal{T} = \mathcal{T} \cup s$
13: **end if**
14: **end for**
15: **if** $\mathcal{F} \neq \emptyset \wedge \mathcal{T} \neq \emptyset$ **then**
16: **for all** $f \in \mathcal{F}$ **do**
17: **for all** $t \in \mathcal{T}$ **do**
18: $\mathcal{TR} = \mathcal{TR} \cup (f, t)$
19: **end for**
20: **end for**
21: **return** RemoveTransicaoInvalida($\mathcal{TR}, \mathcal{TC}$)
22: **else**
23: **return** $\emptyset$
24: **end if**
25: **end function**

Algoritmo 7 Remove Transições Inválidas.

1: **function** RemoveTransicaoInvalida($\mathcal{TR}, \mathcal{TC}$)
2: **for all** $\langle \beta_\circ, \beta_\times \rangle \in \mathcal{TC}$ **do**
3: **for all** $(s, t) \in \mathcal{TR}$ **do**
4: **if** $\texttt{label}(s) \models \beta_\circ \wedge \texttt{label}(t) \not\models \beta_\times$ **then**
5: $\mathcal{TR} = \mathcal{TR} \setminus (s, t)$
6: **end if**
7: **end for**
8: **end for**
9: **return** $\mathcal{TR}$
10: **end function**

um conjunto $\mathcal{R}$ de relações de transição. Para cada estado s em $\mathcal{S}$ (linha 2), se não existir um estado de destino t em S tal que a relação de transição (s,t) pertença a $\mathcal{R}$ (linha 3), a auto-relação (s,s) é adicionada ao conjunto $\mathcal{TR}$ (linha 4). Por fim, o conjunto $\mathcal{TR}$ é retornado (linha 7).

Algoritmo 8 Adiciona Transição de Loop nos Estados Terminais.

```
1: function AdicionaLoopEmTerminal(𝒮, 𝒯ℛ)
2:     for all s ∈ 𝒮 do
3:         if ∄t ∈ 𝒮, (s,t) ∈ 𝒯ℛ then
4:             𝒯ℛ = 𝒯ℛ ∪ (s,s)
5:         end if
6:     end for
7:     return 𝒯ℛ
8: end function
```

Finalmente, a estrutura de Kripke que representa o comportamento excepcional sensível ao contexto, é definida formalmente como segue:

Definição 10 (Estrutura de Kripke do Contexto)**.** *Seja* $\mathcal{CP}$ *um conjunto finito de proposições contextuais,* $\mathcal{G}$ *um conjunto de restrições semânticas estabelecidas sobre* $\mathcal{CP}$*,* $\mathcal{TC}$ *um conjunto de restrições de transição,* $\mathcal{E}$ *um conjunto de exceções contextuais e* Γ *um conjunto de escopos de tratamento, a estrutura de Kripke do comportamento excepcional sensível ao contexto é dada pela tupla* $\mathcal{EK} = \langle S, I, L, \rightarrow \rangle$*, tal que:*

- $\mathcal{AP} = \mathcal{CP}$*;*
- S *é dado pelo Algoritmo 1;*
- I *é dado pelo Algoritmo 2;*
- L *é dada pelo Algoritmo 3; e*
- $\rightarrow$ *é dada pelo Algoritmo 4.*

4.3 Atividade de Especificação

A atividade de especificação consiste na determinação de propriedades sobre o comportamento excepcional sensível ao contexto com o intuito de encontrar determinados tipos de faltas de projeto. Particularmente, nesta tese de doutorado foram catalogadas 5 propriedades comportamentais que, se violadas, indicam a existência de faltas de projeto

no tratamento de exceção sensível ao contexto, são elas: **progresso de detecção**, **progresso de captura**, **progresso de tratador**, **estabilidade de tratamento** e **alcançabilidade**. Essas propriedades foram inspiradas no estudo das propriedades descritas nos trabalhos de Cubo et al. (2009), Huang et al. (2009), Sama et al. (2010) descritos na Seção 3.2 do Capítulo 3 desta tese, porém com um foco voltado para a semântica do comportamento do TESC. Cada um dessas propriedades é apresentada e formalmente definida nas próximas subseções.

4.3.1 Progresso de Detecção

Essa propriedade determina que para cada estado da estrutura de Kripke do contexto, deve existir pelo menos um estado onde cada exceção contextual é detectada. A violação dessa propriedade indica a existência de exceções contextuais que não são detectadas. Esse tipo de falta de projeto é denominada de **exceção morta**. Essa propriedade deve ser verificada para cada uma das exceções contextuais modeladas no sistema. Desse modo, seja e uma exceção contextual, a fórmula (4.4), escrita em CTL, especifica formalmente essa propriedade.

$$\texttt{EF}(\texttt{ece}(e)) \tag{4.4}$$

A fórmula (4.5) representa a especificação desta propriedade para a exceção contextual $fire$ (4.3), apresentada na Seção 4.2.2.

$$\texttt{EF}(\texttt{hasSmoke} \wedge \texttt{tempGEQ35}) \tag{4.5}$$

4.3.2 Progresso de Captura

Essa propriedade estabelece que para cada exceção de contexto levantada, deve existir, pelo menos, um caso de tratamento habilitado a capturar aquela exceção. A violação dessa propriedade indica que existem estados do contexto onde exceções contextuais são levantadas, mas não podem ser capturadas. Esse tipo de falta de projeto é denominada de **tratamento nulo**. É importante observar que, mesmo existindo situações de contexto onde o sistema não pode tratar aquela exceção, é importante que o projetista esteja ciente de que esse fenômeno ocorre no seu modelo. Sendo assim, seja $\langle e, \texttt{HCASE} \rangle$ um escopo de tratamento com h_0, h_1, $\ldots$, $h_n \in \texttt{HCASE}$ casos de tratamento e $\alpha_0 \in h_0$, $\alpha_1 \in h_1$, $\ldots$, $\alpha_n \in h_n$ condições de seleção desses casos de tratamento, a fórmula (4.6), escrita em CTL,

especifica formalmente essa propriedade.

$$\mathtt{EF}\left(\mathtt{ecs}(e) \wedge \left(\bigvee_{0}^{i<|\mathtt{HCASE}|} \alpha_i\right)\right) \tag{4.6}$$

Por exemplo, a fórmula (4.7) representa a especificação desta propriedade considerando o escopo de tratamento $\langle fire, \{\mathtt{hcase}_0, \mathtt{hcase}_1\}\rangle$ para a exceção contextual *fire* onde $\mathtt{hcase}_0 = \langle \alpha_0, \mathcal{H}_0\rangle$ com $\alpha_0 = \mathtt{inMovement} \wedge \mathtt{atParkEntrance}$ e $\mathtt{hcase}_1 = \langle \alpha_1, \mathcal{H}_1\rangle$ com $\alpha_1 = \mathtt{inMovement} \wedge \mathtt{atParkPlace}$.

$$\begin{aligned} &\mathtt{EF}((\mathtt{hasSmoke} \wedge \mathtt{tempGEQ35}) \wedge ((\mathtt{inMovement} \wedge \mathtt{atParkEntrance}) \vee \\ &\qquad (\mathtt{inMovement} \wedge \mathtt{atParkPlace}))) \end{aligned} \tag{4.7}$$

4.3.3 Progresso de Tratador

Essa propriedade determina que para cada estado do contexto onde uma exceção contextual é levantada, deve existir pelo menos um destes estados onde cada caso de tratamento é selecionado para tratar aquela exceção. A violação dessa propriedade indica que existem casos de tratamento de uma exceção de contexto que nunca serão selecionados. Esse tipo de falta de projeto é denominada de **tratador morto**. Desse modo, seja $\langle e, \mathtt{HCASE}\rangle$ um escopo de tratamento com h_0, h_1, $\ldots$, $h_n \in \mathtt{HCASE}$ casos de tratamento e $\alpha_0 \in h_0$, $\alpha_1 \in h_1$, $\ldots$, $\alpha_n \in h_n$ condições de seleção desses casos de tratamento, a fórmula (4.8), escrita em CTL, especifica formalmente essa propriedade.

$$\bigwedge_{0}^{i<|\mathtt{HCASE}|} \mathtt{EF}(\mathtt{ecs}(e) \wedge (\alpha_i)) \tag{4.8}$$

Por exemplo, a fórmula (4.9) representa a especificação desta propriedade considerando o escopo de tratamento $\langle fire, \{\mathtt{hcase}_0, \mathtt{hcase}_1\}\rangle$ para a exceção contextual *fire* onde $\mathtt{hcase}_0 = \langle \alpha_0, \mathcal{H}_0\rangle$ com $\alpha_0 = \mathtt{inMovement} \wedge \mathtt{atParkEntrance}$ e $\mathtt{hcase}_1 = \langle \alpha_1, \mathcal{H}_1\rangle$ com $\alpha_1 = \mathtt{inMovement} \wedge \mathtt{atParkPlace}$.

$$\begin{aligned} &\mathtt{EF}((\mathtt{hasSmoke} \wedge \mathtt{tempGEQ35}) \wedge (\mathtt{inMovement} \wedge \mathtt{atParkEntrance})) \wedge \\ &\quad \mathtt{EF}((\mathtt{hasSmoke} \wedge \mathtt{tempGEQ35}) \wedge (\mathtt{inMovement} \wedge \mathtt{atParkPlace})) \end{aligned} \tag{4.9}$$

4.3.4 Estabilidade de Tratamento

Essa propriedade determina que para cada exceção tratada, o estado de retomada do fluxo de controle não deve ser um estado onde a mesma exceção é levantada. A violação dessa propriedade indica que o comportamento excepcional sensível ao contexto do sistema está em loop. Esse tipo de falta de projeto é chamado de **tratamento cíclico**. É importante mencionar que essa propriedade tem maior relevância para as situações onde o tratamento de exceção sensível ao contexto atua recuperando erros, não provendo compensações. Nesse último caso, o comportamento excepcional busca contingenciar o contexto excepcional, e não eliminar a sua causa. Desse modo, seja $\langle e, \mathtt{HCASE} \rangle$ um escopo de tratamento com $h_0, h_1, \ldots, h_n \in \mathtt{HCASE}$ casos de tratamento, $\alpha_0 \in h_0$, $\alpha_1 \in h_1$, ..., $\alpha_n \in h_n$ condições de seleção desses casos de tratamento, $\mathcal{H}_0 \in h_0$, $\mathcal{H}_1 \in h_1$, ..., $\mathcal{H}_n \in h_n$ conjunto de medidas de tratamento desses casos de tratamento, e $l_{\mathcal{H}_i}$ a última medida de tratamento do conjunto $\mathcal{H}_i$, a fórmula (4.10), escrita em CTL, especifica formalmente essa propriedade.

$$\bigwedge_{0}^{i<|\mathtt{HCASE}|} \neg((\mathtt{ecs}(e) \wedge \alpha_i) \wedge \mathtt{EF}(l_{\mathcal{H}_i} \wedge \mathtt{ecs}(e))) \tag{4.10}$$

Por exemplo, considerando o escopo de tratamento $\langle \mathit{fire}, \{\mathtt{hcase}_0, \mathtt{hcase}_1\} \rangle$, onde $\mathtt{hcase}_0 = \langle \alpha_0, \mathcal{H}_0 \rangle$ com $\alpha_0 = \mathtt{inMovement} \wedge \mathtt{atParkEntrance}$ e $\mathcal{H}_0 = \{\mathtt{isSprinklerOn} \wedge (\neg\ \mathtt{atParkEntrance} \wedge \neg\ \mathtt{atParkPlace} \wedge \neg\ \mathtt{atParkExit})\}$ e $\mathtt{hcase}_1 = \langle \alpha_1, \mathcal{H}_1 \rangle$ com $\alpha_1 = \mathtt{inMovement} \wedge \mathtt{atParkPlace}$ e $\mathcal{H}_1 = \{\mathtt{isSprinklerOn} \wedge \mathtt{atParkExit}, \mathtt{isSprinklerOn} \wedge (\neg\ \mathtt{atParkEntrance} \wedge \neg\ \mathtt{atParkPlace} \wedge \neg\ \mathtt{atParkExit})\}$.

$$\begin{aligned}
&\neg((\mathtt{hasSmoke} \wedge \mathtt{tempGEQ35}) \wedge (\mathtt{inMovement} \wedge \mathtt{atParkEntrance}))\wedge \\
&\mathtt{EF}(\mathtt{isSprinklerOn} \wedge (\neg\mathtt{atParkEntrance} \wedge \neg\mathtt{atParkPlace} \wedge \neg\mathtt{atParkExit})\wedge \\
&\qquad (\mathtt{hasSmoke} \wedge \mathtt{tempGEQ35}))) \wedge \\
&\neg((\mathtt{hasSmoke} \wedge \mathtt{tempGEQ35}) \wedge (\mathtt{inMovement} \wedge \mathtt{atParkPlace}))\wedge \\
&\mathtt{EF}(\mathtt{isSprinklerOn} \wedge (\neg\mathtt{atParkEntrance} \wedge \neg\mathtt{atParkPlace} \wedge \neg\mathtt{atParkExit})\wedge \\
&\qquad (\mathtt{hasSmoke} \wedge \mathtt{tempGEQ35})))
\end{aligned} \tag{4.11}$$

4.3.5 Alcançabilidade

Essa propriedade estabelece que para cada exceção contextual levantada e capturada, sempre é possível executar todas as medidas de tratamento e fazer com que o fluxo de

controle normal seja retomado. A violação dessa propriedade indica que as medidas de tratamento especificadas não conseguem conduzir o sistema a um estado onde a retomada do fluxo de controle seja possível. Essa falta de projeto é denominada de **retomada impossível**. Desse modo, seja $\langle e, \texttt{HCASE} \rangle$ um escopo de tratamento com h_0, h_1, ..., $h_n \in \texttt{HCASE}$ casos de tratamento, $\alpha_0 \in h_0$, $\alpha_1 \in h_1$, ..., $\alpha_n \in h_n$ critérios de seleção desses casos de tratamento, $\mathcal{H}_0 \in h_0$, $\mathcal{H}_1 \in h_1$, ..., $\mathcal{H}_n \in h_n$ conjunto de medidas de tratamento desses casos de tratamento e $t_{j,\mathcal{H}_i}$ a medida de tratamento de índice $j < |\mathcal{H}_i|$ do conjunto $\mathcal{H}_i$, a fórmula (4.12), escrita em CTL, especifica formalmente essa propriedade.

$$\bigwedge_{0}^{i<|\texttt{HCASE}|} ((\texttt{ecs}(e) \wedge \alpha_i) \rightarrow \texttt{EX}(t_{0,\mathcal{H}_i} \wedge \texttt{EX}(t_{1,\mathcal{H}_i} \wedge \texttt{EX}(\ldots \texttt{EX}(l_{|\mathcal{H}_i|-1,\mathcal{H}_i}))))) \tag{4.12}$$

Por exemplo, considerando o escopo de tratamento $\langle fire, \{\texttt{hcase}_0, \texttt{hcase}_1\} \rangle$ onde $\texttt{hcase}_0 = \langle \alpha_0, \mathcal{H}_0 \rangle$ com $\alpha_0 = \texttt{inMovement} \wedge \texttt{atParkEntrance}$ e $\mathcal{H}_0 = \{\texttt{isSprinklerOn} \wedge (\neg\ \texttt{atParkEntrance} \wedge \neg\ \texttt{atParkPlace} \wedge \neg\ \texttt{atParkExit})\}$ e $\texttt{hcase}_1 = \langle \alpha_1, \mathcal{H}_1 \rangle$ com $\alpha_1 = \texttt{inMovement} \wedge \texttt{atParkPlace}$ e $\mathcal{H}_1 = \{\texttt{isSprinklerOn} \wedge \texttt{atParkExit}, \texttt{isSprinklerOn} \wedge (\neg\ \texttt{atParkEntrance} \wedge \neg\ \texttt{atParkPlace} \wedge \neg\ \texttt{atParkExit})\}$.

$$\begin{aligned} &(((\texttt{hasSmoke} \wedge \texttt{tempGEQ35}) \wedge (\texttt{inMovement} \wedge \texttt{atParkEntrance})) \rightarrow \\ &\texttt{EX}(\texttt{isSprinklerOn} \wedge (\neg\texttt{atParkEntrance} \wedge \neg\texttt{atParkPlace} \wedge \neg\texttt{atParkExit}))) \wedge \\ &(((\texttt{hasSmoke} \wedge \texttt{tempGEQ35}) \wedge (\texttt{inMovement} \wedge \texttt{atParkPlace})) \rightarrow \\ &\texttt{EX}((\texttt{isSprinklerOn} \wedge \texttt{atParkExit}) \wedge \\ &\texttt{EX}(\texttt{isSprinklerOn} \wedge (\neg\texttt{atParkEntrance} \wedge \neg\texttt{atParkPlace} \wedge \neg\texttt{atParkExit})))) \end{aligned} \tag{4.13}$$

4.4 Considerações Finais

Este capítulo apresentou o CAEH✓, um método para verificação de modelos do tratamento excepcional sensível ao contexto. A forma como o CAEH✓modela o comportamento excepcional e o mapeia para uma estrutura de Kripke foi apresentado em detalhes. Além disso, um conjunto de propriedades comportamentais que ajudam a identificar um conjunto de faltas de projeto foi proposto e formalmente estabelecido. Adicionalmente, as abstrações propostas com relação a modelagem do comportamento excepcional sensível ao contexto (exceções contextuais, casos de tratamento e escopo de tratamento) se mostram eficientes em representar aspectos comportamentais relacionados com a definição e a detecção de exceções contextuais, como o agrupamento, seleção e execução das medidas de tratamento e a retomada do fluxo de controle.

É importante destacar que, embora o CAEH✓ não enderece o problema de modelagem do comportamento adaptativo do sistema, ele leva em consideração questões importantes sobre a modelagem de comportamento adaptativo descritas na literatura, por meio das abstrações de proposições contextuais, restrições semânticas, estados do contexto e restrições de transição. O próximo capítulo é dedicado a avaliação do CAEH✓, onde uma ferramenta que automatiza o método é apresentada (o JCAEH✓) e alguns cenários de aplicação com injeção de faltas são analisados.

5 *Avaliação do CAEH✓*

Este capítulo é dedicado a avaliação do método CAEH✓. Na Seção 5.1 é dada um visão geral sobre o propósito dessa avaliação. Na Seção 5.2 é dada uma visão geral sobre a organização da ferramenta que automatiza o método (JCAEH✓) e é apresentado o projeto do tratamento de exceção sensível ao contexto para a aplicação exemplo *UbiParking* modelado no JCAEH✓. Na Seção 5.3 são descritos cenários de injeção de faltas para o modelo do *UbiParking* e uma discussão sobre os resultados da análise desses cenários é feita com o JCAEH✓. Por fim, conclusões que sumarizam o capítulo são oferecidas.

5.1 Introdução

A avaliação do CAEH✓foi conduzida com o intuito de evidenciar se o método é passível de automação e se é eficaz. Para avaliar o primeiro aspecto, uma ferramenta que automatiza o método foi desenvolvida, o JCAEH✓. Com relação ao segundo aspecto, um conjunto de cenários de injeção de faltas foram criados para duas exceções contextuais modeladas na aplicação *UbiParking*. Esses cenários foram modelados e submetidos à ferramenta como forma de avaliar a efetividade do método em identificar faltas de projeto. Observe que o segundo aspecto faz uma avaliação indireta do método (i.e., via ferramenta JCAEH✓). A avaliação desses dois aspectos ajudam a constatar o cumprimento das metas MET03 e MET04, descritos no Capítulo 1, corroborando para a aceitação da hipótese de pesquisa estabelecida nesta tese, também declarada no Capítulo 1.

5.2 A Ferramenta JCAEH✓

O JCAEH✓foi implementado na plataforma Java e a sua organização geral está descrita na Figura 5.1. O JCAEH✓recebe do projetista, via API (*Application Programming Interface*), as especificações das proposições de contexto, das restrições semânticas, das

restrições de transição, das exceções contextuais, dos casos de tratamento e dos escopos de tratamento. Após isso, o módulo conversor gera os estados do contexto e constrói o modelo de comportamento excepcional sensível ao contexto e o conjunto de propriedades descritas pelo CAEH✓. É importante mencionar que o projetista pode informar propriedades adicionais a serem verificadas sobre o modelo. De posse do modelo de comportamento e das propriedades, o JCAEH✓submete o modelo e as propriedades ao módulo de verificação de modelos, o qual executa o processo de verificação e gera um relatório de saída contendo os resultados da verificação.

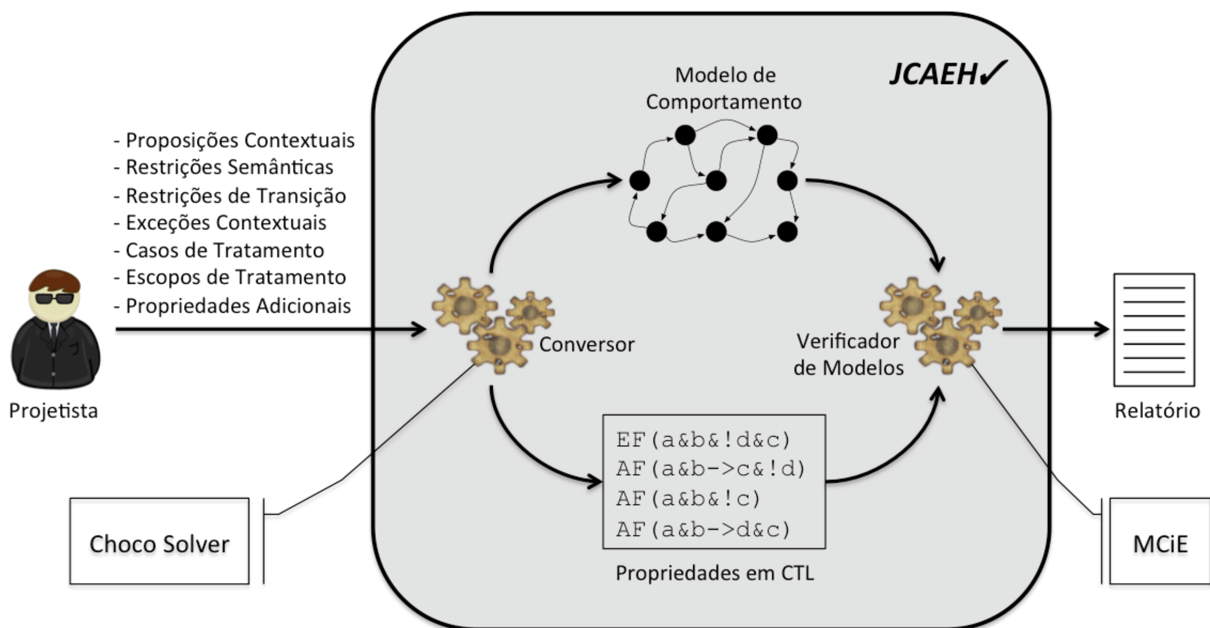


Figura 5.1: Visão geral do JCAEH✓.

Para a geração dos estados do contexto, o JCAEH✓faz uso da ferramenta Choco Solver[1], uma das implementações de referência da JSR (*Java Specification Request*) 331: *Constraint Programming API*[2]. Já no processo de verificação, o JCAEH✓utiliza o verificador de modelos MCiE desenvolvido no projto *The Model Checking in Education*[3]. Esse verificador foi escolhido pelo fato de dar suporte a CTL, lógica temporal na qual as propriedades do CAEH✓foram especificadas, e ser implementado em Java, facilitando a sua integração com o JCAEH✓.

A Listagem 5.1 ilustra o uso da API do JCAEH✓para a especificação de dois cenários excepcionais no *UbiParking*: (i) exceção de incêndio ("Fire" - linha 40); e (ii) exceção

[1]http://www.emn.fr/z-info/choco-solver
[2]http://jcp.org/en/jsr/detail?id=331
[3]http://www2.cs.uni-paderborn.de/cs/kindler/Lehre/MCiE/

de vaga ocupada ("NoFreeSpace" - linha 59). A classe `CAEHModel` (linha 3) da API do JCAEH✓é responsável por agrupar todas as abstrações providas pelo CAEH✓. Nas linhas 6-21 as proposições contextuais são criadas e adicionadas numa instância da classe `CAEHModel` que é referenciado pela variável `caeh`. Na linha 24 é declarada uma fórmula que descreve um contexto de alto nível, que significa que o veículo está fora do estacionamento. Na linhas 27-28 é criada uma restrição semântica que garante que o veículo não estará em mais de um lugar ao mesmo tempo. Nas linhas 31-36 são criadas restrições de transição que impedem que o veículo consiga chegar a saída do estacionamento antes de passar pelo pátio de vagas, ou que ele alcance o pátio de vagas ou a saída do estacionamento antes de passar pela entrada, ou ainda, que saia pela entrada ou que alcance o lado de fora do estacionamento sem passar pela saída.

Listagem 5.1: Código fonte do *UbiParking* usando a API do JCAEH✓.

```
1  public class UbiParking {
2     public static void main(String[] args) throws EvaluationException {
3           CAEHModel caeh = new CAEHModel(``UbiParking'');
4
5           // Context Propositions
6           ContextProposition inMovement = new ContextProposition(``inMovement'');
7           caeh.addContextProposition(inMovement);
8           ContextProposition atParkEntrance = new ContextProposition(``atParkEntrance'');
9           caeh.addContextProposition(atParkEntrance);
10          ContextProposition atParkPlace = new ContextProposition(``atParkPlace'');
11          caeh.addContextProposition(atParkPlace);
12          ContextProposition atParkExit = new ContextProposition(``atParkExit'');
13          caeh.addContextProposition(atParkExit);
14          ContextProposition hasSpace = new ContextProposition(``hasSpace'');
15          caeh.addContextProposition(hasSpace);
16          ContextProposition isHot = new ContextProposition(``isHot'');
17          caeh.addContextProposition(isHot);
18          ContextProposition hasSmoke = new ContextProposition(``hasSmoke'');
19          caeh.addContextProposition(hasSmoke);
20          ContextProposition isSprinklerOn = new ContextProposition(``isSprinklerOn'');
21          caeh.addContextProposition(isSprinklerOn);
22
23          // High Level Context Information
24          ContextExpression outOfPark = and(not(atParkEntrance), not(atParkPlace), not(atParkExit));
25
26          // Semantic Constraints
27          ContextExpression disjoined = or(xor(atParkEntrance, atParkPlace, atParkExit), outOfPark));
28          caeh.addSemanticConstraint(new SemanticConstraint(``AllPlacesDisjoined'', disjoined));
29
30          // Transition Constraints
31          TransitionConstraint transition = new TransitionConstraint(atParkEntrance, not(atParkExit));
32          caeh.addTransitionConstraint(transition);
33          transition = new TransitionConstraint(outOfPark, and(not(atParkPlace), not(atParkExit)));
34          caeh.addTransitionConstraint(transition);
35          transition = new TransitionConstraint(atParkPlace, and(not(atParkEntrance), not(outOfPark)))
36          caeh.addTransitionConstraint(transition);
37
38          // Contextual Exceptions
39          ContextExpression exceptionalContext = and(isHot, hasSmoke, not(isSprinklerOn));
40          ContextualException fire = new ContextualException(``Fire'', exceptionalContext);
41          HandlingScope fireScope = new HandlingScope(fire);
42          HandlingCase handlingCase = new HandlingCase();
43          handlingCase.setCatchConstraint(and(inMovement, atParkPlace));
44          HandlingBehavior handlingBehavior = new HandlingBehavior();
45          handlingBehavior.addHandlingStep(and(isSprinklerOn, inMovement, atParkExit));
46          handlingBehavior.addHandlingStep(and(isSprinklerOn, inMovement, outOfPark));
47          handlingCase.setHandlingBehavior(handlingBehavior);
48          fireScope.addHandlingCase(handlingCase);
```

```
49
50             handlingCase = new HandlingCase();
51             handlingCase.setCatchConstraint(and(inMovement, atParkExit));
52             handlingBehavior = new HandlingBehavior();
53             handlingBehavior.addHandlingStep(and(isSprinklerOn, inMovement, outOfPark));
54             handlingCase.setHandlingBehavior(handlingBehavior);
55             fireScope.addHandlingCase(handlingCase);
56             caeh.addHandlingScope(fireScope);
57
58             exceptionalContext = and(inMovement, atParkPlace, not(hasSpace));
59             ContextualException noSpace = new ContextualException(''NoFreeSpace'', exceptionalContext);
60             HandlingScope noSpaceScope = new HandlingScope(noSpace);
61             handlingCase = new HandlingCase();
62             handlingCase.setCatchConstraint(and(inMovement, atParkPlace));
63             handlingBehavior = new HandlingBehavior();
64             handlingBehavior.addHandlingStep(and(inMovement, atParkExit));
65             handlingCase.setHandlingBehavior(handlingBehavior);
66             noSpaceScope.addHandlingCase(handlingCase);
67             caeh.addHandlingScope(noSpaceScope);
68
69             PropertyGenerator generator = new PropertyGenerator();
70             Set<Property> properties = generator.generate(caeh.getHandlingScopes());
71
72             CAEH2KripkeModel conversor = new CAEH2KripkeModel();
73             KripkeStructureModel kripkeModel = conversor.toKripkeModel(caeh);
74
75             CTLTransitionSystemMCiE checker = new CTLTransitionSystemMCiE();
76             checker.check(kripkeModel, properties);
77
78             KripkeModelTXTReport report = new KripkeModelTXTReport();
79             report.report(kripkeModel);
80         }
81 }
```

Nas linhas 39-40 a exceção contextual de incêndio é declarada, seu nome é "Fire" e seu contexto excepcional é caracterizado pela situação em que existe fumaça (`hasSmoke`), a temperatura está alta (`isHot`) e os aspersores encontra-se desligados (`not(isSprinklerOn)`). Na linha 41 essa exceção é adicionada ao seu escopo de tratamento. Na linha 42 é criado um caso de tratamento e na linha 43 é atribuída a esse caso de tratamento a sua condição de seleção. Essa condição de seleção verifica se o veículo encontra-se em movimento no parque de estacionamento. A classe `HandlingBehavior` (linha 44) é utilizada para manter as medidas de tratamento para aquele caso de tratamento. Nas linhas 45-46 são cridas duas medidas. A primeira estabelece que após o tratamento, o veículo esteja na saída do estacionamento e os aspersores estejam ligados. A segunda medida determina que após a execução da primeira medida, o veículo encontre-se fora do estacionamento e os aspersores permaneçam ligados. Outro caso de tratamento é definido para essa exceção na linha 50. A condição de seleção para esse outro caso de tratamento é que o veículo esteja em movimento na saída do estacionamento (linha 51). Para esse caso de tratamento, apenas uma medida de tratamento é estabelecida (linha 53), idêntica a segunda medida definida para o primeiro caso de tratamento.

Nas linhas 58-59, a exceção de "NoFreeSpace" é declarada. Seu contexto excepcional é caracterizado pela situação em que o veículo está em movimento (`inMovement`) dentro do

pátio de vagas (`atParkPlace`) e a vaga reservada para ele foi ocupada (`not(hasSpace)`), sendo essa a última vaga livre dentro do estacionamento. Essa exceção é adicionada ao seu escopo de tratamento na linha 60. Para essa exceção, apenas um caso de tratamento foi estabelecido nas linhas 61. A condição desse caso de tratamento estabelece que ele só é selecionado se o veículo estiver em movimento (`inMovement`) dentro do pátio de vagas (`atParkPlace`) (linha 62). A medida de tratamento associada a esse caso de tratamento (linhas 63-64) define que após o tratamento, o veículo encontre-se em movimento na saída do estacionamento. Nas linhas 69-70 é estanciado o gerador de propriedades (`PropertyGenerator`), que cria todas as fórmulas temporais para cada escopo de tratamento presente no modelo. Nesse momento, é possível ao projetista adicionar mais propriedades que queira verificar. Nas linhas 72-73 o modelo é convertido para uma estrutura de Kripke, conforme explicado no Capítulo 4. Após isso, o modelo representado como uma estrutura de Kripke e as propriedades são submetidos ao verificador de modelos (linhas 75-76) e, por fim, o relatório é gerado.

5.3 Cenários de Injeção de Faltas

Antes de começar o detalhamento da avaliação descrita nesta seção, é importante deixar claro qual é o seu propósito. O objetivo dessa avaliação é investigar quão eficaz é o método em identificar faltas de projeto. Para atingir esse propósito, foi utilizado o conceito de **injeção de faltas** (*fault injection*). Essa técnica é largamente adotada na comunidade com o intuito de avaliar a confiabilidade de um modelo ou sistema computacional (HSUEH; TSAI; IYER, 1997). Entende-se por injeção de faltas a inserção deliberada e sistemática de faltas em um modelo ou sistema computacional afim de avaliar aspectos de robustez e dependabilidade (CLARK; PRADHAN, 1995). Para a avaliação do CAEH✓, é adotado uma abordagem similar a usada por Ezekiel e Lomuscio (2009) onde faltas de projeto são inseridas automaticamente em um modelo de comportamento e analisadas via verificador de modelos.

É importante ressaltar que está fora do escopo desta avaliação mensurar a produtividade do projetista em construir modelos complexos e corretos (i.e., livres faltas de projeto) utilizando o método por meio da ferramenta. Desse modo, na Seção 5.3.1 são descritos os cenários de injeção de faltas no modelo, e, na Seção 5.3.2, uma discussão sobre os resultados obtidos é conduzida.

5.3.1 Descrição dos Cenários

Os cenários de injeção de faltas levam em consideração as duas exceções descritas na Seção 5.2 e modeladas na Listagem 5.1: (i) exceção de incêndio ("Fire"); e (ii) exceção de vaga ocupada ("NoFreeSpace"). O modelo descrito na Listagem 5.1 foi previamente submetido ao JCAEH✓e nenhuma das faltas de projeto estabelecidas pelo CAEH✓foi encontrada, portanto, trata-se de um modelo correto. A execução desse modelo teve a duração de 58ms e resultou na geração de 129 estados, sendo 23 deles excepcionais. A partir desse modelo correto, para cada propriedade que se deseja verificar (progresso de detecção, progresso de captura, progresso de tratador, estabilidade de tratamento e alcançabilidade) foi feita uma alteração de forma deliberada no modelo com o propósito de violá-la (i.e., faltas de projeto foram injetadas no modelo). Nas próximas subseções são descritas cada uma dessa alterações para cada uma das propriedades.

Cenário 1: Violando o Progresso de Detecção

Essa propriedade é violada quando não existe pelo menos um estado do contexto onde a exceção em questão pode ser detectada. Contudo, o modelo do *UbiParking* não viola essa propriedade. Portanto, para essa situação, optou-se por inserir uma contradição lógica na fórmula que descreve o contexto excepcional como forma de inviabilizar a detecção das exceções. A Listagem 5.2 descreve a forma como o modelo do *UbiParking* foi alterado para incorporar essa mudança para a exceção "Fire". Já a Listagem 5.3 descreve a alteração sofrida pela exceção "NoFreeSpace". Na Listagem 5.2 foi inserido no contexto excepcional a contradição: está quente (`isHot`) e não está quente (`not(isHot)`). Para a exceção "NoFreeSpace" (Listagem 5.3) foi inserida a contradição: há vagas (`hasSpace`) e não há vagas (`not(hasSpace)`).

Listagem 5.2: Cenário 1 do *UbiParking* para a exceção "Fire".

```
(...)
    ContextExpression exceptionalContext = and(isHot, hasSmoke, not(isSprinklerOn), not(isHot));
    ContextualException fire = new ContextualException(''Fire'', exceptionalContext);
(...)
```

Listagem 5.3: Cenário 1 do *UbiParking* para a exceção "NoFreeSpace".

```
(...)
    exceptionalContext = and(inMovement, atParkPlace, not(hasSpace), hasSpace);
    ContextualException noSpace = new ContextualException(''NoFreeSpace'', exceptionalContext);
(...)
```

Cenário 2: Violando o Progresso de Captura e de Tratador

Essas duas propriedades são violadas, simultaneamente, quando não é possível selecionar casos de tratamento quando uma exceção é detectada. Desse modo, foi inserido uma contradição nas condições de seleção dos casos de tratamento para que essa propriedade fosse violada. A Listagem 5.4 descreve a forma com o modelo foi alterado para incorporar essa mudança para a exceção "Fire". Na condição de seleção dos dois casos de tratamento para da exceção "Fire", foi inserido a seguinte contradição: o veículo está em movimento (`isMovement`) e não está (`not(isMovement)`). O mesmo foi feito para a condição de seleção do único caso de tratamento da exceção "NoFreeSpace" (Listagem 5.5).

Listagem 5.4: Cenário 2 do *UbiParking* para a exceção "Fire".

```
(...)
    HandlingScope fireScope = new HandlingScope(fire);
    HandlingCase handlingCase = new HandlingCase();
    handlingCase.setCatchConstraint(and(inMovement, atParkPlace, not(inMovement)));
(...)
    handlingCase = new HandlingCase();
    handlingCase.setCatchConstraint(and(inMovement, atParkExit, not(inMovement)));
(...)
```

Listagem 5.5: Cenário 2 do *UbiParking* para a exceção "NoFreeSpace".

```
(...)
    HandlingScope noSpaceScope = new HandlingScope(noSpace);
    handlingCase = new HandlingCase();
    handlingCase.setCatchConstraint(and(inMovement, atParkPlace, not(inMovement)));
(...)
```

Cenário 3: Violando a Estabilidade de Tratamento

Essa propriedade é violada quando a última medida de tratamento, do caso de tratamento selecionado, leva o sistema a um estado em que a exceção que desencadeou aquele tratamento é relançada. Desse modo, foi feita uma alteração na última medida de tratamento para que essa permita levar o sistema a um estado em que a exceção em questão possa ser relançada. A Listagem 5.6 descreve as alterações sofridas pelo modelo da exceção "Fire" para acomodar essas mudanças.

Listagem 5.6: Cenário 3 do *UbiParking* para a exceção "Fire".

```
(...)
    ContextExpression exceptionalContext = and(isHot, hasSmoke, not(isSprinklerOn));
    ContextualException fire = new ContextualException(``Fire'', exceptionalContext);
(...)
HandlingBehavior handlingBehavior = new HandlingBehavior();
    handlingBehavior.addHandlingStep(and(isSprinklerOn, inMovement, atParkExit));
    handlingBehavior.addHandlingStep(and(inMovement, outOfPark, and(isHot, hasSmoke, not(isSprinklerOn))));
    (...)
    handlingBehavior = new HandlingBehavior();
    handlingBehavior.addHandlingStep(and(inMovement, outOfPark, and(isHot, hasSmoke, not(isSprinklerOn))));
    handlingCase.setHandlingBehavior(handlingBehavior);
```

```
(...)
```

Listagem 5.7: Cenário 3 do *UbiParking* para a exceção “NoFreeSpace”.

```
(...)
    exceptionalContext = and(inMovement, atParkPlace, not(hasSpace));
    ContextualException noSpace = new ContextualException(''NoFreeSpace'', exceptionalContext);
(...)
    handlingBehavior = new HandlingBehavior();
    handlingBehavior.addHandlingStep(and(inMovement, and(inMovement, atParkPlace, not(hasSpace))));
    handlingCase.setHandlingBehavior(handlingBehavior);
(...)
```

Para o primeiro caso de tratamento, a segunda medida de tratamento foi alterada, removeu-se a necessidade dos aspersores estarem ligados (`isSprinklerOn`) e fez-se uma disjunção entre o restante da fórmula e o contexto excepcional da exceção contextual “Fire”. Por outro lado, a alteração feita no modelo, para acomodar essa mudança na exceção “NoFreeSpace”, considerou a remoção da necessidade do veículo estar na saída do estacionamento (`atParkExit`) da última (e única) medida de tratamento do único caso de tratamento desta exceção. Além disso, foi feita uma disjunção entre o restante da fórmula dessa medida de tratamento e o contexto excepcional da exceção contextual “NoFreeSpace” (Listagem 5.7).

Cenário 4: Violando a Alcançabilidade

Essa propriedade é violada quando pelo menos uma das medidas de tratamento não pode ser executada (i.e., não existir estado de contexto que satisfaça a fórmula que descreve essa medida). Desse modo, para violar essa propriedade foi inserida uma contradição na primeira medida de tratamento de todos os casos de tratamento das duas exceções em questão. No caso da exceção “Fire”, a contradição inserida na primeira medida de tratamento do primeiro caso de tratamento foi a necessidade dos aspersores estarem ligados (`isSprinklerOn`) e desligados (`not(isSprinklerOn)`) ao mesmo tempo (Listagem 5.8). O mesmo procedimento foi adotado para a única medida de tratamento do segundo caso de tratamento dessa exceção (Listagem 5.8). Por outro lado, para a exceção “NoFreeSpace” (Listagem 5.9), o procedimento consistiu na inserção da seguinte contradição: o veículo está em movimento (`isMovement`) e não está (`not(isMovement)`).

Listagem 5.8: Cenário 4 do *UbiParking* para a exceção “Fire”.

```
(...)
HandlingBehavior handlingBehavior = new HandlingBehavior();
handlingBehavior.addHandlingStep(and(isSprinklerOn, inMovement, atParkExit, not(isSprinklerOn)));
handlingBehavior.addHandlingStep(and(isSprinklerOn, inMovement, outOfPark));
handlingCase.setHandlingBehavior(handlingBehavior);
(...)
handlingBehavior = new HandlingBehavior();
handlingBehavior.addHandlingStep(and(isSprinklerOn, inMovement, outOfPark, not(isSprinklerOn)));
```

```
handlingCase.setHandlingBehavior(handlingBehavior);
fireScope.addHandlingCase(handlingCase);
caeh.addHandlingScope(fireScope);
(...)
```

Listagem 5.9: Cenário 4 do *UbiParking* para a exceção "NoFreeSpace".

```
(...)
handlingBehavior = new HandlingBehavior();
handlingBehavior.addHandlingStep(and(inMovement, atParkExit, not(inMovement)));
(...)
```

5.3.2 Ambiente de Execução e Discussão dos Resultados

Todos os cenários foram executados no mesmo ambiente operacional: um Mac OS X, Versão 10.8.2, com processador de 2GHz Intel Core i7 e memória de 8GB 1600MHz DDR3. Cada cenário foi executado individualmente e foram consideradas 3 (três) tipos de permutações: (i) a injeção de falta apenas na exceção "Fire"; (ii) a injeção de falta apenas na exceção "NoFreeSpace"; e (iii) a injeção de falta em ambas exceções de forma simultânea. Em resumo, com base nos resultados obtidos em todos os cenários, observa-se que a efetividade da proposta nessa avaliação foi de 100%, visto que os resultados podem ser classificados como desejados ou esperados. Nas próximas subseções é feita uma discussão sobre os resultados de cada cenário e suas permutações, chamadas de agora em diante de rodadas.

Cenário 1: Discussão dos Resultados

A primeira rodada desse cenário consistiu em injetar uma falta de projeto apenas na modelagem da exceção "Fire". Nessa rodada, 129 estados foram gerados, sendo 8 deles excepcionais. O tempo total de execução foi de 463ms. Os resultados dessa rodada foram sumarizados na Tabela 5.1. Por convenção, o símbolo (✓) indica que aquela falta de projeto foi detectada no modelo. Por outro lado, o símbolo (✗), significa exatamente o contrário. Como esperado, a falta injetada foi detectada através da identificação de uma falta de projeto de exceção morta. Além dela, duas outras faltas de projeto foram detectadas: tratamento nulo e tratador morto. O fato dessas outras duas faltas de projeto existirem é compreensível, uma vez que só se pode tratar uma exceção que foi detectada.

Na segunda rodada, o cenário consistiu em injetar uma falta de projeto apenas na exceção "NoFreeSpace". Nessa rodada, 129 estados foram gerados, sendo 16 deles excepcionais. O tempo total de execução foi de 230ms. Como esperado, a falta de projeto foi detectada (exceção morta). A Tabela 5.2 traz um resumo dos resultados obtidos. Além

Tabela 5.1: Sumário da Execução da Rodada 1 do Cenário 1

	Exceções	
Faltas de Projeto	Fire	NoFreeSpace
Exceção Morta	✓	✗
Tratamento Nulo	✓	✗
Tratador Morto	✓	✗
Tratamento Cíclico	✗	✗
Retomada Impossível	✗	✗

da falta de projeto de exceção morta, foram detectadas as faltas de projeto de tratamento nulo e tratador morto. Como discutido na rodada anterior, já era esperado que essas duas faltas de projeto fossem detectadas.

Tabela 5.2: Sumário da Execução da Rodada 2 do Cenário 1

	Exceções	
Faltas de Projeto	Fire	NoFreeSpace
Exceção Morta	✗	✓
Tratamento Nulo	✗	✓
Tratador Morto	✗	✓
Tratamento Cíclico	✗	✗
Retomada Impossível	✗	✗

Por fim, a terceira rodada consistiu em injetar faltas na modelagem das duas exceções simultaneamente. Nessa rodada, 128 estados foram gerados, sendo que nenhuma deles excepcional. O tempo total de execução foi de 39ms. Também como esperado, o modelo não pode ser gerado, uma vez que nenhuma exceção foi detectada.

Cenário 2: Discussão dos Resultados

A primeira rodada desse cenário consistiu em injetar uma falta de projeto apenas na modelagem da exceção "Fire". Nessa rodada, 129 estados foram gerados, sendo 23 deles excepcionais. O tempo total de execução foi de 329ms. Os resultados dessa rodada foram sumarizados na Tabela 5.3. Como esperado, a falta injetada foi detectada através identificação das faltas de projeto de tratamento nulo e tratador morto. Nenhuma falta de exceção morta foi detectada, uma vez que exceções foram levantadas no modelo. Além disso, como as exceções não podem ser tratadas, não existe o risco da propriedade estabilidade de tratamento ser violada, logo a falta de projeto de tratamento cíclico não existe no modelo. Adicionalmente, uma vez que não é possível tratar a exceção, a falta de projeto de retomada impossível também não ocorre.

Na segunda rodada desse cenário, foi injetada uma falta de projeto apenas na mo-

Tabela 5.3: Sumário da Execução da Rodada 1 do Cenário 2

	Exceções	
Faltas de Projeto	Fire	NoFreeSpace
Exceção Morta	✗	✗
Tratamento Nulo	✓	✗
Tratador Morto	✓	✗
Tratamento Cíclico	✗	✗
Retomada Impossível	✗	✗

delagem da exceção “NoFreeSpace”. Nessa rodada, 129 estados foram gerados, sendo 23 deles excepcionais. O tempo total de execução foi de 209ms. Os resultados dessa rodada foram sumarizados na Tabela 5.4. Como esperado, a falta injetada foi detectada através identificação das faltas de projeto de tratamento nulo e tratador morto. Do mesmo modo que na rodada anterior, nenhuma falta de projeto de exceção morta, tratamento cíclico e retomada impossível ocorreu no modelo.

Tabela 5.4: Sumário da Execução da Rodada 2 do Cenário 2

	Exceções	
Faltas de Projeto	Fire	NoFreeSpace
Exceção Morta	✗	✗
Tratamento Nulo	✗	✓
Tratador Morto	✗	✓
Tratamento Cíclico	✗	✗
Retomada Impossível	✗	✗

Por fim, a última roda desse cenário considerou injeção de faltas em ambas exceções. Nessa rodada, 129 estados foram gerados, sendo 23 deles excepcionais. O tempo total de execução foi de 100ms. Os resultados dessa rodada foram sumarizados na Tabela 5.5. Como esperado, para cada exceção, o par de faltas de projeto de tratamento nulo e tratador morto foi detectado. Do mesmo modo que nas rodadas anterioriores, nenhuma falta de projeto de exceção morta, tratamento cíclico e retomada impossível ocorreu no modelo.

Tabela 5.5: Sumário da Execução da Rodada 3 do Cenário 2

	Exceções	
Faltas de Projeto	Fire	NoFreeSpace
Exceção Morta	✗	✗
Tratamento Nulo	✓	✓
Tratador Morto	✓	✓
Tratamento Cíclico	✗	✗
Retomada Impossível	✗	✗

Cenário 3: Discussão dos Resultados

Na primeira rodada desse cenário, foi inserida uma falta de projeto apenas na exceção "Fire". Nessa rodada, 129 estados foram gerados, sendo 23 deles excepcionais. O tempo total de execução foi de 268ms. A Tabela 5.6 resume os resultados obtidos nessa rodada. Como esperado, a falta injetada foi detectada com a identificação da falta de projeto de tratamento cíclico. As demais faltas de projeto não foram detectadas. Em princípio, era de se esperar que a falta de projeto de retomada impossível pudesse ocorrer, uma vez que ela envolve o tratamento. Porém, essa falta de projeto só ocorre quando a propriedade de alcançabilidade é violada (i.e., quando é impossível alcançar o estado que caracteriza a última medida de tratamento do caso de tratamento selecionado para tratar a exceção em questão). Contudo, a propriedade de alcançabilidade admite que esse estado seja um estado excepcional.

Tabela 5.6: Sumário da Execução da Rodada 1 do Cenário 3

Faltas de Projeto	Exceções Fire	 NoFreeSpace
Exceção Morta	✗	✗
Tratamento Nulo	✗	✗
Tratador Morto	✗	✗
Tratamento Cíclico	✓	✗
Retomada Impossível	✗	✗

A segunda rodada desse cenário inseriu uma falta de projeto apenas na exceção "No-FreeSpace". Nessa rodada, 129 estados foram gerados, sendo 23 deles excepcionais. O tempo total de execução foi de 117ms. A Tabela 5.7 resume os resultados obtidos nessa rodada. Como esperado, a falta injetada foi detectada com a identificação da falta de projeto de tratamento cíclico. As demais faltas de projeto não foram detectadas. Em particular, a justificativa para a não ocorrência da falta de projeto de retomada impossível, é similar a dada na rodada anterior.

Tabela 5.7: Sumário da Execução da Rodada 2 do Cenário 3

Faltas de Projeto	Exceções Fire	 NoFreeSpace
Exceção Morta	✗	✗
Tratamento Nulo	✗	✗
Tratador Morto	✗	✗
Tratamento Cíclico	✗	✓
Retomada Impossível	✗	✗

Por fim, na terceira rodada desse cenário, foram inseridas faltas de projeto em ambas

exceções. Nessa rodada, 129 estados foram gerados, sendo 23 deles excepcionais. O tempo total de execução foi de 54ms. A Tabela 5.8 resume os resultados obtidos nessa rodada. Como esperado, as faltas injetadas foram detectadas nas duas exceções com a identificação da falta de projeto de tratamento cíclico em ambas. As demais faltas de projeto não foram detectadas pelas mesmas justificativas apresentadas anteriormente.

Tabela 5.8: Sumário da Execução da Rodada 3 do Cenário 3

	Exceções	
Faltas de Projeto	Fire	NoFreeSpace
Exceção Morta	✗	✗
Tratamento Nulo	✗	✗
Tratador Morto	✗	✗
Tratamento Cíclico	✓	✓
Retomada Impossível	✗	✗

Cenário 4: Discussão dos Resultados

Na primeira rodada desse cenário, foi inserida uma falta de projeto apenas na exceção "Fire". Nessa rodada, 129 estados foram gerados, sendo 23 deles excepcionais. O tempo total de execução foi de 61ms. A Tabela 5.9 resume os resultados obtidos nessa rodada. Como esperado, a falta injetada foi detectada com a identificação da falta de projeto de retomada impossível. As faltas de projeto de exceção morta, tratamento nulo e tratador morto não foram detectadas, uma vez que estas não estão relacionadas com a retomada do fluxo de controle. Com relação à falta de tratamento cíclico, era possível imaginar a sua ocorrência, uma vez que essa falta de projeto analisa se o estado de retomada é excepcional. Porém, a falta de tratamento cíclico indica a existência de tal situação, o que não é o caso desse cenário, logo, ela não ocorre no modelo.

Tabela 5.9: Sumário da Execução da Rodada 1 do Cenário 4

	Exceções	
Faltas de Projeto	Fire	NoFreeSpace
Exceção Morta	✗	✗
Tratamento Nulo	✗	✗
Tratador Morto	✗	✗
Tratamento Cíclico	✗	✗
Retomada Impossível	✓	✗

A segunda rodada desse cenário inseriu uma falta de projeto apenas na exceção "NoFreeSpace". Nessa rodada, 129 estados foram gerados, sendo 23 deles excepcionais. O tempo total de execução foi de 52ms. A Tabela 5.10 resume os resultados obtidos nessa

rodada. Como esperado, a falta injetada foi detectada com a identificação da falta de projeto de retomada impossível. As demais faltas de projeto não foram detectadas pelos mesmos motivos da primeira rodada.

Tabela 5.10: Sumário da Execução da Rodada 2 do Cenário 4

	Exceções	
Faltas de Projeto	Fire	NoFreeSpace
Exceção Morta	✗	✗
Tratamento Nulo	✗	✗
Tratador Morto	✗	✗
Tratamento Cíclico	✗	✗
Retomada Impossível	✗	✓

Por fim, na terceira rodada desse cenário, foram inseridas faltas de projeto em ambas exceções. Nessa rodada, 129 estados foram gerados, sendo 23 deles excepcionais. O tempo total de execução foi de 30ms. A Tabela 5.11 resume os resultados obtidos nessa rodada. Como esperado, as faltas injetadas foram detectadas com a identificação da falta de projeto de retomada impossível em ambas exceções. As demais faltas de projeto não foram detectadas pelos mesmos motivos descritos nas rodadas anteriores.

Tabela 5.11: Sumário da Execução da Rodada 3 do Cenário 4

	Exceções	
Faltas de Projeto	Fire	NoFreeSpace
Exceção Morta	✗	✗
Tratamento Nulo	✗	✗
Tratador Morto	✗	✗
Tratamento Cíclico	✗	✗
Retomada Impossível	✓	✓

5.4 Considerações Finais

Este capítulo apresentou uma avaliação para o método CAEH✓. O JCAEH✓, ferramenta que automatiza o método, foi apresentada e os cenários de injeção de faltas, utilizados para analisar a efetividade do método, foram descritos. Adicionalmente, uma discussão de cada cenário foi conduzida. Os resultados obtidos nesse capítulo demonstram a viabilidade e efetividade do método proposto e o seu alinhamento com as metas MET03 e MET04, além de corroborar a hipótese de pesquisa investigada nessa tese de doutorado, citados no Capítulo 1. No próximo capítulo são presentadas as conclusões finais sobre esta tese e os possíveis direcionamentos para trabalhos futuros.

6 Conclusões e Trabalhos Futuros

Este capítulo é dedicado as considerações finais desta tese. Na Seção 6.1 é dada uma visão geral da tese. A Seção 6.2 é dedicada aos principais resultados alcançados desta tese. Na Seção 6.3 a hipótese de pesquisa é analisada. Por fim, a Seção 6.4 é dedicada aos possíveis trabalhos futuros.

6.1 Visão Geral do Trabalho

Esta seção descreve os assuntos abordados nesta tese. No Capítulo 1 foi apresentada a motivação e a caracterização do problema, a definição do problema, a hipótese, as questões de pesquisa, os objetivos e as metas que nortearam o desenvolvimento desta tese. No Capítulo 2 foram abordados os temas que representam a base teórica desta tese, os quais são: sistemas de software adaptativo sensível ao contexto, tratamento de exceção, verificação de modelos e programação por restrições.

No Capítulo 3 foram apresentados os trabalhos relacionados com esta tese de doutorado. Eles trabalhos foram divididos em duas categorias. A primeira, descreve os principais tipos de exceções contextuais, a forma como o comportamento excepcional funciona e identifica pontos do projeto do tratamento de exceção sensível ao contexto que estão propensos a faltas de projeto. Já na segunda categoria, uma discrição dos principais trabalhos sobre verificação de sistemas software adaptativos sensíveis, que dão embasamento às decisões tomadas com respeito a solução proposta nesta tese, foi oferecida.

O CAEH✓foi apresentado no Capítulo 4. A forma como o método modela o comportamento excepcional sensível ao contexto e o mapeia para uma estrutura de Kripke, foi descrito em detalhes nesse capítulo. Além disso, um conjunto de propriedades comportamentais que ajudam a identificar tipos particulares de faltas de projeto foi proposto e formalmente definido. Adicionalmente, as abstrações propostas (exceções contextuais, casos de tratamento e escopo de tratamento) se mostram eficientes em representar aspectos comportamentais relacionados com a definição e a detecção de exceções contextuais,

como o agrupamento, seleção e execução das medidas de tratamento e a retomada do fluxo de controle. É importante destacar que, embora o CAEH✓não enderece o problema de modelagem do comportamento adaptativo do sistema, ele leva em consideração questões importantes sobre a modelagem de comportamento adaptativo descritas na literatura por meio das abstrações de proposições contextuais, restrições semânticas, estados do contexto e restrições de transição.

No Capítulo 5 foi descrita a avaliação do CAEH✓. O JCAEH✓, ferramenta que automatiza método, foi apresentada e os senários de injeção de faltas utilizados para analisar a efetividade do método foram descritos. Adicionalmente, uma discussão de cada cenário foi conduzida. Os resultados obtidos nesse capítulo demonstram a viabilidade e a efetividade do método proposto.

6.2 Resultados Alcançados

Os principais resultados alcançados desta tese foram:

- O CAEH✓, que possibilita a modelagem do comportamento excepcional sensível ao contexto, permitindo mapeá-lo para uma estrutura de Kripke;
- O conjunto de propriedades comportamentais propostas, as quais permitem identificar tipos de faltas de projeto passíveis de ocorrer durante o projeto do tratamento de exceção sensível ao contexto; e
- O JCAEH✓, que cumpre o seu objetivo de automatizar o método proposto provendo uma API para que o projetista possa modelar o comportamento excepcional sensível ao contexto, especificar propriedades e verificá-las de forma automática.

6.3 Análise da Hipótese de Pesquisa

A hipótese de pesquisa declarada no Capítulo 1 é retomada nesta seção. Para isso, ela é declarada a seguir com o propósito de analisar a sua validade frente aos resultados alcançados:

É possível representar o modelo de comportamento do tratamento de exceção sensível ao contexto via estrutura de Kripke e utilizar a técnica de Verificação

de Modelos para identificar, de forma automática, a ocorrência de faltas de projeto nesse modelo.

Com base nos resultados alcançados e na avaliação do CAEH✓descrita no Capítulo 5, onde faltas de projeto são injetadas em um modelo correto e o JCAEH✓é utilizado como ferramenta de verificação de modelos para encontrar, de forma automática, essas faltas, é possível concluir que esta hipótese de pesquisa é considerada como **aceita**.

6.4 Trabalhos Futuros

Os trabalhos propostos como atividades de pesquisa a serem desenvolvidas posteriormente para dar continuidade a este trabalho são listadas abaixo:

- **Tratamento de Exceções Concorrentes**: o termo exceções concorrentes é utilizado para designar a ocorrência simultânea de mais de uma exceção. Tipicamente, quando isso ocorre existe um mecanismo de resolução que permite selecionar as medidas de tratamento mais adequadas face ao conjunto de exceções levantadas. Desse modo, um dos trabalhos em andamento consiste em definir uma função de resolução que permita resolver exceções contextuais concorrentes no CAEH✓. Duas estratégias existentes na literatura foram implementas no JCAEH✓: uma baseadas em árvore de exceções e a outra em prioridades. A estratégia baseada em árvore de exceções permite ao projetista especificar situações de contexto em determinados nós da árvore como critério de decisão. Porém, nenhuma análise mais aprofundada foi conduzida;

- **Composição dos Modelos Adaptativo e Excepcional**: neste trabalho, apenas o comportamento excepcional sensível ao contexto é modelado. Um possível direcionamento de pesquisa consiste em investigar uma maneira de compor os dois modelos de comportamento e entender a forma como eles interagem entre si. O CAEH✓admite apenas dois pontos de interação entre esses comportamentos, (i) os estados excepcionais e (ii) os estados de retomada. Contudo, existem outros aspectos que precisam ser investigados. Por exemplo, pode ser que uma regra de adaptação seja desencadeada no mesmo estado onde uma exceção contextual é levantada. Esse tipo de conflito pode ser entendido como uma falta de projeto, onde o sistema é (mal) projetado para ter dois tipos de reações frente a mesma situação contextual, caracterizando uma espécie de não determinismo emergente;

- **DSL para Modelagem**: embora o JCAEH✓provenha uma API para que os projetistas construam seus modelos, lidar com as abstrações do CAEH✓nesse nível de granularidade pode fazer com que os projetistas percam o interesse em utilizar o método. Um dos trabalhos futuros consiste no desenvolvimento de uma DSL (*Domain-Specific Language*) que permita aumentar o nível de abstração na atividade de modelagem. Além disso, a criação de alguma metáfora visual para exibir o modelo de comportamento e informar ao projetista a violação de propriedades. Esse trabalho já encontra-se em andamento dentro do grupo de pesquisa GREat[1], inclusive com resultados iniciais publicados (LIMA; ANDRADE; ROCHA, 2012);

- **Mecanismo de Tratamento de Exceção**: embora as abstrações do CAEH✓se mostrem apropriadas para a modelagem do comportamento excepcional sensível ao contexto, existe a necessidade de construir um mecanismo de tratamento de exceção sensível ao contexto que permita aos desenvolvedores construí suas aplicações fazendo uso do tratamento de exceções contextuais. Nessa linha, um dos trabalhos desenvolvidos no GREat disponibiliza um sistema de suporte a ubiquidade para o desenvolvimento de aplicações adaptativas sensíveis ao contexto, denominado de SysSU (LIMA et al., 2011). A partir desse trabalho, uma dissertação de mestrado defendida no GREat propôs a utilização do SysSU como componente auxiliar para a detecção de exceções contextuais em aplicações ubíquas orientadas à tarefas (FILHO, 2012). Contudo, o objetivo de pesquisa é que toda a lógica do tratamento de exceções contextuais fique incorporada ao sistema de suporte. Essa decisão implica em vários desafios de pesquisa, em parte devido a mobilidade, desacoplamento e comunicação assíncrona entre os elementos que compõem o sistema; e

- **Teste de Robustez**: os testes de robustez ajudam a aumentar os níveis de confiabilidade dos sistema de software. Particularmente, trabalhos existentes na literatura exploram a geração de casos de teste para validar o comportamento adaptativo dos sistemas adaptativos sensíveis ao contexto (WANG; ELBAUM; ROSENBLUM, 2007). Um direcionamento de pesquisa seria explorar o modelo de comportamento gerado pelo JCAEH✓para a realização de teste de robustez através da injeção de exceções contextuais nesses sistemas. Desse modo, o modelo de comportamento funcionaria como oráculo de teste para verificar se o sistema reage como esperado às exceções contextuais injetadas no sistema.

[1]Grupo de Redes de Computadores, Engenharia de Software e Sistemas, onde esta desse de doutorado foi desenvolvida.

Referências Bibliográficas

ABOWD, G. D. Software engineering issues for ubiquitous computing. In: *Proceedings of the 21st International Conference on Software Engineering*. New York, NY, USA: ACM, 1999. (ICSE'99), p. 75–84. ISBN 1-58113-074-0.

ANDERSSON, J. et al. Software engineering for self-adaptive systems. In: CHENG, B. H. et al. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. Modeling Dimensions of Self-Adaptive Software Systems, p. 27–47. ISBN 978-3-642-02160-2.

AVIZIENIS, A. et al. Basic concepts and taxonomy of dependable and secure computing. *IEEE Transactions on Dependable and Secure Computing*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 1, n. 1, p. 11–33, 2004. ISSN 1545-5971.

BAIER, C.; KATOEN, J.-P. *Principles of Model Checking (Representation and Mind Series)*. [S.l.]: The MIT Press, 2008. ISBN 9780262026499.

BALDAUF, M.; DUSTDAR, S.; ROSENBERG, F. A survey on context-aware systems. *International Journal of Ad Hoc and Ubiquitous Computing*, Inderscience Publishers, Inderscience Publishers, Geneva, SWITZERLAND, v. 2, n. 4, p. 263–277, jun. 2007. ISSN 1743-8225.

BARDRAM, J. E.; CHRISTENSEN, H. B. Pervasive computing support for hospitals: An overview of the activity-based computing project. *IEEE Pervasive Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 6, p. 44–51, January 2007. ISSN 1536-1268.

BEDER, D. M.; ARAUJO, R. B. de. Towards the definition of a context-aware exception handling mechanism. In: *2011 Fifth Latin-American Symposium on Dependable Computing Workshops (LADCW)*. [S.l.: s.n.], 2011. p. 25–28.

BELLAVISTA, P. et al. A survey of context data distribution for mobile ubiquitous systems. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 44, n. 4, p. 24:1–24:45, sep 2012. ISSN 0360-0300.

BETTINI, C. et al. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, Elsevier Science Publishers B. V., v. 6, p. 161–180, April 2010. ISSN 1574-1192.

BOWEN, J. P.; HINCHEY, M. G. Seven more myths of formal methods. *IEEE Software*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 12, n. 4, p. 34–41, jul. 1995. ISSN 0740-7459.

BRITO, P. et al. Architecting fault tolerance with exception handling: Verification and validation. *Journal of Computer Science and Technology*, Springer Boston, v. 24, p. 212–237, 2009. ISSN 1000-9000.

BUHR, P. A.; MOK, W. Y. R. Advanced exception handling mechanisms. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 26, p. 820–836, September 2000. ISSN 0098-5589.

CAMPBELL, R. H.; RANDELL, B. Error recovery in asynchronous systems. *IEEE Transactions on Software Engineering*, IEEE Press, Piscataway, NJ, USA, v. 12, p. 811–826, August 1986. ISSN 0098-5589.

CATARCI, T. et al. Pervasive software environments for supporting disaster responses. *IEEE Internet Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 12, p. 26–37, January 2008. ISSN 1089-7801.

CHEN, J.-H. et al. A smart kitchen for nutrition-aware cooking. *IEEE Pervasive Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 9, p. 58–65, October 2010. ISSN 1536-1268.

CHENG, B. H. et al. Software engineering for self-adaptive systems. In: CHENG, B. H. et al. (Ed.). Berlin, Heidelberg: Springer-Verlag, 2009. cap. Software Engineering for Self-Adaptive Systems: A Research Roadmap, p. 1–26. ISBN 978-3-642-02160-2.

CHENG, B. H. C. et al. (Ed.). *Software Engineering for Self-Adaptive Systems*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009. (Lecture Notes in Computer Science, v. 5525). ISBN 978-3-642-02160-2.

CHETAN, S.; RANGANATHAN, A.; CAMPBELL, R. Towards fault tolerance pervasive computing. *IEEE Technology and Society Magazine*, v. 24, n. 1, p. 38–44, March 2005. ISSN 0278-0097.

CHO, E.-S.; HELAL, S. A situation-based exception detection mechanism for safety in pervasive systems. In: *2011 IEEE/IPSJ 11th International Symposium on Applications and the Internet (SAINT)*. [S.l.]: IEEE, 2011. p. 196 –201. ISBN 978-1-4577-0531-1.

CHO, E.-S.; HELAL, S. Toward efficient detection of semantic exceptions in context-aware systems. In: *9th International Conference on Ubiquitous Intelligence Computing and 9th International Conference on Autonomic Trusted Computing (UIC/ATC)*. [S.l.: s.n.], 2012. p. 826 –831.

CLARK, J.; PRADHAN, D. Fault injection: A method for validating computer-system dependability. *Computer*, v. 28, n. 6, p. 47 –56, jun 1995. ISSN 0018-9162.

CLARKE JR., E. M.; GRUMBERG, O.; PELED, D. A. *Model checking*. Cambridge, MA, USA: MIT Press, 1999. ISBN 0-262-03270-8.

COUTAZ, J. et al. Context is key. *Communications of the ACM*, ACM Press, New York, NY, USA, v. 48, p. 49–53, March 2005. ISSN 0001-0782.

CRISTIAN, F. Exception handling and software fault tolerance. *IEEE Transactions on Computers*, IEEE Computer Society, Washington, DC, USA, v. 31, p. 531–540, June 1982. ISSN 0018-9340.

CUBO, J. et al. A model to design and verify context-aware adaptive service composition. In: *Proceedings of the 2009 IEEE International Conference on Services Computing*. Washington, DC, USA: IEEE Computer Society, 2009. (SCC '09), p. 184–191. ISBN 978-0-7695-3811-2.

DAMASCENO, K. et al. Context-aware exception handling in mobile agent systems: The moca case. In: *Proceedings of the 2006 international workshop on Software Engineering for Large-Scale Multi-Agent Systems*. New York, NY, USA: ACM, 2006. (SELMAS'06), p. 37–44. ISBN 1-59593-395-6.

DEY, A. K. Understanding and using context. *Personal Ubiquitous Computing*, Springer-Verlag, London, UK, v. 5, n. 1, p. 4–7, 2001. ISSN 1617-4909.

DEY, A. K.; ABOWD, G. D. *Towards a Better Understanding of Context and Context-Awareness*. Atlanta, GA, USA, 1999.

EZEKIEL, J.; LOMUSCIO, A. Combining fault injection and model checking to verify fault tolerance in multi-agent systems. In: *Proceedings of The 8th International Conference on Autonomous Agents and Multiagent Systems - Volume 1*. Richland, SC: International Foundation for Autonomous Agents and Multiagent Systems, 2009. (AAMAS '09), p. 113–120. ISBN 978-0-9817381-6-1.

FERNáNDEZ, A. J.; HILL, P. M. A comparative study of eight constraint programming languagesover the boolean and finite domains. *Constraints*, Kluwer Academic Publishers, Hingham, MA, USA, v. 5, n. 3, p. 275–301, jul 2000. ISSN 1383-7133.

FILHO, C. A. B. de Q. *Um Mecanismo de Tratamento de Exceções Sensível ao Contexto para Sistemas Ubíquos Orientados a Tarefas*. Dissertação (Mestrado) — Universidade Federal do Ceará, 2012.

FILHO, F. C.; BRITO, P. H. da S.; RUBIRA, C. M. F. Specification of exception flow in software architectures. *Journal of Systems and Software*, v. 79, n. 10, p. 1397–1418, 2006. ISSN 0164-1212.

FRANCE, R.; RUMPE, B. Model-driven development of complex software: A research roadmap. In: *2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. (FOSE'07), p. 37–54. ISBN 0-7695-2829-5.

GARCIA, A. F. et al. A comparative study of exception handling mechanisms for building dependable object-oriented software. *Journal of Systems and Software*, v. 59, n. 2, p. 197–222, November 2001. ISSN 01641212.

GARLAN, D. et al. Project aura: Toward distraction-free pervasive computing. *IEEE Pervasive Computing*, v. 1, n. 2, p. 22–31, apr-jun 2002. ISSN 1536-1268.

GOODENOUGH, J. B. Exception handling: Issues and a proposed notation. *Communications of the ACM*, ACM Press, New York, NY, USA, v. 18, p. 683–696, December 1975. ISSN 0001-0782.

HENRICKSEN, K.; INDULSKA, J.; RAKOTONIRAINY, A. Modeling context information in pervasive computing systems. In: *Proceedings of the First International Conference on Pervasive Computing*. London, UK: Springer-Verlag, 2002. (Pervasive '02), p. 167–180. ISBN 3-540-44060-7.

HENTENRYCK, P. V.; SARASWAT, V. Strategic directions in constraint programming. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 28, n. 4, p. 701–726, dec 1996. ISSN 0360-0300.

HSUEH, M.-C.; TSAI, T.; IYER, R. Fault injection techniques and tools. *Computer*, v. 30, n. 4, p. 75 –82, apr 1997. ISSN 0018-9162.

HUANG, Y. et al. Concurrent event detection for asynchronous consistency checking of pervasive context. In: *IEEE International Conference on Pervasive Computing and Communications*. Los Alamitos, CA, USA: IEEE Computer Society, 2009. p. 1–9. ISBN 978-1-4244-3304-9.

HUEBSCHER, M. C.; MCCANN, J. A. A survey of autonomic computing–degrees, models, and applications. *ACM Computing Survey*, ACM, New York, NY, USA, v. 40, p. 7:1–7:28, August 2008. ISSN 0360-0300.

INTILLE, S. S. Designing a home of the future. *IEEE Pervasive Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 1, p. 76–82, April 2002. ISSN 1536-1268.

ISO9126. *ISO/IEC 9126:2001: Software Engineering – Product Quality*. Geneva, Switzerland: International Organization for Standardization (ISO), 2001.

JANG, M.; SUH, S.-T. U-city: New trends of urban planning in korea based on pervasive and ubiquitous geotechnology and geoinformation. In: *Proceedings of the 2010 international conference on Computational Science and Its Applications - Volume Part I*. Berlin, Heidelberg: Springer-Verlag, 2010. (ICCSA'10), p. 262–270. ISBN 3-642-12155-1, 978-3-642-12155-5.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. *Computer*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 36, n. 1, p. 41–50, jan. 2003. ISSN 0018-9162.

KIENZLE, J. On exceptions and the software development life cycle. In: *Proceedings of the 4th International Workshop on Exception Handling*. New York, NY, USA: ACM Press, 2008. (WEH'08), p. 32–38. ISBN 978-1-60558-229-0.

KINDBERG, T.; FOX, A. System software for ubiquitous computing. *IEEE Pervasive Computing*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 1, p. 70–81, January 2002. ISSN 1536-1268.

KNIGTH, J. *Fundamentals of Dependable Computing for Software Engineers*. [S.l.]: Chapman and Hall/CRC, 2011. ISBN 1439862559.

KNUDSEN, J. L. Better exception-handling in block-structured systems. *IEEE Software*, IEEE Computer Society Press, Los Alamitos, CA, USA, v. 4, p. 40–49, May 1987. ISSN 0740-7459.

KRAMER, J.; MAGEE, J. Self-managed systems: an architectural challenge. In: *2007 Future of Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. (FOSE'07), p. 259–268. ISBN 0-7695-2829-5.

KRAMER, J.; MAGEE, J. A rigorous architectural approach to adaptive software engineering. *Journal of Computer Science and Technology*, Institute of Computing Technology, Beijing, China, v. 24, p. 183–188, March 2009. ISSN 1000-9000.

KULKARNI, D.; TRIPATHI, A. A framework for programming robust context-aware applications. *IEEE Transactions on Software Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 36, n. 2, p. 184–197, 2010. ISSN 0098-5589.

LADDAGA, R. *Self-Adaptive Software*. [S.l.], 1997. Technical Report 98-12, DARPA BAA.

LEE, P. A.; ANDERSON, T. *Fault Tolerance: Principles and Practice*. 2nd. ed. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 1990. ISBN 0387820779.

LEMOS, R. D. et al. Software engineering for self-adaptive systems: A second research roadmap. In: LEMOS, R. de et al. (Ed.). *Software Engineering for Self-Adaptive Systems*. [S.l.]: Springer, 2012, (Dagstuhl Seminar Proceedings, v. 7475). p. 1–26.

LIMA, F. F. P. et al. Uma arquitetura desacoplada e interoperável para coordenação em sistemas ubíquos. In: *V Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS'2011)*. [S.l.: s.n.], 2011.

LIMA, R.; ANDRADE, R. M. C.; ROCHA, L. S. Uma dsl para modelagem de comportamento de sistemas ubíquos sensíveis ao contexto. In: *II WorkShop de Teses e Dissertações do CBsoft*. [S.l.]: III Congresso Brasileiro de Software de 2012: Teoria e Prática, 2012. (CBSoft'2012).

LIU, Y.; XU, C.; CHEUNG, S. C. Afchecker: Effective model checking for context-aware adaptive applications. *Journal of Systems and Software*, v. 86, n. 3, p. 854–867, 2013.

LOKE, S. W. Building taskable spaces over ubiquitous services. *IEEE Pervasive Computing*, IEEE Computer Society, Los Alamitos, CA, USA, v. 8, n. 4, p. 72–78, oct.-dec. 2009. ISSN 1536-1268.

LOPES, F.; CACHO, N.; BATISTA, T. Um mecanismo de composição de eventos para resolução de exceções sensíveis ao contexto. In: *XXI Simpósio Brasileiro de Engenharia de Software (SBES'2007)*. [S.l.: s.n.], 2007. p. 182–198.

MAGERKURTH, C. et al. Pervasive games: Bringing computer entertainment back to the real world. *Computers in Entertainment*, ACM, New York, NY, USA, v. 3, p. 4–4, jul. 2005. ISSN 1544-3574.

MAIA, M. E.; ROCHA, L. S.; ANDRADE, R. M. C. Requirements and challenges for building service-oriented pervasive middleware. In: *ICPS'09: Proceedings of the 2009 International Conference on Pervasive Services*. New York, NY, USA: ACM Press, 2009. p. 93–102.

MARINHO, F. G. et al. A software product line for the mobile and context-aware applications domain. In: BOSCH, J.; LEE, J. (Ed.). *Software Product Lines: Going Beyond*. [S.l.]: Springer Berlin - Heidelberg, 2010, (Lecture Notes in Computer Science, v. 6287). p. 346–360.

MERCADAL, J. et al. A domain-specific approach to architecturing error handling in pervasive computing. In: *Proceedings of the ACM international conference on Object oriented programming systems languages and applications*. New York, NY, USA: ACM, 2010. (OOPSLA '10), p. 47–61. ISBN 978-1-4503-0203-6.

MEYER, B. *Object-Oriented Software Construction (2nd ed.)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997. ISBN 0-13-629155-4.

MILLER, R.; TRIPATHI, A. Issues with exception handling in object-oriented systems. In: AKSIT, M.; MATSUOKA, S. (Ed.). *ECOOP'97 - Object-Oriented Programming*. [S.l.]: Springer Berlin / Heidelberg, 1997, (Lecture Notes in Computer Science, v. 1241). p. 85–103.

MILNER, R. Ubiquitous computing: Shall we understand it? *The Computer Journal*, v. 49, n. 4, p. 383–389, May 2006. ISSN 0010-4620.

MILNER, R. *The Space and Motion of Communicating Agents*. New York, NY, USA: Cambridge University Press, 2009. ISBN 0521738334, 9780521738330.

OREIZY, P. et al. An architecture-based approach to self-adaptive software. *IEEE Intelligent Systems*, IEEE Educational Activities Department, Piscataway, NJ, USA, v. 14, n. 3, p. 54–62, maio 1999. ISSN 1541-1672.

PARNAS, D. L.; WüRGES, H. Response to undesired events in software systems. In: *Proceedings of the 2nd International Conference on Software Engineering*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1976. (ICSE'76), p. 437–446.

PASCOE, M. J. Adding generic contextual capabilities to wearable computers. In: *ISWC'98: Proceedings of the 2nd IEEE International Symposium on Wearable Computers*. Washington, DC, USA: IEEE Computer Society, 1998. p. 92. ISBN 0-8186-9074-7.

PATIKIRIKORALA, T. et al. A systematic survey on the design of self-adaptive software systems using control engineering approaches. In: *2012 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*. [S.l.: s.n.], 2012. p. 33–42. ISSN 2157-2305.

RANGANATHAN, A.; CAMPBELL, R. H. Provably correct pervasive computing environments. In: *Proceedings of the 2008 Sixth Annual IEEE International Conference on Pervasive Computing and Communications*. Washington, DC, USA: IEEE Computer Society, 2008. p. 160–169. ISBN 978-0-7695-3113-7.

RAYCHOUDHURY, V. et al. Middleware for pervasive computing: A survey. *Pervasive and Mobile Computing*, n. 0, 2012. ISSN 1574-1192.

ROCHA, L.; ANDRADE, R. Towards a formal model to reason about context-aware exception handling. In: *5th International Workshop on Exception Handling (WEH) at ICSE'2012*. [S.l.: s.n.], 2012. p. 27–33.

ROCHA, L. S. et al. Utilizando reconfiguração dinâmica e notificação de contextos para o desenvolvimento de software ubíquo. In: *XXI Simpósio Brasileiro de Engenharia de Software (SBES'2007)*. [S.l.: s.n.], 2007. p. 219–235.

ROCHA, L. S. et al. Ubiquitous software engineering: Achievements, challenges and beyond. In: *Software Engineering (SBES), 2011 25th Brazilian Symposium on*. [S.l.: s.n.], 2011. p. 132–137. ISSN 978-1-4577-2187-8.

SADRI, F. Ambient intelligence: A survey. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 43, p. 36:1–36:66, oct. 2011. ISSN 0360-0300.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: Landscape and research challenges. *ACM Transactions on Autonomous and Adaptive Systems*, ACM, New York, NY, USA, v. 4, n. 2, p. 14:1–14:42, may 2009. ISSN 1556-4665.

SAMA, M. et al. Context-aware adaptive applications: Fault patterns and their automated identification. *IEEE Transactions on Software Engineering*, IEEE Computer Society, Los Alamitos, CA, USA, v. 36, n. 5, p. 644–661, 2010. ISSN 0098-5589.

SATYANARAYANAN, M. Pervasive computing: Vision and challenges. *IEEE Personal Communications*, v. 8, n. 4, p. 10–17, August 2001. ISSN 1070-9916.

SCHILIT, B.; THEIMER, M. Disseminating active map information to mobile hosts. *IEEE Network*, v. 8, n. 5, p. 22 –32, sep/oct 1994. ISSN 0890-8044.

SCHNEIDER, K. *Verification of Reactive Systems: Formal Methods and Algorithms*. [S.l.]: Springer Verlag, 2004. ISBN 3540002960.

SIEWE, F.; ZEDAN, H.; CAU, A. The calculus of context-aware ambients. *Journal of Computer and System Sciences*, Academic Press, Inc., Orlando, FL, USA, v. 77, p. 597–620, jul. 2011. ISSN 0022-0000.

SOYLU, A.; CAUSMAECKER, P. D.; DESMET, P. Context and adaptivity in pervasive computing environments: Links with software engineering and ontological engineering. *Journal of Software*, Academy Publisher, v. 4, n. 9, p. 992–1013, nov. 2009.

SYKES, D. et al. From goals to components: A combined approach to self-management. In: *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*. New York, NY, USA: ACM, 2008. (SEAMS '08), p. 1–8. ISBN 978-1-60558-037-1.

VARSHNEY, U. *Pervasive Healthcare Computing: EMR/EHR, Wireless and Health Monitoring*. [S.l.]: Springer, 2009. ISBN 978-1-4419-0214-6.

WANG, Z.; ELBAUM, S.; ROSENBLUM, D. S. Automated generation of context-aware tests. In: *Proceedings of the 29th International Conference on Software Engineering*. Washington, DC, USA: IEEE Computer Society, 2007. (ICSE '07), p. 406–415. ISBN 0-7695-2828-7.

WEISER, M. The computer for the 21st century. *Scientific American*, v. 265, n. 3, p. 66–75, September 1991.

WEYNS, D. et al. A survey of formal methods in self-adaptive systems. In: *Proceedings of the Fifth International C* Conference on Computer Science and Software Engineering*. New York, NY, USA: ACM, 2012. (C3S2E'12), p. 67–79. ISBN 978-1-4503-1084-0.

WOODCOCK, J. et al. Formal methods: Practice and experience. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 41, n. 4, p. 19:1–19:36, out. 2009. ISSN 0360-0300.

XU, C. et al. Adam: Identifying defects in context-aware adaptation. *Journal of Systems and Software*, Elsevier Science Inc., New York, NY, USA, v. 85, n. 12, p. 2812–2828, dez. 2012. ISSN 0164-1212.

ZHANG, J.; CHENG, B. H. Using temporal logic to specify adaptive program semantics. *Journal of Systems and Software*, v. 79, n. 10, p. 1361–1369, 2006. ISSN 0164-1212.