UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE CIÊNCIAS

DEPARTAMENTO DE COMPUTAÇÃO

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ERICK BARROS DOS SANTOS

RETAKE: ABORDAGEM PARA TESTE EM TEMPO DE EXECUÇÃO DE SISTEMAS DINAMICAMENTE ADAPTATIVOS

FORTALEZA

2020

ERICK BARROS DOS SANTOS

RETAKE: ABORDAGEM PARA TESTE EM TEMPO DE EXECUÇÃO DE SISTEMAS DINAMICAMENTE ADAPTATIVOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software.

Orientadora: Profa. Dra. Rossana Maria de Castro Andrade

Coorientador: Dr. Ismayle de Sousa Santos

FORTALEZA

2020

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S234r Santos, Erick Barros dos.
RETAkE : abordagem para teste em tempo de execução de sistemas dinamicamente adaptativos / Erick Barros dos Santos. – 2020.
116 f. : il. color.

Dissertação (mestrado) – Universidade Federal do Ceará, Centro de Ciências, Programa de Pós-Graduação em Ciência da Computação, Fortaleza, 2020.
Orientação: Profa. Dra. Rossana Maria de Castro Andrade.
Coorientação: Prof. Dr. Ismayle de Sousa Santos.

1. Sistemas adaptativos. 2. Sensibilidade ao contexto. 3. Teste em tempo de execução. I. Título.
CDD 005

ERICK BARROS DOS SANTOS

RETAKE: ABORDAGEM PARA TESTE EM TEMPO DE EXECUÇÃO DE SISTEMAS DINAMICAMENTE ADAPTATIVOS

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Ciência da Computação. Área de Concentração: Engenharia de Software.

Aprovada em: 11/09/2020

BANCA EXAMINADORA

Profa. Dra. Rossana Maria de Castro Andrade (Orientadora)
Universidade Federal do Ceará (UFC)

Dr. Ismayle de Sousa Santos (Coorientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Marcio Espíndola Freire Maia
Universidade Federal do Ceará (UFC)

Profa. Dra. Valéria Lelli Leitão Dantas
Universidade Federal do Ceará (UFC)

Prof. Dr. Pedro de Alcântara dos Santos Neto
Universidade Federal do Piauí (UFPI)

Dedico esse trabalho à Cristina Barros de Abreu, Cláudio dos Santos e Lisandra Juvêncio da Silva. Para essas pessoas eu só devo amor, respeito e lealdade.

AGRADECIMENTOS

Aos meus pais, Cristina Barros e Cláudio dos Santos, que permitiram que eu me tornasse o que sou hoje. Eu teria alcançado pouquíssimo sem o apoio da minha família.

A minha companheira e melhor amiga, Lisandra Juvêncio, por ter sido meu alicerce nos dias mais difíceis. Ficar perto do coração dela sempre tornou a vida simples do jeito que deve ser.

Aos meus orientadores, Prof.ª Dr.ª Rossana Andrade e Dr. Ismayle Santos, por me guiarem na condução deste trabalho e no meu desenvolvimento pessoal e profissional.

Aos meus amigos, Belmondo Rodrigues, Bruno Sabóia, Nayana Carneiro, Rute Castro e Ticiane Pinheiro, por terem me lembrado que ninguém consegue ficar isolado de seus pares.

A toda equipe do laboratório GREat, por me proporcionar um ambiente que me ajudou a alcançar meus objetivos.

Por fim, as instituições Universidade Federal do Ceará e Conselho Nacional de Desenvolvimento Científico e Tecnológico, pelo suporte financeiro durante o meu mestrado. O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

"O mundo não passa de mudança. Nossa vida é apenas percepção." (ANTONINUS; HAYS, 2003, p. 79, tradução nossa).

RESUMO

Um Sistema Dinamicamente Adaptável (DAS, da sigla em inglês) provê suporte para adaptações dinâmicas em tempo de execução a fim de lidar com mudanças no contexto. Essas adaptações podem alterar a estrutura ou comportamento do sistema, assim como a lógica do seu mecanismo de adaptação. Logo, um dos principais desafios da área é a execução de atividades de verificação e validação. Por exemplo, considerando um sistema executando em um smartphone, que adapta suas funcionalidades através de regras no formato de condição-ação, pode ser necessário alterar as regras em tempo de execução para ajustar o sistema a um contexto como o esgotamento rápido de energia, exigindo a redução no nível de bateria que ativa uma funcionalidade desse sistema. No entanto, isso pode inserir defeitos no DAS, levando o mesmo a falhar na execução correta da adaptação. O teste em tempo de execução pode ser realizado para verificar a adaptação do sistema durante suas operações em ambiente de execução final. Para auxiliar na execução desses testes podem ser utilizados modelos de *features*, que constituem representações de alto-nível das funcionalidades do sistema. Entretanto, poucos trabalhos na literatura realizam testes de adaptação durante a execução do sistema e com foco nas regras de adaptação. Dessa forma, este trabalho de mestrado propõe a RETAkE, uma abordagem para teste de DAS em tempo de execução que se baseia na modelagem de contexto e *features*. O foco da abordagem está na execução de sequências de teste no mecanismo de adaptação, permitindo a verificação das regras de adaptação alteradas. Adicionalmente, a RETAkE permite a checagem de propriedades como técnica de suporte aos testes. Como contribuição secundária, foi implementada uma ferramenta que automatiza a execução da abordagem proposta neste trabalho. Para a avaliação do RETAKE, dois DAS móveis são usados para realizar uma prova de conceito com faltas, um teste de mutantes e uma análise do tempo de execução. A ferramenta identificou faltas inseridas na prova de conceito e no teste de mutantes e, por fim, a última avaliação identificou que a ferramenta impacta no tempo de execução dos DASs.

Palavras-chave: sistemas adaptativos; sensibilidade ao contexto; teste em tempo de execução.

ABSTRACT

A Dynamically Adaptive System (DAS) supports dynamic runtime adaptations to handle context changes. These adaptations can change the structure or behavior of the system and the logic of its adaptation mechanism. Therefore, one of the main challenges in the area is the execution of verification and validation activities. For instance, considering a system executing in a smartphone, which adapts its features through rules in the condition-action format, it may be necessary to change the rules at runtime to better fit the system to a context as the rapidly depleting of power resources, which may require a reduction in the battery level that activates a functionality. However, this adaptation may insert defects in the DAS, causing it to fail to correctly perform the adaptation. Runtime testing can be performed to verify the system's adaptation during its operations in the final execution environment. To assist in the execution of these tests, features models can be used, which are high-level representations of the system's functionalities. However, few work in the literature perform adaptation tests during system execution focusing on adaptation rules. Thus, this master's thesis proposes RETAkE, an approach for the DAS test at runtime based on the system context and feature modeling. The approach focuses on executing test sequences in the adaptation mechanism, verifying the changed adaptation rules. Additionally, RETAkE also has a property checking technique that supports the testing process. As a secondary contribution, it was implemented a tool that automates the execution of the approach proposed in this work. Moreover, two mobile DASs are used in three different RETAkE evaluations, as follows: a proof of concept for detecting faults, a mutations test, and an analysis of the DASs adaptations in execution time. The tool identified the injected faults in the proof of concept as well as in the mutations testing and the last evaluation showed the impact of the tool during the DASs execution time.

Keywords: adaptive systems; context awareness; runtime testing.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS E SIGLAS

V&V	Verificação e Validação
DAS	Dynamically Adaptive System
RA	Regras de Adaptação
LPS	Linha de Produto de Software
LPSD	Linha de Produto de Software Dinâmica
CFM	Context Feature Model
eCFM	Extended Context Feature Model
DFTS	Dynamic Feature Transition System
C-KS	Context Kripke Structure
RETAkE	RuntimE Testing of dynamically Adaptive systEms
CONTroL	CONtext-variability-based software Testing Library
UML	Unified Modeling Language
JSON	JavaScript Object Notation
XML	Extensible Markup Language

SUMÁRIO

1 INTRODUÇÃO

Esta dissertação de mestrado tem o objetivo de propor uma abordagem que executa testes para Sistemas Dinamicamente Adaptativos em tempo de execução. A abordagem foca na verificação da adaptação do sistema durante suas operações regulares, sendo que essa é suportada por uma atividade de checagem de propriedades comportamentais. Também é apresentado uma ferramenta de monitoramento e controle que executa as principais atividades da abordagem automaticamente.

A Seção 1.1 deste capítulo discorre sobre o contexto no qual a presente dissertação está inserida. A Seção 1.2 expõe brevemente uma visão geral do tema de pesquisa e os pontos que motivam a execução deste trabalho. A Seção 1.3 apresenta o objetivo e a metodologia utilizada para alcançá-lo. Por fim, a Seção 1.4 sumariza como toda a dissertação está estruturada.

1.1 Contextualização

Um Sistema Dinamicamente Adaptativo (DAS) é um sistema que se adapta em resposta à mudança de condições ambientais (ALVES *et al.*, 2009). Essas adaptações são acionadas para que o DAS possa, entre outras razões, recuperar-se de falhas (KRUPITZER *et al.*, 2015) ou dar continuidade ao cumprimento de seus objetivos (FREDERICKS; DEVRIES; CHENG, 2014). Entre as opções para decidir sobre as adaptações, um DAS pode reconfigurar-se usando regras que alteram as *features* do sistema (ALVES *et al.*, 2009). Dessa forma, é possível que o DAS adapte seu próprio comportamento ou estrutura.

Em adição às alterações previamente mencionadas, o DAS também pode alterar sua própria lógica de adaptação durante sua execução. Assim, é possível que novas funcionalidades sejam vinculadas em tempo de execução (CAPILLA; ORTIZ; HINCHEY, 2014), afetando as regras de adaptação do sistema.

Se as regras de adaptação não forem corretamente implementadas, o DAS pode ter falhas na adaptação de suas funcionalidades (SANTOS, 2017). O problema pode ser agravado porque uma falha no início do processo de adaptação pode ser propagada através de todas as etapas restantes do processo (PÜSCHEL *et al.*, 2014b). Isso pode levar o sistema a falhar continuamente durante suas reconfigurações.

Com o objetivo de verificar as adaptações do DAS, é importante avaliar as suas adaptações disparadas por mudanças de contexto durante a execução do sistema (SANTOS,

2017). Logo, é necessário realizar atividades de Verificação e Validação (V&V) em tempo de execução através do monitoramento de informações do DAS (TAMURA *et al.*, 2013). Dentre as técnicas disponíveis, trabalhos na literatura realizam tanto a verificação de execuções de modo passivo (QIN *et al.*, 2019) como de forma proativa através testes em tempo de execução[1] (FREDERICKS; CHENG, 2015).

1.2 Motivação

Considerando a necessidade de avaliar as adaptações desencadeadas por mudanças de contexto, teste de software é uma técnica promissora para essa atividade. Nessa direção, revisões sistemáticas da literatura listam desafios de teste de software para DAS (SANTOS *et al.*, 2017b; SIQUEIRA *et al.*, 2016), de forma que podem ser citados como lidar com configurações incorretas definidas em tempo de execução, e exercer mais controle sobre o sistema durante sua execução. Especificamente relacionado aos testes em tempo de execução, vale mencionar que essa atividade pode impactar no desempenho do sistema (LAHAMI; KRICHEN; JMAIEL, 2016), e o dinamismo do DAS pode tornar os testes inconsistentes (FREDERICKS; DEVRIES; CHENG, 2014).

Abordagens de testes em tempo de execução atuam de formas variadas para lidar com os desafios mencionados. Alguns trabalhos focam na utilização do contexto de execução do sistema para reduzir a quantidade de testes para executar (FREDERICKS; CHENG, 2015; EBERHARDINGER; HABERMAIER; REIF, 2017). Outros trabalhos propõem arquiteturas de testes que se integram com o DAS para ter mais controle sobre o mesmo (LAHAMI; KRICHEN; JMAIEL, 2016; LEAL; CECCARELLI; MARTINS, 2019). Entretanto, o foco dessas abordagens é a verificação das funcionalidades do sistema após a adaptação.

Neste sentido, torna-se relevante pesquisar como verificar não apenas as funcionalidades do DAS, mas também o mecanismo de adaptação em tempo de execução. Isso é relevante principalmente em casos de alteração na lógica de adaptação, sendo importante executar técnicas proativas como testes, que estimulam o DAS com entradas de contexto. Adicionalmente, pode ser relevante integrar os testes com técnicas passivas visando detectar falhas de adaptações (METZGER, 2011).

[1] Nesta dissertação, testes em tempo de execução (em inglês, *runtime*) são testes realizados enquanto o sistema está em execução no seu ambiente de operação final. Já testes em tempo de projeto (em inglês, *design time*) são testes realizados durante todo o ciclo de desenvolvimento do sistema.

1.3 Objetivo e Metodologia

Este trabalho tem como objetivo propor uma abordagem de testes para detectar falhas de adaptação em tempo de execução de DAS. A abordagem objetiva a checagem da variabilidade do sistema, garantindo que as regras de adaptação foram corretamente executadas. Para isso, realiza-se a verificação das respostas de adaptação através de uma técnica passiva e outra proativa: checagem de propriedades comportamentais e testes de software. Dessa forma, a abordagem deve permitir a identificação de falhas no sistema ao longo de suas sequências de adaptações.

A abordagem proposta nesta dissertação baseia-se no trabalho de Santos, I. S. et al. (SANTOS *et al.*, 2016; SANTOS, 2017; SANTOS *et al.*, 2018). Esses trabalhos têm o propósito de verificar o DAS em tempo de projeto, desconsiderando cenários dinâmicos que podem ocorrer durante a execução no ambiente de operação. Para evoluir esses trabalhos, a abordagem atual permite a geração e execução de testes mais compatíveis com o estado de contexto do DAS. Para isso, utiliza-se um modelo para especificar a variabilidade de contexto (SANTOS *et al.*, 2016) (Subseção 2.2.3) e outro para a variabilidade de *features* (SANTOS *et al.*, 2017a) (Subseção 2.2.2), em que este último mapeia as *features* com seus respectivos contextos.

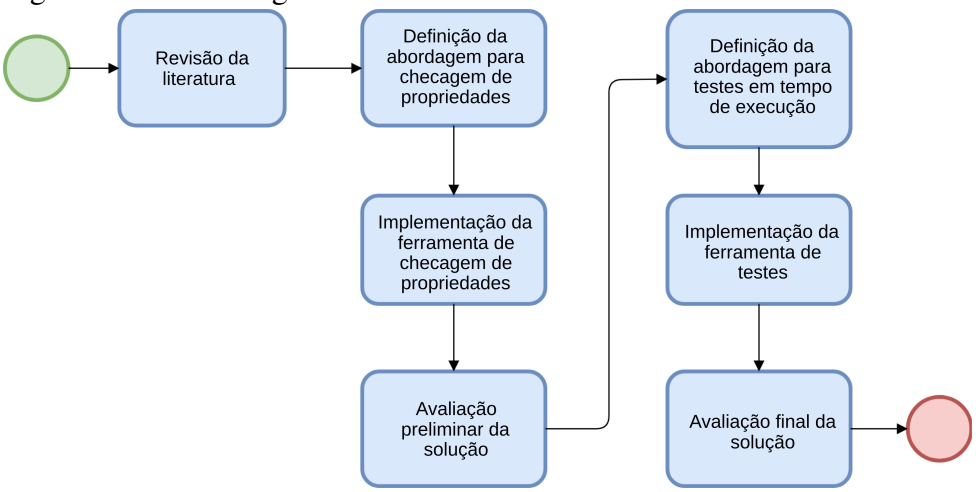
Outra contribuição relacionada a este trabalho é uma ferramenta que automatiza as atividades da abordagem, executando lado a lado com o DAS. Essa ferramenta evolui um trabalho anterior (SANTOS *et al.*, 2018) de testes de DAS em tempo de projeto. A utilização do artefato de software viabiliza a instrumentação do sistema através de anotações no código-fonte, permitindo controlar o processo de adaptação do sistema para não só checar o seu estado atual, mas também inserir entradas de contexto que exercitem adaptações.

Vale ressaltar que o escopo deste trabalho está relacionado à verificação do mecanismo de adaptação do DAS, de forma que essa atividade acontece após as adaptações. Esse ponto de verificação é importante para detectar falhas logo após seu aparecimento, podendo auxiliar na sua não propagação para as demais fases de adaptação (PÜSCHEL *et al.*, 2014b).

Para alcançar o objetivo do trabalho foi seguida a metodologia ilustrada na Figura 1.

O primeiro passo consiste na realização de uma revisão da literatura na área de verificação de DAS em tempo de execução, mas com ênfase em testes de software. Essa revisão permitiu obter um panorama da área de pesquisa e a identificação de trabalhos relacionados. Para isso, a revisão foi conduzida com atividades baseadas no método do mapeamento sistemático (PETERSEN *et al.*, 2008). Este método foi selecionado como base por ser uma revisão com características rigorosas, tais como: strings de busca, seleção de bases e critérios de inclusão/exclusão

Figura 1 – Metodologia



Fonte: elaborada pelo autor.

de trabalhos.

Após a revisão e análise das técnicas de verificação, optou-se por dividir a concepção da abordagem em uma técnica passiva e outra proativa. A começar pela técnica passiva, definiram-se primeiro as atividades relativas à checagem de propriedades para DAS. Esse tipo de atividade é relevante para checar o sistema em seu estado atual. Na sequência, realizou-se uma implementação parcial da ferramenta para a automação das atividades de checagem de propriedades. Os resultados dessa primeira parte foram publicados em (SANTOS; ANDRADE; SANTOS, 2019a).

Optou-se por fazer uma prova de conceito como avaliação preliminar. A prova de conceito é conduzida com o uso da ferramenta em dois exemplos de aplicações adaptativas: GREat Tour (LIMA *et al.*, 2013) e Phone Adapter (SAMA *et al.*, 2010). A prova de conceito foi selecionada para verificar se é factível utilizar a solução gerada em tempo de execução dos DASs.

Na sequência, definiram-se as atividades para os testes de adaptação. De forma similar ao passo anterior, implementou-se uma ferramenta para automatizar a execução das atividades de testes. Essa ferramenta também utiliza conceitos semelhantes aos da checagem de propriedades para executar os testes.

No que diz respeito à avaliação final da solução, foi escolhida a técnica de testes de mutantes (JIA; HARMAN, 2010). A avaliação também poderia ter se sustentado na existência de falhas reais ou na inserção de falhas baseadas na experiência (ANDREWS; BRIAND; LABICHE, 2005). Entretanto, o teste de mutantes habilita a inserção de falhas de maneira sistematizada através de operadores sintáticos. Adicionalmente, o teste de mutantes tem sido

aplicado com sucesso por outras abordagens de teste na literatura (QIN *et al.*, 2016; SANTOS, 2017). Novamente, a avaliação é realizada com os dois exemplares de DAS previamente mencionados.

Dado que atividades de verificação em tempo de execução podem sobrecarregar o DAS, torna-se necessário medir o impacto nas adaptações. Para isso, foi utilizada a versão completa da implementação da abordagem, de forma que se mediu o tempo para o DAS completar as adaptações em configurações variadas.

1.4 Estrutura da Dissertação

Neste capítulo foi apresentada uma visão geral da pesquisa desenvolvida nesta dissertação. Foi apresentada a contextualização e motivação que levaram a condução do trabalho. Também foram descritos o objetivo e a metodologia seguida para alcançá-lo.

O restante da dissertação está organizada em cinco capítulos que são sumarizados a seguir.

No **Capítulo 2** são discutidos os conceitos que fundamentam este trabalho. Neste sentido, são explorados os conceitos de contexto, modelos para sistemas adaptativos, checagem de propriedades e testes em tempo de execução.

No **Capítulo 3** são apresentados os principais trabalhos relacionados a presente dissertação. É sumarizado como esses trabalhos atuam no problema tratado e como contribuíram para a construção da solução proposta.

No **Capítulo 4** é discutida a abordagem proposta para a verificação em tempo de execução. Também é detalhado o artefato de software que implementa as atividades apresentadas na abordagem.

No **Capítulo 5** são apresentadas três avaliações. A primeira é uma prova de conceito focada na detecção de falhas das propriedades comportamentais. A segunda é um teste de mutantes aplicado aos teste em tempo de execução e versões defeituosas de exemplares e DAS. A terceira avaliação diz respeito à medição do impacto da abordagem no tempo de adaptação dos exemplares.

Finalmente, no **Capítulo 6**, esta dissertação é concluída com o resumo das suas principais contribuições. Também são listados as limitações e os possíveis trabalhos futuros para a evolução da abordagem proposta.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo discorre sobre a fundamentação teórica necessária para este trabalho, apresentando assim as principais definições para Sistema Dinamicamente Adaptativo (Dynamically Adaptive System (DAS)) e Teste de Software em Tempo de Execução. Para isso, este capítulo está organizado como segue: na Seção 2.1 é descrito um cenário que será utilizado como exemplo ao longo da dissertação; na Seção 2.2 são apresentados os principais conceitos de DAS; na Seção 2.3 são resumidos os conceitos relativos à garantia de adaptação para DAS; e, finalmente, a Seção 2.4 apresenta a conclusão do capítulo.

2.1 Aplicação Exemplo

Para ilustrar os conceitos relacionados com o presente trabalho, a aplicação GREat Tour (MARINHO *et al.*, 2013) será usada como exemplo. Esta mesma aplicação foi utilizada por Santos (2017), cujo trabalho foi estendido nesta dissertação. O GREat Tour é uma aplicação móvel para guiar visitantes em *tours* pelo laboratório GREat[1]. Essa aplicação tem o propósito de ajudar usuários através das *features* texto, vídeo, fotos e login.

O GREat Tour é capaz de ativar/desativar suas *features* baseada nas seguintes informações de contexto: nível de bateria do dispositivo, conexão com fonte de energia e conexão com a Internet. Para realizar a adaptação das *features*, a aplicação utiliza Regras de Adaptação (RA) na forma de <condição de guarda de contexto, ações de adaptação> como sumarizado na Tabela 1. Nessa tabela, os valores *on*/*off* indicam ações de adaptação para ativar/desativar as *features*. Por exemplo, a regra de adaptação RA1 desativa as *features Vídeo* e *Foto* quando o nível de bateria é menor ou igual à 30% e a fonte de energia está desconectada.

Tabela 1 – Regras de Adaptação do GREat Tour

Identificador da Regra	condição de guarda de contexto	Login	Vídeo	Foto	Texto
RA1	bateria $\leq$ 30% $\wedge$ carregando = falso		off	off	
RA2	internet = verdadeiro	on			
RA3	internet = falso	off			
RA4	carregando = verdadeiro		on	on	
RA5	bateria > 30% $\wedge$ carregando = falso		on	on	off
RA6	bateria $\leq$ 30%				on
RA7	bateria >30%				off

Fonte: elaborada pelo autor.

[1] http://www.great.ufc.br/

Dado um cenário em que o GREat Tour fosse capaz de adaptação dinâmica, o mecanismo de adaptação poderia decidir alterar suas próprias regras em tempo de execução. Por exemplo, a seguinte situação pode ser considerado para o GREat Tour: as regras RA1, RA5, RA6 e RA7 mudam suas condições para ativar as *features Vídeo* e *Foto* somente quando o nível de bateria é alta. Adicionalmente, a *feature Texto* é ativada quando o nível de bateria for menor que médio (<70%). Essa adaptação mimetiza o cenário em que o mecanismo de adaptação identifica que o nível de bateria está reduzindo muito rápido. Por conta disso, o mecanismo atualiza as regras para ativar *features* que precisem de mais recursos energéticos (e.g., *Vídeo*) apenas quando a bateria tiver com nível de carga alta. A Tabela 2 resume as regras adaptadas destacadas em cinza.

Tabela 2 – Regras de adaptação alteradas da aplicação GREat Tour

Identificador da Regra	**condição de guarda de contexto**	**Login**	**Vídeo**	**Foto**	**Texto**
RA1	bateria $\leq$ 70% $\wedge$ carregando = falso		off	off	
RA2	internet = verdadeiro	on			
RA3	internet = falso	off			
RA4	carregando = verdadeiro		on	on	
RA5	bateria > 70% $\wedge$ carregando = falso		on	on	off
RA6	bateria $\leq$ 70%				on
RA7	bateria >70%				off

Fonte: elaborada pelo autor.

A alteração nas regras anteriormente apresentadas pode resultar em cenários de falhas. Por exemplo, é possível que o mecanismo de adaptação do DAS altere o código-fonte das regras, mas não seus modelos internos que permitam a correta adaptação (PÜSCHEL *et al.*, 2014b). Essa falha de sincronia pode fazer com que as regras sejam incorretamente executadas. Nesse sentido, é relevante verificar essas alterações de regras após mudanças de contexto.

2.2 Sistema Dinamicamente Adaptativo

As atividades que compõem o processo de evolução dos sistemas de software geralmente são executadas manualmente durante os períodos em que o mesmo não está em operação (DE LEMOS *et al.*, 2013). No entanto, este ciclo de vida tornou-se incompatível com novos cenários de uso dos sistemas, sendo que esses devem se adaptar para lidar com seu ambiente e com requisitos em constante evolução (BARESI; GHEZZI, 2010).

Por exemplo, sistemas distribuídos e baseados em componentes tendem a se adaptar

em tempo de execução não apenas para lidar com as mudanças de requisitos, mas também para aumentar sua confiabilidade e mitigar possíveis falhas que possam ocorrer (LAHAMI; KRICHEN; JMAIEL, 2016). Esse tipo de sistema que é capaz de realizar adaptações em tempo de execução será referenciado ao longo deste trabalho como Sistema Dinamicamente Adaptativo (i.e., DAS da sigla em inglês) (ALVES *et al.*, 2009).

Como apresentado por Krupitzer *et al.* (2015), as técnicas que realizam as adaptações em sistemas adaptativos podem ser separadas pelo tipo de mudança realizada: parâmetro, estrutural e contexto. A adaptação de parâmetro consiste na alteração de comportamento de um componente. Já a adaptação de estrutura consiste na integração de novos componentes ou algoritmos, de forma que esses também podem implicar em mudanças de comportamento (MCKINLEY *et al.*, 2004). Por fim, o sistema pode não só alterar seu comportamento ou estrutura, como atuar no seu próprio contexto de execução.

Uma especialização importante de DAS são os sistemas capazes de auto-adaptação: a habilidade do sistema em realizar adaptações a partir do monitoramento tanto do ambiente como de si próprio (CHENG *et al.*, 2009). Essas informações podem auxiliar na tomada de decisões de adaptação mais adequadas dentro de limites de custo e tempo (SALEHIE; TAHVILDARI, 2009). Sistemas com capacidade de auto-adaptação possuem as chamadas propriedades auto-* (*self-**), sendo cada uma delas descrita a seguir, de acordo com Salehie e Tahvildari (2009):

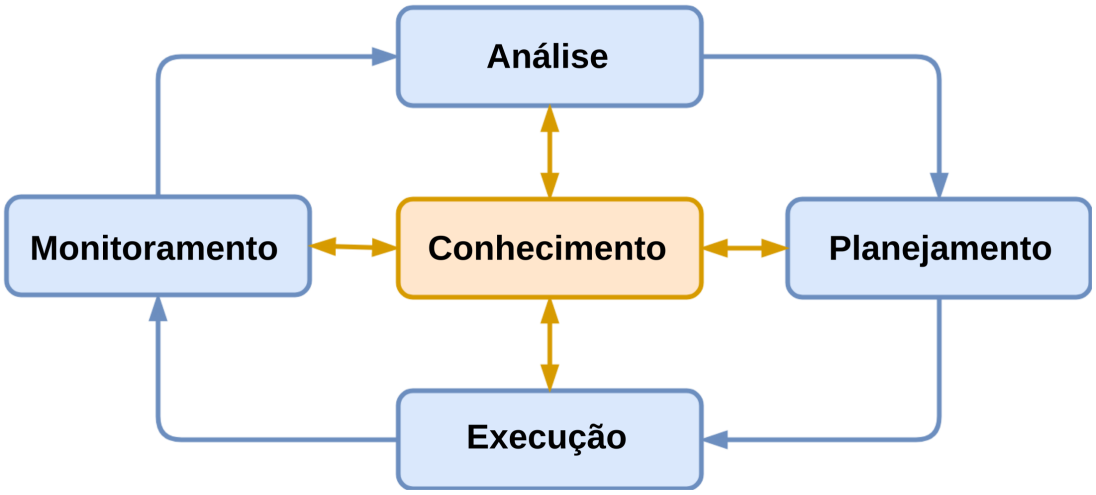
a) **Auto-configuração:** habilidade do sistema de alterar a própria estrutura de seus componentes através da inserção, remoção e substituição;
b) **Auto-cura:** capacidade do sistema de identificar inconsistências em sua execução e agir para se recuperar;
c) **Auto-otimização:** esta propriedade permite que o sistema otimize seus próprios atributos de qualidade, tais como tempo de resposta e taxa de vazão;
d) **Auto-proteção:** capacidade do sistema de se proteger de falhas de segurança relacionadas a ataques.

Em um nível mais elevado de abstração, a estrutura dos sistemas dinâmicos pode seguir a visão clássica de elementos autonômicos proposta por Kephart e Chess (2003). Essa estrutura implica a composição de dois elementos: recursos gerenciados e uma lógica de adaptação (KRUPITZER *et al.*, 2015). O recurso gerenciado pode compreender de elementos de hardware, tais como memória e processador, e software, como serviços de aplicação e sistemas legados. A lógica de adaptação controla os recursos gerenciados e executa suas adaptações

(KEPHART; CHESS, 2003), de forma que essa lógica pode compor um mecanismo de adaptação (KRUPITZER *et al.*, 2015).

De acordo com Brun *et al.* (2009), o ciclo de feedback compreende um mecanismo genérico para que a lógica de adaptação controle a adaptação do sistema. O ciclo de feedback é um elemento central derivado da teoria do controle e que consiste em quatro atividades: coleta, análise, decisão e ação. É possível relacionar essas atividades genéricas com o ciclo de feedback projetado por Kephart e Chess (2003) para sistemas autonômicos: Monitoramento (M), Análise (A), Planejamento (P) e Execução (E). Essas atividades, apoiadas por uma base de Conhecimento (K, da sigla em inglês), compõem o chamado ciclo MAPE-K, amplamente utilizado como modelo de referência para adaptações (KRUPITZER *et al.*, 2015). A Figura 2 ilustra os elementos do ciclo MAPE-K de acordo com a arquitetura proposta pela IBM (COMPUTING *et al.*, 2006). As descrições para esses elementos são providas a seguir:

Figura 2 – Ciclo MAPE-K



Fonte: adaptado de Computing *et al.* (2006).

a) **Monitoramento**: o elemento de monitoramento é responsável pela coleta, agregação, filtragem e comunicação dos dados relevantes obtidos do recurso gerido;
b) **Análise:** esse elemento faz uso dos dados coletados no *Monitoramento* para correlacionar e modelar situações complexas, permitindo ao mecanismo gestor analisar mudanças no ambiente ou no recurso gerenciado. Assim, o papel do elemento de *Análise* é prever situações futuras que requerem ações de adaptação;
c) **Planejamento:** com base nas informações geradas pelo elemento de *Análise*, o elemento de *Planejamento* deve fornecer o plano de adaptação para alcançar os objetivos do recurso gerenciado;

d) **Execução:** o elemento de *Execução* é responsável pela execução das ações do plano de adaptação gerado;
e) **Conhecimento:** o *Conhecimento* é um elemento que funciona como um repositório compartilhado que envia e recebe dados para os demais elementos. Os dados armazenados aqui incluem sintomas de adaptação, políticas, requisições de mudança e planos de mudança.

2.2.1 Sensibilidade ao Contexto

Na definição apresentada por Alves *et al.* (2009), é afirmado que um DAS adapta sua estrutura e comportamento em tempo de execução para lidar com mudanças de contexto. Abowd *et al.* (1999, p. 304, tradução nossa) define contexto como "[...] qualquer informação que pode ser usada para caracterizar a situação de uma entidade. Uma entidade é uma pessoa, lugar ou objeto considerado relevante para a interação entre um usuário e uma aplicação, incluindo o usuário e as próprias aplicações". Por exemplo, a localização geográfica de um usuário caracteriza sua situação e uma aplicação, que pode reagir a mudanças de contexto e que pode usar essas informações para oferecer funcionalidades personalizadas (por exemplo, uma descrição textual sobre a localização).

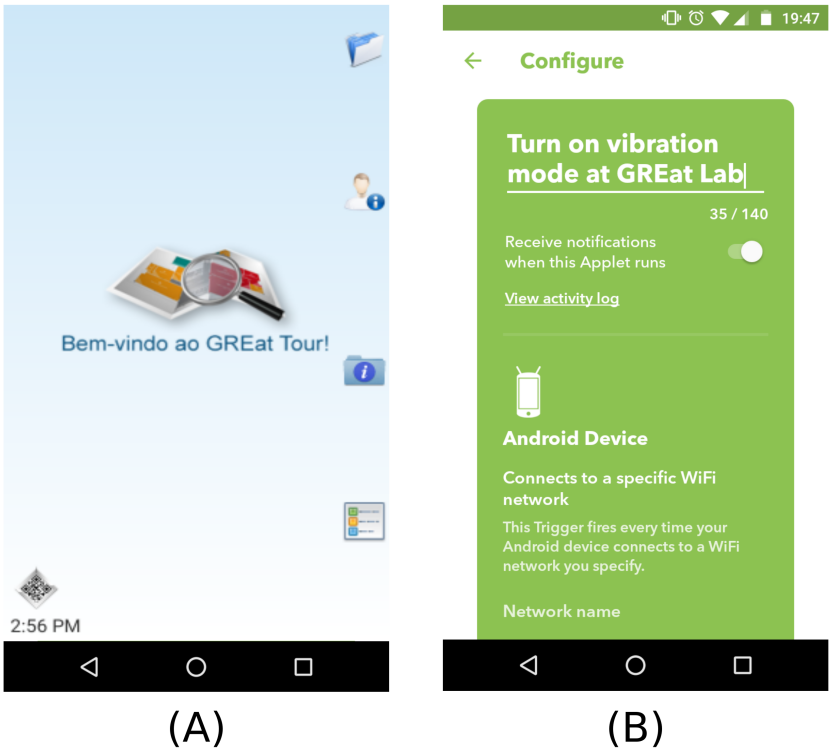
Uma vez que uma mudança de contexto é detectada, a lógica de sistemas adaptativos pode usar diferentes tipos de critérios em seu processo de tomada de decisão: modelos, regras/-políticas e objetivos (KRUPITZER *et al.*, 2015). Como discutido por Krupitzer *et al.* (2015), as adaptações baseadas em alterações de parâmetros são possíveis a partir de abordagens baseadas em regras. Nesses sistemas, uma regra pode ser representada na forma condição-ação, em que a condição representa uma alteração de contexto, e a ação determina uma reconfiguração do sistema (SANTOS, 2017).

Este trabalho de mestrado se concentra em adaptações desencadeadas pela mudança no contexto e que usam regras para representar as respostas de adaptação. O padrão de condição-ação é especialmente relevante para sistemas que precisam de abordagens mais leves e determinísticas (ALVES *et al.*, 2009), podendo ser citado como exemplo as aplicações móveis.

Diversas aplicações têm capturado contexto dos seus usuários para disparar adaptações com regras e, com isso, decidir sobre os eventos que desencadeiam ações de adaptação. O GREat Tour (LIMA *et al.*, 2013) (Figura 3-A), aplicação de exemplo desta dissertação (ver Seção 2.1), é um DAS guiado por regras de adaptação. O IFTTT[2] é outro exemplo de aplicativo

[2] https://ifttt.com/

Figura 3 – Exemplos de sistemas que se adaptam baseado em contexto: GREat Tour (A) e IFTTT (B)



Fonte: elaborada pelo autor.

sensível ao contexto que permite aos seus usuários definirem suas próprias regras de adaptação. A Figura 3-B mostra um exemplo de regra do IFTTT que ativa a vibração do smartphone em uma localização específica. Como exemplo de tecnologia para especificação de regras ao nível de código-fonte, a Google Awareness API[3] oferece uma solução que abstrai do desenvolvedor a definição de contextos que disparam a execução de funções.

2.2.2 Modelagem da Variabilidade Dinâmica para DAS

Macías-Escrivá *et al.* (2013) identificaram várias abordagens que têm sido usadas para apoiar a adaptação em DAS: baseados em componentes, baseados em modelos, multi-agentes e inspiradas na natureza.

Além dessas opções, um DAS também pode utilizar o conceito de modelo de *features* de Linha de Produto de Software (LPS), que consiste de um conjunto de sistemas que compartilham uma coleção de recursos desenvolvidos a partir de artefatos reutilizáveis (HALLSTEINSEN *et al.*, 2008). Um modelo de *features* (KANG *et al.*, 1990) serve para modelar a variabilidade das Linhas de Produto de Software através de uma representação

[3] https://developers.google.com/awareness/

hierárquica.

Para permitir o dinamismo do DAS, podem ser aplicados conceitos de Linha de Produto de Software Dinâmica (LPSD), em que as variações dos produtos de software são geradas em tempo de execução para lidar com situações específicas (HALLSTEINSEN *et al.*, 2008). Por exemplo, foi encontrado na literatura trabalhos que têm aplicado modelos de *features* para garantir seu comportamento em tempo de execução de aplicações móveis auto-adaptativas (MIZOUNI *et al.*, 2014), Sistemas Ciber-Físicos (ISLAM; AZIM, 2018) e sistemas baseados em Software como Serviço (MOUSA; BENTAHAR; ALAM, 2019).

No presente trabalho utiliza-se a definição de *feature* apresentada por Kang *et al.* (1990, p. 1, tradução nossa) "[...] aspectos ou características do domínio visíveis aos usuários.". Um modelo de *features* organiza seus elementos de maneira hierárquica onde cada *feature* pode ser filha de uma única *feature* pai. Adicionalmente, cada *feature* pode ser opcional ou mandatória, de forma que está última demanda que a *feature* sempre esteja disponível em uma variante do sistema. Por fim, o modelo de *features* clássico disponibiliza a seguinte lista de relacionamentos entre *features* (BENAVIDES; SEGURA; RUIZ-CORTÉS, 2010):

a) **OR**: uma ou mais das *features* filhas de uma *feature* pai pode estar presente;
b) **XOR**: apenas uma das *features* filhas de uma *feature* pai pode estar presente;
c) **Requer**: se uma *feature* A tem uma relação do tipo *Requer* com a *feature* B, então B deve estar presente sempre que A também estiver;
d) **Exclui**: se uma *feature* A tem uma relação do tipo *Exclui* com a *feature* B, então B não deve estar presente sempre que A estiver.

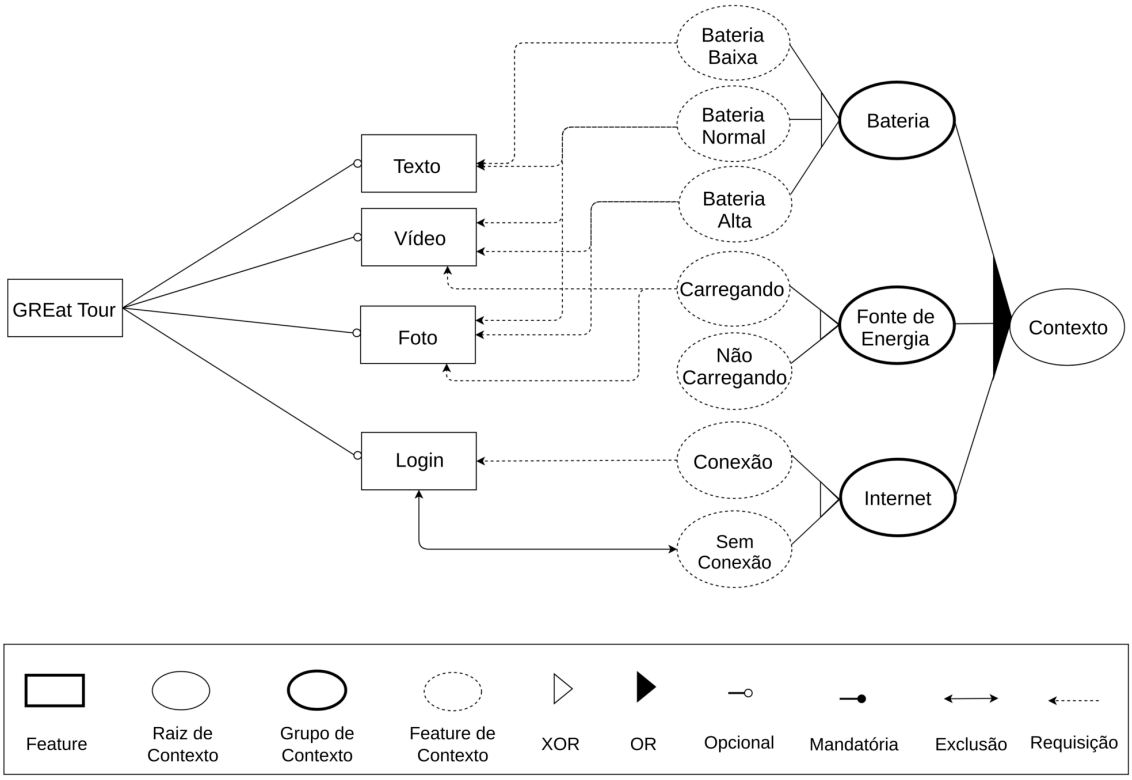
Esse modelo pode ser utilizado para representar as funcionalidades do DAS em alto-nível, permitindo verificar se sua estrutura está correta em tempo de execução. Entretanto, o modelo de *features* clássico é definido em tempo de projeto, sendo incapaz de alterar o estado de suas *features* de acordo com mudanças ambientais (CAPILLA; ORTIZ; HINCHEY, 2014). Para permitir a variação dos produtos de um DAS em tempo de execução, o sistema precisa realizar a chamada variabilidade dinâmica. Capilla *et al.* (2014) definem a variabilidade dinâmica como a capacidade do sistema de alterar não só o estado de suas *features*, como também alterar a estrutura dos modelos em tempo de execução.

Dentre os tipos de modelos de *features*, o Context Feature Model (CFM) foi proposto por Saller, Lochau e Reimund (2013) como uma opção para modelar sistemas que adaptam a disponibilidade de suas *features* em tempo de execução após mudanças do ambiente. O

diferencial desse modelo é representar tanto as *features* quanto as *features* de contexto, que são as variáveis de contexto responsáveis pela ativação/desativação das *features*.

Uma vez que o CFM tem limitações na representação de restrições entre *features* de contexto, o trabalho de Santos *et al.* (2017a) propôs o Extended Context Feature Model (eCFM) como uma extensão ao CFM. A vantagem de usar o eCFM é que ele permite especificar restrições nas *features* de contexto através do uso *Grupos de Contexto*: elementos que permitem grupos OR e XOR semelhantes ao modelo de *feature* original. Após a criação do modelo eCFM, o engenheiro de software tem uma visão em alto-nível sobre quais contextos afetam as *features* do sistema.

Figura 4 – Extended Context Feature Model da aplicação GREat Tour



Fonte: adaptado de Santos (2017)

A Figura 4 ilustra o eCFM da aplicação exemplo. A parte esquerda da figura ilustra as *features*, tal qual o modelo clássico, e a parte direita, as *features* de contexto. A aplicação de exemplo fornece aos usuários as *features Texto*, *Vídeo*, *Foto* e *Login*, que podem ser ativadas/desativadas após mudanças no nível da bateria do dispositivo, na conexão com uma fonte de energia externa ou na conexão com a Internet. Por exemplo, se as *features* de contexto *Bateria Alta* ou *Bateria Normal* estiverem ativas, então a aplicação deve ativar as *features Vídeo* e *Foto*. Essa configuração também é válida quando o dispositivo móvel está ligado a uma fonte

de alimentação externa. A regra de adaptação para o cenário é apresentada no modelo como uma linha tracejada conectando as *features* de contexto *Bateria Normal* e *Bateria Alta* às *features Vídeo* e *Foto*.

A vantagem de se utilizar o modelo eCFM é que o mesmo representa as regras de adaptação do DAS em alto-nível. Dessa forma, as alterações nas regras podem ser analisadas a partir do modelo, permitindo assim a verificação do sistema após adaptações. Neste trabalho, é utilizada a definição de Santos (2017) para regras de adaptação: um conjunto de ações atômicas de ativação/desativação que mudam simultaneamente o estado das *features* quando uma condição de guarda de contexto é satisfeita. Nesse sentido, o cenário de adaptação anteriormente discutido pode ser representado como uma tupla <Bateria Baixa, *desativar*(Vídeo, Foto)>, onde o primeiro elemento é a condição de guarda de contexto e o segundo é uma ação de ativação das *features Vídeo* e *Foto*.

Mesmo que as *features* de um DAS tenham seu comportamento adaptativo modelado antes das operações do sistema em seu ambiente de produção, esse comportamento pode sofrer alterações em tempo de execução. Por exemplo, um sistema pode adicionar/remover suas *features* baseado no projeto de adaptação do sistema (CAPILLA; ORTIZ; HINCHEY, 2014) ou preferências do usuário (MAURO *et al.*, 2018). Dessa forma, para sistemas dinâmicos, é possível reconfigurar tanto sua própria estrutura quanto sua lógica de adaptação, de forma que esse último pode ser alcançado pela alteração das regras de adaptação.

2.2.3 Modelo de Estado para DAS

Trabalhos na literatura (XU *et al.*, 2012; PÜSCHEL *et al.*, 2014a; EBERHARDINGER; HABERMAIER; REIF, 2017) têm modelado a evolução do ambiente de execução com representações baseadas em estados do sistema e de contexto. A utilização desses modelos pode auxiliar na análise do estado atual do DAS para selecionar novas adaptações ou para a verificação das adaptações. No que diz respeito a esse último, modelos de estados permitem verificar propriedades (XU *et al.*, 2012) e selecionar testes para executar (EBERHARDINGER; HABERMAIER; REIF, 2017).

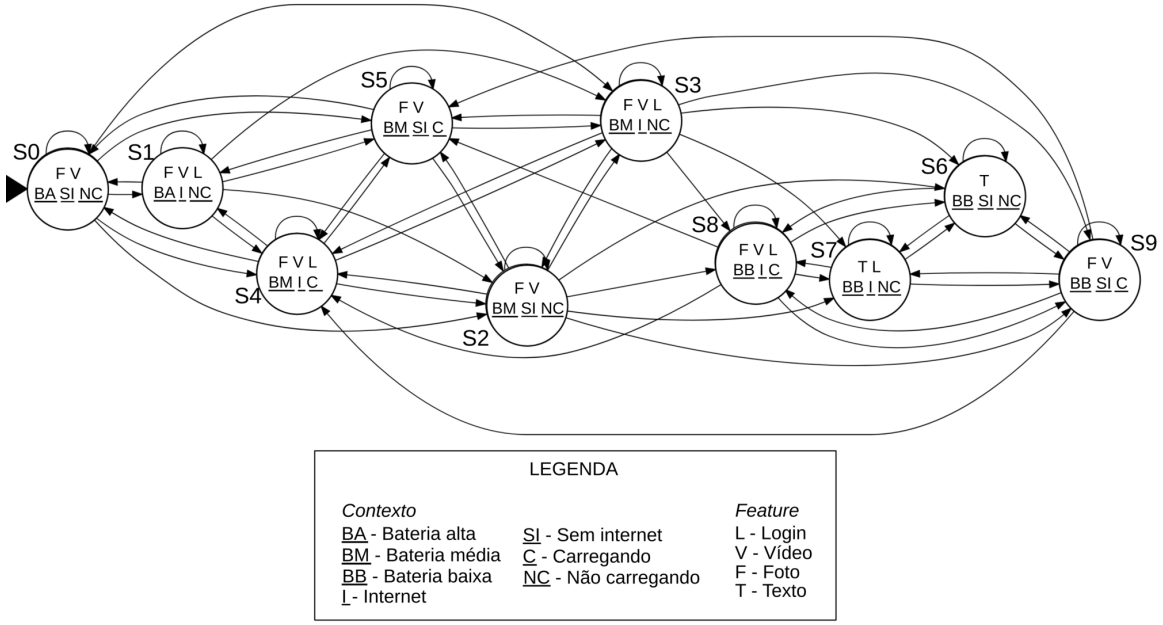
Em relação à modelagem do comportamento adaptativo, no trabalho de Santos *et al.* (2016) foi proposto o modelo Dynamic Feature Transition System (DFTS), que é criado com base no modelo Context Kripke Structure (C-KS) (ROCHA; ANDRADE, 2012), no modelo de *features* e nas regras de adaptação do DAS. O DFTS é um grafo cujos nós representam os

estados de contexto e as *features* ativas, enquanto as arestas representam as variações de contexto do sistema. O diferencial do DFTS é modelar tanto as *features* e *features* de contexto, como também o impacto das mudanças de contexto na configuração do sistema. A definição a seguir formaliza o conceito de DFTS.

> Dado um Context Kripke Structure $CK = \langle S, I, C, L, \rightarrow \rangle$, uma LPSD com modelo de *features* FM, um conjunto R de regras de adaptação e um conjunto E de configurações inicias do projeto, um Dynamic Feature Transition System (DFTS) é definido por uma função $dfts(CK, FM, R, E) = \langle S', I', P, L', \rightarrow' \rangle$ onde S' é um conjunto de estados, $I' \subseteq S'$ é um conjunto de estados de configuração, $P = P_C \uplus P_F$ é um conjunto de proposições atômicas que assumimos ser particionada em proposições de contexto e *features* com $P_C = C$ e $PF = afp(FM)$, L' é uma função de rotulamento tal que $L' : S' \rightarrow' 2^P$ e $\rightarrow' \subseteq (S' \times P_c \times S')$ é uma relação de transição. (SANTOS *et al.*, 2016, p. 17, tradução nossa).

A Figura 5 ilustra o DFTS da aplicação exemplo considerando os contextos *Bateria*, *Conexão com fonte de energia* e *Conexão com a internet* (ver Seção 2.1). Nesse modelo, cada nó representa um estado das *features* do sistema e do contexto. Por exemplo, no estado inicial do sistema (S0), a configuração demanda que ambas as *features Vídeo* e *Foto* sejam ativadas por conta do estado de contexto inicial (*BA, SI, NC*). Quando o contexto muda para *BA, I, NC*, o sistema muda sua configuração para o estado S1, onde a *feature Login* é ativada.

Figura 5 – Modelo DFTS da aplicação de exemplo



Fonte: elaborada pelo autor.

Cada estado do DFTS é rotulado com um conjunto de proposições atômicas que utilizam operadores lógicos com variáveis de contexto. Portanto, um conjunto de expressões lógicas avaliadas como verdadeiras determina o estado atual do sistema. Por exemplo, para

determinar que o DAS está no estado *S0* da Figura 5, a proposição *BA* representa uma proposição para bateria cheia e precisa ser avaliada como verdadeira. Na aplicação exemplo, o sistema estaria no estado *S0* se seu nível fosse maior ou igual à 70%.

Como o DFTS herda os conceitos das C-KS, cada transição de estado válida no modelo é chamada *execução do sistema* (e.g., $S0 \rightarrow S3 \rightarrow S4$), e uma sequência de execuções é chamada de *caminho* (e.g., $\text{path}(p) = S0, S3, S1$) (SANTOS *et al.*, 2016). Na Figura 5, um caminho válido seria S0, S4, S5.

2.3 Verificação da Adaptação para Sistemas Dinamicamente Adaptativos

Schmerl *et al.* (2017) apontam que a variedade de atividades executadas por um sistema adaptativo demanda a necessidade de múltiplas técnicas de garantias. Essas técnicas devem gerar evidências que o DAS cumpre os objetivos previamente definidos. Esse tópico de prover garantias da correta adaptação tem sido relevante para sistemas adaptativos (FREDERICKS; CHENG, 2015; LAHAMI; KRICHEN; JMAIEL, 2016; EBERHARDINGER; HABERMAIER; REIF, 2017).

Considerando o caráter dinâmico de um DAS, é necessário que as garantias sejam continuamente revisadas para verificar seu correto funcionamento após as mudanças (DE LEMOS *et al.*, 2017). Sendo assim, as partes adaptativas do sistema precisam ser verificadas em tempo de execução, de forma que técnicas tradicionais de V&V precisam ser adaptadas para um uso adequado (TAMURA *et al.*, 2013).

Entretanto, esse mesmo dinamismo, que pode estar relacionado com a integração de novos componentes ou alteração de configurações, torna desafiadora a verificação de DAS. No que diz respeito aos testes de software, importantes desafios surgem durante todo o uso da técnica em DAS (SIQUEIRA *et al.*, 2016): (i) "determinar o impacto das adaptações do sistema nos casos de teste" (SIQUEIRA *et al.*, 2016, p. 5, tradução nossa); (ii) permitir ou não que o sistema se adapte durante os testes; e (iii) determinar a cobertura adequada para os testes gerados.

Trabalhos na literatura têm utilizado técnicas variadas de V&V para DAS, podendo ser citado como exemplos os testes (FREDERICKS; CHENG, 2015; EBERHARDINGER; HABERMAIER; REIF, 2017) e a checagem de propriedades (XU *et al.*, 2012; MONGIELLO; PELLICCIONE; SCIANCALEPORE, 2015; BARBOSA *et al.*, 2017). A principal diferença entre as duas técnicas está relacionada à forma como agem. O teste de software pode ser interpretado como uma atividade ativa (LEUCKER; SCHALLHART, 2009), já que inserem

entradas no sistema para analisar as saídas geradas. Por outro lado, a checagem de propriedades é passiva, pois considera apenas o estado atual da execução do sistema.

A abordagem de teste para DAS é discutida na Subseção 2.3.1 com foco para testes em tempo de projeto. A Subseção 2.3.2 apresenta os conceitos relativos a testes para DAS em tempo de execução. Por fim, a Subseção 2.3.3 discorre sobre técnicas para checagem de propriedades.

2.3.1 Teste de Software em Tempo de Projeto

Dentre as opções para garantir a qualidade em sistemas de software, pode-se aplicar a clássica técnica de teste de software. Myers, Sandler e Badgett (2011, p. 6, tradução nossa) definem teste como a atividade que "[...] executa um programa com a intenção de encontrar erros". Neste sentido, o teste determina se o software faz o que deveria (MYERS; SANDLER; BADGETT, 2011). De acordo com IEEE (2010), as definições de anomalias de software incluem erro, defeito, falta e falha. Um defeito ocorre quando o software tem uma deficiência que impede o atendimento aos seus requisitos. Uma falta ocorre quando um erro cometido por um humano se revela durante a execução do mesmo. Por fim, uma falha determina o descumprimento do software em relação aos seus requisitos após a execução.

De acordo com Sommerville (2011) e Hass (2008), os testes podem ser executados em diferentes níveis: teste unitário (teste de componente), que tem como objetivo testar pequenas unidades do sistema (métodos e classes de objetos); teste de integração, cujo foco é testar a integração das unidades anteriormente testadas; teste de sistema, que objetiva testar as funcionalidades no sistema já totalmente integrado; e teste de aceitação, cujo objetivo é identificar se o sistema atende as necessidades do usuário final.

Com relação à testes de software para DAS, trabalhos na literatura tem proposto a verificação desse tipo de sistema em tempo de projeto (WANG; CHAN, 2009; PÜSCHEL *et al.*, 2014a; QIN *et al.*, 2016). Dentre os trabalhos que focam no comportamento de DASs baseados em *features*, Santos (2017) propôs o TestDAS: um método que tem como propósito testar a implementação das regras de adaptação no DAS, checando assim a correta reconfiguração do sistema com relação a sua especificação. A abordagem foca em sistemas desenvolvidos com engenharia de Linhas de Produto de Software Dinâmicas e faz uso dos modelos DFTS (SANTOS *et al.*, 2016) e eCFM (SANTOS *et al.*, 2017a).

O método TestDAS é dividido em quatro etapas: modelagem do DAS, checagem

de modelos, geração e execução de testes. O foco da abordagem é testar as reconfigurações das funcionalidades do DAS após mudanças de contexto. Primeiramente, o DAS deve ser modelado com o DFTS para checar um total de cinco propriedades comportamentais, descritas a seguir (SANTOS *et al.*, 2016):

a) ***Configuration Correctness:*** Esta propriedade determina que os estados da DAS devem estar em conformidade com o modelo de *features* e as regras de adaptação. A violação dessa propriedade é chamada de *configuração inválida*;
b) ***Rule Liveness:*** Esta propriedade especifica que deve existir pelo menos um estado responsável por ativar uma determinada regra de adaptação. A violação da propriedade é chamada de *predicado morto*;
c) ***Interleaving Correctness:*** Esta propriedade demanda que as ações das regras de adaptação devem ser executadas no DAS, mesmo quando elas atuam sobre o mesmo conjunto de *features*. A violação dessa propriedade é nomeada *intercalação não determinística*;
d) ***Feature Liveness:*** "A propriedade determina que cada *feature* opcional do modelo de *features* deve ser ativada em algum estado do DFTS.". (SANTOS *et al.*, 2016, p. 119, tradução nossa) A violação dessa propriedade indica uma *feature morta*;
e) ***Variation Liveness:*** A propriedade determina que cada *feature* opcional do modelo de *features* deve ser desativada em algum estado do DFTS. A violação dessa propriedade indica uma *falsa feature variável*.

Em seguida, essas propriedades são utilizadas como critérios de cobertura para a geração de casos de testes de adaptação. O TestDAS é suportado pelas ferramentas TestDAS Tool e CONTroL (SANTOS *et al.*, 2018). Os casos de teste de adaptação do método são projetados com os seguintes elementos: uma configuração inicial de *features* ativas *S1*, um estado de contexto *C* e uma configuração de *features* esperadas após a adaptação *S2*. A configuração *S2* deve ser realizada quando o estado de contexto *C* ativar as regras que demandam as *features* de *S2*. Nesse sentido, um caso de teste de adaptação é interpretado como uma transição no modelo DFTS.

Os casos de teste de adaptação são utilizados para compor as chamadas sequências de testes de adaptação. Dessa forma, uma sequência de teste é um número finito de transições consecutivas no DFTS, determinando se as *features* do estado *S2* estão de acordo com a configuração exigida por *C*. Como afirma Santos (2017), essas variações em sequência são capazes de executar adaptações sucessivas no DAS e podem revelar mais falhas de adaptação.

A definição a seguir apresenta o formalismo da sequência de teste (SANTOS, 2017). Nessa definição, ω representa a condição de guarda de contexto das regras de adaptação e α, as ações ativação/desativação sobre *features* do DAS.

> Seja $D = \langle S', I', P', L', \rightarrow' \rangle$ um DFTS. Uma n-sequência de teste é uma sequência finita de n transições de estado de sistema $s_0 \xrightarrow{ctx_1, \omega_1 : \alpha_1(F_1)} s_1 \xrightarrow{ctx_2, \omega_2 : \alpha_2(F_2)} s_2 \ldots \xrightarrow{ctx_n, \omega_n : \alpha_n(F_n)} s_n$, onde $s_i \in S'$ e $ctx_i \models \omega_i$. Testar essas sequências significa avaliar se as features ativas no estado s_{i+1} estão de acordo com a ação $\alpha(i)$, disparada pelo contexto ctx_i. Quando n=1 (1-sequência de teste), a sequência é o teste de uma única transição no DFTS. (SANTOS, 2017, p. 79 , tradução nossa).

2.3.2 Teste de Software em Tempo de Execução

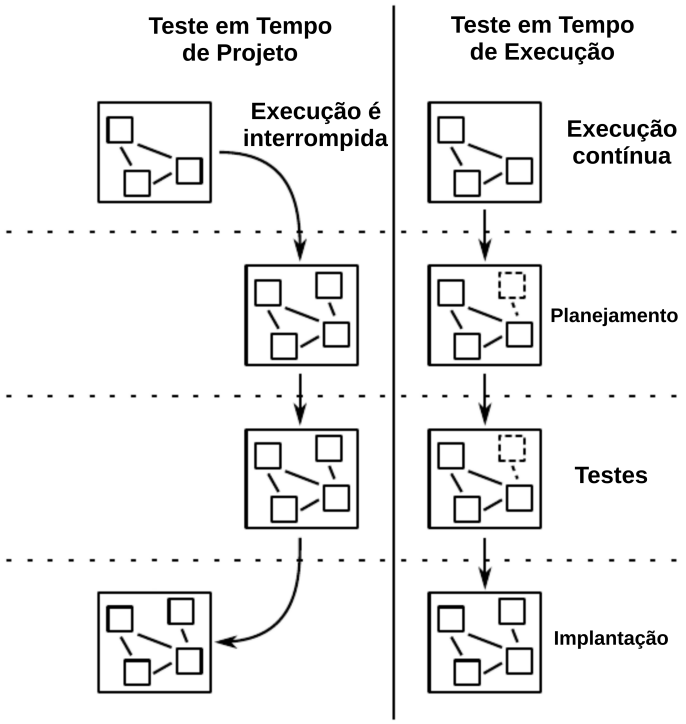
Embora a adaptação em tempo de execução, discutida anteriormente na Subseção 2.2, permita maior confiabilidade no DAS, ela também pode gerar novos defeitos. Portanto, é necessário realizar tarefas de V&V em diferentes fases do ciclo de vida do DAS e não apenas em tempo de projeto (TAMURA *et al.*, 2013). Sendo assim, o teste de software pode ajudar a verificar o comportamento do sistema em um conjunto finito de suas configurações.

Conforme mencionado anteriormente, o teste de software deve ser uma atividade aplicada antes de um sistema entrar em produção (denominado de tempo de projeto). Entretanto, ele também pode ser usado depois da implantação do sistema, durante a sua fase de execução (denominado de tempo de execução). Isso se torna necessário principalmente nos casos em que a estrutura do sistema não é estática em tempo de execução (em DAS, por exemplo), exigindo assim a verificação após a sua implantação. Neste sentido, o teste em tempo de execução é o teste realizado no "[...] sistema executando em seu ambiente de execução final" BRENNER *et al.* (2007, p.154, tradução nossa).

Na literatura de testes em tempo de execução, abordagens têm lidado principalmente com sistemas baseados em componentes (LAHAMI; KRICHEN; JMAIEL, 2016) ou orientados a serviços (METZGER, 2011; LEAL; CECCARELLI; MARTINS, 2019). Assim, a maioria dos conceitos desta subseção estarão relacionados a esses tipos de sistemas.

A Figura 6 ilustra a principal diferença entre os testes em tempo de projeto e em tempo de execução. Em tempo de projeto, realiza-se testes após mudanças no sistema (e.g., a adição de um novo componente ou mudança de requisitos) enquanto o mesmo não está executando. Por outro lado, no teste em tempo de execução, executam-se testes nas modificações sem interromper as operações do sistema.

Figura 6 – Comparação entre teste em tempo de projeto e em tempo de execução



Fonte: adaptado de González, Piel e Gross (2009).

Um dos principais conceitos de teste de tempo de execução é a sensibilidade ao teste (GONZÁLEZ; PIEL; GROSS, 2009), que caracteriza as operações de testes em tempo de execução que podem interferir no sistema. Segundo González, Piel e Gross (2009), a sensibilidade de um componente ao teste em tempo de execução é afetada por quatro fatores, sendo cada um deles descritos a seguir.

O estado interno de um componente pode ser responsável por influenciar os resultados de um teste, levando à possíveis falsos positivos, da mesma forma que a execução de testes também pode influenciar o estado do próprio sistema.

Durante a execução dos testes, pode ocorrer a interação entre componentes, afetando assim o comportamento desses sistemas. Adicionalmente, outros sistemas que ultrapassem os limites do sistema em teste também podem ser afetados durante a execução dos testes. Esta interação com sistemas externos pode realizar ações irreversíveis no mundo real.

Os testes em tempo de execução podem ser uma operação computacionalmente onerosa (FREDERICKS; CHENG, 2015), acrescentando uma carga de processamento extra na execução do sistema em teste. Portanto, esse tipo de teste pode afetar a disponibilidade do sistema em um cenário de recursos limitados, uma vez que alguns componentes precisam permanecer bloqueados. Se o sistema tiver requisitos como alta disponibilidade, os testes em

tempo de execução podem não ser viáveis.

A partir dos fatores apresentados anteriormente, autores na literatura (GONZÁLEZ; PIEL; GROSS, 2009; PIEL; GONZÁLEZ; GROSS, 2010; LAHAMI; KRICHEN; JMAIEL, 2016) afirmam que técnicas de isolamento são um meio de minimizar a sensibilidade de um sistema aos testes em tempo de execução, reduzindo possíveis impactos no sistema e no ambiente. As principais técnicas de isolamento de teste são descritas a seguir.

(1) Bloqueio. A técnica de bloqueio isola os testes através da interrupção das operações de um componente (GONZÁLEZ; PIEL; GROSS, 2009) e, com isso, impede sua execução. Esta técnica permite a separação dos estados em que o sistema está executando suas operações regulares ou está executando testes.

(2) Separação Interna. A separação interna controla as saídas e entradas de componentes que podem afetar outros componentes ou o ambiente (GONZÁLEZ; PIEL; GROSS, 2009). Para isso, as saídas do componente durante os testes podem ser suprimidas. Caso componente demande entrada de dados para funcionar durante os testes, então essas podem ser simuladas para a correta execução (GONZÁLEZ; PIEL; GROSS, 2009).

(3) Agendamento. Para prevenir o impacto na disponibilidade dos componentes durante o teste em tempo de execução, a execução dos casos de teste podem ser agendadas (GONZÁLEZ; PIEL; GROSS, 2009). Dessa forma, é possível que o sistema continue operando para atender seus requisitos não-funcionais.

(4) Orientado por aspectos. Lahami, Krichen e Jmaiel (2016) introduzem o uso do paradigma de Programação orientada a aspectos para isolamento aos testes. O paradigma é utilizado para instrumentar o sistema em teste, facilitando assim a observação e controle do comportamento durante a execução dos testes (LAHAMI; KRICHEN; JMAIEL, 2016). O uso do paradigma pode, por exemplo, permitir o bloqueio de operações irreversíveis durante a execução dos testes.

2.3.3 Checagem de Propriedades em Tempo de Execução

De acordo com Leucker e Schallhart (2009), serviços de software tornam-se mais similares com sistemas autônomos que devem estar em conformidade com seus contratos. Devido ao comportamento complexo inerente de tais sistemas em tempo de execução, técnicas tradicionais, como prova de teorema e verificação de modelos, podem não ser suficientes para verificar o sistema antes de sua execução no ambiente de produção.

Nesse sentido, a verificação de propriedades comportamentais em tempo de execução pode ser usada como uma técnica complementar às tradicionais realizadas em tempo de projeto. Esse tipo de técnica relaciona-se com a área de pesquisa denominada verificação em tempo de execução. Apesar da similaridade com a terminologia de V&V da Engenharia de Software que tem uma definição própria: "disciplina de ciência da computação que lida com o estudo, desenvolvimento e aplicação de técnicas de verificação que permitem verificar se uma execução de um sistema em escrutínio satisfaz ou viola uma determinada propriedade de correção" (LEUCKER; SCHALLHART, 2009, p. 294, tradução nossa).

A técnica de verificação de propriedades comportamentais em tempo de execução diz respeito principalmente ao monitoramento de rastros do sistema (e.g., eventos, estados ou sinais) durante sua execução, objetivando verificar propriedades definidas em especificações formais como lógicas temporais e autômatos (AHRENDT *et al.*, 2019). Devido ao foco em analisar o sistema com base na sua execução atual, considera-se a verificação em tempo de execução uma técnica *passiva* em contraste com o teste de software (LEUCKER; SCHALLHART, 2009). A verificação também pode ser realizada através de assertivas no código-fonte através de lógica de primeira ordem (CHIMENTO; AHRENDT; SCHNEIDER, 2018).

A Figura 7 apresenta uma visão geral de como os principais tópicos de verificação de tempo de execução se relacionam. Por exemplo, uma técnica de verificação pode utilizar uma especificação para gerar um modelo do sistema e, a partir disso, obter monitores que checam propriedades através da análise de rastros do sistema. Normalmente essas aguardam a chegada de novos dados para iniciar sua execução.

Tamura *et al.* (2013) afirmam a necessidade de executar tarefas de V&V em tempo de execução pelo mecanismo de adaptação para avaliar propriedades. No entanto, os mesmos autores também argumentam sobre a necessidade de adaptar as técnicas tradicionais de V&V para execução durante as operações do DAS. Isso é relevante considerando que as técnicas de verificação estáticas, tais como a checagem de modelos (SANTOS *et al.*, 2016), podem ser proibitivas em tempo de execução devido ao problema de explosão de estados.

Técnicas de verificação em tempo de execução para DAS consideram o contexto de execução do sistema durante a garantia de propriedades (XU *et al.*, 2012; BARBOSA *et al.*, 2017; QIN *et al.*, 2019). Ao fazer isso, é possível lidar com variáveis dependentes de contexto que mudam livremente antes da implantação do sistema, mas que só variam em um espaço limitado em tempo de execução (TAMURA *et al.*, 2013). Com relação à verificação

Figura 7 – Áreas relacionadas com verificação em tempo de execução.

Fonte: adaptado de Ahrendt *et al.* (2019).

de propriedades em DAS, trabalhos na literatura têm utilizado tanto técnicas mais tradicionais como propostas próprias. Uma abordagem comum tem sido utilizar formalismos derivados de lógica linear temporal (IFTIKHAR; WEYNS, 2017; QIN *et al.*, 2019) para especificar propriedades complexas. Entretanto, essas abordagens têm a desvantagem de exigir considerável conhecimento de lógica formal por parte do engenheiro.

Em um direcionamento mais prático, pode-se realizar a checagem de propriedades a partir de assertivas espalhadas pelo código-fonte. Logo, é possível usar lógica de primeira ordem para especificar as propriedades como pré e pós-condições, representando assim contratos que devem ser mantidos antes e após a adaptação do sistema (MONGIELLO; PELLICCIONE; SCIANCALEPORE, 2015). Por exemplo, esses contratos podem ser definidos utilizando Java Modeling Language[4], uma tecnologia que permite anotar métodos com o comportamento do sistema em tempo de execução (CHIMENTO; AHRENDT; SCHNEIDER, 2018).

[4] http://www.openjml.org/

Xu *et al.* (2012) utilizam o conceito de *adaptação de um ponto* para checar essas propriedades. A partir de informações do estado atual, checa-se um conjunto fixo de expressões lógicas antes, durante a após o processo de adaptação, de forma que essas expressões representam propriedades genéricas do sistema. Outra estratégia diz respeito a verificação de propriedades através da execução de assertivas para características de qualidade em entidades de software alteradas em tempo de execução (LIM *et al.*, 2015).

2.4 Resumo do Capítulo

Este capítulo apresentou os principais conceitos para o entendimento do presente trabalho. Primeiro foram discutidos os fundamentos de sistemas dinamicamente adaptativos, explorando a característica de sensibilidade ao contexto que permite o processo de adaptação. Em seguida foram apresentados conceitos relativos à modelos utilizados no processo de decisão de adaptação de *features* e de estados.

Na sequência foi discutida a fundamentação teórica relativa à verificação para DAS em tempo de execução do sistema. Nesse tema foram apresentadas definições relativas a testes de software, níveis de testes e anomalias. Também foi discutido como a técnica de testes pode ser utilizada em ambiente de execução do sistema. Por fim, foi apresentada a verificação de propriedades comportamentais como uma técnica adicional.

O próximo capítulo apresenta os trabalhos que mais se relacionam com a abordagem desenvolvida na presente dissertação bem como o processo de obtenção desses trabalhos. Esses trabalhos relacionados foram organizados em técnicas de checagem de propriedades e testes de software em tempo de execução e em tempo de projeto.

3 TRABALHOS RELACIONADOS

Este capítulo discute os principais trabalhos relacionados encontrados durante a revisão da literatura, resumindo artigos sobre a verificação de DAS em tempo de projeto e em tempo de execução.

A Seção 3.1 detalha o procedimento utilizado para obter os trabalhos relacionados. Em seguida, a Seção 3.2 apresenta os trabalhos de checagem de propriedades em tempo de execução e a Seção 3.3 sumariza os artigos sobre testes em tempo de execução. A Seção 3.4 discute os trabalhos sobre testes em tempo de projeto. A Seção 3.5 compara e discute os trabalhos apresentados nas seções anteriores e, por fim, a Seção 3.6 conclui o capítulo.

3.1 Procedimento de Busca

A revisão da literatura deste trabalho foi feita utilizando conceitos do método do mapeamento sistemático (PETERSEN *et al.*, 2008). Esse método foi selecionado como base por proporcionar uma revisão rigorosa da literatura, a qual teve como objetivo obter uma visão geral da área pesquisada. Para isso, foram utilizadas algumas atividades do mapeamento sistemático de acordo com Kitchenham e Charters (2007), sendo elas: elaboração de uma *string* de busca, critérios de inclusão e exclusão de trabalhos e busca em bases automatizadas.

A *string* de busca elaborada continha alguns termos da revisão sistemática de Santos *et al.* (2017b) e eram relacionados com três grupos: DAS, verificação em tempo de execução e tipos de soluções. Os termos de cada grupo foram individualmente combinados com o operador lógico de disjunção e, posteriormente, combinados com o operador lógico de conjunção. Essa combinação permitiu a geração da *string* de busca final, que é apresentada a seguir:

a) "context aware"OR "context driven"OR "context sensitivity"OR "context sensitive"OR "self-adaptive"OR "self-adapt"OR "pervasive"OR "ubiquitous"OR "autonomic system"OR "autonomic software"OR "adaptive system"OR "adaptive software";
b) "runtime test"OR "runtime testing"OR "run-time test"OR "run-time testing"OR "online test"OR "online testing"OR "self-testing"OR "self-test"OR "runtime monitoring"OR "run-time monitoring"OR "runtime verification"OR "run-time verification"OR "runtime validation"OR "run-time validation";
c) "approach"OR "strategy"OR "method"OR "methodology"OR "technique"OR "process"OR "algorithm"OR "tool"OR "framework".

A *string* de busca gerada foi então utilizada na opção de busca avançada das principais bases de pesquisa da área de Ciência da Computação: Scopus[1], IEEE Xplorer[2] e ACM Digital Library[3]. Adicionalmente, também foi feita uma busca manual nos anais das conferências mais relacionadas com a área de sistemas adaptativos, que são: SEAMS (Software Engineering for Adaptive and Self-Managing)[4] e SASO (Self-Adaptive and Self-Organizing)[5].

Para realizar a primeira seleção dos trabalhos retornados na busca, foram utilizados critérios de inclusão e de exclusão. Um período de cinco anos foi escolhido para que se obtivessem os trabalhos mais recentes relacionados ao tema. Adicionalmente, foi dado ênfase aos trabalhos que tivessem como foco a execução de testes em tempo de execução. Então, os seguintes critérios de inclusão foram usados: trabalhos de conferências e periódicos, publicados entre 2014 e 2019, escritos no idioma inglês e cuja temática fosse verificação em tempo de execução. Já os critérios de exclusão foram: trabalhos que não tivessem relação com verificação em tempo de execução para DAS, capítulos de livro e estudos secundários (e.g., revisões e mapeamentos sistemáticos). Os capítulos de livros e estudos secundários foram excluídos porque geralmente apresentam visões gerais de uma área de pesquisa e não tem como foco abordagens. Os estudos secundários são trabalhos dedicados em levantar evidências que respondam questões de pesquisa sobre uma área.

Para complementar as buscas nas bases automatizadas, foram analisados os trabalhos citados em três revisões da literatura de testes para DAS, sendo elas: Siqueira *et al.* (2016), Matalonga, Rodrigues e Travassos (2017), Santos *et al.* (2017b). Os trabalhos referenciados nessas revisões abrangem tanto testes em tempo de projeto como em tempo de execução.

Na sequência foi realizada uma seleção dos trabalhos com base no título e, caso houvesse relação com checagem e testes em tempo de execução, o seu resumo também era analisado. Após a análise desses dois pontos, o trabalho era selecionado para análise completa caso tivesse relação com a área pesquisada. Após a seleção dos trabalhos com base no título e resumo, os mesmos foram analisados por completo. Adicionalmente, as referências dos trabalhos analisados foram checadas para identificar outros possíveis trabalhos relevantes não retornados pela busca nas bases, mesmo que esses fossem mais antigos do que período selecionado. No total foram selecionados 34 trabalhos para a análise completa.

[1] https://www.scopus.com/
[2] https://ieeexplore.ieee.org/
[3] https://dl.acm.org/
[4] https://conf.researchr.org/home/seams-2019
[5] https://saso2019.cs.umu.se/

Após a primeira revisão, foi identificada a necessidade de expandir a revisão para que houvesse uma maior garantia de que nenhum trabalho com abordagens para sistemas adaptativos fosse omitido. Sendo assim, foi realizada uma nova consulta na base Scopus utilizando a mesma *string* de busca e critérios, mas dessa vez o período utilizado foi de 2009 até 2013. Foi escolhido um intervalo reduzido para complementar a busca, uma vez que as revisões da literatura (SIQUEIRA *et al.*, 2016; MATALONGA; RODRIGUES; TRAVASSOS, 2017; SANTOS *et al.*, 2017b) já contemplavam um período de busca abrangente. Entretanto, nenhum novo trabalho foi adicionado para a revisão. Um possível motivo para isso é que os trabalhos mais relacionados já haviam sido retornados pela análise das referências da primeira revisão. Por fim, dos trabalhos encontrados foram selecionados os trabalhos relacionados apresentados nas próximas seções.

3.2 Checagem de Propriedades em Tempo de Execução

Esta seção apresenta os trabalhos relacionados com checagem de propriedades comportamentais. Trabalhos na literatura têm focado na verificação da correta adaptação baseada em seu estado atual com objetivos variados: checar propriedades de alcançabilidade (BARBOSA *et al.*, 2017; XU *et al.*, 2012), geração automática de propriedades (QIN *et al.*, 2019) e dar suporte ao desenvolvimento com garantias de adaptação (IFTIKHAR; WEYNS, 2017; MIZOUNI *et al.*, 2014).

Xu *et al.* (2012) apresentam Adam, uma abordagem para detectar erros na adaptação de aplicações baseadas em modelos. Adam realiza a verificação através da modelagem das adaptações do DAS como uma máquina de estados, sendo que esta determina o estado atual do sistema e suas regras de adaptação. Então, um conjunto pré-definido de asserções é executada para verificar propriedades de consistência, alcançabilidade e vivacidade quando o DAS transita entre os estados. Esse processo de checagem consiste em avaliar as condições de guarda do estado atual e as regras de adaptação do DAS, permitindo checar se o estado atual do DAS está correto em relação a sua especificação. Entretanto, esse trabalho não considera a modelagem de *features* de um DAS ou a análise do comportamento individual delas. Outra desvantagem é que durante a checagem de propriedades, o trabalho só permite que uma regra de adaptação esteja ativa ao mesmo tempo. Esse último ponto compromete o cenário de intercalação de regras de adaptação.

Mizouni *et al.* (2014) apresentam um *framework* que permite o desenvolvimento de aplicações móveis adaptativas e sensíveis ao contexto. O *framework* faz uso de conceitos

de Linhas de Produto de Software Dinâmicas para modelar a variabilidade da aplicação. Em tempo de projeto, o *framework* fornece um modelo de *features* estendido para representar as funcionalidades da aplicação e o contexto que as adapta. Em tempo de execução, o *framework* coleta informações de contexto e usa o modelo de *features* e políticas de adaptação para decidir sobre a melhor configuração do DAS. Após a geração de um plano de reconfiguração do DAS, é utilizado um modelo de *features* para determinar a corretude da configuração escolhida. Mesmo que cheque a consistência do DAS com seu modelo de *features*, o trabalho de Mizouni *et al.* (2014) não considera a checagem de outras propriedades.

Iftikhar e Weyns (2017) apresentam o ActivFORMS, um ambiente de tempo de execução para conduzir adaptações em DAS. Usando o ActivFORMS, deve-se configurar o DAS gerenciado, especificar o modelo MAPE-K, as propriedades e requisitos do sistema a serem verificados em tempo de projeto. Em tempo de execução, o ActivFORMS faz a checagem estatística de um modelo de estados para checar a satisfação das propriedades e, com isso, selecionar novas configurações para o DAS. O ambiente proposto por Iftikhar e Weyns (2017) também realiza a atualização do modelo de ciclo de *feedback*, bem como dos objetivos de adaptação de acordo com a evolução do sistema. Uma das desvantagens desse trabalho é realizar checagem de modelos em tempo de execução, o que pode ser custoso para sistemas complexos. Assim, a verificação das propriedades com base na análise dos traços do sistema pode ser mais adequada.

Barbosa *et al.* (2017) apresentam o Lotus@Runtime, uma ferramenta que verifica propriedades de alcançabilidade relacionadas à adaptação do DAS. Para isso, o Lotus@Runtime utiliza um Sistema de Transição Rotulado anotado com probabilidades para realizar a checagem de modelos em tempo de execução. Uma vez identificada uma falha, a ferramenta sugere uma nova adaptação para o DAS. O trabalho de Barbosa *et al.* (2017) utiliza as transições anotadas com probabilidade para executar uma simulação no DAS e, com isso, fazer a verificação das propriedades. Para isso, o engenheiro de software deve manualmente adicionar as probabilidades do DAS de acordo com seu perfil operacional. Outra limitação do trabalho é que ele não verifica as regras de adaptações do DAS.

Islam e Azim (2018) propõem uma abordagem para garantir as adaptações de sistemas ciber-físicos com capacidade de auto-adaptação. Para isso, deve-se modelar o ambiente de operação e as configurações do sistema, cada uma utilizando um modelo de *features*. Em tempo de execução, os modelos são utilizados para gerar configurações que garantem requisitos

não funcionais (proteção e desempenho) e funcionais (comportamento do sistema). Na garantia dos requisitos não funcionais, a abordagem gera uma adaptação que altera os modos de operação do sistema. Enquanto na garantia funcional é verificado as ações executadas pelo sistema dado uma configuração. Similar ao trabalho de Mizouni *et al.* (2014), a abordagem atual é limitada em garantir planos de adaptação para o DAS, não verificando o comportamento com propriedades após a execução da adaptação.

Qin *et al.* (2019) apresentam a CoMID, uma abordagem para o monitoramento de propriedades invariantes em sistemas ciber-físicos sensíveis ao contexto. O processo é realizado através da geração de invariantes do sistema de maneira automática, de modo que auxilia na detecção de estados anormais do DAS. A geração consiste no agrupamento de rastros de contexto do programa e do ambiente para identificar as invariantes. Em seguida, geram-se múltiplos grupos de invariantes para decidir sobre uma violação caso hajam incertezas na leitura de contextos. A limitação desse trabalho é a falta de detalhes sobre como automatizar as atividades da abordagem através de um ferramental.

3.3 Testes em Tempo de Execução

Trabalhos na literatura exploram o uso de técnicas de teste em tempo de execução para garantir o comportamento em DAS, conforme apresentado nesta seção. Entretanto, como também descrito nesta seção, poucos trabalhos têm visado exercitar o mecanismo de adaptação com entradas para estimular adaptações em tempo de execução.

Teste em tempo de execução tem sido uma técnica comumente aplicada para sistemas com arquiteturas orientadas a serviços. Nesta direção, Metzger (2011) combina monitoramento de propriedades e testes em tempo de execução na verificação de sistemas dinâmicos com arquiteturas orientadas a serviços. A abordagem aplica as duas técnicas para obter um nível mais elevado de confiança da conformidade do sistema com seu Acordo de Nível de Serviço. O SALMon (ORIOL; FRANCH; MARCO, 2015) é um exemplo de *framework* que implementa essas atividades de verificação. Entretanto, esses conceitos relacionam-se mais com a previsão de violações de requisitos não funcionais do sistema, de forma que não foca no comportamento adaptativo após mudanças de contexto.

Cafeo *et al.* (2011) abordam os testes de tempo de execução para Linhas de Produto de Software Dinâmicas, sendo o diferencial da abordagem inferir resultados de testes durante a execução do DAS. Para isso, o sistema deve ser representado com grafos de chamadas e após

uma adaptação do sistema, o grafo é utilizado para inferir os resultados. O processo consiste na análise desse modelo para comparar a nova configuração gerada com as outras que já foram testadas. Caso haja similaridade, os resultados de testes de configurações anteriores podem ser reaproveitados para a nova configuração. Ainda que foque em sistemas modelados pelo conceito de *features*, o trabalho de Cafeo *et al.* (2011) não testa a disponibilidade dessas *features* durante as reconfigurações.

Fredericks e Cheng (2015) apresentam Proteus, uma abordagem orientada por requisitos para gerar suítes de testes adaptativas para os objetivos do sistema. Os autores assumem que os testes precisam se adaptar devido às incertezas no ambiente de operação, o que poderia tornar os oráculos de teste incompatíveis com o estado atual do sistema. Proteus então seleciona os casos de teste com base em um modelo de objetivos e no ambiente do sistema. Finalmente, Proteus usa um algoritmo evolutivo para determinar a relevância de um caso de teste dado um contexto através de funções de utilidade. Apesar do trabalho objetivar a seleção e execução de testes, a abordagem apresentada não realiza a inserção de entradas no sistema e limita-se a checar asserções no estado atual do DAS. Esse tipo de execução pode prevenir a detecção de falhas de forma proativa pelos testes em tempo de execução, necessitando que o estado com falha seja alcançado pelo sistema antes da execução dos testes.

Lahami, Krichen e Jmaiel (2016) apresentam uma solução caixa-preta para gerar suítes de teste para sistemas distribuídos e orientados a componentes. As suítes de testes são cientes dos recursos disponíveis no sistema e realizam a detecção de falhas após as adaptações estruturais. Os autores apresentam um *framework* que realiza os seguintes passos: (i) análise de dependências entre componentes; (ii) recuperação de casos de teste para componentes afetados; (iii) análise de restrições de recursos no ambiente de execução; e, finalmente, (iv) execução dos testes. O trabalho de (LAHAMI; KRICHEN; JMAIEL, 2016) utiliza múltiplas técnicas de isolamento para diminuir a sensibilidade do sistema aos testes, de forma que a técnica mais adequada é selecionada. A limitação do trabalho é testar apenas os componentes após mudanças estruturais, ao invés de testar as reorganizações desses componentes. Nesse sentido, a abordagem não exercita a reconfiguração para identificar possíveis falhas estruturais.

Hänsel e Giese (2017) propõem combinar testes funcionais em tempo de projeto e de execução para testar a variação de Linhas de Produto de Software Dinâmicas. A técnica suporta a definição de perfis operacionais do sistema, que, por sua vez, auxiliam na seleção de suítes de testes especificados em tempo de projeto. Essas suítes de testes são refinadas e executadas em

tempo de execução, permitindo que resultados de tempo de projeto sejam complementados. O engenheiro deve usar esses perfis operacionais para gerar classes de equivalência no sistema que agrupam os testes unitários. Semelhante à proposta de Cafeo *et al.* (2011), esse trabalho também objetiva a checagem de sistemas que se adaptam utilizando o conceito de *features*. Entretanto, ele não realiza a geração de testes em tempo de execução.

Eberhardinger, Habermaier e Reif (2017) propõem uma abordagem de simulação e testes baseados em modelos para sistemas com capacidade de auto-organização. A abordagem requer a modelagem dos estados do sistema e do ambiente operacional, sendo esse último anotado com probabilidades nas transições. A análise dos modelos permite identificar o estado atual do sistema, que é ponto de partida para os testes, e obter a suíte de testes que deve ser executada. Os testes representam falhas que devem estimular adaptações corretivas no sistema. Entretanto, a abordagem considera a simulação do DAS como sendo a própria execução do sistema, logo, não verifica o mesmo em seu ambiente de execução final. O trabalho também não apresenta nenhuma técnica de isolamento de testes.

Por fim, Leal, Ceccarelli e Martins (2019) apresentam SAMBA, uma abordagem baseada em modelos para testar orquestrações de serviços. Os autores disponibilizam um mecanismo de teste auto-adaptável inspirada no ciclo MAPE-K que decide, através de mudanças nos modelos de orquestração, se as atividades de testes funcionais e de integração são necessárias. O mecanismo inclui as seguintes funcionalidades: atualização automática dos modelos de orquestração, e geração e execução de testes funcionais e de regressão. De forma similar a outros trabalhos, a abordagem de Leal, Ceccarelli e Martins (2019) limita-se em testar apenas as funcionalidades do DAS.

3.4 Testes em Tempo de Projeto

Vários trabalhos de testes em tempo de projeto do DAS podem ser encontrados na literatura. Apesar do objetivo desta revisão ser a identificação de trabalhos com atividades de verificação em tempo de execução, conceitos de testes em tempo de projeto podem ser adaptados para os testes durante as operações do sistema. Por exemplo, várias abordagens dependem de modelos de estados para descrever a variação de contexto do sistema, de forma que exploram esses modelos para gerar os casos de teste. Modelos similares podem ser utilizados para determinar o contexto do sistema em tempo de execução e gerar testes. Nesta seção, são apresentados alguns trabalhos desta categoria de teste, dando ênfase para os que embasam as

técnicas utilizadas nesta dissertação.

Püschel *et al.* (2014a) propõem uma abordagem caixa-preta que combina simulação e testes para DAS. Para isso, o engenheiro de software deve modelar simulações de contexto, configurações inicias do ambiente, reconfigurações do ambiente, ações do sistema e as ações disparadas por mudanças ambientais. Esse último modelo especifica detalhadamente o comportamento do sistema e incorpora asserções para verificar suas reações após adaptações. Todos esses modelos verificam a corretude do DAS desde o início de sua adaptação até a finalização da reconfiguração, de forma que a simulação visa avaliar o DAS em cenários complexos em adição aos testes. A aplicação de asserções e testes auxilia na verificação do DAS, mas o trabalho não apresenta nenhum algoritmo para a geração de testes ou simulação do DAS.

Qin *et al.* (2016) propõem a abordagem Sample-based Interactive Testing (SIT) para testar aplicações adaptativas. A SIT permite a exploração do espaço de entradas de contexto através de uma técnica de amostragem sistemática. A exploração é realizada com o auxílio de um modelo que representa a interação do DAS com o ambiente e as restrições associadas aos valores de entrada e saída. Com isso, são retornadas sequências de valores aos parâmetros de entrada da aplicação para estimular diferentes comportamentos. Vale ressaltar que, para aplicar a técnica de amostragem, Qin *et al.* (2016) focam em sistemas cujos parâmetros de entrada são números reais relativos aos sensores. A abordagem SIT possui como limitação uma representação simples de informação de contexto relacionada apenas com números reais. Dessa forma, não é possível representar outros valores que demandem *strings* ou booleanos.

O último trabalho relacionado a testes em tempo de projeto é o de Santos (2017). Sendo um dos trabalhos que fundamentam a presente dissertação, já foi oferecida uma visão geral na Seção 2.3.1. Em resumo, o trabalho de Santos (2017), chamado de TestDAS, usa modelos de *features* com foco na checagem das regras de adaptação e, adicionalmente, apresenta a ferramenta CONTroL (SANTOS *et al.*, 2018). Entretanto, o TestDAS assume que o comportamento das *features* não muda em tempo de execução (e.g., alterações nas regras de adaptação ou modelos de *features*). Outra limitação é que tanto a técnica de checagem de modelos como os testes podem requerer a análise de todos os estados do DFTS, exigindo uma exploração exaustiva.

3.5 Comparação entre os Trabalhos

Esta seção discute os trabalhos sumarizados nas Seções 3.2 e 3.3, realizando assim uma comparação entre os mesmos e ressaltando as diferenças em relação à abordagem proposta

nesta dissertação.

Com relação aos trabalhos de testes em tempo de projeto (Seção 3.4), as principais similaridades dizem respeito ao uso de um modelo de estados do DAS, geração de testes guiado por um critério e controle de contexto automatizado para exercitar adaptações. Nesse tipo de trabalho, os modelos de estados são utilizados para declarar o comportamento do sistema, de forma que os estados podem ser visitados utilizando algum tipo de critério (ex.: randômico) para gerar sequências de testes. As sequências geradas são então utilizadas para injetar dados de teste no DAS e, consequentemente, exercitar adaptações. A abordagem apresentada no presente trabalho utiliza um sequenciamento semelhante de atividades, mas de modo a ser utilizado durante a execução do DAS.

Tabela 3 – Comparação de trabalhos relacionados à verificação em tempo de execução

Trabalho	**Modelo de *features***	**Tipo de checagem**	**Foco da abordagem**	**Suporte de ferramental**
Xu *et al.* (2012)	Não	Asserções	Verificação	Não
Mizouni *et al.* (2014)	Sim	Asserções	Desenvolvimento	Não
Iftikhar e Weyns (2017)	Não	Checagem de modelos	Desenvolvimento	Sim
Barbosa *et al.* (2017)	Não	Checagem de modelos	Verificação	Sim
Islam e Azim (2018)	Sim	Asserções	Desenvolvimento	Sim
Qin *et al.* (2019)	Não	Asserções	Verificação	Não
Trabalho Atual	**Sim**	**Asserções**	**Verificação**	**Sim**

Fonte: elaborada pelo autor.

A discussão dos trabalhos relacionados será realizada em duas partes, iniciando pela checagem de propriedades comportamentais, e, em seguida, com testes para DAS em tempo de execução.

A Tabela 3 sumariza as características dos trabalhos da Seção 3.2, sendo elas: modelo de *features*, que determina se a abordagem utiliza esse tipo de modelo; tipo de checagem, podendo ser asserções e checagem de modelos; foco da abordagem, que podem ter como foco a verificação da adaptação ou suporte ao desenvolvimento; por fim, o suporte ferramental, que determina se o trabalho oferece ferramenta automatizada.

A partir dessa tabela é possível verificar que apenas os trabalhos de Mizouni *et al.* (2014) e Islam e Azim (2018) checam a variabilidade das *features* do sistema, mas seu principal objetivo é auxiliar no processo de adaptação do DAS. Dos seis trabalhos citados, dois disponibilizam ferramental para execução automatizada de checagem de modelos e verificação de propriedades de alcançabilidade.

Em tempo de execução do DAS é importante realizar a checagem da satisfação das propriedades comportamentais no DAS. O monitoramento da variabilidade das *features* em tempo de execução, após a adaptação do DAS, permite a verificação de restrições de configurações, mas não foram encontrados trabalhos com esse objetivo. Além disso, as abordagens não têm considerado a verificação de propriedades relacionadas com *features* individuais somado as regras de adaptação.

Tabela 4 – Comparação de trabalhos relacionados à testes em tempo de execução

Trabalho	**Checagem de Propriedades**	**Critério de Seleção**	**Tipo de Teste**	**Testes Adaptáveis**	**Mecanismo de Adaptação**	**Regras de Adaptação**
Metzger (2011)	Sim	Não Especificado	Desempenho	Não	Não	Não
Cafeo *et al.* (2011)	Não	Contexto	Funcional	Não	Não	Não
Fredericks e Cheng (2015)	Não	Contexto	Não Especificado	Sim	Não	Não
Lahami, Krichen e Jmaiel (2016)	Não	Contexto	Funcional	Não	Não	Não
Hänsel e Giese (2017)	Não	Randômico	Funcional	Não	Não	Não
Eberhardinger, Habermaier e Reif (2017)	Não	Contexto	Funcional	Não	Sim	Não
Leal, Ceccarelli e Martins (2019)	Não	Randômico	Funcional	Sim	Não	Não
Trabalho Atual	**Sim**	**Contexto**	**Funcional**	**Sim**	**Sim**	**Sim**

Fonte: elaborada pelo autor.

Os trabalhos da Seção 3.2 propõem a modelagem de DAS (e.g., máquinas de estado finito), mas eles não estão preocupados em utilizar modelos que expressem a variabilidade do contexto e como isso afeta cada *feature* DAS em tempo de execução. Esses modelos podem ser utilizados para gerar simulações que auxiliem na verificação do DAS.

A Tabela 4 apresenta uma visão geral dos trabalhos relacionados com testes para DAS em tempo de execução. Esses trabalhos foram classificados de acordo com os seguintes critérios: execução de checagem de propriedades em conjunto dos testes; critério de seleção de testes; tipo de teste executado; testes adaptáveis de acordo com o estado do sistema; foco no mecanismo de adaptação; e o exercício das regras de adaptação.

Apenas Metzger (2011) realiza a combinação das técnicas de checagem de propriedades e testes, mas não apresenta detalhes sobre como o processo é feito. Hielscher *et al.* (2008) discute essa combinação denominando-a de *adaptação proativa* através de testes para sistemas orientados a serviços, de forma que o uso de ambas as técnicas podem proporcionar maiores

níveis de confiança sobre falhas. Em uma direção similar, uma solução com foco na verificação da reconfiguração das funcionalidades pode auxiliar na garantia dessa adaptação.

Os trabalhos apresentados utilizam técnicas variadas para realizar a seleção de testes em tempo de execução, mas a maioria utiliza o contexto atual de execução do DAS durante essa seleção. Dentre os trabalhos que especificam o tipo de teste executado, mais da metade foca em testes funcionais (IEEE, 2015). Além disso, somente Fredericks e Cheng (2015) e Leal, Ceccarelli e Martins (2019) apresentam técnicas que realizam adaptação de oráculos dos testes. Permitir a adaptação desses oráculos é uma característica relevante dado o dinamismo de DAS. Desse modo, os casos de teste podem permanecer consistentes com o estado do sistema.

De todos os trabalhos de testes discutidos, apenas Eberhardinger, Habermaier e Reif (2017) testa o mecanismo de adaptação do DAS. Os demais trabalhos são focados nas funcionalidades gerais do sistema. Contudo, o trabalho de Eberhardinger, Habermaier e Reif (2017) pode ser considerado como parcialmente relacionado com tempo de execução, já que a abordagem simula a execução do sistema ao invés de verificar suas operações reais. Isso limita a aplicabilidade da abordagem durante a execução do sistema por conta de sua técnica de injeção de falhas.

Levando em consideração os trabalhos encontrados na literatura e os desafios ainda abertos, o presente trabalho objetiva a checagem de propriedades comportamentais em conjunto com testes em tempo de execução. Com relação a checagem de propriedades, não foram identificados trabalhos que foquem na verificação de propriedades relacionadas às *features* do DAS e regras de adaptação após a reconfiguração do DAS. A modelagem dos sistemas utilizando o conceito de *features* auxilia na visualização de produtos comuns após mudanças de contexto. O modelo de *features* estendido auxilia na definição das regras de adaptação do DAS, que pode ser utilizado para verificar o comportamento adaptativo em tempo de execução.

Outro diferencial do presente trabalho em relação aos demais é seu foco em testar o mecanismo de adaptação durante a execução do sistema. Os testes executados focam na avaliação de regras de adaptação, permitindo também que os casos de teste sejam gerados baseados em informações do sistema. Como apresentado na Tabela 4, essas características não estavam presentes no escopo da maioria dos trabalhos de testes em tempo de execução.

3.6 Resumo do Capítulo

Este capítulo discorreu sobre os principais trabalhos relacionados a esta dissertação de mestrado. Primeiro foi explicado o procedimento seguido para realizar a revisão da literatura. Na sequência foram sumarizados os principais trabalhos relacionados com checagens de propriedades e testes para DAS, salientando as limitações mais relevantes de cada um. Por fim, foi apresentada uma discussão geral do diferencial deste trabalho em comparação aos trabalhos relacionados.

Com relação à checagem de propriedades, os trabalhos analisados utilizam principalmente uma verificação baseada em asserções ou checagem de modelos. As abordagens apresentadas também têm um foco variado de suporte ao desenvolvimento e verificação dos sistemas. Apenas um dos trabalhos apresentados tem o propósito de verificar o comportamento individual das *features* do sistema, mas a checagem é utilizada apenas para garantir a adaptação do DAS antes de sua execução.

Em se tratando dos testes, a maioria dos trabalhos têm executado testes para verificar as funcionalidades do DAS. Esses trabalhos também utilizam informações de contexto para a seleção automática dos casos de teste que devem ser executados. Entretanto, não foram encontrados trabalhos com o objetivo de testar as regras de adaptação do mecanismo de adaptação em tempo de execução. Além disso, poucos trabalhos têm integrado a checagem de propriedades comportamentais com testes durante a verificação.

O próximo capítulo detalha a abordagem proposta neste trabalho. Essa abordagem tem o objetivo de mitigar as limitações de trabalhos anteriores no teste de DAS em tempo de execução. Para isso, são apresentados tanto os detalhes da execução das atividades de verificação em tempo de execução como da ferramenta que viabiliza a execução dessas atividades de forma automatizada.

4 RETAKE

Neste capítulo é apresentado RuntimE Testing of dynamically Adaptive systEms (RETAkE), uma abordagem de teste em tempo de execução para DAS. O presente capítulo está organizado como segue. A Seção 4.1 apresenta uma visão geral da abordagem. A Seção 4.2 detalha as atividades da abordagem. A Seção 4.3 descreve a ferramenta desenvolvida para dar suporte à execução da abordagem. A Seção 4.4 detalha as atividades realizadas pela ferramenta. Por fim, a Seção 4.5 apresenta a conclusão do capítulo.

4.1 Visão Geral

O presente trabalho de mestrado propõe a abordagem RETAkE cujo objetivo é verificar o comportamento de sistemas que se adaptam baseado em contexto. A abordagem executa testes em tempo de execução que exercitam a variabilidade do mecanismo de adaptação, que é o componente responsável pelo monitoramento e análise de contexto e execução das adaptações. Os testes são realizados durante a execução do código-fonte do mecanismo de adaptação, permitindo a verificação proativa em diferentes contextos. Sendo assim, a abordagem auxilia na identificação de falhas de adaptação, detectando execuções incorretas do sistema em relação à sua especificação.

A abordagem foca em verificar a adaptação comportamental de um DAS durante sua execução, tendo como ênfase DASs cujas adaptações podem ser modeladas com o conceito de *features*. A relevância deste tema advém do fato de que o mecanismo de adaptação do DAS, seguindo o clássico ciclo de *feedback* MAPE-K, pode apresentar diversos cenários de falhas durante sua operação (PÜSCHEL *et al.*, 2014b). Por exemplo, o monitoramento incorreto do contexto no DAS tem o potencial de propagar falhas para as outras fases do ciclo do MAPE-K, o que pode levar a falhas difíceis de detectar em tempo de projeto.

Esta dissertação de mestrado é uma evolução de (SANTOS *et al.*, 2016; SANTOS, 2017; SANTOS *et al.*, 2018). O objetivo desses trabalhos era checar propriedades comportamentais através de checagem de modelos (SANTOS *et al.*, 2016) e executar testes de adaptação (SANTOS, 2017; SANTOS *et al.*, 2018). Entretanto, esses trabalhos eram limitados ao tempo de projeto, não considerando informações do DAS durante sua execução (e.g., alteração de regras). Em contrapartida, a abordagem RETAkE foca no teste da variabilidade sensível ao contexto do DAS durante sua execução no ambiente final, considerando assim falhas decorridas

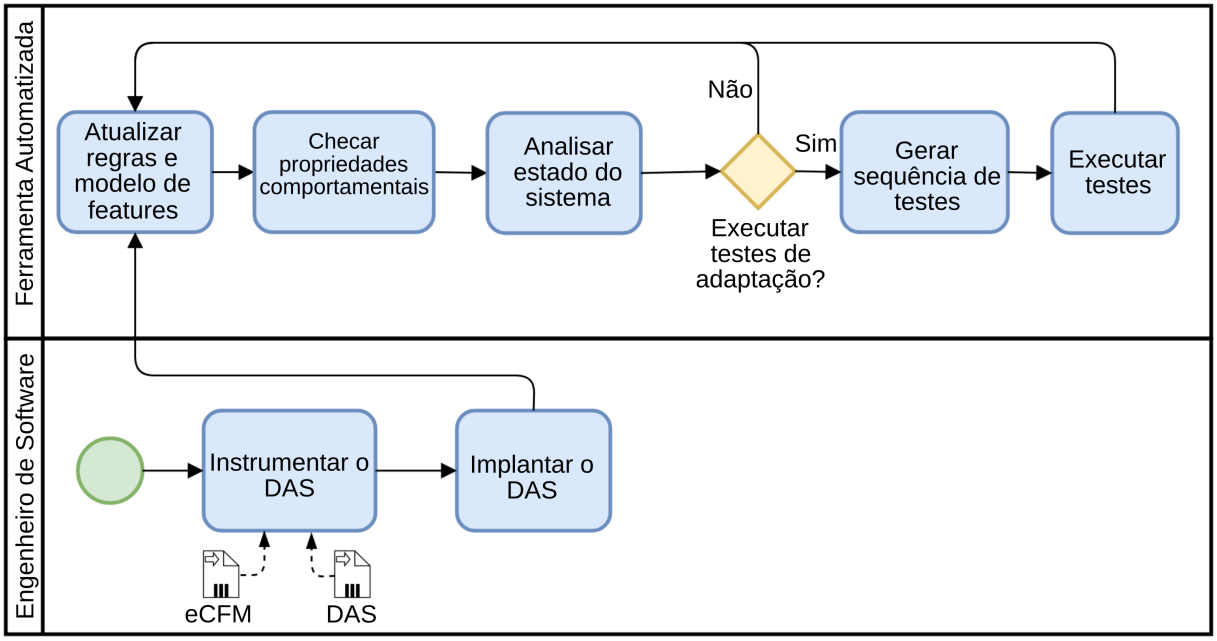
do comportamento em tempo de execução.

Dos trabalhos citados anteriormente (SANTOS *et al.*, 2016; SANTOS, 2017; SANTOS; ANDRADE; SANTOS, 2019a), foram utilizados os modelos DFTS e eCFM. O DFTS foi selecionado para modelar o contexto e gerar sequências de teste por seu foco na variação de contexto e representação dos estados do sistema. O eCFM, que modela restrições entre *features* de contexto e features do sistema, é utilizado para gerar os oráculos de testes.

A abordagem RETAkE também propõe adaptações em como as propriedades propostas por Santos *et al.* (2016) são verificadas, além da geração e execução de testes em tempo de execução. Por fim, a ferramenta disponível em (SANTOS *et al.*, 2018) foi utilizada como base para a implementação que executa as atividades da abordagem.

A Figura 8 apresenta a visão geral das atividades da abordagem. As primeiras atividades devem ser realizadas manualmente pelo engenheiro de software, sendo elas a instrumentação do sistema (ver Seção 4.3) e a implantação do DAS. Em seguida, as atividades de verificação (checagem de propriedades e testes) são realizadas em tempo de execução de forma automatizada. Essas atividades são suportadas por uma ferramenta que deve ser instalada no DAS.

Figura 8 – Visão Geral da RETAkE



Fonte: elaborada pelo autor.

A primeira atividade (**Instrumentar o DAS**) recebe como entrada os modelos de suporte DFTS e o eCFM, em que o primeiro representa as *features* ativas e o contexto de um dado momento, e o segundo modela a variabilidade do DAS (ver Capítulo 2). A instrumentação do sistema tem o propósito de monitorar e controlar o DAS para a execução dos testes. Esse

processo é realizado a partir de anotações de variáveis e métodos no código-fonte, sendo a execução do sistema interceptada através de programação orientada a aspectos (ver Seção 4.3).

Após a modelagem e instrumentação do sistema, o engenheiro de software deve implantar o DAS integrado com a ferramenta desenvolvida para dar suporte a abordagem (**Implantar o DAS**). O engenheiro é capaz de implantar o sistema em seu ambiente final de operação ou em um ambiente controlado (ver Seção 4.3). Caso opte pelo ambiente controlado, é possível simular o contexto do sistema para testar e configurar a solução antes da implantação (ver Seção 4.4). As próximas atividades são realizadas de forma automática.

Primeiro, a ferramenta proposta realiza a atividade de atualização da representação do sistema (**Atualizar Regras e Modelo de Features**), atualizando suas regras e modelo eCFM (ver Subseção 4.2.1).

Uma vez alterado o estado de contexto do sistema, é inciada a verificação em tempo de execução (**Checar propriedades comportamentais**) propriedades comportamentais do DAS (ver Subseção 4.2.2). Essas propriedades foram propostas por Santos *et al.* (2016) e dizem respeito às regras de adaptação do sistema e o impacto delas no comportamento individual das *features*. Vale ressaltar que essa é uma atividade de suporte para a execução dos testes, permitindo assim a verificação constante do sistema quando os testes não forem executados. Como discutido por Hielscher *et al.* (2008), os resultados da checagem de propriedades comportamentais podem condicionar a execução de testes para encontrar mais falhas. Por exemplo, se uma propriedade é violada, então testes podem ser executados no DAS.

A próxima atividade é a análise do sistema para determinar seu contexto (**Analisar Estado do Sistema**) e, com isso, determinar seu estado no DFTS (ver Subseção 4.2.3). A atividade que segue são os testes em tempo de execução (**Gerar sequência de testes**). Considerando que os testes estimulam respostas do mecanismo de adaptação, essa atividade tem o potencial de revelar falhas de adaptação em estados além do estado atual do DAS. Para isso, RETAkE é responsável por gerar uma sequência de testes com mudanças de contexto (ver Subseção 4.2.4).

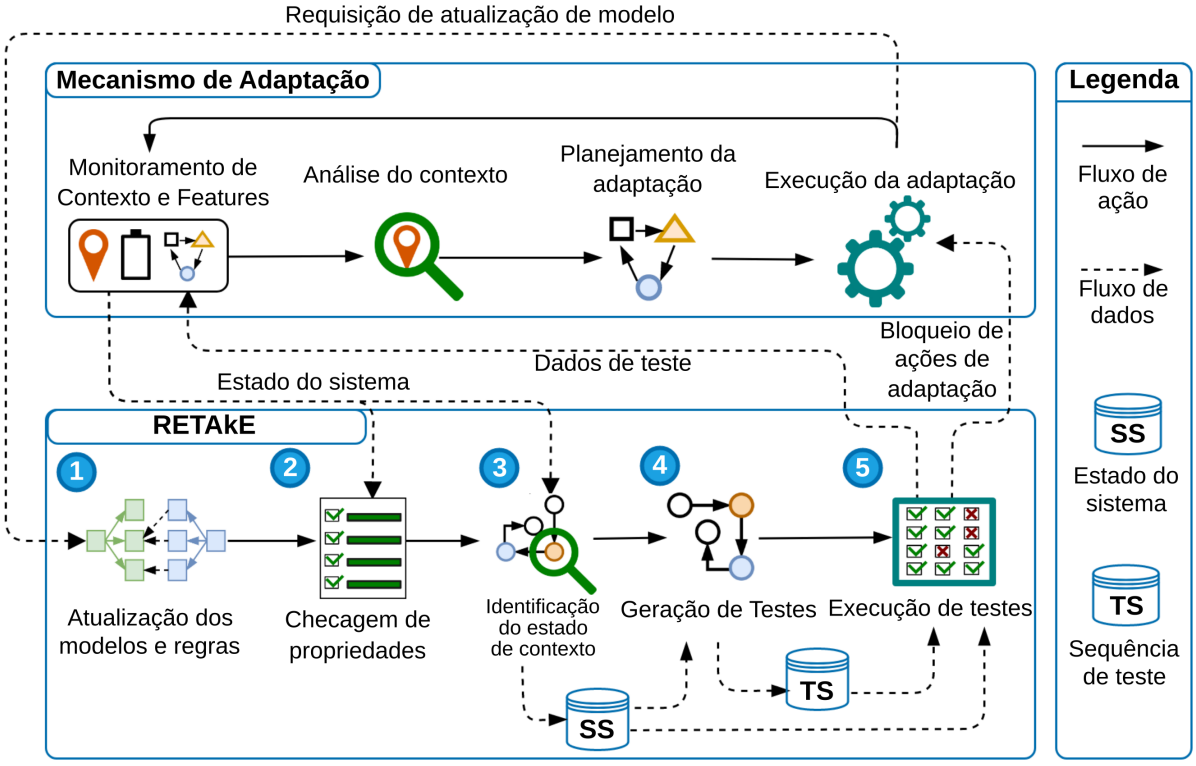
A continuidade da atividade de geração é a execução das sequências testes (**Executar testes**). Essa atividade é repetida até que toda a sequência seja executada (ver Subseção 4.2.5). Dessa forma, os testes estimulam respostas de reconfiguração e as respostas obtidas são comparadas com os modelos especificados. Uma vez que a técnica pode impactar nas funcionalidades do sistema, sua execução pode ser condicionada à condições, como resultados das propriedades comportamentais e adaptação das regras do DAS (ver Seção 4.4).

4.2 Atividades da Abordagem

Como discutido na Seção 2.2, um DAS pode se adaptar em resposta às mudanças de contextos. Além de ajustar seus parâmetros ou reorganizar sua arquitetura, um DAS também pode mudar sua lógica de adaptação. Entretanto, essas mudanças podem levar à falhas de adaptação em tempo de execução se não for corretamente testado (SANTOS *et al.*, 2017b).

Neste cenário, é relevante verificar a adaptação de sistemas sensíveis ao contexto durante sua execução. Os testes realizados pela abordagem RETAkE exercitam as respostas de reconfigurações através do mecanismo de adaptação, permitindo assim a verificação da conformidade em outros estados de contexto. O foco da abordagem são sistemas que têm sua adaptação orientada por regras e que as mesmas possam ser representadas como modelos de *features* estendidos. Nesse sentido, as condições das regras devem ser modeladas com *features* de contexto e as ações devem ativar/desativar *features* do DAS.

Figura 9 – Fluxo de interações da RETAkE com o mecanismo de adaptação do DAS



Fonte: elaborada pelo autor.

A Figura 9 ilustra o fluxo de atividades realizadas pela RETAkE durante as adaptações do DAS. A atualização de modelos (atividade 1) é discutida na Subseção 4.2.1. A Subseção 4.2.2 detalha o processo de checagem de propriedades (atividade 2). A atualização de modelo e regras do DAS (atividade 3) é apresentada na Subseção 4.2.3. A Subseção 4.2.4 discute os

conceitos utilizados durante o processo de geração de testes (atividade 4). Por fim, a Subseção 4.2.5 (atividade 5) discute as técnicas necessárias para a execução dos testes.

4.2.1 Atualização de Modelo e Regras em Tempo de Execução

No escopo deste trabalho, considera-se que o mecanismo de adaptação do DAS pode alterar suas regras e seu modelo de *features*. Desse modo, a abordagem mantém um modelo de *features* do sistema que é atualizado em tempo de execução para refletir o estado atual do sistema.

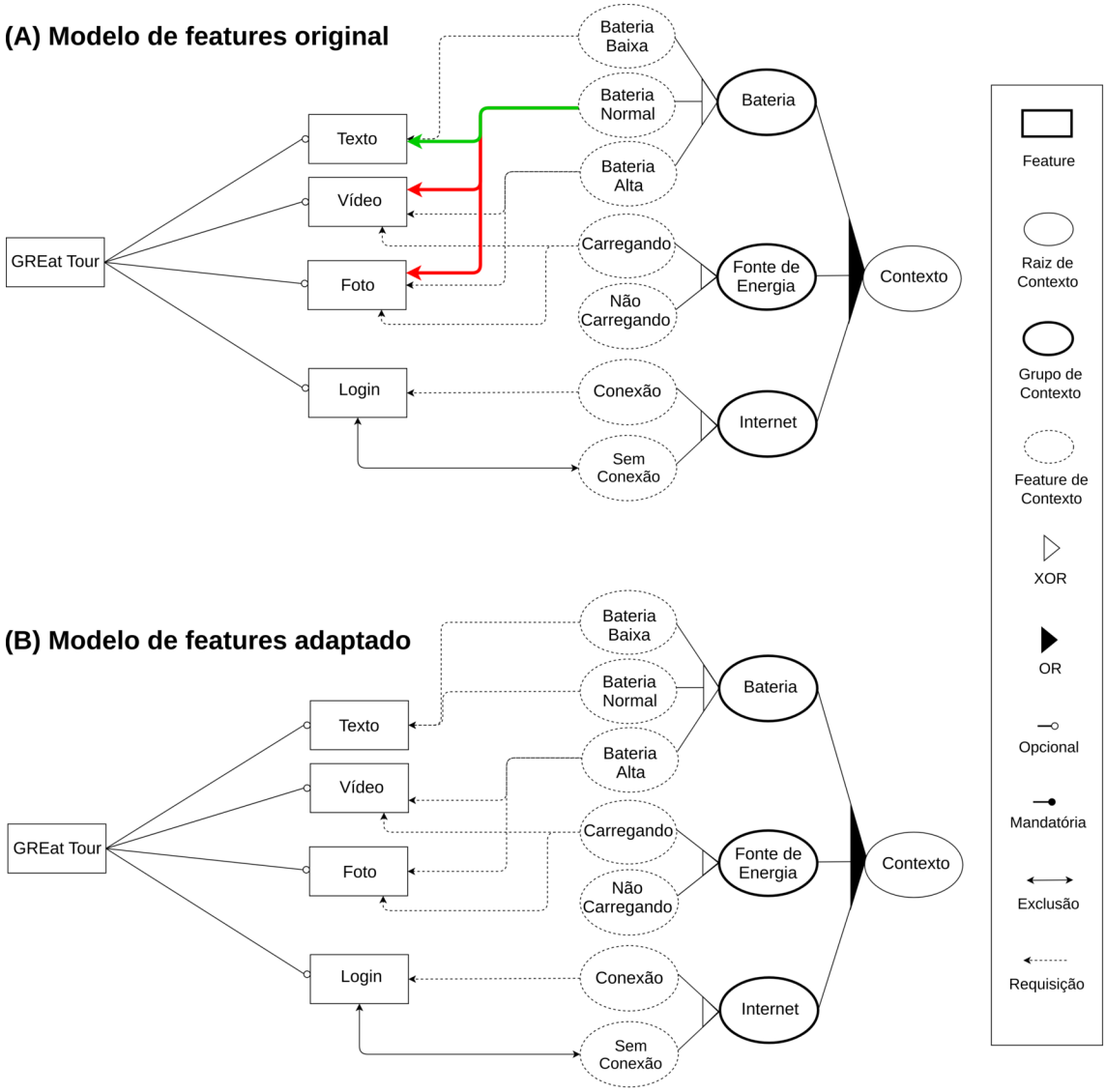
Assim, o sistema deve ser responsável por comunicar essas informações, de modo que elas podem ser de dois tipos: adição ou remoção de relações do tipo requer/exclui entre uma *feature* de contexto e uma *feature* sensível ao contexto. Outros trabalhos na literatura também permitem esse tipo de atualização em modelos de variabilidade (CAPILLA; VALDEZATE; DÍAZ, 2016; MAURO *et al.*, 2018). Por exemplo, o trabalho de Capilla, Valdezate e Díaz (2016) discute alterações na estrutura das *features* no modelo de variabilidade, e Mauro *et al.* (2018) altera as restrições das *features* de contexto com base nos perfis dos usuários.

Em tempo de execução, o DAS deve solicitar a atualização do modelo e das regras logo após a execução da adaptação. Por exemplo, o DAS pode requisitar a remoção da relação *requer* entre a *feature* de contexto *Bateria Normal* e a *feature Vídeo*. Outra modificação pode ser a adição de uma relação do tipo *requer* entre a *feature Texto* e o contexto *Bateria Normal*.

A Figura 10 exemplifica a adaptação do modelo eCFM de acordo com a aplicação de exemplo da Seção 2.1. Na Figura 10-A é destacado em vermelho as relações que devem ser removidas e em verde as que devem adicionadas. Já a Figura 10-B mostra o resultado após a adaptação do modelo. O cenário exemplifica modificações que poderiam auxiliar o sistema na economia de bateria quando houver necessidade de racionamento. A relevância de manter esse modelo atualizado advém do seu uso como oráculo de teste.

Vale ressaltar que está fora do escopo deste trabalho checar restrições ou propriedades através da análise estática dos modelos. Para identificar falhas na atualização dos modelos, o engenheiro de software pode usar técnicas auxiliares para verificação do modelo (SANTOS *et al.*, 2016; MARINHO *et al.*, 2012).

Figura 10 – Exemplo de atualização de modelo de *features* em tempo de execução



Fonte: elaborada pelo autor.

4.2.2 Checagem de Propriedades em Tempo de Execução

Uma técnica proativa, tal como testes em tempo de execução, auxilia na obtenção de evidências da correta adaptação após reconfigurações do sistema. Entretanto, essa técnica pode impactar nas funcionalidades do sistema, uma vez que pode ser necessário bloquear sua execução regular (LAHAMI; KRICHEN; JMAIEL, 2016). Essa carga adicional no sistema implica na importância do uso de técnicas auxiliares que monitoram o sistema em tempo de execução. Essas técnicas podem também auxiliar na decisão do momento de execução dos testes (HIELSCHER *et al.*, 2008).

Uma técnica de verificação que pode auxiliar os testes é a checagem de propriedades comportamentais, as quais devem ser satisfeitas logo após a adaptação do DAS. Essa técnica está

relacionada com a área de Verificação em Tempo de Execução (LEUCKER; SCHALLHART, 2009), em que rastros finitos do sistema são confrontados com propriedades.

Esta seção, portanto, discute a verificação de um conjunto de quatro propriedades. As propriedades foram propostas por Santos *et al.* (2016) e são relacionadas ao comportamento individual das *features* e regras de adaptação. Os resultados obtidos podem ajudar na identificação do nível de confiança das adaptações na ausência de testes.

4.2.2.1 Propriedades Comportamentais para DAS

Santos *et al.* (2016) definiram cinco propriedades comportamentais para checar o comportamento de DASs modelados com modelos de *features*. Essas propriedades foram criadas para checagem de modelo através do DFTS, objetivando garantir que a especificação do DAS estava correta em relação ao seu modelo de variabilidade e regras de adaptação.

Entretanto, a checagem de modelos se sustenta na construção de modelos completos para checar todas as possíveis execuções do sistema (LEUCKER; SCHALLHART, 2009). Quando se trata de um DAS complexo, essa verificação das propriedades pode ser inviável em tempo de execução devido ao problema da explosão de estados (TAMURA *et al.*, 2013), necessitando a visita de todos os estados do modelo. Dessa forma, é relevante realizar uma checagem mais leve e focada no estado atual.

Esta dissertação objetiva a checagem de quatro propriedades definidas por Santos *et al.* (2016), sendo a principal diferença o processo de checagem que foca no estado atual com relação à execução da implementação do DAS.

A seguir são explicadas as propriedades originais consideradas e as adaptações necessárias para verificá-las em tempo de execução pela abordagem RETAkE.

(1) Configuration Correctness. Esta propriedade determina que os estados do DAS devem estar em conformidade com as restrições do modelo de *features*. Para verificar esta propriedade, o modelo de *features* é convertido em uma expressão lógica como proposto por Czarnecki e Wasowski (2007). A fórmula é obtida através da criação de proposições com conjunções de implicações de acordo com os relacionamentos das *features*. A seguir são enumeradas as etapas deste processo:

a) Criar uma implicação da *feature* filha para suas respectivas *features* pai;
b) Criar uma implicação das *features* pai para todas as suas *features* filhas que são mandatórias;

c) Criar uma implicação da *feature* pai para uma disjunção das *features* de um grupo OR;
d) Criar uma implicação da *feature* pai para uma disjunção exclusiva das *features* de um grupo XOR;
e) Criar implicações adicionais para relações do tipo requer entre *features*.

Considerando o modelo de *features* da aplicação de exemplo, a fórmula final seria: (Texto $\rightarrow$ GREat) $\wedge$ (Vídeo $\rightarrow$ GREat) $\wedge$ (Foto $\rightarrow$ GREat) $\wedge$ (Login $\rightarrow$ GREat). Originalmente, essa fórmula deveria ser avaliada como verdadeira em todos os estados do modelo DFST.

Neste trabalho, o processo de checagem deve consistir na atribuição do estado atual de cada *feature* do DAS na expressão lógica. Se a expressão for avaliada como verdadeira, então a configuração atual está em conformidade com o modelo de *features* do DAS. A violação desta propriedade implica que alguma regra de adaptação realiza incorretamente a ativação/desativação de uma ou mais *features*. Também pode implicar na violação de outras restrições (e.g., grupos XOR ou OR) do modelo de features.

(2) Interleaving Correctness. Essa propriedade exige que as ações de adaptação sejam executadas mesmo quando mais de uma regra é ativada. A propriedade de *Interleaving Correctness* deve ser checada em cada estado do DAS. Ela foi adaptada neste trabalho como segue: se uma ou mais regras de adaptação for disparada e suas *features* não estiverem com o estado correto no estado atual, então a propriedade é violada.

(3) Feature Liveness. A verificação dessa propriedade também pode exigir a checagem de todos os estados do DFTS, mas desta vez com o objetivo de determinar se existe um estado onde uma *feature* não obrigatória é ativada. A verificação dessa propriedade foi adaptada para uma restrição que determina o tempo máximo esperado para a ativação de uma *feature*. A definição desse tempo deve ser realizada pelo engenheiro de software com base no modelo de estados de contexto e no comportamento do DAS. Foi utilizado um tempo máximo para que não fosse necessário visitar uma grande quantidade de estados em tempo de execução para validar a propriedade. Logo, uma violação dessa propriedade é a não ativação das features não-obrigatórias no tempo máximo. A verificação se assemelha com as restrições de vínculo de *features* em tempo de execução (BÜRDEK *et al.*, 2014), mas com a diferença de não haver foco na checagem do momento em que isso acontece.

(4) Variation Liveness. Tal como a *Feature Liveness*, esta propriedade também poderia exigir a visita de muitos estados para identificar se uma *feature* ativa era desativada em algum estado. Sendo assim, ela também foi modificada para ser uma restrição de tempo que

configura o inverso da *Feature Liveness*. Uma violação desta propriedade é a não satisfação desse tempo máximo.

Optou-se por não verificar a propriedade de *Rule Liveness* (SANTOS *et al.*, 2016). Como explicado no Capítulo 2, a propriedade determina que uma regra esteja ativa em pelo menos um estado do DFTS. Entretanto, dado o foco de verificar as propriedades no estado atual de contexto do DAS, não é possível aguardar que o sistema execute até que a regra em verificação seja avaliada como verdadeira. Também optou-se por não definir uma restrição de tempo tal qual nas propriedades de *Feature/Variation Liveness*, uma vez que para as regras de adaptação, considerando o dinamismo e a imprevisibilidade do contexto, esse tipo de restrição não se aplica.

4.2.2.2 Atividades para Checagem de Propriedades Comportamentais

A verificação das propriedades consiste na avaliação de assertivas logo após a adaptação do DAS. Dessa forma, os resultados das propriedades comportamentais são analisados comparando estados consecutivos do sistema. Esse processo é guiado pela análise das regras de adaptação e o estado das *features*. A análise das *features* objetiva a identificação de violações das propriedades *Feature Liveness* e *Variation Liveness*. Já a avaliação das regras de adaptação serve para checar as propriedades *Configuration Correctness* e *Interleaving Correctness*. Nesse último passo é utilizada uma forma de verificação semelhante ao trabalho de Xu *et al.* (2012): as condições de guarda das regras modeladas são avaliadas com o contexto atual do DAS, e os resultados são utilizados para verificar a conformidade em relação às propriedades.

Dado que a abordagem RETAkE permite a alteração da lógica de adaptação, outra informação que deve ser recebida do sistema são as regras de adaptação atualizadas. As novas regras são processadas na segunda atividade da Figura 9. Ainda que o modelo de *features* seja capaz de representar essas regras em um nível mais alto, a checagem das propriedades pode ser realizada de forma mais direta utilizando uma especificação no formato condição-ação. A seguir são listadas as formas de checagem de cada uma das propriedades.

(1) Configuration Correctness: Como discutido anteriormente, o modelo de *features* é convertido em uma expressão lógica, que se avaliada como verdadeira implica na validade da configuração. A expressão é avaliada pela abordagem RETAkE através da atribuição do estado das *features* após as adaptações.

(2) Interleaving Correctness: Esta propriedade é verificada através da avaliação de todas as regras de adaptação especificadas na abordagem. O processo é realizado através

da atribuição do contexto atual nas variáveis das regras e, em seguida, as ações de adaptação são comparadas com os estados das *features*. Caso uma ou mais regras sejam avaliadas como verdadeiras e os estados das *features* estejam incorretos, então a propriedade é violada.

Algoritmo 1: Checagem de propriedades comportamentais

Entrada: Lista de *features* com dados de tempo de ativação/desativação
Entrada: Regras de adaptação
Entrada: Intervalo de tempo do ciclo de adaptação
Entrada: Estado de contexto atual do sistema

```
1  início
2      R ← Regras de adaptação
3      F ← Lista de features com estados e tempo de ativação/desativação
4      t ← Intervalo de tempo do ciclo de adaptação
5      InterleavingRules ← ∅
6      C ← Estado de contexto atual do sistema
7      se (avaliarRegraModeloFeatures(F) ≠ Verdadeiro) então
8      Marcar propriedade Configuration Correctness como violada
9      fim
10     para cada r ∈ R faça
11         se (avaliarRegra(r,C) ≠ Verdadeiro) então
12         Marcar propriedade de Interleaving Correctness como violada
13         InterleavingRules ← InterleavingRules ∪ r
14         fim
15     fim
16     para cada f ∈ F faça
17         se (t > tempo desde a última ativação de f) então
18         Marcar Feature Liveness como violada
19         fim
20         se (t > tempo desde a última desativação de f) então
21         Marcar Variation Liveness como violada
22         fim
23     fim
24 fim
```

(3) Feature e (4) Variation Liveness: Por fim, as propriedades de *Feature e Variation Liveness* implicam em checar se as *features* ativaram/desativaram no intervalo de tempo

especificado. Para isso, a abordagem RETAkE deve guardar os tempos relativos à ativação e desativação das *features* e, após cada ciclo de adaptação, é realizada a checagem dos tempos. Caso o tempo decorrido desde a última ativação/desativação seja maior que o especificado, a propriedade é identificada como violada.

O Algoritmo 1 detalha as operações para verificar as propriedades. Das linhas 3 à 7 são inicializadas as variáveis utilizadas no algoritmo. Na linha 8, a função *avaliarRegraModelo-Features()* checa a expressão lógica correspondente à propriedade *Configuration Correctness*. Em seguida, todas as regras de adaptação são avaliadas utilizando o método *avaliarRegra()* e, caso uma ou mais regras não sejam satisfeitas, a propriedade de *Interleaving Correctness* é marcada como violada.

O Algoritmo 1 é encerrado com a checagem das propriedades *Feature/Variation Liveness*. Para isso, verifica-se o tempo desde a última ativação e desativação para cada *feature*. Caso o tempo seja maior que o especificado para ativação, a propriedade *Feature Liveness* é marcada como violada. O mesmo vale para a *Variation Liveness* para o tempo de desativação.

Por fim, a verificação de propriedades pode ter seus resultados integrados com a execução de testes. Dessa forma, o engenheiro pode decidir por iniciar a execução de testes com diferentes tamanhos de sequências em caso de sucesso ou falha de propriedades. A vantagem dessa abordagem é iniciar a execução de testes com base em outras condições além das alterações de regras. Assim, o mecanismo de adaptação do DAS pode realizar testes a partir do estado atual e, com isso, identificar mais falhas de adaptação.

4.2.3 Identificação do Estado Atual de Contexto

Após a atualização dos modelos e checagem de propriedades, o próximo passo diz respeito à identificação do estado atual de contexto no modelo DFTS. Essa atividade é a primeira que compõe os testes em tempo de execução, sendo necessária para que as sequências de testes possam ser geradas a partir do estado atual do sistema. Identificar o contexto do sistema é relevante para auxiliar na seleção de casos de teste relacionados ao estado atual (FREDERICKS; CHENG, 2015; EBERHARDINGER; HABERMAIER; REIF, 2017).

A execução deste passo pode estar condicionada às situações listadas a seguir. Todas essas condições devem ser configuradas pelo engenheiro de software:

a) Alterações de regras de adaptação e/ou modelo de *features*;
b) Adaptações no estado de *features*;

c) Resultados de propriedades comportamentais, tanto em falhas quanto em sucessos (a ser elaborado na Seção 4.2.2).

Se as condições especificadas pelo engenheiro de software para iniciar os testes forem satisfeitas, inicia-se o processo de identificação do estado atual. Considerando que o eCFM modela os possíveis contextos de um DAS, a abordagem pode explorar o conjunto de *features* de contexto e identificar quais estão ativas baseadas no contexto do ambiente. Os nomes das *features* de contexto ativas são utilizadas para construir um identificador que é mapeado para um estado do DFTS. Por exemplo, se a bateria é baixa (BB) e a fonte de energia está conectada (C), então o identificador para esse estado de contexto será *{BB,C}*.

Dado que todos os estados do DFTS definem contextos únicos, o Algoritmo 2 detalha os passos necessários para a obtenção do identificador de estado de contexto. Para isso, itera-se todos os valores associados às variáveis de contexto do sistema (linha 6). Em seguida, avaliam-se os valores que correspondem as *features* de contexto do eCFM (função *avaliarContexto()* na linha 8) e, caso sejam verdadeiras, as variáveis identificadas são utilizadas para construir um identificador.

Algoritmo 2: Identificação do estado atual do sistema

```
   Entrada: Lista de features de contexto do eCFM
   Entrada: Lista de valores de contexto
1  Saída: Estado atual do sistema no modelo DFTS
2  início
3      C ← Lista de valores de contexto
4      ECFM ← Lista de features de contexto do eCFM
5      Chave ← ∅
6      para cada c ∈ C faça
7          para cada fc ∈ ECFM faça
8              se (avaliarContexto(c, fc)) então
9              Chave ← Chave∪ identificador do contexto c
10             fim
11         fim
12     fim
13 fim
```

Assume-se que o sistema em teste pode fazer transições inválidas de estados de contexto no DFTS, mas não pode estar em um estado inexistente no modelo. Considerando o DFTS para o cenário de exemplo (Figura 5), o sistema poderia transitar do estado S6 (bateria

baixa) para S1 (bateria alta). No caso do sistema se encontrar em um estado de contexto não mapeado, os testes não podem ser iniciados.

A partir desse ponto, as adaptações do mecanismo de adaptação devem ser bloqueadas para evitar interferências no DAS. Na sequência, são gerados os testes de adaptação.

4.2.4 Geração de Testes de Adaptação

Após a análise do estado atual do sistema, o próximo passo da abordagem consiste na geração dos testes que possam identificar falhas no comportamento adaptativo. Tal qual o trabalho de Santos (2017), o processo de geração dos testes consiste na exploração do modelo DFTS para gerar casos de teste. Os testes gerados focam na funcionalidade do mecanismo de adaptação que é abstraído como um único componente. Por conta disso, o nível dos testes gerados neste trabalho é o unitário.

A presente dissertação não tem o propósito de propor uma nova técnica para a geração dos testes, mas adaptar conceitos utilizados em tempo de projeto para o tempo de execução. Dessa forma, será utilizado o conceito de sequência de teste apresentado por Santos (2017) (Ver Subseção 2.3.1). Porém, a RETAkE não usa o estado das *features* antes de adaptação e foca apenas no estado pós-reconfiguração. Dessa forma, a configuração será analisada exclusivamente a partir do estado de contexto atual e das *features* demandadas nesse estado. Adicionalmente, o estado correto das *features* será obtido a partir do modelo eCFM, ao invés do DFTS como originalmente proposto em (SANTOS, 2017).

Primeiramente, o engenheiro de software deve selecionar um tamanho n para a sequência. Então, RETAkE seleciona n estados no DFTS seguindo um conceito similar ao da métrica chamada Diversidade de Contexto (WANG; CHAN, 2009; WANG; CHAN; TSE, 2014). A definição a seguir apresenta o conceito original da Diversidade de Contexto. Nessa definição, uma *instância de contexto* é um valor aplicado para cada variável de contexto em um determinado tempo. Já um fluxo de contexto é uma sequência de instâncias de contexto. Wang e Chan (2009) afirmam que uma sequência de valores de contexto com elevadas mudanças consecutivas causam mais adaptações no DAS, podendo revelar mais defeitos de adaptação.

> A Diversidade de Contexto (DC) de um fragmento de fluxo de contexto *cstream(C)* é denotado por *DC(cstream(C))* e é definido pela seguinte equação:
> $DC(cstream(C)) = \sum_{i=1}^{n-1} HD(ins(C)_i, ins(C)_{i+1})\ n = |cstream(c)|$

> onde $HD(ins(C)_i, ins(C)_{i+1})_i$ é a distância de Hamming de uma par de instâncias de contexto $ins(C)_i$ e $ins(C)_{i+1}$, e n é o tamanho de um fragmento de *stream* de contexto C. (WANG; CHAN, 2009, p. 613, tradução nossa).

Algoritmo 3: Geração de Sequência de Testes

Entrada: Estado atual do DAS no DFTS

Entrada: Tamanho da sequência de testes

Saída: Sequência de Teste

início

- $s \leftarrow$ *Estado atual do DAS no DFTS*
- $S \leftarrow$ *Conjunto de estados vizinhos ao estado atual s*
- $n \leftarrow$ *Tamanho da sequência de testes*
- $SC \leftarrow \emptyset$
- $H \leftarrow \emptyset$
- **para** *cada* $k \in \{1, \ldots, n\}$ **faça**
  - **para** *cada* $sn \in S$ **faça**
    - $H \leftarrow H \cup hamming(sn, s)$
  - **fim**
  - $r \leftarrow$*número aleatório entre 0 e 1*
  - **para** *cada* $h \in H$ **faça**
    - $p \leftarrow \mathrm{h} \div \sum_{n=1}^{|H|} H_n$
    - **se** $(r < p)$ **então**
    - $SC \leftarrow SC \cup$ *estado com transição relacionada a distância h*
    - encerrar o iteração
    - **fim**
  - **fim**
  - $H \leftarrow \emptyset$
  - $s \leftarrow$ *último estado selecionado na geração*
  - $S \leftarrow$ *estados vizinhos de s*
- **fim**

fim

A geração de testes utiliza um algoritmo que calcula a distância de Hamming entre o contexto do estado atual do DAS no DFTS em relação aos seus vizinhos. Então, os valores obtidos são atribuídos como pesos nas transições para o estado vizinho. Quanto maior a distância de Hamming entre um estado e seu vizinho, maior a chance desse vizinho ser selecionado para a sequência. Essa abordagem difere da métrica de Diversidade de Contexto (WANG; CHAN,

2009) porque não compara a distância entre todos os pares de contextos.

A sequência de teste gerada neste trabalho consiste em um caminho no modelo DFTS, de forma que é possível obter uma sequência de variações de contexto. Quando comparado com o trabalho de Santos (2017), as sequências geradas tem maior chance de ter uma alta diversidade de contexto entre estados.

O Algoritmo 3 detalha os passos relativos à geração da sequência de testes. Da linha três à sete são inicializadas as variáveis. A partir da linha nove calcula-se a distância de Hamming entre o contexto do estado atual e de todos os estados vizinhos, de forma que os valores são guardados na variável *H*. Considerando o cenário de exemplo com a Figura 5, a distância entre os estados *S4* (BM, I, C) e *S3* (BM, I, NC) é igual ao número de contextos diferentes entre eles. Como o *S4* e *S3* diferem em um contexto, a distância entre eles é igual a um.

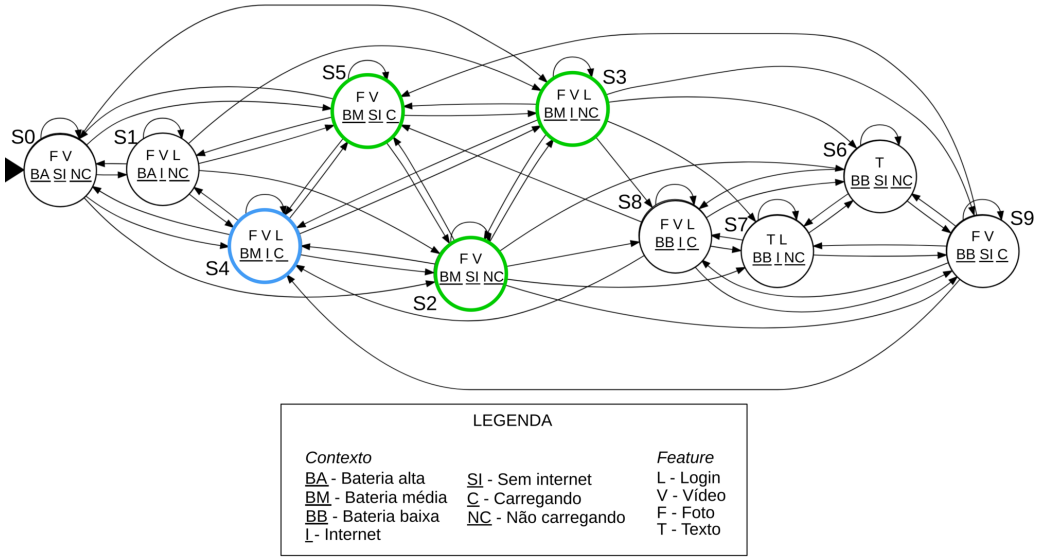
Em seguida, é gerado um número aleatório entre zero e um na linha 12. A partir da linha 13, itera-se sobre as distâncias de cada estado vizinho e calculam-se seus pesos *p* em relação as demais transições. O peso *p* é obtido na linha 14 dividindo-se a distância de Hamming *h* pelo somatório de todas as distâncias. Caso o peso da transição seja menor ou igual a *p*, então o estado é selecionado para compor a sequência de teste. O processo é repetido até que uma sequência de teste do tamanho estipulado seja gerada.

Dado que a sequência de teste é gerada com maiores pesos para os estados com maior Diversidade de Contexto, a transição de estados deve disparar mais adaptações. O uso da diversidade de contexto entre estados consecutivos de modelos e altas variações de contexto (MUNOZ; BAUDRY, 2009) foram explorados em DAS em tempo de projeto (SIQUEIRA *et al.*, 2018). Porém, se a geração dos testes fosse realizada exatamente como proposto por Wang e Chan (2009), qualquer um dos itens da sequência deveria ser passível de substituição. Isso não é possível dado a evolução de contexto presente no DFTS.

A Figura 11 ilustra uma sequência de testes para o modelo de contexto da aplicação de exemplo. Supondo que o sistema estivesse no estado *S4* quando suas regras de adaptação foram alteradas, então esse estado seria identificado pela RETAkE. A partir desse ponto, seriam explorados outros estados de contexto diferentes do atual (destacado na cor azul). Utilizando o processo de geração de teste descrito anteriormente, seria possível obter a sequência de estados *S5*, *S2* e *S3* (destacados na cor verde).

De forma similar ao processo anteriormente descrito, o engenheiro de software também tem a opção de gerar uma sequência de testes de forma totalmente randômica. A

Figura 11 – Exemplo de Sequência de Teste



Fonte: elaborada pelo autor.

principal diferença é que no lugar de calcular as distâncias de Hamming entre o estado de contexto atual e seus vizinhos, os novos estados são selecionados aleatoriamente.

4.2.5 Execução de Testes

A última etapa relacionada aos testes da abordagem é a execução das sequências. A ideia principal é consumir a sequência de testes gerada na etapa anterior e atribuir seus valores às variáveis na fase de monitoramento do DAS (atividade cinco da Figura 9). Estas mudanças nos valores de contexto impulsionam a avaliação das regras de adaptação do sistema e, consequentemente, executam as reconfigurações das *features* do sistema.

Nesse sentido, cada item da sequência de testes é utilizado para gerar um caso de teste de adaptação baseado na definição de Santos (2017). A avaliação do resultado da adaptação é feita através da análise do estado das *features*, de forma que essas devem corresponder a uma configuração válida. Para isso, as *features* sensíveis ao contexto são representadas através de uma expressão lógica que determina a validade da reconfiguração. Desse modo, cada *feature* de um modelo eCFM pode estar ativa/desativa, sendo representada pelo domínio de valores $D = \{verdadeiro, falso\}$.

RETAkE gera uma fórmula lógica que representa o oráculo em cada caso de teste, sendo essa composta por uma conjunção de: (1) uma proposição para cada *feature* requerida, (2) uma negação de proposição para cada *feature* não requerida (excluída ou não declarada) e uma (3) fórmula que determina a validade de *Configuration Correctness* (ver Subseção 4.2.2.1).

Tabela 5 – Exemplo de oráculo para um caso de teste

	Fórmula
***Features* requeridas**	Texto , Login
***Features* não requeridas**	¬ Vídeo , ¬ Foto
Configuration Correctness	(Texto → GREat) ∧ (Vídeo → GREat) ∧ (Foto → GREat) ∧ (Login → GREat)
Exemplo de fórmula	Texto ∧ Login ∧¬ Vídeo ∧ ¬ Foto ∧ (Texto → GREat) ∧ (Video → GREat) ∧ (Foto → GREat) ∧ (Login → GREat)

Fonte: elaborada pelo autor.

A Tabela 5 apresenta um exemplo da fórmula gerada. A última linha da tabela exemplifica um oráculo para o modelo de *features* da Figura 4. De acordo com o eCFM da Figura 4, se o contexto da aplicação for *Bateria Baixa*, *Não Carregando* e *Conexão*, então as *features Texto* e *Login* devem estar ativas. Essas *features* devem compor proposições verdadeiras e as demais compõem proposições negadas. A última linha exemplifica a fórmula final.

Como a execução de testes pode causar perturbações no estado do sistema, é necessário isolar o mecanismo de adaptação durante os testes em tempo de execução. O isolamento do mecanismo é realizado bloqueando ações de adaptação que podem causar ações irreversíveis (e.g., ações que afetam o mundo real) ou indesejadas (e.g., mudanças na interface do usuário) (LAHAMI; KRICHEN; JMAIEL, 2016). Para isso, a RETAkE permite que o mecanismo de adaptação entre em modo de teste através das técnicas de bloqueio (GONZÁLEZ; PIEL; GROSS, 2009) e orientação a aspectos (LAHAMI; KRICHEN; JMAIEL, 2016). O isolamento é realizado através da interceptação e bloqueio das adaptações disparadas por contexto durante os testes. Nesse sentido, os oráculos dos casos de teste concentram-se na verificação da resposta de reconfiguração a ser executada pelo mecanismo de adaptação, não permitindo a adaptação real do sistema.

O teste em tempo de execução consiste no controle das adaptações e no contexto do DAS. Cada teste inicia um ciclo de adaptação (monitoramento de contexto, análise e execução da adaptação) e injeta contexto no mecanismo de adaptação. Em seguida, avalia-se a resposta de adaptação produzida pelo mecanismo com o estado das *features*. A abordagem RETAkE repete esse processo até que a sequência de testes termine, recomeçando após alterações nas regras de adaptação. Portanto, cada caso de teste precisa de uma execução completa do ciclo de adaptação do DAS e seu mecanismo de adaptação permanece bloqueado.

Vale mencionar que se uma regra de adaptação muda durante a execução dos testes, o modelo de variabilidade também é atualizado, mas essa mudança não gera um novo ciclo

de testes de forma imediata. Isso impede a verificação de entrar em um ciclo de execuções de testes. Após a finalização dos testes, RETAkE para de controlar o início dos ciclos de adaptação e de injetar contexto no DAS. Como essa última atividade torna inconsistente o contexto do DAS, a abordagem libera o mecanismo para perceber novamente o contexto atual e se adaptar regularmente.

4.3 Implementação da RETAkE

CONtext-variability-based software Testing Library (CONTroL) é uma ferramenta de suporte para a verificação de DAS em tempo de projeto (SANTOS *et al.*, 2018). O objetivo original da CONTroL era a execução de sequências de teste no mecanismo de adaptação do DAS.

A ferramenta é inserida no projeto do DAS para permitir a instrumentação do código-fonte, monitorar e controlar as variáveis relacionadas às *features* e contextos. O paradigma facilitador da instrumentação é a Programação Orientada a Aspectos (KISELEV, 2002). O uso desse paradigma permite a interceptação do fluxo de execução de forma transparente e fazendo com que a verificação seja um interesse entrecortante. CONTroL é disponibilizado como uma biblioteca Android[1] escrita em Java com AspectJ[2]. Dessa forma, a ferramenta pode assumir o processo de compilação e produzir arquivos de classe Java modificados.

Figura 12 – Exemplo de uso das anotações (A) *@ControlContext*, (B) *@ControlContextGroup*, (C) *@ControlFeature* e (D) *ControlStatusAdapted*

(A)
```
@ControlContext(contextName = "hasInternetConnection")
public boolean hasInternetConnection() {
    Context context = ContextManager.getInstance().getAppContext();
    // Ommited source code
    return (activeNetwork != null && activeNetwork.isConnectedOrConnecting());
}
```

(B)
```
@ControlFeature(feature = "video")
public boolean isFeatureActivated() {
    return super.isActivated();
}
```

(C)
```
@ControlContextGroup(contextGroupName = "battery")
public double getmBatteryLevel() {
    if (mContext == null) {
        mContext = ContextManager.getInstance().getAppContext();
    }
    Intent batteryIntent = mContext.registerReceiver(null,
            new IntentFilter(Intent.ACTION_BATTERY_CHANGED));
    int level = batteryIntent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
    int scale = batteryIntent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
    return level / (double) scale;
}
```

(D)
```
private Runnable mContextRunnable = new Runnable() {
    @ControlStatusAdapted
    @Override
    public void run() {
        for (DeviceContext deviceContext : mContextList) {
            if (deviceContext.updateContextInfo(mAppContext)) {
                deviceContext.notifyObservers();
            }
        }
        // Ommited source code
    }
};
```

Fonte: adaptado de Santos *et al.* (2018)

Como forma de instrumentação, CONTroL fornece um conjunto de quatro anotações

[1] https://developer.android.com/studio/projects/android-library
[2] https://www.eclipse.org/aspectj/

Java para atributos de classes que são descritas a seguir. A Figura 12 ilustra essas anotações utilizadas no código-fonte da aplicação GREat Tour. A seguir são detalhados as definições de cada uma:

a) *@ControlContext* anota métodos que indicam a presença de um contexto específico como um valor booleano (e.g., conexão de internet) e recebe como parâmetro o nome da variável de contexto;
b) *@ControlContextGroup* anota métodos que retornam valores de outros tipos de domínios (e.g., porcentagens com o nível de carga da bateria de um dispositivo);
c) *@ControlFeature* monitora o estado das *feature* e recebe como parâmetro o nome da mesma. Isso é feito através da anotação de métodos que indicam se as *features* foram ativadas/desativadas;
d) *@ControlStatusAdapted* é utilizada para anotar o método que inicia as ações de adaptação ao DAS na fase de execução.

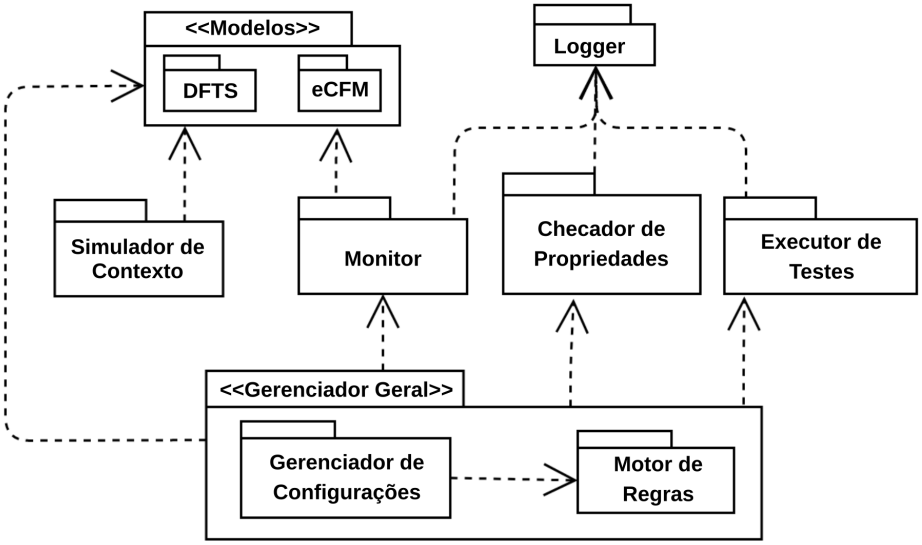
A ferramenta CONTroL foi projetada para testes em tempo de projeto, então ela foi utilizada como base para a implementação da RETAkE, sendo então denominada de CONTroL@Runtime. Esta reestruturação habilitou que a mesma pudesse ser executada em conjunto com o DAS durante sua execução em ambiente final. Dessa forma, o usuário tem disponível as seguintes funcionalidades: (i) criar arquivos de configuração e modelos; (ii) instrumentar o DAS para controle e monitoramento de variáveis de interesse; (iii) executar a checagem de propriedades; (iv) executar testes em tempo de execução; e (v) configurar integração de resultados das técnicas.

Para que que as atividades fossem executadas automaticamente, foi planejado um artefato que pudesse ser instalado no projeto do DAS. A ferramenta foi desenvolvida em Java e disponibiliza interfaces para o desenvolvedor do DAS. A Figura 13 apresenta a arquitetura da ferramenta que implementa a RETAkE na forma de um diagrama de pacotes Unified Modeling Language (UML)[3]. A seguir são descritas as responsabilidades de cada um desses pacotes:

a) **Gerenciador Geral**: Este pacote é responsável pelo gerenciamento de todo o processo de verificação, sendo composto pelos pacotes *Gerenciador de Configurações* e *Motor de Regras*. O pacote *Gerenciador de Configurações* é responsável por coordenar as atividades de verificação configuradas pelo engenheiro de software. Para isso, é feito uso do *Motor de Regras* que interpreta as regras definidas nas configurações e auxilia

[3] https://www.uml.org/

Figura 13 – Diagrama de pacotes da CONTroL@Runtime



Fonte: elaborada pelo autor.

no processo de verificação. O pacote *Gerenciador Geral* faz uso dos pacotes utilizados na verificação (*Monitor*, *Checador de Propriedades* e *Executor de Testes*) e o *Modelos*, possuindo também uma máquina de estados que alterna entre os seguintes modos de operação: monitoramento, simulação, checagem e testes;

b) **Simulador de Contexto**: O pacote *Simulador de Contexto* é responsável por gerar sequências de contexto para que se possa testar e configurar a CONTroL@Runtime. Essas sequências são geradas através do pacote *Modelos*, que fornece o modelo DFTS a ser percorrido. O *Simulador de Contexto* pode utilizar simulações randômicas ou baseadas na métrica de Diversidade de Contexto;

c) **Monitor**: Este pacote contêm as anotações utilizadas para instrumentar o sistema. Os modos de execução do *Gerenciador Geral* também influenciam nas estratégias do *Monitor* em relação ao contexto. Caso esteja monitorando as variáveis de contexto, o pacote apenas armazena seus estados. Caso esteja sendo executada uma simulação ou testes, o *Monitor* também modifica as variáveis de contexto para inserir os valores das sequências geradas. Independente do modo de execução, o estado das *features* é apenas coletado;

d) **Checador de Propriedades**: O pacote *Checador de Propriedades* efetua a verificação de propriedades comportamentais da adaptação, obtendo os modelos e regras de adaptação através do pacote *Modelos*;

e) **Executor de Testes**: O pacote *Executor de Testes* executa as suítes de teste em tempo de execução. Esse pacote faz uso do *Logger* para obter as informações de resultados

anteriores que possam auxiliar na decisão de inicio dos testes;

f) **Modelos**: O pacote *Modelos* atua como repositório dos modelos e regras de adaptação utilizados pela abordagem. Os modelos armazenados são o DFTS e o eCFM;

g) **Logger**: Esse pacote é utilizado pelo pacote *Monitor* e pelos pacotes de verificação (*Checador de Propriedades* e *Executor de Testes*) para salvar os resultados de verificação, respectivamente.

As anotações originais da CONTroL só eram capazes de anotar métodos de classes. Para aumentar a capacidade de instrumentação da CONTroL@Runtime, foram criadas quatro novas anotações de código-fonte. A anotações @*ControlContextMethod* e @*ControlContextGroupMethod* realizam, respectivamente, as interceptações feitas anteriormente por @*ControlContext* e @*ControlContextGroup*. Essas últimas anotações agora são utilizadas para anotar atributos de classe e interceptar seus valores durante a leitura ou escrita. Além disso, a anotação @*ControlFeature* foi alterada para ser utilizada com os atributos de classe. A Figura 14-A e 14-B destaca em vermelho, respectivamente, as anotações @*ControlContextGroupMethod* e @*ControlContextMethod*.

Figura 14 – Exemplo de uso das novas anotações: (A) @*ControlContextGroupMethod*, (B) @*ControlContextMethod*, (C) @*ControlDelay* e (D) @*ControlEffectorIsolation*

```
(A)
@ControlContextGroupMethod(contextGroupName = "battery")
public double getmBatteryLevel() {
    if (mContext == null) {
      mContext = ContextManager.getInstance().getAppContext();
    }

    Intent intent = mContext.registerReceiver(null,
      new IntentFilter(Intent.ACTION_BATTERY_CHANGED));

    int level = batteryIntent.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
    int scale = batteryIntent.getIntExtra(BatteryManager.EXTRA_SCALE, -1);

    this.mBatteryLevel = level / (double) scale;
```

```
(B)
@ControlContextMethod(contextName = "internet")
public boolean hasInternetConnection() {
    Context context = ContextManager.getInstance().getAppContext();
    ConnectivityManager connectivityManager = (ConnectivityManager)
        context.getSystemService(context.CONNECTIVITY_SERVICE);
    NetworkInfo activeNetwork = connectivityManager.getActiveNetworkInfo();
    return (activeNetwork != null && activeNetwork.isConnectedOrConnecting());
}
```

```
(C)
public class ContextManager {

    private static ContextManager contextManagerInstance = null;

    @ControlDelay
    private long adaptationDelay;

    private Context mAppContext;
    private Activity mCurrentActivity;

    private Collection<DeviceContext> mContextInterests;
```

```
(D)
@Override
@ControlEffectorIsolation
public void activate(String currentActivity) {
    if (currentActivity.equals(FilesActivity.class.getSimpleName())) {
        mContextManager.getCurrentActivity().runOnUiThread(new Runnable() {
          public void run() {
          // Omitted source code
            btnImg.setEnabled(true);
            Toast.makeText(
                mContextManager.getAppContext(),
                "TEXT FEATURE ACTIVATED",
                Toast.LENGTH_SHORT
             ).show();
          }
        });
    }
}
```

Fonte: elaborada pelo autor.

As outras duas novas anotações são @*ControlDelay* e @*ControlEffectorIsolation*. Para o correto uso da CONTroL@Runtime, considera-se que o DAS inicia cada novo ciclo de adaptação após um tempo de atraso pré-determinado. A anotação @*ControlDelay* controla

esse tempo de atraso do ciclo de adaptação durante a execução dos testes, permitindo que o mesmo seja reduzido para diminuir o impacto no tempo de adaptação. A anotação *@ControlEffectorIsolation* deve ser utilizada para isolar efeitos de adaptações que possam perturbar o sistema durante a execução dos testes. Por exemplo, se um método altera a disponibilidade de um botão do sistema baseado na ativação de regras, então esse método precisa ser anotado com *@ControlEffectorIsolation* para não afetar as funcionalidades para o usuário durante a execução de testes. As Figuras 14-C e 14-D exemplificam, respectivamente, as anotações *@ControlDelay* e *@ControlEffectorIsolation*.

Para exemplificar o mapeamento entre o modelo de *features* e as anotações, a Figura 14-B contêm a anotação necessária para o contexto *Internet*, que é demandado pela *feature* Login da Figura 4. A *feature* Login por sua vez é instrumentada na Figura 14-D. Essas anotações de contexto e *features* no código-fonte representam o mapeamento da regra RA2 da Tabela 1.

A ferramenta CONTroL@Runtime possui um forma de instrumentação intrusiva e exige que o código-fonte do DAS esteja anotado antes da primeira implantação. Com isso, a implementação apresentada nesta seção não permite a integração de novas *features* em tempo de execução.

4.4 Exemplo de Uso

Nesta seção são descritos detalhes da execução das atividades da CONTroL@Runtime (ver Subseção 4.3).

No primeiro passo na abordagem, o engenheiro de software tem que especificar os modelos (eCFM e DFTS), o arquivo de configuração e as regras de adaptação a serem usadas pela CONTroL@Runtime. O modelo DFTS pode ser especificado utilizando a ferramenta TestDAS (SANTOS, 2017). Já o arquivo de configuração contém dados como o tamanho das sequências da simulação e testes, o domínio para variáveis de contexto, expressões para avaliação de contextos e outras configurações para as atividades de verificação. Todos esses arquivos devem seguir o formato JavaScript Object Notation (JSON)[4]. Outras notações com esquema como o Extensible Markup Language (XML)[5] poderiam ter sido utilizadas, mas o JSON foi escolhido dado a sua popularidade em projetos de software.

A Figura 15 ilustra trechos do modelo DFTS e das regras de adaptação para a

[4] https://www.json.org/
[5] https://www.w3.org/XML/

aplicação de exemplo (ver Seção 2.1).

Figura 15 – Exemplos de (A) modelo DFTS e (B) regras de adaptação da CONTroL@Runtime


```
[
    {
        "id":0,
        "features":["login" ,"video"],
        "contexts":{
              "battery":"baterryHigh",
              "internet":"hasNotInternetConnection",
              "charging":"isNotCharging"
        },
        "edges": {
              "0": {
                   "battery": "batteryHigh",
                   "internet": "hasNotInternetConnection",
                   "charging": "isNotCharging",
              },
             "1": {
                   "battery": "batteryHigh",
                   "internet": "hasInternetConnection",
                   "charging": "isNotCharging",
             },
             "3": {
                   "battery": "batteryMedium",
                   "internet": "hasInternetConnection",
                   "charging": "isNotCharging",
             },
             "2": {
                   "battery": "batteryMedium",
                   "internet": "hasNotInternetConnection",
                   "charging": "isNotCharging",
             },
             "5": {
                   "battery": "batteryMedium",
```

```
{
    "1": {
        "ruleId": "1",
        "description":"",
        "contextConditions":[
              {
                   "contextVariable":"battery",
                   "contextValue":"0.30",
                   "operator": "<="
              }
        ],
        "features": ["text"],
        "expression":  null,
        "wasRuleActivated": false,
        "lastStatus": false,
        "lastResultInterleavingCorrectness": false,
        "activationTimeInterval": 10000,
        "lastActivationTime": 10000,
        "lastResultRuleLiveness": false,
        "lastResultAdaptationActions":true,
        "lastDeactivatedFeatures": []
    },
    "2": {
        "ruleId": "2",
        "description":"",
        "contextConditions":[
              {
                   "contextVariable":"internet",
                   "operator": "=="
                   "contextValue":"true",
              }
        ],
```

Fonte: elaborada pelo autor.

O engenheiro de software deve instrumentar o DAS utilizando o conjunto de anotações Java previamente descritas, de forma que isso cria um vínculo direto entre o código-fonte e os modelos. Dado a necessidade de anotar atributos e métodos, a implementação do DAS precisa seguir o paradigma de programação orientada a objetos e ter variáveis bem definidas para armazenar informações de contexto, estados das *features* e o tempo de atraso para ciclo de adaptação. Adicionalmente, o DAS precisa ter métodos para controlar o início de adaptações e para executar adaptação que ativa/desativa as *features*. Esses últimos são importantes para que as ações de adaptações possam ser interceptadas e bloqueadas.

Em seguida, é realizada a implantação do sistema e a CONTroL@Runtime instancia os modelos e regras de adaptação. O engenheiro tem a possibilidade de iniciar uma simulação de contexto caso queria utilizar um ambiente controlado para configuração. Para isso, o modelo DFTS é utilizado para gerar uma sequência de contexto do tamanho especificado no arquivo de configuração. Vale ressaltar que o engenheiro tem a opção de utilizar o mesmo algoritmo da geração de testes para realizar a simulação do DAS. Sendo assim, a execução da simulação não está relacionada com o comportamento real do DAS.

Essa atividade é relevante para que o engenheiro configure a CONTroL@Runtime

antes da implantação final do DAS. Por exemplo, o engenheiro pode analisar diferentes configurações (e.g., verificações e tamanhos de sequências) da CONTroL@Runtime para identificar o tempo de atraso necessário durante os testes.

Após a inicialização do sistema, tanto por simulação quanto em ambiente real, inicia-se o monitoramento e controle do DAS. Em caso de simulação de contexto, a CONTroL@Runtime intercepta as variáveis de contexto e atribui os valores gerados pela sequência de testes. O monitoramento também tem a função de armazenar o estado dos sistemas para análise posterior, com a diferença de que esse comportamento está presente durante toda a execução. Essas informações são guardadas diretamente na instância do modelo eCFM.

Para a atualização das regras e modelos especificados, o DAS deve executar declarações semelhantes à chamadas de uma biblioteca. A Figura 16 ilustra as declarações necessárias para realizar a atualização na aplicação GREat Tour. A figura ilustra o seguinte cenário: uma relação do tipo *requer* em que as *features Vídeo* e *Foto* sejam ativadas apenas no contexto com bateria alta. Das linhas 1 até 17 são declaradas as condições de guarda de contextos, estado das *features* e criação de regra. As linhas 19 à 21 fazem essas mesmas atualizações no modelo de *features*. As linhas 23 e 24 enviam as requisições para a CONTroL@Runtime.

Figura 16 – Trecho de código-fonte com requisição para atualização de modelos e regras de adaptação

```
 1 ContextCondition cc1 = new ContextCondition("battery", "0.70", "<=");
 2 ContextCondition cc2 = new ContextCondition("charging", "false", "==");
 3 ArrayList<ContextCondition> contextConditions = new ArrayList<>();
 4 contextConditions.add(cc1);
 5 contextConditions.add(cc2);
 6
 7 HashSet<String> features = new HashSet<>();
 8 HashSet<String> featuresToDeactivate = new HashSet<>();
 9 featuresToDeactivate.add("video");
10 featuresToDeactivate.add("photo");
11
12 AdaptationRule newRule = new AdaptationRule(
13     String.valueOf(mRuleId),
14     "", contextConditions,
15     features,
16     featuresToDeactivate
17 );
18
19 RuleUpdate ruleUpdate = new RuleUpdate();
20 ruleUpdate.removeFeatureFromContext("battery", "batteryMedium", "video", null);
21 ruleUpdate.removeFeatureFromContext("battery", "batteryMedium", "photo", null);
22
23 ControlManager.getInstance().addControlAdaptationRule(newRule);
24 ControlManager.getInstance().addControlRuleUpdate(ruleUpdate);
```

Fonte: elaborada pelo autor.

Dando início a primeira atividade de verificação, a checagem de propriedades é iniciada logo após o método anotado com *@ControlStatusAdapted* é executado (indicando o fim da fase de execução). O processo é majoritariamente guiado pelo motor de regras da

CONTroL@Runtime, que avalia as regras dado um determinado contexto. Em seguida, todas as *features* do modelo instanciado do eCFM são analisadas, e as operações do DAS permanecem bloqueadas durante a verificação.

Em seguida, as atividades de testes são iniciadas caso sejam identificadas mudanças no modelo de *features* ou dependendo dos resultados da checagem de propriedades. Dessa forma, será gerada uma sequência de testes, em que cada teste de adaptação utiliza um ciclo de adaptação do DAS. Cada item da sequência de testes representa uma volta no ciclo de adaptação, mantendo as operações de adaptações do sistema bloqueadas através da anotação *@ControlEffectorIsolation*.

Durante a execução dos testes, a anotação *@ControlDelay* intercepta a variável com o tempo de atraso da adaptação para reduzi-lo e, consequentemente, acelera as atividades de testes. O tempo deve ser configurado pelo engenheiro de software de acordo com o atraso necessário para processamento da adaptação no DAS.

Quando a sequência termina e o método anotado com *@ControlStatusAdapted* é executado, o DAS inicia novamente a leitura de contexto do ambiente e atualiza o estado de suas *features*. Por fim, os resultados das verificações são armazenados após cada ciclo de adaptação do DAS.

Figura 17 – Trecho de arquivo de configuração com declaração de técnicas de verificação

A

```
{
    "featureModelPath":"/feature.json",
    "contextModelPath":"/context_feature.json",
    "adaptationRulesPath":"/rules.json",
    "stateModelPath":"/state_model.json",
    "adaptationDelayPeriod":5,
    "isHighLevelContext":false,

    "loggerConfiguration":{
        "isActivated": true,
        "traceLogPath": "/logs",
    },

    "simulatorConfiguration":{
        "isActivated":true,
        "contextSequenceSize":50,
        "contextSimulationType":"random",
        "randomInitialSimulation":false,
        "initialState":0,
        "initialRandomStates":[1, 2, 3]
    },
```

B

```
    "assertionCheckerConfiguration":{
        "isActivated":true,
        "afterAdaptation":false
        "afterChangeAdaptationRule":true,
    },

    "runtimeTestConfiguration":{
        "isActivated":true,
        "isAlways":true,
        "afterAdaptation":false,
        "afterChangeAdaptationRule":false,
        "defaultSequenceSize":10,
        "defaultGenerationType":"diversity",
        "executionMode":"state",
        "propertiesSuccess":{
            "testSequenceSize":"5",
            "testSequenceGeneration":"random",
        },
        "propertiesFailure":{
            "testSequenceSize":"10",
            "testSequenceGeneration":"diversity",
        }
    },
```

Fonte: elaborada pelo autor.

A Figura 17 ilustra um trecho do arquivo de configuração utilizado para declarar as técnicas de verificação da CONTroL@Runtime. Por exemplo, as linhas 2 à 7 da Figura 17-A listam configurações gerais como a localização de arquivos e o tempo de atraso de adaptação

durante o modo teste. As linhas 9 à 21 declaram configurações especificas para o *Logger* e *Simulador de Contexto*, em que neste último é possível determinar o tamanho da sequência de simulação e estados iniciais randômicos no modelo de contexto. Já nas linhas 23 à 27 da Figura 17-B é possível verificar configurações da checagem de propriedades, de forma que é determinado se a atividade é executada apenas após adaptações das regras e modelo.

A Figura 17-B também apresenta detalhes da configuração da atividade de testes nas linhas 29 à 43. Por exemplo, pode ser configurado o algoritmo para a geração de testes e o tamanho da sequência utilizado. No caso da integração dos testes com os resultados de propriedades, as linhas 37 à 40 detalham que deve ser executada uma sequência de tamanho cinco caso não ocorram falhas nas propriedades. Já as linhas 41 à 44 exemplificam que deve ser executada uma sequência de tamanho 10 em caso de falha.

Essa integração pode ser útil no cenário em que logo que uma falha seja identificada pelas propriedades, os testes iniciam para analisar o estado do sistema. O DAS pode utilizar essas informações para garantir a correta execução da adaptação. Sendo assim, é possível obter um nível maior de confiança sobre as reconfigurações baseado na execução do código-fonte.

4.5 Resumo do Capítulo

Este capítulo apresentou a RETAkE, uma abordagem para a verificação da adaptação de DAS durante sua execução. Foram apresentadas as atividades relativas à execução de testes de adaptação e checagem de propriedades comportamentais em sistemas que se adaptam baseados em contexto. Também foi apresentado uma ferramenta de suporte que automatiza a execução dessas atividades junto das operações do DAS.

A execução dos testes foi proposta como uma forma de verificar a conformidade das adaptações do mecanismo de adaptação do DAS e suas regras. O foco dessa atividade é revelar possíveis adaptações incorretas após a execução da lógica de adaptação. Para isso, a abordagem realiza os seguintes passos: processamento das alterações do modelo e regras, geração de sequências de testes a partir do estado de contexto atual e execução de testes de adaptação no sistema.

Para apoiar a execução dos testes, foi proposto também uma atividade de suporte para checar propriedades comportamentais relacionadas às *features*. Nesse ponto, a abordagem checa um total de quatro propriedades com foco nas regras de adaptação. O processo de verificação acontece apenas no estado de contexto atual do sistema, podendo revelar falhas que ocorreram

no sistema.

Por fim, foi apresentado a CONTroL@Runtime, uma ferramenta de verificação que permite a instrumentação do código-fonte do DAS para controle e monitoramento do mesmo. A CONTroL@Runtime possui um componente gerenciador responsável por coordenar toda a execução das atividades de forma automatizada. Disponível para sistemas desenvolvidos com a linguagem de programação Java, a ferramenta proposta apoia a geração de testes em tempo de execução do DAS.

O próximo capítulo detalha uma avaliação da abordagem através do uso do artefato de software proposto no presente capítulo. O foco é avaliar a capacidade da abordagem em detectar falhas de adaptação e, para esse fim, foi utilizada uma técnica de mutantes com operadores de mutação específicos para regras de adaptação. Adicionalmente, será apresentada uma prova de conceito com a checagem de propriedades comportamentais. Por fim, é realizada uma avaliação para identificar a sobrecarga no tempo de adaptação gerada pela abordagem.

5 AVALIAÇÃO

Este capítulo detalha as três avaliações realizadas com a RETAkE, de forma que todas elas foram conduzidas utilizando dois exemplos de DASs (Seção 5.1). A primeira avaliação foi uma prova de conceito para analisar a viabilidade de detecção de falhas de propriedades comportamentais (Seção 5.2). Na segunda avaliação foi executado um teste de mutantes para determinar a eficácia dos testes em tempo de execução (Seção 5.3). Por fim, foi realizada uma avaliação focada no impacto causado pela ferramenta que implementa a abordagem RETAkE na adaptação do DAS (Seção 5.4). Todas as avaliações desta seção tiveram como foco o mecanismo de adaptação dos DASs, de forma que as funcionalidades de cada *feature* foram abstraídas.

5.1 Exemplos de DAS Utilizados

Para utilizar e avaliar a RETAkE é necessário utilizar implementação de DAS que seja baseada em regras. Adicionalmente, é necessário que o código-fonte do DAS esteja disponível e possa ser modificado para instrumentação. Considerando esses requisitos, foram selecionadas as aplicações GREat Tour (LIMA *et al.*, 2013) e Phone Adapter (SAMA *et al.*, 2010). Foram utilizadas versões modificadas das aplicações que simulavam a atualização de regras em tempo de execução. Ambas as versões são descritas a seguir.

5.1.1 GREat Tour

Para as três avaliações foi utilizada uma versão do GREat Tour que implementa o cenário de uso descrito na Seção 2.1. A versão original do GREat Tour tinha suas regras de adaptação implementadas como declarações *se-então* no código-fonte, e não suportava atualização de regras de adaptação em tempo de execução.

Para melhorar a declaração e avaliação das regras de adaptação, a aplicação foi manualmente modificado para incluir uma lógica simplificada de atualização de regras. Dessa forma, foi utilizado o SUCCEEd (ARAGÃO JUNIOR *et al.*, 2018), um framework que objetiva a criação de *workflows* para executar ações em sistemas adaptativos de Internet das Coisas. Mesmo com esse foco, um subconjunto de suas funcionalidades habilita o desenvolvimento de um mecanismo de regras desacoplado que executa ações baseadas em filtros. Este framework permite a implementação das regras de adaptação como classes da linguagem Java, em que as instâncias são analisadas e executadas por um módulo de avaliação. O SUCCEEd foi

escolhido pela sua simplicidade de configuração, viabilizando a definição de regras de adaptação legíveis por um operador humano. A implementação de regras como classes também facilitou a atualização das mesmas em tempo de execução através dos *workflows*.

Originalmente a aplicação não permitia a alteração de suas regras de forma automatizada, então ela foi modificada para simular o cenário apresentado na Seção 2.1. Sendo assim, as regras são alteradas imediatamente após o início da execução da aplicação.

Tanto o eCFM quanto o DFTS já foram apresentados nas Figuras 4 e 5, respectivamente. Vale ressaltar que todos os estados foram modelados com auto-transição para gerar simulações que verifiquem a reconfiguração sem mudanças de contexto.

5.1.2 Phone Adapter

Phone Adapter (SAMA *et al.*, 2010) é uma aplicação móvel que permite aos usuários automatizar configurações no smartphone através da criação de perfis. Cada perfil contém configurações específicas (toque, vibração e modo avião) para o dispositivo e um conjunto de regras de adaptação. A aplicação monitora continuamente o contexto de execução do dispositivo e, em seguida, realiza o processo de adaptação e ativa o perfil da regra satisfeita.

Tabela 6 – Regras de adaptação do Phone Adapter

Identificador da Regra	**Condição de guarda de contexto**	**Modo Avião**	**Toque**	**Vibração**
RA1	GPS = verdadeiro $\wedge$ Localização $\neq$ Casa $\wedge$ Localização $\neq$ Escritório	off	off	on
RA2	GPS = verdadeiro $\wedge$ Localização = Casa	off	on	on
RA3	GPS = verdadeiro $\wedge$ Localização = Escritório $\wedge$ Hora $\neq$ Reunião	off	on	on
RA4	GPS = falso $\wedge$ Localização $\neq$ Casa $\wedge$ Localização $\neq$ Escritório	off	on	off
RA5	GPS = verdadeiro $\wedge$ Velocidade $\geq$ 8,0	off	on	on
RA6	GPS = verdadeiro $\wedge$ Localização = Escritório $\wedge$ Hora = Reunião $\wedge$ bluetooth > 3	on	off	off
RA7	GPS = verdadeiro $\wedge$ Localização = Escritório $\wedge$ Hora $\neq$ Reunião $\wedge$ bluetooth < 3	on	off	off
RA8	GPS = falso $\wedge$ Localização $\neq$ Casa $\wedge$ Localização $\neq$ Escritório	off	on	off
RA11	GPS = falso $\wedge$ Localização $\neq$ Casa $\wedge$ Localização $\neq$ Escritório	off	on	off
RA10	GPS = verdadeiro $\wedge$ Velocidade < 8,0	off	off	on

Fonte: elaborada pelo autor.

Para este trabalho, foi utilizada uma versão do Phone Adapter que contém um subconjunto de perfis e regras de adaptação baseadas no trabalho original. Como o Phone Adapter foi desenvolvido para permitir a criação manual de perfis por um usuário, foi necessário automatizar a inicialização dos perfis para esta avaliação. Os perfis considerados para esta prova

de conceito são os seguintes: *Geral*, *Casa*, *Rua*, *Corrida*, *Escritório* e *Reunião*. A Tabela 6 lista o conjunto de regras de adaptação utilizadas. Por exemplo, a regra RA2 é ativada quando o contexto GPS é verdadeiro e a localização é *Casa*, determinando que a aplicação está no perfil *Casa*. Nesse perfil são ativadas as *features Vibração* e *Toque*.

O Phone Adapter permitia ao usuário criar, editar e excluir regras de adaptação, mas não permitia a atualização dessas regras de forma automatizada. Logo, foi utilizada uma versão da aplicação que simulava um cenário de atualização de regras. Para isso, Phone Adapter foi manualmente modificado para alterar as ações de regras tão logo a execução iniciasse.

Para o Phone Adapter foi criado um cenário fictício de atualização em que as regras RA2, RA3 e RA5 alteram as ações de adaptação sobre suas *features* em tempo de execução. O novo estado das regras é apresentado na Tabela 7 e as que são atualizadas aparecem destacadas em cinza.

Outra diferença com relação à versão original do Phone Adapter é que o estado inicial teve suas *features* fixadas. Desse modo, o Phone Adapter não pode configurar as *features* atualmente ativas no dispositivo como sendo as corretas para o estado inicial. A última consideração é que a *feature Toque* é tida como ativa quando seu valor é maior que zero.

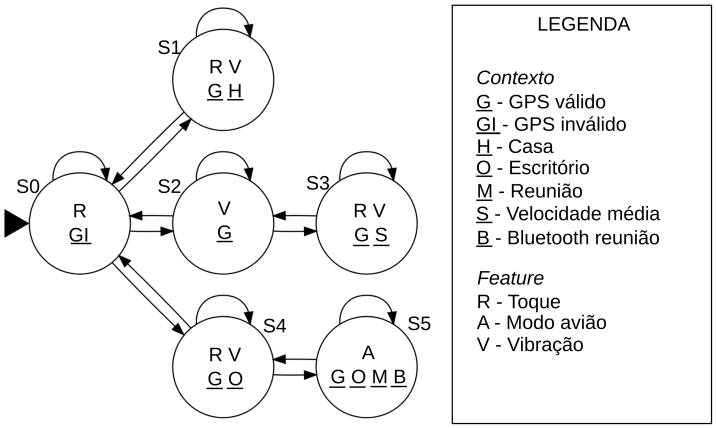
Tabela 7 – Regras de adaptação do Phone Adapter após a alteração.

Identificador da Regra	**Condição de guarda de contexto**	**Modo Avião**	**Toque**	**Vibração**
RA1	GPS = verdadeiro $\wedge$ Localização $\neq$ Casa $\wedge$ Localização $\neq$ Escritório	off	off	on
RA2	GPS = verdadeiro $\wedge$ Localização = Casa	off	on	off
RA3	GPS = verdadeiro $\wedge$ Localização = Escritório $\wedge$ Hora $\neq$ Reunião	off	off	on
RA4	GPS = falso $\wedge$ Localização $\neq$ Casa $\wedge$ Localização $\neq$ Escritório	off	on	off
RA5	GPS = verdadeiro $\wedge$ Velocidade $\geq$ 8,0	on	off	off
RA6	GPS = verdadeiro $\wedge$ Localização = Escritório $\wedge$ Hora = Reunião	on	off	off
RA7	GPS = verdadeiro $\wedge$ Localização = Escritório $\wedge$ Hora $\neq$ Reunião $\wedge$ bluetooth < 3	off	off	on
RA8	GPS = falso $\wedge$ Localização $\neq$ Casa $\wedge$ Localização $\neq$ Escritório	off	on	off
RA9	GPS = falso $\wedge$ Localização $\neq$ Casa $\wedge$ Localização $\neq$ Escritório	off	on	off
RA10	GPS = verdadeiro $\wedge$ Velocidade < 8,0	off	off	on

Fonte: elaborada pelo autor.

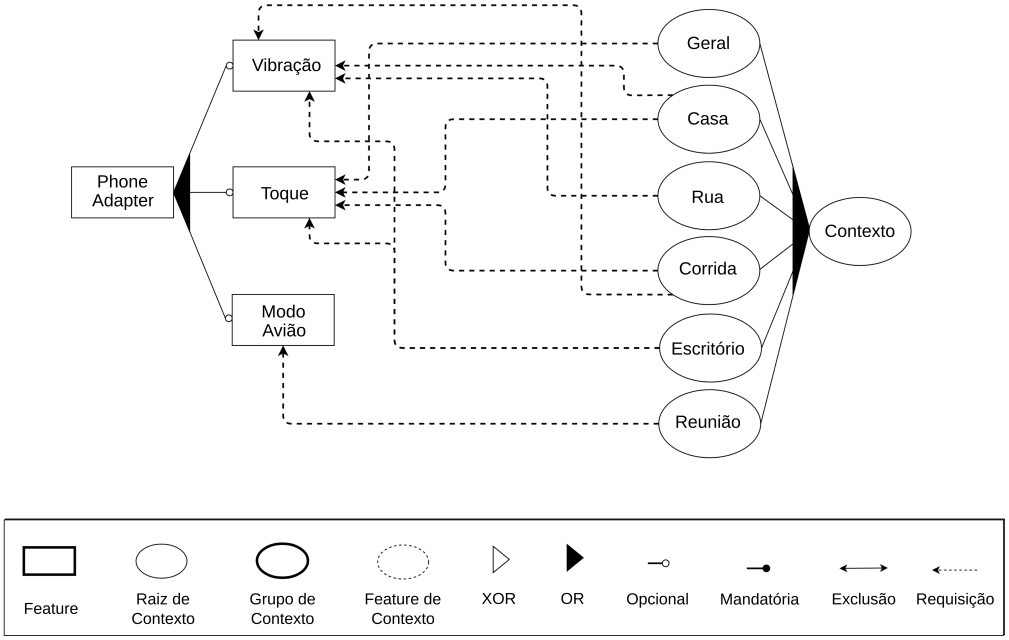
Com relação ao eCFM, as *features* de contexto foram modeladas de forma a representar os perfis da aplicação. Isso foi necessário porque a alta granularidade dos contextos fazia com que muitas *features* de contextos afetassem simultaneamente outras *features* do sistema. Dado que o oráculo de teste é gerado através do eCFM, caso uma *feature* seja requerida e excluída por *features* de contexto simultaneamente, podem ocorrer falsas detecções de falhas. Por exemplo, se

Figura 18 – Modelo DFTS do Phone Adapter



Fonte: elaborada pelo autor.

Figura 19 – Modelo eCFM do Phone Adapter



Fonte: elaborada pelo autor.

no modelo da Figura 4 a *feature* de contexto *Conexão* excluísse *Texto*, então haveria um conflito quando o contexto *Bateria Baixa* estivesse ativo. Nesse caso, os testes com essas duas *features* iriam falhar.

As Figuras 18 e 19 ilustram, respectivamente, os modelos DFTS e eCFM criados para a aplicação Phone Adapter. No que diz respeito ao DFTS, cada estado de contexto representa um determinado perfil da aplicação de forma que o estado do perfil padrão é o S0.

Ainda com relação ao eCFM, foi adicionada a restrição OR entre as *features Vibração*, *Toque* e *Modo Avião*. A versão original do Phone Adapter não possuía esse requisito, mas elas foram inseridas para verificar a capacidade da abordagem RETAkE em detectar falhas relacionadas às restrições de modelos de *features*.

5.2 Prova de Conceito

Esta seção apresenta uma prova de conceito relacionada a checagem de propriedades comportamentais em tempo de execução da abordagem RETAkE. Essa prova de conceito objetivou responder à seguinte questão: *é viável aplicar a abordagem RETAkE para identificar falhas de propriedades comportamentais em tempo de execução?*

Para isso, os DASs móveis GREat Tour (MARINHO *et al.*, 2013) e Phone Adapter (SAMA *et al.*, 2010) foram modelados e instrumentados utilizando a ferramenta CONTroL@Runtime apresentada na Seção 4.3.

As Subseções 5.2.1 e 5.2.2 apresentam a prova de conceito para as aplicações GREat Tour e Phone Adapter, respectivamente. A Subseção 5.2.3 discute os resultados obtidos. Por fim, a Subseção 5.2.4 discute as ameaças à validade.

5.2.1 GREat Tour

Para utilizar a CONTroL@Runtime, foram criados os modelos DFTS e eCFM apresentados nas Figuras 5 e 4, respectivamente. Também foram especificadas as regras de adaptação na ferramenta CONTroL@Runtime. Como essa aplicação segue uma sequência de atividades semelhante ao ciclo MAPE-K, poucos ajustes foram necessários para anotar o código-fonte para o monitoramento e controle das *features* e variáveis de contexto. Para simular o contexto do ambiente, foi gerada uma simulação de contexto randômica utilizando o DFTS.

Para verificar a capacidade de detecção de falhas das propriedades, faltas foram manualmente inseridas no código-fonte que controla a adaptação da aplicação. Primeiramente, o código-fonte foi inspecionado para selecionar expressões condicionais relacionadas às regras modeladas, e depois alteraram-se as expressões para violar as propriedades. A Tabela 8 sumariza as faltas inseridas no GREat Tour.

Tabela 8 – Descrição das faltas inseridas nas regras do GREat Tour

Identificador da Falta	**Descrição**	**Ação Executada**
F1	*Feature* login como mandatória	Modelar a *feature* login como mandatória e permitir que seja desabilitada pelas regras.
F2	Alteração de regras para afetarem as mesmas *features* simultaneamente	Alterar a condição de guarda de contexto da regra RA1 para ficar igual à RA5.
F3	Alteração de regra para causar falha	Trocar ativação por desativação na regra RA4.

Fonte: elaborada pelo autor.

A inserção da falta F1 da Tabela 8 fez com que a CONTroL@Runtime identificasse uma violação na propriedade *Configuration Correctness*. Essa falta está relacionada com a *feature Login*, uma vez que a mesma é obrigatória no modelo de *features*.

Já a F2 causou uma violação de *Interleaving Correctness* durante o processo de verificação. Isso aconteceu porque as regras RA1, e RA5 atuam sobre as *features Vídeo* e *Foto*, de forma que RA1 demanda o desativamento e RA5 demanda o ativamento. Para identificar essas falhas foi necessário determinar se havia regras ativas e não satisfeitas.

A última falta inserida foi a F3, que também foi identificada pela CONTroL@Runtime quando se manifestou como falha. Essa falta está relacionada com a violação individual de regras de adaptação, que também não satisfaz a propriedade de *Interleaving Correctness*. O código-fonte relativo à regra RA4 foi manualmente modificado para sempre desativar a *feature Vídeo* quando a regra era ativada. Isso levou à uma violação da especificação original da regra nos arquivos da CONTroL@Runtime. A violação é detectada sempre que a simulação passava por estados do DFTS que tinha o contexto *Carregando* = *verdadeiro*.

5.2.2 Phone Adapter

Mesmo que o Phone Adapter não tenha sido originalmente projetado para realizar adaptações em tempo de execução com base em modelos de *features*, optou-se por usá-lo para analisar a viabilidade da abordagem proposta com outros tipos de aplicações sensíveis ao contexto. A avaliação com o Phone Adapter foi focada na propriedade *Variation Liveness*. Ainda que o Phone Adapter não tivesse restrições de tempo que pudessem ser verificadas, a *feature Modo Avião* foi especificada com um tempo mínimo de desativação para o propósito desta avaliação. Essa restrição simula o cenário em que um celular não poderia passar muito tempo sem poder realizar chamadas de emergência.

Primeiro foram especificados o DFTS, eCFM e as regras de adaptação criadas no código-fonte (ver Subseção 5.1.2). Todas as *features* de contexto do eCFM que requerem uma *feature* excluem as demais. As exclusões foram removidas da figura para facilitar a legibilidade. Para instrumentar corretamente o Phone Adapter, seu código-fonte precisou ser modificado para criar variáveis que guardassem o estado das *features* e contextos na classe responsável por gerenciar as adaptações. Como o Phone Adapter foi implementado com um fluxo de atividades semelhante ao ciclo MAPE-K, outras modificações não foram necessárias.

Em seguida, foi executada uma simulação de contexto randômica e as falhas foram

injetadas da seguinte forma: definiu-se uma restrição de tempo de 999 milissegundos para que a *feature Modo Avião* permanecesse ativa. Dada a sequência de contexto gerada para este exemplo, foi identificada uma violação da propriedade, uma vez que a *Modo Avião* ficou ativada por mais de 999 milissegundos. Dado que a *feature Modo Avião* foi ativada, as transições de estados da simulação não permitiram que a regra de desativação da *feature* fosse avaliada no tempo especificado. Isso fez com que a propriedade de *Variation Liveness* fosse violada. Esse cenário pretende apenas imitar a passagem do tempo através de adaptações consecutivas, portanto, mais cenários precisam ser executados para reunir mais dados sobre a precisão da identificação.

5.2.3 Discussão dos Resultados

As Subseções 5.2.1 e 5.2.2 apresentaram uma prova de conceito para analisar a viabilidade da abordagem RETAkE em checar propriedades comportamentais para DAS.

Considerando que as faltas inseridas são relativamente simples, elas poderiam ser detectadas através de testes em tempo de projeto. Entretanto, levando em conta cenários de modelos grandes ou que possuem dinamismos nas regras, realizar a checagem dessas propriedades em tempo de execução auxilia na garantia da corretude das adaptações do sistema.

Em relação ao uso do DFTS, sua correta representação do DAS é uma suposição que deve ser mantida para a execução da simulação. Para isso, pode-se recorrer ao trabalho de Santos *et al.* (2016) para realizar a checagem do modelo antes da instrumentação do DAS. Mesmo assim, o DFTS pode não representar o comportamento real do DAS. Dependendo da implementação do sistema, a necessidade de especificar uma auto-transição no modelo pode tornar o DAS inconsistente. Por exemplo, a aplicação Phone Adapter entram em um ciclo de adaptações quando um perfil tem uma regra que leva ao mesmo perfil. A abordagem RETAkE poderia ser adaptada para usar outros modelos de variabilidade que representam o DAS, tal como discutido na seção de trabalhos futuros desta dissertação (Seção 6.4).

Em relação aos modelos, não houve a intenção de verificar se as transições eram realizadas corretamente entre estados consecutivos no DFTS, mas checar o estado atual do DAS. Por exemplo, a abordagem não identifica a transição do estado *S1* para *S3* na Figura 5 como uma falha, contanto que as *features* estejam corretamente ativas em *S3*. O foco deste trabalho é verificar o comportamento correto de adaptação conforme definido no modelo eCFM e as regras de adaptação. No entanto, a presente abordagem se beneficia de uma ferramenta baseada em anotações para instrumentar o código-fonte do DAS e monitorar variáveis importantes.

Como apresentado anteriormente, a abordagem foi bem sucedida em identificar todas as quatro faltas previamente inseridas. Nesse sentido, a técnica obteve êxito em cumprir seu objetivo de identificar violações nas propriedades comportamentais no estado atual. Mesmo que não faça uso de técnicas mais complexas como lógica linear temporal, ainda assim essa checagem de propriedades pode oferecer suporte quando os testes não estiverem sendo executados. Revisitando à questão apresentada na introdução da Seção 5.2 (*é viável aplicar a abordagem RETAkE para identificar falhas de propriedades comportamentais em tempo de execução?*), é possível afirmar que a técnica é viável para ser aplicada no cenário da avaliação.

5.2.4 Ameaças à Validade

A avaliação conduzida nesta prova de conceito teve ameaças à validade que serão discutidas de acordo com Wohlin *et al.* (2012). As ameaças podem ser classificadas em validade de conclusão, validade interna, validade de construção e validade externa (WOHLIN *et al.*, 2012). Para essa avaliação foram identificadas ameaças relativas à validade interna e externa.

A validade interna está relacionada com fatores não medidos ou incontroláveis, afetando os resultados observados durante o estudo (WOHLIN *et al.*, 2012). Dessa forma, uma ameaça à validade interna diz respeito à instrumentação do sistema. Por exemplo, o GREat Tour e o Phone Adapter precisaram ser alterados para facilitar a modelagem das *features* e a coleta de contexto. A principal alteração consistiu na criação de variáveis no código-fonte que armazenassem valores de contexto e estado de *features*.

Outra ameaça interna foi a criação dos modelos para as aplicações. Por exemplo, o Phone Adapter não possuía um modelo de *features* ou DFTS. Esses modelos foram criados em conformidade com as regras de adaptação da aplicação. Entretanto, para analisar as propriedades *Feature/Variation Liveness*, a aplicação foi modelada com restrições de tempo para uma de suas *features*.

As ameaças à validade externas dizem respeito a generalização dos resultados obtidos (WOHLIN *et al.*, 2012). O tamanho das aplicações é uma ameaça externa do presente trabalho. Como apresentado na Seção 5.1, as aplicações possuem regras de adaptação simples e em pequena quantidade, o que pode levar a cenários de falhas com pouca complexidade. Para mitigar isso, as aplicações foram modificadas para aumentar a complexidade de suas regras e criar um cenário fictício de adaptação de regras. Esse cenário permitiu a inserção de variadas faltas que poderiam ocorrer em tempo de execução.

5.3 Teste de Mutantes

Esta seção apresenta uma avaliação da abordagem RETAkE em relação à execução de testes em tempo de execução. Essa avaliação foi inspirada no trabalho de Santos (2017) que utilizou a técnica de teste de mutantes. A presente avaliação objetivou responder à seguinte questão: *a abordagem RETAkE é eficaz em detectar reconfigurações incorretas através de testes em tempo de execução?*

Para isso, foram geradas versões faltosas dos exemplos de DAS utilizando a técnica de teste mutantes (Subseção 5.3.1). Na sequência, os resultados são apresentados e discutidos (Subseções 5.3.2, 5.3.3 e 5.3.4). Por fim, são discutidas as ameaças à validade do estudo (Subseção 5.3.5).

5.3.1 Processo de Execução do Teste de Mutantes

A avaliação com teste de mutantes foi realizada utilizando uma versão da CONTroL@Runtime apenas com a funcionalidade de testes em tempo de execução. A CONTroL@Runtime foi configurada para iniciar os testes tão logo houvesse requisições para atualização de regras.

Como a abordagem RETAkE se concentrou na execução de testes, foi selecionada a técnica de teste de mutação (JIA; HARMAN, 2010) para analisar a sua eficácia. O teste de mutação consiste em gerar versões faltosas de um software através de alterações sintáticas no código-fonte (BOURQUE; FAIRLEY, 2014). Essas versões alteradas do código-fonte são chamadas de *mutantes* (JIA; HARMAN, 2010). Nesse tipo de técnica, deve-se executar a suíte de testes do software nas versões faltosas e caso a falha seja identificada, o *mutante* é considerado *morto* (BOURQUE; FAIRLEY, 2014). Por fim, deve-se calcular o escore de mutação, que é a proporção de mutantes mortos em relação ao total. A seguir são detalhados os passos de como a técnica foi utilizada para esta avaliação.

(I) Geração de Mutantes. A primeira fase do teste de mutação é a injeção de faltas, levando à criação das versões de código-fonte chamadas de mutantes. Assim, foi realizada a inspeção manual das regras de adaptação de cada exemplo de DAS e a execução de testes iniciais para verificar as adaptações. Em seguida, os mutantes foram manualmente criados através da aplicação de operadores de mutação na implementação original.

Como a RETAkE foca na reconfiguração de *features* pelo mecanismo de adaptação,

as regras de adaptação devem ser o alvo das mutações. Nesse sentido, o trabalho de Santos (2017) propõe operadores de mutação focados em modelos de *features* e regras de adaptação. Foram escolhidas especificamente as classes de operadores de mutação focados nas regras de adaptação: (i) baseados em contexto, que aplicam mudanças sintáticas nas condições de guarda de contexto; (ii) baseados em ação, que mudam as ações (ativar/desativar) das *features*.

Mesmo que os operadores definidos por Santos (2017) foquem na especificação das regras de adaptação, eles podem ser aplicados para criar mutantes no nível do código-fonte. Considerando a preocupação da abordagem RETAkE em testar a relação entre as condições de guarda de contexto e as *features*, optou-se por usar os operadores baseados em contexto e em ação, que estão resumidos na Tabela 9. Além desses operadores, também foram injetadas faltas de acordo com os dois cenários a seguir: (i) o código-fonte da regra se adapta, mas o sistema não envia a mensagem para atualizar o modelo de *features* e regras; (ii) o sistema envia a mensagem para atualizar o modelo de *features* e regras, mas não adapta suas regras de adaptação. Esses cenários representam faltas na manipulação de modelos pelo DAS (PÜSCHEL *et al.*, 2014b).

Tabela 9 – Operadores de mutação para regras de adaptação

Classe	**Identificador**	**Descrição**
Baseado em contexto	CtxINV	alterar a condição de guarda de contexto da regra para uma inexistente.
	CtxUNR	mudar a condição de guarda de contexto da regra para uma expressão que nunca é avaliada como verdadeira.
	CtxINT	alterar as condições de guarda de contexto das regras que têm diferentes ações sobre a mesma *feature* para que sejam avaliadas como verdadeiras ao mesmo tempo.
Baseado em ação	DelRl	remover a regra de adaptação.
	ActToDea	converter uma ação de ativação em uma ação de desativação.
	DeaToAct	converter uma ação de desativação em uma ação de ativação.
	AddRl	adição de ativação/desativação de *feature* na regra de adaptação.

Fonte: Santos (2017)

(II) Execução de Testes. Após a geração dos mutantes, iniciaram-se os testes de tempo de execução dos mutantes utilizando a ferramenta CONTroL@Runtime. Para isso, os exemplos de DAS foram inicializados em um dispositivo móvel real executando o sistema operacional Android. Foi utilizada uma simulação de ambiente inicial da aplicação e para estimular apenas o primeiro ciclo de reconfigurações. A simulação inicia sempre no estado S0 do modelo DFTS e, após a primeira adaptação, as regras se alteram automaticamente sem que seja realizada uma análise do contexto.

Cada mutante foi executado 30 vezes e, depois que uma falha é identificada, a

execução de testes é finalizada. Os mutantes foram executados mais de uma vez por conta da randomicidade associada à geração dos testes, de forma que cada execução poderia gerar sequências diferentes que não detectassem as falhas. Sendo assim, as execuções múltiplas permitiram obter as médias das sequências de testes que detectaram as falhas.

Após a execução, foi calculado o escore de mutação: a proporção de mutantes identificados sobre o número total de mutantes não equivalentes (JIA; HARMAN, 2010).

(III) Análise de Mutantes. A terceira fase consistiu na análise dos resultados dos testes para determinar sua eficácia. O primeiro passo nesta fase foi identificar os chamados Mutantes Equivalentes, que são as versões sintaticamente diferentes do código-fonte, mas que tem o mesmo comportamento da versão original (JIA; HARMAN, 2010). Depois disso, o cálculo do escore de mutação é refeito para subtrair o número de mutantes equivalentes do total de mutantes criados.

Ainda relacionado à atividade de análise dos resultados, também foram coletados dados relativos aos tamanhos das sequências executadas. Esses dados não fazem parte da avaliação de mutantes, mas o impacto nos tamanhos das sequências de testes será brevemente discutido na Seção 5.4.

5.3.2 Resultados para o GREat Tour

Os mutantes do GREat Tour foram gerados através da aplicação dos operadores nas três regras que adaptam sua definição em tempo de execução (i.e., RA1, RA5, RA6), porém, não foi possível aplicar os operadores ActToDea e DeaToAct nas regras RA1 e RA6, respectivamente. Ao final do processo, obteve-se 25 mutantes do código-fonte. Com relação à sequência de testes, foi configurado um tamanho cinco vezes maior ao número de estados do DFTS, totalizando um tamanho igual à 50. Esse valor foi escolhido por conta do indeterminismo da geração de testes utilizada, então sequências pequenas poderiam não explorar o DFTS o suficiente para encontrar as falhas.

Para identificar os mutantes equivalentes, realizou-se uma análise manual das execuções que não detectaram falhas. A Tabela 10 resume as ações executadas para gerar os mutantes, a quantidade e o percentual de execuções que detectaram cada mutante. Considerando apenas os 17 mutantes identificados em todas as 30 execuções e sem análise de equivalentes, seria possível obter um escore de mutação igual à 0,68.

A análise dos mutantes não identificados revelou que todos os seis eram mutantes

Tabela 10 – Ação executada para gerar mutante e resultados de detecção para o GREat Tour

Identificador da Regra	**Identificador do Operador**	**Ação executada para gerar mutante**	**Nº de sequências falhas**	**Percentual de sequências falhas**
RA1	DelRl	Remoção da própria regra de adaptação	30	100%
	DeaToAct	Converter a desativação da *feature* Vídeo em ativação	30	100%
	AddRl	Adicionar a desativação da *feature* Texto	0	0%
	CtxINV	Mudar a condição de guarda de contexto do nível de bateria para < 0%	30	100%
	CtxUNR	Mudar a condição de guarda de contexto para carregando == verdadeiro && não carregando == verdadeiro	30	100%
	CtxINT	Alterar a condição de guarda de contexto para incluir carregando == verdadeiro	30	100%
	Falhar em adaptar o código	Remover a adaptação das condições de guarda de contexto	30	100%
	Falhar em adaptar o modelo	Remover a solicitação de atualização do modelo	0	0%
RA5	DelRl	Remoção da própria regra de adaptação	0	0%
	ActToDea	Converter a ativação da *feature* Foto em desativação	30	100%
	DeaToAct	Converter a desativação da *feature* Texto em ativação	0	0%
	AddRl	Adicionar a ativação da *feature* Login	29	96,67%
	CtxINV	Mudar o estado da condição de guarda de contexto do nível da bateria para > 120%	0	0%
	CtxUNR	Alterar a condição de guarda de contexto para carregando == verdadeiro && nível de bateria > 70%	30	100%
	CtxINT	Mudar a condição de guarda de contexto para carregando == falso && nível da bateria <= 70%	30	100%
	Falhar em adaptar o código	Remover a adaptação das condições de guarda de contexto	30	100%
	Falhar em adaptar o modelo	Remover a requisição de atualização do modelo	0	0%
RA6	DelRl	Remoção da própria regra de adaptação	30	100%
	ActToDea	Converter a ativação da *feature* Texto em desativação	30	100%
	AddRl	Adicionar a ativação da *feature* Vídeo	30	100%
	CtxINV	Mudar a condição de guarda de contexto do nível da bateria para < 0%	30	100%
	CtxUNR	Mudar a condição de guarda de contexto para nível da bateria alta && nível da bateria baixa	30	100%
	CtxINT	Mudar a condição de guarda de contexto para carregamento == falso && nível da bateria > 70%	30	100%
	Falhar em adaptar o código	Remover a adaptação das condições de guarda de contexto	30	100%
	Falhar em adaptar o modelo	Remover a requisição de atualização do modelo	30	100%

Fonte: elaborada pelo autor.

equivalentes. Por exemplo, um dos equivalentes ocorreu por conta da combinação entre a regra RA1 e o operador AddRlRA. Esse operador adicionou uma ação para desativar a *feature Texto*, mas o mutante gerado foi equivalente porque a aplicação avaliava RA1 antes das demais, levando a *feature Texto* para o estado correto. Outro equivalente foi a falha da regra RA1 em informar as alterações do modelo eCFM, que foi obtido removendo o código-fonte necessário para a operação. O mutante é equivalente porque a regra RA5 também realiza as mesmas operações de alteração, levando o DAS para um estado consistente após a atualização das regras. Essa explicação também se aplica ao mutante que falha em alterar o modelo na regra RA5. A maioria dos outros mutantes são relacionados com a *feature Texto* por conta da regra RA7, sendo esta avaliada após todas as outras.

Dos 25 mutantes gerados, 17 foram detectados em todas as 30 execuções e sete eram equivalentes. Esses valores permitem obter um escore de mutação igual à 0,95. Entretanto, o único mutante não identificado em todas as execuções foi detectado em 29 sequências de teste. Considerando os mutantes identificados por pelo menos uma execução, então o escore de mutação final será de 1. Uma das razões para esse resultado de detecção está no tipo de atualização das regras: um ajuste de um nível na bateria que ativa/desativa as *features*. Sendo assim, a maior parte dos mutantes se manifesta em estados muito próximos ao estado inicial *S0* do modelo.

5.3.3 Resultados para o Phone Adapter

Os operadores de mutação foram aplicados no Phone Adapter com foco em um subconjunto das regras que se atualizavam. Dos 27 operadores originalmente planejados, foi possível realizar a aplicação de 22, obtendo assim o mesmo número de versões mutantes do código-fonte. A simulação para o Phone Adapter foi iniciada no estado S0.

Na Tabela 11 é possível verificar os resultados para a execução dos testes, as ações realizadas para gerar os mutantes, o número e o percentual de sequências de teste que identificaram os mutantes. Dentre as 30 execuções realizadas para o Phone Adapter, cinco mutantes foram identificados em todas as execuções.

Após a análise dos mutantes equivalentes, foi identificado que todos os mutantes não identificados na regra RA2 eram equivalentes. O comportamento do mutante é equivalente ao original porque os estados S0 e S1 possuem as mesmas *features* após a atualização das regras. Logo, sempre que o estado S1 não pode ser alcançado por conta de alguma falta nas

Tabela 11 – Ação executada para gerar os mutantes e resultados de detecção para o Phone Adapter

Identificador da Regra	**Identificador do Operador**	**Ação executada para gerar os mutantes**	**Nº de sequências falhas**	**Percentual de sequências falhas**
RA2	DelRl	Remoção da própria regra após a adaptação	0	0,00%
	ActToDea	Converter a ativação da *feature Toque* em desativação	28	93,33%
	DeaToAct	Converter a desativação da *feature Modo Avião* em ativação	30	100,00%
	CtxINV	Mudar a condição de guarda de contexto para *Velocidade* <0,0	0	0,00%
	CtxUNR	Mudar a condição de guarda de contexto para GPS = falso	0	0,00%
	CtxINT	Mudar a condição de guarda de contexto para ficar igual a da regra RA1	25	83,33%
	Falhar em adaptar o código	Remover a adaptação das condições de guarda de contexto	29	96,67%
	Falhar em adaptar o modelo	Remover a solicitação de atualização do modelo	28	93,33%
RA5	DelRl	Remoção da própria regra após a adaptação	23	76,67%
	DeaToAct	Converter a desativação da *feature Vibração* para ativação	22	73,33%
	ActToDea	Converter a ativação da *feature Modo Avião* para desativação	26	86,67%
	CtxINV	Mudar a condição de guarda de contexto de *Localização* para ficar igual a regra R2	24	80,00%
	CtxUNR	Mudar a condição de guarda de contexto para *Localização*=Casa e Velocidade=20,0	27	90,00%
	Falhar em adaptar o código	Remover a adaptação das condições de guarda de contexto	23	76,67%
	Falhar em adaptar o modelo	Remover a solicitação de atualização do modelo	21	70,00%
RA7	DelRl	Remoção da própria regra após a adaptação	28	93,33%
	ActToDea	Mudar a ativação da *feature Vibração* para desativação	30	100,00%
	DeaToAct	Mudar a desativação da *feature Toque* para ativação	30	100,00%
	CtxINV	Mudar a condição de guarda de contexto para *Hora* = 25:00	28	93,33%
	CtxUNR	Mudar a condição de guarda de contexto para *Localização* = Escritório e GPS = falso	27	90,00%
	Falhar em adaptar o código	Remover a adaptação das condições de guarda de contexto	30	100,00%
	Falhar em adaptar o modelo	Remover a solicitação de atualização do modelo	30	100,00%

Fonte: elaborada pelo autor.

regras, as mesmas *features* são ativadas e o restante da sequência de teste pode ser executada normalmente. Com isso, a análise dos mutantes permitiu identificar que três dos mutantes gerados eram equivalentes.

Calculando-se o escore de mutação em que os mutantes foram identificados em todas as 30 execuções, é possível obter um valor de 0,26. Ainda assim, a CONTroL@Runtime obteve êxito em identificar todos os mutantes em 20 execuções ou mais. Os outros mutantes não foram identificados em todas as execuções porque as sequências não passavam pelas transições com a regra defeituosa. Considerando os mutantes identificados em pelos menos uma das execuções, a abordagem foi capaz de obter um escore de mutação igual à 1.

5.3.4 Discussão dos Resultados

A partir dos resultados apresentados nas Subseções 5.3.2 e 5.3.3 é possível verificar que o GREat Tour obteve resultados melhores quando comparado ao Phone Adapter. Um fator que pode ter impactado nos resultados foi o fato do algoritmo permitir repetir estados já visitados anteriormente. Isso pode ter travado a exploração em sucessivas adaptações de poucos estados de contexto.

No cálculo dos escores de mutação foram considerados dois tipos de contagem para os mutantes detectados: (i) um quando todas as execuções detectavam e (ii) outro quando pelos menos uma sequência de teste detectava. Para o primeiro tipo de contagem o GREat Tour obteve 0,95, e o Phone Adapter, 0,26. Logo, o indeterminismo do processo de geração de testes pode impactar negativamente na geração das sequências. Ainda assim, considerando os casos em que ao menos uma sequência de teste identificou as falhas, todos os mutantes foram detectados pela abordagem.

Com relação aos tamanhos de sequências de teste, foram escolhidos valores iguais a cinco vezes o total de estados do DFTS de cada DAS. Sequências desse tamanho seriam capazes de visitar todos os estados do modelo, porém optou-se por não realizar explorações exaustivas. Vale ressaltar que na atual versão da abordagem o tamanho de sequência de testes precisa ser manualmente definido pelo engenheiro de software.

Considerando os resultados anteriormente obtidos, a CONTroL@Runtime pode ser configurada para executar testes logo após a adaptação do DAS. Dessa forma, é possível testar um conjunto limitado de configurações partindo de diferentes estados de contexto.

Vale ressaltar que a abordagem é focada na detecção de falhas de adaptação, sendo

necessário que as faltas presentes no código-fonte sejam executadas e se manifestem como comportamento incorreto. A CONTroL@Runtime indica quais os estados de contexto ocasionaram a falha, mas não é capaz de determinar quais regras estão incorretas. Por exemplo, se a regra RA1 do GREat Tour foi implementada como menor 0% (quando deveria ser menor ou igual à 30%), então a falha é detectada no estado S7 (ver Figura 5). Entretanto, a ferramenta não indica que a causa foi a regra RA1.

Revisitando a questão que guiou a condução da avaliação (*a abordagem RETAkE é eficaz em detectar reconfigurações incorretas através de testes em tempo de execução?*), é possível afirmar que a abordagem é eficaz porque todas faltas inseridas foram encontradas quando elas se manifestaram como falhas. Porém, o caráter randômico das sequências de testes pode não revelar a falta tão logo elas sejam inseridas no DAS. Isso demonstra a necessidade de executar testes em outros momentos do fluxo de execução do DAS (e.g., após adaptações de *features*).

5.3.5 Ameaças à Validade

Durante a condução desta avaliação foram identificadas ameaças relacionadas à validade interna e externa (WOHLIN *et al.*, 2012). Essas ameaças são discutidas a seguir.

Uma ameaça à validade interna neste trabalho é a implementação do algoritmo que gerou as sequências de teste. Para garantir seu correto funcionamento, foram executados testes unitários.

Outra ameaça à validade interna são os mutantes e a forma como eles foram inseridos no DAS. Dado que não foi encontrado na literatura trabalhos que criem mutantes para regras de adaptação automaticamente, os mutantes foram criados manualmente. Para reduzir o viés durante a criação, tentou-se criar as faltas de forma aleatória e considerando o escopo individual de cada regra.

A última ameaça à validade interna é a instrumentação dos exemplos de DAS, de forma que este pode ter afetado os resultados da avaliação de mutantes e de propriedades. Para garantir o correto funcionamento da CONTroL@Runtime, foram executados diversos testes em conjunto dos exemplos de DAS, tanto em versões corretas como faltosas dos mesmos. Adicionalmente, a CONTroL@Runtime foi modificada para permitir a recuperação do seu estado após a execução de testes. Essa ação foi necessária principalmente para o Phone Adapter, já que o mesmo não é capaz de se recuperar após entrar em um estado não condizente com o contexto

do ambiente.

A principal ameaça à validade externa é o uso de duas aplicações acadêmicas que foram manualmente modificadas para mimetizar cenários de adaptação de regras. Para reduzir o viés da modificação, as aplicações foram alteradas de forma a permanecer o mais semelhante possível ao seu projeto original. Ainda assim, os mecanismos de adaptação não realizam nenhum tipo de análise para atualizar as regras.

Outra ameaça relacionada à validade externa é a escolha dos modelos da abordagem, que podem causar problemas de representatividade dependendo do tipo de aplicação selecionada. O uso do eCFM com muitas *features* de contexto afetando as mesmas *features* pode fazer com que a abordagem identifique erroneamente falhas de adaptação. Para mitigar isso, as *features* de contexto foram modeladas utilizando representações em mais alto-nível quando necessário.

5.4 Impacto da Abordagem

Teste em tempo de execução pode ser uma operação computacionalmente onerosa, requirindo recursos de hardware e impactando no comportamento adaptativo do DAS (FREDERICKS; CHENG, 2015). Por conta disso, trabalhos na literatura tem apresentado soluções cientes de recursos para testes em tempo de execução (LAHAMI; KRICHEN; JMAIEL, 2016). Mesmo que a abordagem RETAkE não tenha sido projetada para lidar com recursos limitados, é relevante avaliar o impacto das suas técnicas de verificação no sistema. Nesse sentido, a presente avaliação teve o objetivo de responder a seguinte questão: *as atividades em tempo de execução da abordagem RETAkE impactam no tempo de execução do DAS?*

A Subseção 5.4.1 apresenta uma discussão sobre os tamanhos de sequências de testes executadas na avaliação de mutantes. A Subseção 5.4.2 detalha uma avaliação sobre o impacto no tempo de adaptação com diferentes configurações da ferramenta proposta. A Subseção 5.4.3 discute os resultados da avaliação de impacto. Por fim, a Subseção 5.4.4 apresenta as ameaças à validade da avaliação apresentada no capítulo.

5.4.1 Tamanhos de Sequências de Testes Executadas

Esta subseção apresenta dados adicionais sobre o teste de mutantes da Seção 5.3. A seguir são discutidos os tamanhos de sequências relativos às avaliações com o GREat Tour e Phone Adapter, respectivamente.

Tabela 12 – Resultados de tamanhos de sequências de testes para o GREat Tour

Identificador da Regra	Identificador do Operador	Média	Tamanho Máximo	Tamanho Mínimo	Desvio Padrão
RA1	DelRl	2,43	9	1	1,65
	DeaToAct	3,17	15	1	3
	AddRl	50	50	50	0
	CtxINV	2,63	6	1	1,45
	CtxUNR	3,10	16	1	2,96
	CtxINT	3,23	12	1	2,75
	Falha em adaptar o código	3,17	10	1	2,38
	Falha em adaptar o modelo	50	50	50	0
RA5	DelRl	50	50	50	0
	ActToDea	6,07	26	1	5,73
	DeaToAct	50	50	50	0
	AddRl	11,03	50	2	12,34
	CtxINV	50	50	50	0
	CtxUNR	50	50	50	0
	CtxINT	2,80	8	1	1,97
	Falha em adaptar o código	2,90	7	1	1,81
	Falha em adaptar o modelo	50	50	50	0
RA6	DelRl	1,10	3	1	0,40
	ActToDea	1,07	2	1	0,25
	AddRl	3,43	13	1	2,76
	CtxINV	1,13	2	1	0,35
	CtxUNR	1,20	3	1	0,48
	CtxINT	1,07	2	1	0,25
	Falha em adaptar o código	1,10	2	1	0,30
	Falha em adaptar o modelo	1,10	2	1	0,30

Fonte: elaborada pelo autor.

A Tabela 12 apresenta os dados de média, desvio padrão, tamanho máximo e mínimo relativos aos tamanhos de sequências no teste de mutantes do GREat Tour. Com exceção dos valores dos mutantes equivalentes, cujas falhas nunca são detectadas, a maior média obtida foi igual a 11,03 casos de teste para o operador AddRl. O mutante identificado mais rapidamente precisou de uma sequência de testes de tamanho 1, e o que foi identificado mais lentamente precisou de uma sequência de tamanho 26. A variação dos tamanhos das sequências pode estar

relacionada com duas características da configuração da avaliação: a randomicidade tanto do estado inicial de simulação quanto do próprio processo de geração das sequências de teste.

No que se refere as informações relativas aos tamanhos de sequências de testes, pode-se verificar na Tabela 13 que a maior média foi de 21 para o operador DeaToAct da regra RA5. Os tamanhos mínimos e máximos foram, respectivamente, 1 e 30.

Tabela 13 – Resultados de tamanhos de sequências de testes para o Phone Adapter

Identificador da Regra	Identificador do Operador	Média	Tamanho Máximo	Tamanho Mínimo	Desvio Padrão
RA2	DelRl	30	30	1	0
	ActToDea	8,33	19	1	9,22
	DeaToAct	5,2	30	1	6,86
	CtxINV	30	30	1	0
	CtxUNR	30	30	1	0
	CtxINT	14,83	30	1	10,37
	Falha em adaptar o código	6,9	21	1	7,59
RA5	Falha em adaptar o modelo	8,33	21	1	8,91
	DelRl	17,4	30	2	10,99
	DeaToAct	17,6	30	2	10,65
	ActToDea	13,87	30	2	9,61
	CtxINV	15,13	30	2	10,93
	CtxUNR	13,13	30	2	9,35
	Falha em adaptar o código	16,87	30	2	11,21
	Falha em adaptar o modelo	15,13	30	2	10,68
RA6	DelRl	10,8	30	3	8,07
	ActToDea	4,53	17	1	4,06
	DeaToAct	4,2	15	1	5,67
	CtxINV	12,27	30	3	10,02
	CtxUNR	10,57	30	3	8,6
	Falha em adaptar o código	4,4	23	1	4,11
	Falha em adaptar o modelo	4,73	21	1	5,7

Fonte: elaborada pelo autor.

A partir dos resultados das Tabela 12 é possível verificar que a maioria das sequências de testes precisou de um à três testes para identificar as falhas no GREeat Tour. Como explicado

anteriormente na Subseção 5.3.4, a razão para valores pequenos no tamanho da sequência é que as falhas ocorrem muito próximas ao estado inicial da simulação. Os valores de média, tamanho máximo e mínimo iguais à 50 na Tabela 12 são todos relativos aos mutantes equivalentes, configurando os maiores valores por que toda a sequência foi executada.

Com relação à aplicação Phone Adapter, a Tabela 13 indica que os tamanhos de sequência de teste tiveram, em geral, médias maiores do que o GREat Tour. Desconsiderando três mutantes para a regra RA2, que são todos equivalentes, a maior e menor média foram 4,2 e 17,6, respectivamente. A razão para esses valores mais variados está na relação entre as regras e o DFTS: os estados de contexto que ativam as regras que se atualizam estão em direções mais espalhados pelo grafo. Isso pode ter contribuído para que as sequências necessitassem de mais estados antes de detectar as falhas.

5.4.2 Impacto no Tempo de Adaptação

Com o objetivo de analisar o impacto causado pela RETAkE, foi realizada uma medição do tempo necessário pelas aplicações para finalizarem uma execução simulada. Seguindo uma direção similar a de outros trabalhos da área de testes em tempo de execução (FREDERICKS; CHENG, 2015; LAHAMI; KRICHEN; JMAIEL, 2016), foram coletados dados relativos ao uso da ferramenta CONTroL@Runtime operando em diferentes configurações. Para isso, a avaliação conduzida em (FREDERICKS; CHENG, 2015) foi utilizada como base, em que se mediu o tempo de execução do DAS em uma simulação com variadas configurações da solução. Ao fim, foram definidas hipóteses semelhantes para determinar se a introdução da abordagem afetava o tempo de execução.

Esta avaliação foi conduzida utilizando uma versão completa da CONTroL@Runtime, compreendendo os módulos de checagem de propriedades, testes em tempo de execução e gerenciador de atividades. Para esse fim, a avaliação considerou o seguinte cenário: as propriedades comportamentais são checadas após cada adaptação, então os testes sempre iniciam sem considerar nenhuma condição de início (ver Subseção 4.2.3). A primeira configuração determina a ausência total da CONTroL@Runtime. As demais configurações representam a introdução de diferentes configurações de checagem. A Tabela 14 sumariza as diferentes combinações utilizadas que são listadas a seguir.

Dado que na configuração *1* não havia a presença da implementação da RETAkE, foram adicionadas sequências com o DFTS diretamente no código-fonte da aplicação. Nas

Tabela 14 – Configurações para a avaliação de impacto na adaptação

Identificador da Configuração	**Descrição**
1	Execução com ausência da solução.
2	Checagem de propriedades ativada.
3	Checagem de propriedades e testes ativados com sequências de tamanho 10.
4	Checagem de propriedades e testes ativados com sequências de tamanho 30.
5	Checagem de propriedades e testes ativados com sequências de tamanho 50.

Fonte: elaborada pelo autor.

demais configurações, a ferramenta foi utilizada para exercitar adaptações no DAS em execuções diferentes daquelas realizadas nos testes de mutantes. Dessa forma, a CONTroL@Runtime esteve presente nas configurações 2-5 da Tabela 14, seja controlando contexto ou monitorando o estado das *features*.

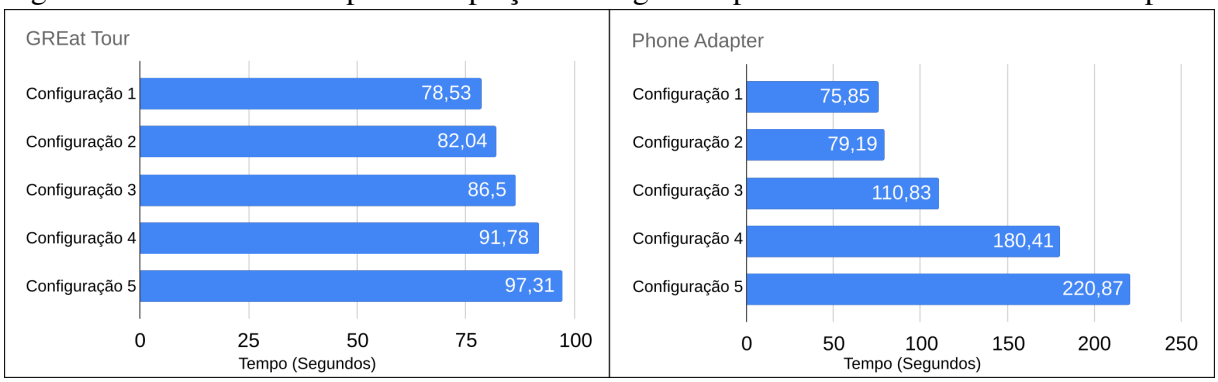
Para avaliar o tempo total de adaptação, foram geradas sequências de contextos de 25 estados do DFTS para o GREat Tour e outra para o Phone Adapter. Cada sequência de contexto foi executada 30 vezes para as cinco configurações listadas acima.

A medição dos tempos foi iniciada na partida da execução do DAS e finalizada no último item da sequência de contexto. Levando em conta que a abordagem bloqueia as adaptações do DAS enquanto os testes são executados, todas as atividades a seguir foram contidas em um único intervalo de adaptação: coleta de contexto, análise de regras, checagem de propriedades e execução de testes.

Para habilitar a execução da solução completa, foi necessário alterar a lógica de início das adaptações. O Phone Adapter possuía um serviço em segundo plano que executava indefinidamente. Por outro lado, o GREat Tour utilizava *threads* com execução postergada. Após uma refatoração, ambas as aplicações passaram a ter interfaces bem definidas para disparar as adaptações. Durante a execução dos testes em tempo de execução, o atraso para iniciar uma nova adaptação foi reduzido para 5 e 75 milissegundos no GREat Tour e Phone Adapter, respectivamente. O atraso mais alto no Phone Adapter foi necessário por conta de sua maneira assíncrona de iniciar os serviços de contexto e adaptação.

As simulações foram executados através de um computador pessoal executando o sistema operacional Ubuntu Linux 18.04, CPU Intel(R) Core(TM) i5-5200U 2.20GHz, 8GB de RAM. Ambas as aplicações foram executadas em dispositivos móveis reais com sistema operacional Android 7.1.1, CPU Quad Core 1.2 GHz e 2GB de RAM.

Figura 20 – Médias do tempo de adaptação em segundos para o GREat Tour e Phone Adapter



Fonte: elaborada pelo autor.

A Figura 20 apresenta a média dos tempos em segundos para o GREat Tour e Phone Adapter com as Configurações 1-5. Já a Tabela 15 apresenta as médias e o desvio padrão para cada configuração. A abordagem introduziu uma sobrecarga relativamente baixa até a configuração 2. Como esperado, o impacto no tempo de execução das aplicações aumentou para sequências de testes maiores. Por exemplo, a média de tempo de execução para os tamanhos das sequências de teste de mutação para o GREat Tour e Phone Adapter foram 97,31 e 180,41 segundos, respectivamente.

É possível verificar que o tempo de adaptação com execução de testes foi maior para o Phone Adapter quando comparado com o GREat Tour. Isso aconteceu porque o Phone Adapter precisa de um tempo de atraso maior durante a execução dos testes, utilizando esse tempo como intervalo para controlar o mecanismo de adaptação em segundo plano.

Tabela 15 – Médias e desvio padrão dos tempos de adaptação em segundos para o GREat Tour e Phone Adapter.

	Configuração	1	2	3	4	5
GREat Tour	Média	78,53	82,04	86,5	91,78	97,31
	Desvio Padrão	0,01	0,03	0,06	0,13	0,25
Phone Adapter	Média	75,85	79,19	110,83	180,41	220,87
	Desvio Padrão	0,03	0,03	0,64	7,62	0,91

Fonte: elaborada pelo autor.

A introdução da ferramenta CONTroL@Runtime a partir da configuração 2 foi suficiente para aumentar o tempo de execução dos DASs com as sequências de 25 estados. Para analisar a significância da diferença de tempo de execução entre as configurações, foi utilizado um teste estatístico para comparar as configurações 2 à 5 com a configuração 1. Dessa forma, as hipóteses nula e alternativa do teste foram declaradas como seguem.

a) *H0: Não existe diferença no tempo de execução do DAS após a inserção da RETAkE.*

b) *H1: Existe diferença no tempo de execução do DAS após a inserção da RETAkE.*

Para realizar um teste de hipótese, o primeiro passo consiste na checagem da normalidade dos dados (WOHLIN *et al.*, 2012). Para isso, foi utilizada a função *shapiro* da biblioteca SciPy[1] para a linguagem Python. A execução dessa função utiliza o método Shapiro-Wilk cujo resultado com alfa maior ou igual à 0,05 demonstrou que nem todas as amostras seguiam a distribuição normal.

Levando em conta que a normalidade de todas as amostras não era uma suposição válida, foi aplicado um teste estatístico não paramétrico. Seguindo a recomendação de Wohlin *et al.* (2012), foi selecionado o teste de Wilcoxon para amostras pareadas, também disponibilizado na biblioteca Scipy. Manteve-se o alfa igual à 0,05 na execução dos testes de hipótese. Entretanto, esse teste demanda que as amostras sejam simetricamente distribuídas em torno da mediana. Suposição essa que foi desrespeitada pelo par de configuração [1-4] da aplicação Phone Adapter. Dessa forma, foi utilizado o Sign Test como alternativa ao teste de Wilcoxon (WOHLIN *et al.*, 2012), sendo esse aplicado através da ferramenta SPSS[2].

O teste de hipótese foi executado para as duas aplicações e considerando os seguintes pares de configurações: [1-2], [1-3], [1-4] e [1-5]. Para todos os pares de configuração do GREat Tour e os pares [1-2], [1-3] e [1-5] do Phone Adapter, o Valor-P obtido do Wilcoxon foi de 1.7344^{-06}. No caso do par de configuração [1-4] foi o obtido o Valor-P de 0,00 para o Sign Test. Como em todos os casos o Valor-P obtido foi menor do que o valor de alfa de 0,05, é possível rejeitar a hipótese nula *H0* de forma que existe diferença no tempo de execução após a inserção da RETAkE nos dois DASs.

5.4.3 Discussão dos Resultados

A Subseção 5.4.2 apresentou os resultados relativos ao impacto no tempo de adaptação. A partir dos testes de hipótese realizados com os dados coletados, foi possível averiguar que a execução de testes impacta no tempo de execução das aplicações. A Figura 20 ilustra o aumento no tempo de execução de acordo com o aumento das sequências de testes.

Ainda que os objetivos da RETAkE sejam diferentes da solução apresentada por Fredericks e Cheng (2015), é possível fazer um paralelo com os resultados desse trabalho. Como

[1] https://www.scipy.org/

[2] https://www.ibm.com/analytics/spss-statistics-software

apresentado na Seção 3.3, o trabalho de Fredericks e Cheng (2015) utiliza uma técnica de otimização para adaptar os oráculos de "teste passivos" (desconsideram a inserção de dados de teste). Os autores concluíram que a execução de testes gerava impacto significativo. A partir da avaliação conduzida no presente trabalho, é possível concluir que outros tipos de atividades de teste em tempo de execução também impactam significativamente no tempo de adaptação de DAS.

A inserção da ferramenta CONTroL@Runtime no Phone Adapter teve um aumento considerável durante a execução dos testes. Uma das razões para isso é a comunicação assíncrona entre os serviços de contexto e adaptação do Phone Adapter, bloqueando o sistema por extensos períodos entre os ciclos de adaptação. O Phone Adapter precisou de lógica adicional para identificar e recuperar o seu estado de contexto após as perturbações causadas pelos testes. Ainda assim, o bloqueio das adaptações precisou ser feito para impedir que as ações afetassem as funcionalidades das aplicações durante os testes. Caso não fosse utilizado, o usuário veria as *features* sendo habilitadas/desabilitadas durante seu uso.

Ainda sobre a instrumentação dos DASs, foi identificada a necessidade de configurar corretamente o tempo de atraso do ciclo de adaptação nos testes. Caso o tempo de atraso não seja configurado corretamente, podem ocorrer falsos positivos na detecção. Por exemplo, o tempo de atraso configurado no Phone Adapter foi de 75 milissegundos. Se um tempo menor for configurado, o mecanismo de adaptação permanece no mesmo estado de contexto e o caso de teste falha.

A resposta para a questão apresentada na introdução do capítulo (*as atividades em tempo de execução da abordagem RETAkE impactam no tempo de execução do DAS?*) é que a RETAkE aumenta o tempo de execução do DAS. A partir dos resultados do Phone Adapter é possível concluir que o impacto de adaptação não depende apenas das atividades da abordagem, mas também da complexidade da adaptação do DAS. Quanto maior a quantidade de atividades executadas na adaptação (e.g., avaliação de regras e comunicação entre serviços), maior será o impacto dos testes executados.

5.4.4 Ameaças à Validade

A presente Subseção irá discutir as ameaças à validade (WOHLIN *et al.*, 2012) da avaliação conduzida ao longo da Seção 5.3. Foram identificadas ameaças relativas à validade interna e externa.

Similar às avaliações anteriores (ver Seção 5.2 e Seção 5.3.5), a presente avaliação também teve a ameaça interna relacionada à instrumentação dos dois DAS. Para que a CONTroL@Runtime controlasse as adaptações desses DASs foi preciso alterar suas lógicas de adaptação. Essas modificações podem ter contribuído para a alteração do tempo total de adaptação. As alterações são necessárias por conta da inviabilidade de se ter uma ferramenta genérica o suficiente para todos os tipos de sistemas. A mitigação para essa ameaça foi modificar as aplicações de forma a manter seu comportamento mais próximo possível do original.

Outra ameaça interna é o caráter randômico das sequências de teste que podem ter influenciado a avaliação. A ferramenta gerou diferentes combinações de sequências de testes para cada estado de contexto e, portanto, diferentes regras de adaptação foram ativadas. Para obter dados relativos a essas diferentes combinações, cada configuração apresentada na avaliação foi executada 30 vezes. Ainda relacionado as sequências geradas, foi utilizada uma sequência de contexto para simular o DAS. Essas sequência estimulam mudanças bruscas de contexto, mas não representam o comportamento real do sistema.

Como ameaça externa pode ser citado os DASs utilizados, que são exemplos pequenos e acadêmicos. Isso pode ter afetado o desempenho, pois os DASs da avaliação possuíam uma pequena quantidade de regras de adaptação, *features* e *features* de contexto.

5.5 Resumo do Capítulo

Este capítulo apresentou as avaliações da abordagem proposta nesta dissertação. As avaliações foram divididas em uma prova de conceito, um teste de mutantes e uma análise de impacto na adaptação. Em todas as avaliações foram utilizadas duas aplicações móveis baseadas em regras que se adaptam em tempo de execução.

A prova de conceito realizada teve como objetivo analisar a viabilidade da atividade de checagem de propriedades comportamentais da abordagem. Para isso, foram inseridas falhas nos dois DASs selecionados e foram executadas simulações. A atividade de checagem obteve êxito na identificação de falhas no estado imediato após a execução da adaptação.

Em seguida, foi realizado um teste de mutantes com as aplicações adaptativas utilizadas na prova de conceito. Para esta avaliação, as aplicações foram manualmente modificadas para ter regras mais complexas e simular cenários simples de adaptação de regras em tempo de execução. Foram criados 25 mutantes para o GREat Tour e 22 mutantes para o Phone Adapter, sendo todos eles especificamente voltados para modelos de *features* e regras de adaptação. A

abordagem conseguiu detectar todas as falhas injetadas nas aplicações em pelo menos uma das execuções realizadas.

Por fim, foi realizada uma avaliação para determinar o impacto das atividades de checagem de propriedades e testes na adaptação. Nessa avaliação foi coletado o acréscimo no tempo de adaptação após a inserção da abordagem RETAkE, comparando seus valores com uma configuração sem a abordagem. Foi possível verificar que a ferramenta impactou significativamente no tempo de adaptação de umas das aplicações, mesmo sem executar atividades de testes.

Os resultados das avaliações apresentaram indícios da eficácia da abordagem proposta, mas também apontaram a necessidade de avaliações futuras. A discussão da avaliação e dos resultados foi estendida em uma seção de ameaças à validade. Como principais ameaças podem ser destacadas a dificuldade de generalizar os resultados por conta dos exemplos dos DASs e o processo de geração randômico dos testes.

O próximo capítulo conclui esta dissertação através de apresentação de uma visão geral do trabalho desenvolvido, discussão dos principais resultados obtidos, limitações do estudo realizado e os possíveis trabalhos futuros.

6 CONCLUSÃO

Esta dissertação de mestrado apresentou uma abordagem para testes em tempo de execução para a verificação de DASs baseados em regras de adaptação. O presente capítulo apresenta a conclusão desta dissertação, sendo organizado como segue. A Seção 6.1 apresenta uma visão geral do texto. A Seção 6.2 resume os resultados alcançados durante o desenvolvimento deste trabalho. A Seção 6.3 discute as limitações da abordagem proposta. A Seção 6.4 conclui o capítulo com os trabalhos futuros.

6.1 Visão Geral

A popularidade dos dispositivos móveis, que possuem variados tipos de sensores, tem aumentado o desenvolvimento de sistemas capazes de adaptar suas *features* através de regras baseadas em contexto. O uso de informações de contexto e a reconfiguração do software trazem vários desafios para sua verificação. Para evitar falhas de adaptação, é necessário analisar as informações do sistema para checar suas adaptações.

Trabalhos na literatura têm utilizado diferentes técnicas de verificação para sistemas adaptativos, sendo o teste de software uma dessas técnicas. Entretanto, no que diz respeito a garantia da adaptação durante a execução do DAS, poucos trabalhos têm testado as regras do mecanismo de adaptação em tempo de execução e limitam-se às funcionalidades do sistema. Outra limitação é que dificilmente realizam a combinação de técnicas, ficando restritos à checagem do estado atual ou a uma custosa execução de testes.

Neste sentido, este trabalho propôs a abordagem RETAkE cujo objetivo é executar testes em tempo de execução no mecanismo de adaptação, sendo esses suportados por checagem de propriedades. Com foco nas regras responsáveis por reconfigurar o sistema, RETAkE evolui trabalhos da literatura e apresenta uma organização de técnicas de verificação para tempo de execução. A técnica de checagem de propriedades auxilia na checagem do estado do DAS imediato à adaptação. Com execução condicional e de forma proativa, a execução dos testes em tempo de execução auxilia na identificação de falhas de execução em estados posteriores.

Adicionalmente, foi criada a CONTroL@Runtime, uma ferramenta que pode ser instalada no DAS e automatizar a execução das checagens. Esse artefato permiti aos desenvolvedores instrumentar o sistema com anotações Java, habilitando o monitoramento e controle do código-fonte do DAS. Baseando-se em informações enviadas pelo DAS, a CONTroL@Runtime

mantém um modelo de *features* atualizado para gerar casos de teste consistentes com o sistema.

Com o intuito de realizar uma avaliação inicial da abordagem, foi apresentada uma prova de conceito focada na checagem de propriedades comportamentais. A avaliação foi conduzida através da injeção manual de faltas em dois exemplares de aplicações móveis adaptativas. Como resultado, a implementação da abordagem RETAkE obteve êxito em identificar as falhas previamente inseridas.

Em seguida, foi realizada uma avaliação de mutantes para analisar a eficácia dos testes em tempo de execução. Os mutantes foram gerados manualmente com operadores para regras de adaptação, mas com a diferença de que foram utilizados em versões mais complexas dos exemplares de DAS que simulavam alterações de regras. Todos os mutantes foram executados múltiplas vezes, sendo todos detectados em alguma execução. Em um dos exemplares, os mutantes foram detectados em quase 100% das execuções, porém, no segundo exemplar, obtiveram-se índices de detecção inferiores. Esses resultados ressaltam a necessidade de melhorar o algoritmo de geração de testes.

Finalmente, foi conduzida uma avaliação para medir o impacto na execução do DAS após o inicio das atividades de testes. Das cinco configurações utilizadas na avaliação, a abordagem impactou pouco no tempo de adaptação quando apenas a checagem de propriedades estava ativa. Após a introdução das atividades de testes, ocorreu um aumento considerável no tempo até que uma nova adaptação ocorresse. O aumento era esperado, principalmente porque a execução dos testes bloqueia o mecanismo de adaptação até a conclusão da atividade. A sobrecarga foi maior no exemplar que utilizava comunicação assíncrona nas adaptações. Mesmo assim, o bloqueio momentâneo das adaptações pode compensar caso os requisitos de tempo de resposta do DAS não sejam muito restritos.

6.2 Resultados

Os principais resultados obtidos durante a realização desta pesquisa são listados a seguir:

a) **RETAkE.** Uma abordagem para teste de DAS que é suportada por uma checagem de propriedades comportamentais. Com relação à checagem de propriedades, a abordagem RETAkE possibilitou verificar um conjunto finito de propriedades relacionadas às *features* e regras de adaptação do DAS. Na sequência, a abordagem RETAkE dá suporte à geração e execução de sequências de testes em tempo de execução. Por fim, a abordagem habilita

uma integração parcial entre os resultados das duas técnicas;

b) **CONTroL@Runtime.** Uma ferramenta que automatiza as atividades da abordagem RETAkE realizadas em tempo de execução. Esse artefato permitiu controlar as atividades de adaptação do DAS e executar a verificação das regras de adaptação. O processo é realizado sem a necessidade de um operador humano.

Além disso, durante a realização deste trabalho foram publicados três artigos diretamente relacionados ao tema, sendo estes listados na Tabela 16. O trabalho em (SANTOS; ANDRADE; SANTOS, 2019a) apresenta os passos relativos à checagem de propriedades comportamentais e parte da ferramenta que implementa a abordagem. O segundo artigo (SANTOS; ANDRADE; SANTOS, 2019b) discute a ideia inicial desta dissertação. Por fim, o artigo (SANTOS *et al.*, 2018) apresenta o CONTroL, a ferramenta de tempo de projeto que serviu como base para a CONTroL@Runtime.

Tabela 16 – Artigos relacionados ao tema desta pesquisa

Autor	**Trabalho**	**Evento**	**Qualis (2019)**
SANTOS, E. B.; ANDRADE, R. M. C.; SANTOS, I. S.	Runtime monitoring of behavioral properties in dynamically adaptive systems	XXXIII Simpósio Brasileiro de Engenharia de Software (2019)	A3
SANTOS, E. B.; ANDRADE, R. M. C.; SANTOS, I. S.	Towards runtime testing of dynamically adaptive systems based on behavioral properties	XV Simpósio Brasileiro de Sistemas de Informação: Workshop de Teses e Dissertações (2019)	-
SANTOS, I. S.; **SANTOS, E. B.**; ANDRADE, R. M. C.; NETO, P. A. S.	Control: Context-based reconfiguration testing tool	XXXIII Simpósio Brasileiro de Engenharia de Software: Sessão de Ferramentas (2018)	A3

Fonte: elaborada pelo autor.

Tabela 17 – Outros artigos publicados

Autor	**Trabalho**	**Evento**	**Qualis (2019)**
SANTOS, E. B.; COSTA, L. S.; ARAGÃO, B.; SANTOS, I. S.; ANDRADE, R. M. C.	Extraction of test cases procedures from textual use cases to reduce test effort: Test Factory Experience Report	XVIII Simpósio Brasileiro de Qualidade de Software (2019)	B1
VIEIRA, L.; LIMA, C.; **SANTOS, E. B.**; ARAGÃO, B. S.; SANTOS, I. S.; ANDRADE, R. M. C.	Automação de Testes em uma Fábrica de Testes: Um Relato de Experiência	XIV Simpósio Brasileiro de Sistemas de Informação (2018)	B1

Fonte: elaborada pelo autor.

Os trabalhos presentes em (VIEIRA *et al.*, 2018) e (SANTOS *et al.*, 2019) também foram publicados durante a realização desta dissertação, de forma que os mesmos encontram-se listados na Tabela 17. Esses artigos foram produzidos a partir da participação do autor da

dissertação em projetos da Fábrica de Testes do GREat[1]. Mesmo que os artigos não estejam diretamente relacionados ao tema deste trabalho, ainda assim eles representam contribuições na área de testes de software.

6.3 Limitações

Após a finalização deste trabalho, foram identificadas limitações que estão relacionadas com decisões da criação da abordagem e com o próprio escopo do trabalho. As principais limitações são discutidas a seguir.

A primeira limitação é a checagem das propriedades comportamentais. Apesar de algumas dessas propriedades serem comuns a sistemas adaptativos, elas ainda são vinculadas ao conceito de *features* e modelos de *features* (KANG *et al.*, 1990). Dessa forma, a abordagem é mais adequada para sistemas derivados de LPSD (CAPILLA *et al.*, 2014). Isso pode impedir que elas sejam generalizadas para a aplicação em outros tipos de sistemas (e.g., DAS que se adaptam baseados em métricas de qualidade de serviço). Adicionalmente, também é verificado um conjunto fixo de propriedades.

A segunda limitação é relacionada à ferramenta CONTroL@Runtime e sua forma de identificar a alteração de regras. Para que essa identificação seja feita, o sistema precisa enviar uma requisição com as novas regras e alteração do modelo de *features*. Isso pode impactar na geração dos testes caso as alterações contenham erros. A adição de atividades de checagem de modelos poderia auxiliar na garantia da validade das alterações.

A terceira e última limitação diz respeito a condução das avaliações, que tiveram como objetivo obter indícios sobre a eficiência da solução proposta. Mesmo que não tenha sido realizado um experimento controlado, objetivou-se conduzir as avaliações de forma sistematizada.

6.4 Trabalhos Futuros

Nesta seção são listadas a seguir possibilidades de evolução da presente pesquisa e também são discutidos possíveis direcionamentos que auxiliem na condução desses novos trabalhos:

a) **Geração de sequências de testes mais eficazes.** Em sua versão atual, a abordagem

[1] http://fabricadetestes.great.ufc.br/

RETAkE utilizou um algoritmo para a geração de sequências de testes baseado na literatura. Essa geração poderia ser melhorada com a criação de uma heurística que, baseando-se em informações da execução do sistema, construa sequências de testes mais eficazes. Os trabalhos de Wang e Chan (2009) e Munoz e Baudry (2009) podem servir de base para a concepção de tais heurísticas;

b) **Integração das técnicas de verificação.** Neste trabalho foi apresentada uma opção de integração parcial dos resultados das propriedades comportamentais com a atividade de testes. Trabalhos na literatura voltados para arquiteturas orientadas a serviços (METZGER, 2011; ORIOL; FRANCH; MARCO, 2015) realizam a combinação dessas duas técnicas, mas com o propósito de medir a qualidade de serviço (e.g., tempo de resposta). Em direção similar, os resultados de propriedades poderiam ser utilizados como critério de cobertura para geração dos testes em tempo de execução;

c) **Integração de novas *features*.** Como apresentado por Capilla *et al.* (2014), sistemas com variabilidade dinâmica podem mudar a estrutura de seus modelos em tempo de execução. Desse modo, um DAS pode ter novas *features* integradas em tempo de execução. Um direcionamento para permitir isso seria evoluir a RETAkE com o conceito de *supertypes* (CAPILLA; VALDEZATE; DÍAZ, 2016). Para isso, o modelo eCFM também precisaria ser evoluído para possibilitar a inclusão e remoção de novas *features*. Adicionalmente, as propriedades comportamentais podem ser verificadas utilizando checagem de modelos (CLARKE JR *et al.*, 2018) para garantir a corretude do modelo alterado. Entretanto, essas verificações precisariam ser agendadas para momentos que não impactem nas operações do DAS;

d) **Desacoplamento da ferramenta de verificação do sistema.** Em sua versão atual, a ferramenta CONTroL@Runtime precisa estar instalada no projeto do DAS. Uma melhoria seria desacoplar a solução, permitindo que a mesma seja executada como um serviço independente, tal como ocorre nos modelos *Test as a Service* (MOHANTY; MOHANTY; BALAKRISHNAN, 2017). Isso possibilitaria a execução das atividades utilizando um serviço escalável e independente de plataforma. Caso o desacoplamento tivesse uma instrumentação menos intrusiva, tal como anotações no código-fonte, também seria possível facilitar a inserção de novas *features* em tempo de execução;

e) **Adaptação da ferramenta.** A CONTroL@Runtime possui componentes fixos com atividades pré-determinadas. Uma melhoria seria habilitar a adaptação da mesma para

acompanhar o dinamismo do DAS. Neste sentido, o artefato poderia ser ciente de contexto para que ativasse suas funcionalidades baseando-se no ambiente e no DAS. O *framework* ReMinds (VIERHAUSER *et al.*, 2016) pode ser citado como um exemplo para o domínio de Sistemas de Sistemas que possui tal característica. Esse trabalho utiliza componentes desacoplados que modificam sua disponibilidade durante a execução;

f) **Utilização de outros modelos.** A RETAkE foi originalmente projetada para utilizar o modelo eCFM para representar as *features* e o DFTS, para a representação dos estados de contexto. Entretanto, a abordagem pode ser estendida para utilizar outros modelos de variabilidade, tais como o modelos que representam *features* de sistema e de contexto em uma única estrutura (CAPILLA *et al.*, 2014). Essa adaptação pode beneficiar a adequação da abordagem para projetos de diferentes contextos;

g) **Avaliação com exemplos de DAS complexos.** As avaliações conduzidas no presente trabalho utilizaram dois exemplos de DAS simples e acadêmicos. Os DASs utilizados possuíam poucas *features* e regras de adaptação, o que pode ter impactado na avaliação de tempos. Novas avaliações podem ser conduzidas com DASs mais complexos com um número maior de regras de adaptação e variáveis de contexto. Essas novas avaliações podem ser realizadas através de um experimento controlado (WOHLIN *et al.*, 2012).

REFERÊNCIAS

ABOWD, G. D. *et al.* Towards a better understanding of context and context-awareness. *In*: GELLERSEN, H W. (org.). **Handheld and Ubiquitous Computing**. Heidelberg: Springer, 1999. p. 304–307.

AHRENDT, W. *et al.* **COST Action IC 1402 ArVI: Runtime Verification Beyond Monitoring–Activity Report of Working Group 1**. [*S. l.*]: arXiv preprint arXiv:1902.03776, 2019.

ALVES, V. *et al.* Comparative study of variability management in software product lines and runtime adaptable systems. *In*: INTERNATIONAL WORKSHOP ON VARIABILITY MODELLING OF SOFTWARE-INTENSIVE SYSTEMS, 2009, [*S. l.*]. **Proceedings** [...]. [*S. l.: s. n.*], 2009. p. 9–17.

ANDREWS, J. H.; BRIAND, L. C.; LABICHE, Y. Is mutation an appropriate tool for testing experiments? *In*: 27TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING, 27., 2005, St. Louis. **Proceedings** [...]. New York: ACM, 2005. p. 402–411.

ANTONINUS, M. A.; HAYS, G. **Meditations**: a new translation. 1. st. New York: Modern Library, 2003.

ARAGÃO JUNIOR, B. R. *et al.* SUCCEEd: support mechanism for creating and executing workflows for decoupled SAS in IoT. *In*: IEEE 42ND ANNUAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 42., 2018, Kyoto. **Proceedings** [...]. [*S. l.*]: IEEE, 2018. p. 738–743.

BARBOSA, D. M. *et al.* Lotus@Runtime: a tool for runtime monitoring and verification of self-adaptive systems. *In*: IEEE/ACM 12TH INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 12., 2017, Buenos Aires. **Proceedings** [...]. [*S. l.*]: IEEE, 2017. p. 24–30.

BARESI, L.; GHEZZI, C. The disappearing boundary between development-time and run-time. *In*: FSE/SDP WORKSHOP ON FUTURE OF SOFTWARE ENGINEERING RESEARCH, 2010, Santa Fe. **Proceedings** [...]. New York: ACM, 2010. p. 17–22.

BENAVIDES, D.; SEGURA, S.; RUIZ-CORTÉS, A. Automated analysis of feature models 20 years later: A literature review. **Information systems**, [*S. l.*], v. 35, n. 6, p. 615–636, set. 2010.

BOURQUE, P.; FAIRLEY, R. E. **Guide to the software engineering body of knowledge (SWEBOK (R))**: Version 3.0. 3. rd. [*S. l.*]: IEEE Computer Society Press, 2014.

BRENNER, D. *et al.* Reducing verification effort in component-based software engineering through built-in testing. **Information Systems Frontiers**, [*S. l.*], v. 9, n. 2-3, p. 151–162, jul. 2007.

BRUN, Y. *et al.* Engineering self-adaptive systems through feedback loops. *In*: CHENG, B. H. C. *et al.* (org.). **Software engineering for self-adaptive systems**. Heidelberg: Springer, 2009. p. 48–70.

BÜRDEK, J. *et al.* Staged configuration of dynamic software product lines with complex binding time constraints. *In*: EIGHTH INTERNATIONAL WORKSHOP ON VARIABILITY

MODELLING OF SOFTWARE-INTENSIVE SYSTEMS, 8., 2014, Sophia Antipolis. **Proceedings** [...]. New York: ACM, 2014. p. 16.

CAFEO, B. B. P. *et al.* Inferring test results for dynamic software product lines. *In*: 19TH ACM SIGSOFT SYMPOSIUM AND THE 13TH EUROPEAN CONFERENCE ON FOUNDATIONS OF SOFTWARE ENGINEERING, 19., 2011, Szeged. **Proceedings** [...]. New York: ACM, 2011. p. 500–503.

CAPILLA, R.; ORTIZ, Ó.; HINCHEY, M. Context variability for context-aware systems. **Computer**, Los Alamitos, v. 47, n. 2, p. 85–87, fev. 2014.

CAPILLA, R. *et al.* An overview of dynamic software product line architectures and techniques: observations from research and industry. **Journal of Systems and Software**, [*S. l.*], v. 91, p. 3–23, maio 2014.

CAPILLA, R.; VALDEZATE, A.; DÍAZ, F. J. A runtime variability mechanism based on supertypes. *In*: IEEE 1ST INTERNATIONAL WORKSHOPS ON FOUNDATIONS AND APPLICATIONS OF SELF* SYSTEMS (FAS* W), 1., 2016, Augsburg. **Proceedings** [...]. [*S. l.*]: IEEE, 2016. p. 6–11.

CHENG, B. H. C. *et al.* Software engineering for self-adaptive systems: a research roadmap. *In*: CHENG, B. H. C. *et al.* (org.). **Software engineering for self-adaptive systems**. Heidelberg: Springer, 2009. p. 1–26.

CHIMENTO, J. M.; AHRENDT, W.; SCHNEIDER, G. Testing meets static and runtime verification. *In*: IEEE/ACM 6TH INTERNATIONAL FME WORKSHOP ON FORMAL METHODS IN SOFTWARE ENGINEERING, 6., 2018, Gothenburg. **Proceedings** [...]. [*S. l.*]: IEEE, 2018. p. 30–39.

CLARKE JR, E. M. *et al.* **Model checking**. 2. nd. London: MIT press, 2018.

COMPUTING, A. *et al.* An architectural blueprint for autonomic computing. **IBM White Paper**, [*S. l.*], v. 31, n. 2006, p. 1–6, jun. 2006.

CZARNECKI, K.; WASOWSKI, A. Feature diagrams and logics: there and back again. *In*: 11TH INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE, 11., 2007, Kyoto. **Proceedings** [...]. [*S. l.*]: IEEE, 2007. p. 23–34.

DE LEMOS, R. *et al.* Software engineering for self-adaptive systems: a second research roadmap. *In*: DE LEMOS, R. *et al.* (org.). **Software Engineering for Self-Adaptive Systems II**. Heidelberg: Springer, 2013. p. 1–32.

DE LEMOS, R. *et al.* Software engineering for self-adaptive systems: Research challenges in the provision of assurances. *In*: DE LEMOS, R. *et al.* (org.). **Software Engineering for Self-Adaptive Systems III. Assurances**. Cham: Springer, 2017. p. 3–30.

EBERHARDINGER, B.; HABERMAIER, A.; REIF, W. Toward adaptive, self-aware test automation. *In*: 12TH INTERNATIONAL WORKSHOP ON AUTOMATION OF SOFTWARE TESTING, 12., 2017, Buenos Aires. **Proceedings** [...]. [*S. l.*]: IEEE, 2017. p. 34–37.

FREDERICKS, E. M.; CHENG, B. H. C. Automated generation of adaptive test plans for self-adaptive systems. *In*: IEEE/ACM 10TH INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 10., 2015, Florence. **Proceedings** [...]. [*S. l.*]: IEEE, 2015. p. 157–168.

FREDERICKS, E. M.; DEVRIES, B.; CHENG, B. H. C. Towards run-time adaptation of test cases for self-adaptive systems in the face of uncertainty. *In*: 9TH INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 9., 2014, Hyderabad. **Proceedings** [...]. New York: ACM, 2014. p. 17–26.

GONZÁLEZ, A.; PIEL, E.; GROSS, H.-G. A model for the measurement of the runtime testability of component-based systems. *In*: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE TESTING, VERIFICATION, AND VALIDATION WORKSHOPS, 2009, Denver. **Proceedings** [...]. [*S. l.*]: IEEE, 2009. p. 19–28.

HALLSTEINSEN, S. *et al.* Dynamic software product lines. **Computer**, [*S. l.*], v. 41, n. 4, p. 93–95, abr. 2008.

HÄNSEL, J.; GIESE, H. Towards collective online and offline testing for dynamic software product line systems. *In*: IEEE/ACM 2ND INTERNATIONAL WORKSHOP ON VARIABILITY AND COMPLEXITY IN SOFTWARE DESIGN, 2., 2017, Buenos Aires. **Proceedings** [...]. [*S. l.*]: IEEE, 2017. p. 9–12.

HASS, A. M. **Guide to advanced software testing**. Norwood: Artech House, 2008.

HIELSCHER, J. *et al.* A framework for proactive self-adaptation of service-based applications based on online testing. *In*: MÄHÖNEN, P.; POHL, K.; PRIOL, T. (org.). **Towards a Service-Based Internet**. Heidelberg: Springer, 2008. p. 122–133.

IEEE. Standard classification for software anomalies. **IEEE Std 1044-2009 (Revision of IEEE Std 1044-1993)**, New York, jan. 2010.

IEEE. IEEE draft international standard for software and systems engineering–software testing–part 4: Test techniques. **ISO/IEC/IEEE P29119-4-FDIS April 2015**, [*S. l.*], p. 1–147, fev. 2015.

IFTIKHAR, M. U.; WEYNS, D. ActivFORMS: a runtime environment for architecture-based adaptation with guarantees. *In*: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE WORKSHOPS, 2017, Gothenburg. **Proceedings** [...]. [*S. l.*]: IEEE, 2017. p. 278–281.

ISLAM, N.; AZIM, A. Assuring the runtime behavior of self-adaptive cyber-physical systems using feature modeling. *In*: 28TH ANNUAL INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING, 28., 2018, Markham. **Proceedings** [...]. Riverton: IBM, 2018. p. 48–59.

JIA, Y.; HARMAN, M. An analysis and survey of the development of mutation testing. **IEEE transactions on software engineering**, [*S. l.*], v. 37, n. 5, p. 649–678, jun. 2010.

KANG, K. C. *et al.* **Feature-oriented domain analysis (FODA) feasibility study**. [*S. l.*]: Carnegie-Mellon Software Engineering Institute, 1990.

KEPHART, J. O.; CHESS, D. M. The vision of autonomic computing. **Computer**, [*S. l.*], v. 36, n. 1, p. 41–50, jan. 2003.

KISELEV, I. **Aspect-oriented programming with AspectJ**. 1. st. Indianapolis: Sams, 2002.

KITCHENHAM, B.; CHARTERS, S. **Guidelines for performing systematic literature reviews in software engineering**. [*S. l.*]: Citeseer, 2007.

KRUPITZER, C. *et al.* A survey on engineering approaches for self-adaptive systems. **Pervasive and Mobile Computing**, [*S. l.*], v. 17, p. 184–206, fev. 2015.

LAHAMI, M.; KRICHEN, M.; JMAIEL, M. Safe and efficient runtime testing framework applied in dynamic and distributed systems. **Science of Computer Programming**, [*S. l.*], v. 122, p. 1–28, jun. 2016.

LEAL, L.; CECCARELLI, A.; MARTINS, E. The SAMBA approach for self-adaptive model-based online testing of services orchestrations. *In*: IEEE 43RD ANNUAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 43., 2019, Milwaukee. **Proceedings** [...]. [*S. l.*]: IEEE, 2019. p. 495–500.

LEUCKER, M.; SCHALLHART, C. A brief account of runtime verification. **The Journal of Logic and Algebraic Programming**, [*S. l.*], v. 78, n. 5, p. 293–303, maio/jun. 2009.

LIM, Y. J. *et al.* Efficient testing of self-adaptive behaviors in collective adaptive systems. *In*: IEEE 39TH ANNUAL COMPUTER SOFTWARE AND APPLICATIONS CONFERENCE, 39., 2015, Taichung. **Proceedings** [...]. [*S. l.*]: IEEE, 2015. p. 216–221.

LIMA, E. R. R. *et al.* GREat Tour: um guia de visitas móvel e sensível ao contexto. *In*: WORKSHOP DE FERRAMENTAS E APLICAÇÕES - SIMPÓSIO BRASILEIRO DE SISTEMAS MULTIMÍDIA E WEB, 2013, Salvador. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2013. p. 53–56.

MACÍAS-ESCRIVÁ, F. D. *et al.* Self-adaptive systems: a survey of current approaches, research challenges and applications. **Expert Systems with Applications**, [*S. l.*], v. 40, n. 18, p. 7267–7279, dez. 2013.

MARINHO, F. G. *et al.* Safe adaptation in context-aware feature models. *In*: 4TH INTERNATIONAL WORKSHOP ON FEATURE-ORIENTED SOFTWARE DEVELOPMENT, 4., 2021, Dresden. **Proceedings** [...]. New York: ACM, 2012. p. 54–61.

MARINHO, F. G. *et al.* MobiLine: a nested software product line for the domain of mobile and context-aware applications. **Science Computer Programming**, [*S. l.*], v. 78, n. 12, p. 2381–2398, dez. 2013.

MATALONGA, S.; RODRIGUES, F.; TRAVASSOS, G. H. Characterizing testing methods for context-aware software systems: results from a quasi-systematic literature review. **Journal of Systems and Software**, [*S. l.*], v. 131, p. 1–21, set. 2017.

MAURO, J. *et al.* Context-aware reconfiguration in evolving software product lines. **Science of Computer Programming**, [*S. l.*], v. 163, p. 139–159, out. 2018.

MCKINLEY, P. K. *et al.* Composing adaptive software. **Computer**, [*S. l.*], v. 37, n. 7, p. 56–64, jul. 2004.

METZGER, A. Towards accurate failure prediction for the proactive adaptation of service-oriented systems. *In*: 8TH WORKSHOP ON ASSURANCES FOR SELF-ADAPTIVE SYSTEMS, 8., 2011, Szeged. **Proceedings** [...]. New York: ACM, 2011. p. 18–23.

MIZOUNI, R. *et al.* A framework for context-aware self-adaptive mobile applications SPL. **Expert Systems with applications**, [*S. l.*], v. 41, n. 16, p. 7549–7564, nov. 2014.

MOHANTY, H.; MOHANTY, J. R.; BALAKRISHNAN, A. **Trends in software testing**. 1. st. [*S. l.*]: Springer Singapore, 2017.

MONGIELLO, M.; PELLICCIONE, P.; SCIANCALEPORE, M. AC-Contract: run-time verification of context-aware applications. *In*: IEEE/ACM 10TH INTERNATIONAL SYMPOSIUM ON SOFTWARE ENGINEERING FOR ADAPTIVE AND SELF-MANAGING SYSTEMS, 10., 2015, Florence. **Proceedings** [...]. [*S. l.*]: IEEE, 2015. p. 24–34.

MOUSA, A.; BENTAHAR, J.; ALAM, O. Context-aware composite SaaS using feature model. **Future Generation Computer Systems**, [*S. l.*], v. 99, p. 376–390, out. 2019.

MUNOZ, F.; BAUDRY, B. Artificial table testing dynamically adaptive systems. **arXiv preprint arXiv:0903.0914**, [*S. l.*], 2009.

MYERS, G. J.; SANDLER, C.; BADGETT, T. **The art of software testing**. 3. rd. New Jersey: John Wiley & Sons, 2011.

ORIOL, M.; FRANCH, X.; MARCO, J. Monitoring the service-based system lifecycle with SALMon. **Expert Systems with Applications**, [*S. l.*], v. 42, n. 19, p. 6507–6521, nov. 2015.

PETERSEN, K. *et al.* Systematic mapping studies in software engineering. *In*: 12TH INTERNATIONAL CONFERENCE ON EVALUATION AND ASSESSMENT IN SOFTWARE ENGINEERING, 12., 2008, Bari. **Proceedings** [...]. Swindon: BCS Learning & Development, 2008. p. 68–77.

PIEL, É.; GONZÁLEZ, A.; GROSS, H.-G. Automating integration testing of large-scale publish/subscribe systems. *In*: HINZE, A. M.; BUCHMANN A. (org.). **Principles and Applications of Distributed Event-Based Systems**. Hershey: IGI Global, 2010. p. 140–163.

PÜSCHEL, G. *et al.* A combined simulation and test case generation strategy for self-adaptive systems. **Journal On Advances in Software**, [*S. l.*], v. 7, n. 3–4, p. 686–696, 2014.

PÜSCHEL, G. *et al.* Testing self-adaptive software: requirement analysis and solution scheme. **International Journal on Advances in Software**, [*S. l.*], v. 2628, n. 1–2, p. 88–100, 2014.

QIN, Y. *et al.* SIT: sampling-based interactive testing for self-adaptive apps. **Journal of Systems and Software**, [*S. l.*], v. 120, p. 70–88, out. 2016.

QIN, Y. *et al.* CoMID: context-based multiinvariant detection for monitoring cyber-physical software. **IEEE Transactions on Reliability**, [*S. l.*], v. 69, n. 1, p. 106–123, ago. 2019.

ROCHA, L. S.; ANDRADE, R. M. C. Towards a formal model to reason about context-aware exception handling. *In*: 5TH INTERNATIONAL WORKSHOP ON EXCEPTION HANDLING, 5., 2012, Zurich. **Proceedings** [...]. [*S. l.*]: IEEE, 2012. p. 27–33.

SALEHIE, M.; TAHVILDARI, L. Self-adaptive software: landscape and research challenges. **ACM transactions on autonomous and adaptive systems**, [*S. l.*], v. 4, n. 2, p. 14, maio 2009.

SALLER, K.; LOCHAU, M.; REIMUND, I. Context-aware DSPLs: model-based runtime adaptation for resource-constrained systems. *In*: 17TH INTERNATIONAL SOFTWARE PRODUCT LINE CONFERENCE CO-LOCATED WORKSHOPS, 17., 2013, Tokyo. **Proceedings** [...]. New York: ACM, 2013. p. 106–113.

SAMA, M. *et al.* Context-aware adaptive applications: fault patterns and their automated identification. **IEEE Transactions on Software Engineering**, [*S. l.*], v. 36, n. 5, p. 644–661, set. 2010.

SANTOS, E. B.; ANDRADE, R. M. C.; SANTOS, I. S. Runtime monitoring of behavioral properties in dynamically adaptive systems. *In*: XXXIII BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 33., 2019, Salvador. **Proceedings** [...]. New York: ACM, 2019. p. 377–386.

SANTOS, E. B.; ANDRADE, R. M. C.; SANTOS, I. S. Towards runtime testing of dynamically adaptive systems based on behavioral properties. *In*: WORKSHOP OF THESIS AND DISSERTATIONS ON INFORMATION SYSTEMS - BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEMS, 15., 2019, Aracaju. **Proceedings** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2019. p. 49–52.

SANTOS, E. B. *et al.* Extraction of test cases procedures from textual use cases to reduce test effort: test factory experience report. *In*: XVIII BRAZILIAN SYMPOSIUM ON SOFTWARE QUALITY, 18., 2019, Fortaleza. **Proceedings** [...]. New York: ACM, 2019. p. 266–275.

SANTOS, I. S. **TestDAS**: Testing method for dynamically adaptive systems. 2017. 183 f. Tese (Doutorado em Ciência da Computação) – Universidade Federal do Ceará, Fortaleza, 2017.

SANTOS, I. S. *et al.* Model verification of dynamic software product lines. *In*: XXX BRAZILIAN SYMPOSIUM ON SOFTWARE ENGINEERING, 30., 2016, Maringá. **Proceedings** [...]. New York: ACM, 2016. p. 113–122.

SANTOS, I. S. *et al.* Dynamically adaptable software is all about modeling contextual variability and avoiding failures. **IEEE Software**, [*S. l.*], v. 34, n. 6, p. 72–77, nov./dez. 2017.

SANTOS, I. S. *et al.* Test case design for context-aware applications: are we there yet? **Information and Software Technology**, [*S. l.*], v. 88, p. 1–16, ago. 2017.

SANTOS, I. S. *et al.* CONTroL: context-based reconfiguration testing tool. *In*: TOOLS SESSION OF CBSoft, 2018, São Carlos. **Proceedings** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2018. p. 7–12.

SCHMERL, B. *et al.* Challenges in composing and decomposing assurances for self-adaptive systems. *In*: DE LEMOS, R. *et al.* (org.). **Software Engineering for Self-Adaptive Systems III. Assurances**. Cham: Springer, 2017. p. 64–89.

SIQUEIRA, B. R. *et al.* Characterisation of challenges for testing of adaptive systems. *In*: 1ST BRAZILIAN SYMPOSIUM ON SYSTEMATIC AND AUTOMATED SOFTWARE TESTING, 1., 2016, Maringá. **Proceedings** [...]. New York: ACM, 2016. p. 1–10.

SIQUEIRA, B. R. *et al.* Experimenting with a multi-approach testing strategy for adaptive systems. *In*: 17TH BRAZILIAN SYMPOSIUM ON SOFTWARE QUALITY, 17., 2018, Curitiba. **Proceedings** [...]. New York: ACM, 2018. p. 111–120.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Education, 2011.

TAMURA, G. *et al.* Towards practical runtime verification and validation of self-adaptive software systems. *In*: DE LEMOS, R. *et al.* (org.) **Software Engineering for Self-Adaptive Systems II**. Heidelberg: Springer, 2013. p. 108–132.

VIEIRA, L. S. *et al.* Automação de testes em uma fábrica de testes: um relato de experiência. *In*: XIV SIMPÓSIO BRASILEIRO DE SISTEMAS DE INFORMAÇÃO, 14., 2018, Caxias do Sul. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2018. p. 80–73.

VIERHAUSER, M. *et al.* ReMinds: a flexible runtime monitoring framework for systems of systems. **Journal of Systems and Software**, [*S. l.*], v. 112, p. 123–136, fev. 2016.

WANG, H.; CHAN, W. K. Weaving context sensitivity into test suite construction. *In*: IEEE/ACM INTERNATIONAL CONFERENCE ON AUTOMATED SOFTWARE ENGINEERING, 2009, Auckland. **Proceedings** [...]. [*S. l.*]: IEEE, 2009. p. 610–614.

WANG, H.; CHAN, W. K.; TSE, T. H. Improving the effectiveness of testing pervasive software via context diversity. **ACM Transactions on Autonomous and Adaptive Systems**, [*S. l.*], v. 9, n. 2, p. 1–28, jul. 2014.

WOHLIN, C. *et al.* **Experimentation in software engineering**. 1. st. Heidelberg: Springer-Verlag Berlin Heidelberg, 2012.

XU, C. *et al.* Adam: identifying defects in context-aware adaptation. **Journal of Systems and Software**, [*S. l.*], v. 85, n. 12, p. 2812–2828, dez. 2012.