UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

RAFAEL GONÇALVES LIMA

UM MÉTODO DE ADAPTAÇÃO PARA MODELOS DE *FEATURES* DE LINHA DE PRODUTOS DE SOFTWARE DINÂMICA

QUIXADÁ

2021

RAFAEL GONÇALVES LIMA

UM MÉTODO DE ADAPTAÇÃO PARA MODELOS DE *FEATURES* DE LINHA DE PRODUTOS DE SOFTWARE DINÂMICA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientadora: Profª. Dra. Carla Illane Moreira Bezerra

QUIXADÁ

2021

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)

L71m Lima, Rafael Gonçalves.
Um Método de Adaptação para Modelos de Features de Linha de Produtos de Software Dinâmica / Rafael Gonçalves Lima. – 2021.
70 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2021.
Orientação: Profa. Dra. Carla Illane Moreira Bezerra.

1. Engenharia de Linha de Produto de Software. 2. Modelo de Características. 3. Sistema Autoadaptativo. I. Título.

CDD 005.1

RAFAEL GONÇALVES LIMA

UM MÉTODO DE ADAPTAÇÃO PARA MODELOS DE *FEATURES* DE LINHA DE PRODUTOS DE SOFTWARE DINÂMICA

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: _____/_____/_____.

BANCA EXAMINADORA

Profª. Dra. Carla Illane Moreira Bezerra (Orientadora)
Universidade Federal do Ceará (UFC)

Prof. Dr. Ivan do Carmo Machado
Universidade Federal da Bahia (UFBA)

Prof. Dr. Enyo José Tavares Gonçalves
Universidade Federal do Ceará (UFC)

Me. Anderson Gonçalves Uchôa
Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio)

À minha família, por acreditarem em minhas escolhas e me apoiarem sempre. Às famílias dos 353 mil brasileiros que perderam a vida por conta do COVID-19, apesar de imersos em tristezas e incertezas, dias melhores virão.

AGRADECIMENTOS

Primeiramente gostaria de agradecer a YONARA, meu amor de longa data, com quem compartilho minha vida, a pessoa que esteve a meu lado durante esses anos difíceis, me ajudando a vencer os desafios e a me manter íntegro nos meus objetivos.

Agradeço aos meus pais, pela força e apoio. Eles são os responsáveis por me fazer seguir em frente e a buscar meus sonhos. Tudo que sou e tudo que construí foi graças aos seus ensinamentos. Agradeço também as minhas irmãs, por acreditarem em mim e por estarem sempre ao meu lado.

Agradeço de um modo especial a Profª. Dra. Carla Ilane, com quem pude trabalhar ao longo dos últimos anos e aprender bastante. As suas orientações foram importantes na construção desse trabalho, obrigado pela confiança e pelo compromisso. Aos demais professores da UFC Campus Quixadá, todos vocês contribuíram para minha formação profissional. Obrigado pelo carinho, conhecimentos e experiências compartilhadas.

Aos meus colegas de curso, foi muito bom ter vivido tudo isso ao lado de vocês. Em especial, gostaria de agradecer aos irmãos que a vida me deu... Clebson, Josué, Pedro, Victor e Wilton, como eu pude aprender com vocês, obrigado pelas conversas, brincadeiras e discussões. Vocês moram no meu coração.

Agradeço a todo mundo que me deu apoio, acreditou e torceu por mim. Por fim, agradeço à Deus pela saúde e pela força para seguir rumo aos meus sonhos.

MUITO OBRIGADO!

“O conhecimento serve para encantar as pessoas,
não para humilhá-las.”

(Mario Sergio Cortella)

RESUMO

Linha de Produtos de Software (LPS) é um paradigma utilizado no gerenciamento de *features* comuns a um conjunto de sistemas complexos, que auxilia a gestão da variabilidade, diminui os custos e tempo de implementação e aumenta o suporte a manutenibilidade. Em uma LPS a gestão da variabilidade é feita de forma estática com as configurações do produto sendo escolhidas em tempo de desenvolvimento. A partir do momento que os sistemas começam a fazer parte do cotidiano das pessoas, há a necessidade de que esses softwares executem continuamente, adaptando-se as mudanças do ambiente e das necessidades do usuário. Para o gerenciamento da variabilidade desses novos sistemas, surgiu o conceito de Linha de Produtos de Softwares Dinâmica (LPSD). A representação de uma LPSD pode ser feita utilizando-se a notação do modelo de *features* (MF). Um dos principais desafios de uma LPSD é o gerenciamento das configurações do MF em tempo de execução, detectando as mudanças no contexto a partir de um conjunto de cenários de adaptação e variando entre as configurações possíveis. Neste contexto, este trabalho tem como objetivo o desenvolvimento de um mecanismo de adaptação para MF de LPSD que permite a modelagem de agentes e contextos, e o suporte a variabilidade dinâmica. O mecanismo transforma as restrições em regras que condicionam a ativação de cada *feature*, os contextos também são anotados nas *features* correspondente para serem ativadas quando ocorrerem mudanças. O desenvolvimento do mecanismo utilizou os conceitos do modelo MAPE-K e foi implementado como extensão da ferramenta DyMMer *web*. Para avaliação do mecanismo, foi realizado um teste de desempenho com modelos de *features* simples e complexos e uma prova de conceito junto a um especialista. Os resultados da avaliação de desempenho revelaram que o mecanismo realiza a adaptação em um ótimo tempo, perdendo um pouco de performance na medida em que o MF fica mais complexo, mas se mantendo ainda dentro de um limite aceitável. Os resultados da avaliação com o especialista demonstraram que o mecanismo auxilia na modelagem dos agentes e contextos de adaptação, bem como apresenta de forma intuitiva o processo de reconfiguração em tempo de execução. Dessa forma, verificou-se que o mecanismo de adaptação compre bem o papel de modelar os agentes e contextos de um MF de LPSD e a execução do processo de reconfiguração dinâmica.

Palavras-chave: Engenharia de Linha de Produto de Software. Modelo de Características. Sistema Autoadaptativo

ABSTRACT

Software Product Line (SPL) is a paradigm used in the management of features common to a set of complex systems, which helps in managing variability, decreasing costs and time of implementation and increasing support for maintainability. In an SPL the management of variability is done in a static way with the product configurations being chosen at development time. From the moment that systems start to be part of people's daily lives, there is a need for these software to run continuously, adapting to changes in the environment and user needs. For the management of the variability of these new systems, the concept of Dynamic Software Product Line (DSPL) emerged. The representation of an DSPL can be done using the feature model (FM) notation. One of the main challenges of an DSPL is the management of FM configurations at runtime, detecting changes in context from a set of adaptation scenarios and varying between possible configurations. In this context, this work aims to develop an adaptation mechanism for DSPL FM that allows the modeling of agents and contexts, and to support dynamic variability. The mechanism transforms the restrictions into rules that condition the activation of each feature, the contexts are also noted in the corresponding features to be activated when changes occur. The development of the mechanism used the concepts of the MAPE-K model and was implemented as an extension of the DyMMer web tool. To evaluate the mechanism, a performance test was carried out with models of simple and complex features and a proof of concept with an expert. The results of the performance evaluation revealed that the mechanism performs the adaptation in a great time, losing a little performance as the FM becomes more complex, but still remaining within an acceptable limit. The results of the evaluation with the specialist showed that the mechanism helps in the modeling of agents and adaptation contexts, as well as intuitively presenting the reconfiguration process at runtime. Thus, it was found that the adaptation mechanism well buys the role of modeling the agents and contexts of an DSPL FM and the execution of the dynamic reconfiguration process.

Keywords: Software Product Line Engineering. Feature Model. Self-Adaptive System

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

LPS	Linhas de Produtos de Softwares
JSON	*JavaScript Object Notation*
MF	Modelo de *features*
MAPE-K	*Monitor, Analyze, Plan, Execute, Knowledge*
LPSD	Linhas de Produtos de Softwares Dinâmicas
KAOS	"*Keep All Objectives Satisfied*"
FODA	*Feature Oriented Domain Analysis*
PIBITI	Programa Institucional de Bolsas de Iniciação em Desenvolvimento Tecnológico e Inovação
SDA	Sistema Dinamicamente Adaptativos
NFP	*Non-Functional Properties* / Propriedades não Funcionais

SUMÁRIO

1 INTRODUÇÃO

Há pouco mais de uma década era comum que os softwares fossem implementados embutidos no mesmo hardware individualmente, de modo que cada produto tinha sua própria variação e podia ser comprado com todo o conjunto de *features* necessárias ao cumprimento de seu propósito (POHL *et al.*, 2005). Com os avanços no campo da tecnologia, gradativamente os softwares passaram a ser mais complexos (HINCHEY *et al.*, 2012). Sistemas mais complexos são mais propensos a gerar erros, difíceis de implementar, além de terem custos elevados e um maior tempo de desenvolvimento. De acordo com Pohl *et al.* (2005), todos esses aspectos exigem a adoção de abordagens de desenvolvimento de software que suporte a variabilidade dos produtos e permita a implementação de novas funcionalidades ao longo da vida útil da aplicação.

Nesse sentido, Pohl *et al.* (2005) indicam que a utilização da engenharia de Linhas de Produtos de Software (LPS) é um paradigma que atende ao desenvolvimento de sistemas mais complexos. Uma LPS permite a construção de blocos de software reusáveis e o gerenciamento da variabilidade através do mapeamento das semelhanças e diferenças do produto com relação aos requisitos, arquitetura, componentes e artefatos (POHL *et al.*, 2005). Essas características ajudam no gerenciamento da complexidade, uma vez que permite a customização do software de acordo com as necessidades dos clientes, reduzindo os custos e o tempo de desenvolvimento (POHL *et al.*, 2005).

Com as sucessivas mudanças nas demandas de software, os desafios de desenvolvimento só aumentaram. Os sistemas passaram a fazer parte do cotidiano das pessoas e se tornaram essenciais nos mais diversos campos do conhecimento humano (HALLSTEINSEN *et al.*, 2008). Essa nova realidade passou a exigir que as aplicações fossem capazes de executar continuamente, independente das condições do ambiente e adaptando-se às mudanças de acordo com as necessidades dos usuários (HINCHEY *et al.*, 2012). Para Hinchey *et al.* (2012), esses novos desafios exigiram diferentes abordagens para suportar o desenvolvimento de sistemas capazes de se adaptar as variações em tempo de execução.

Nesse contexto, uma das abordagens amplamente utilizada para lidar com softwares sensíveis ao contexto é de Linhas de Produtos de Softwares Dinâmicas (LPSD), no qual estende a LPS convencional para modelar a variabilidade de sistemas adaptativos em tempo de execução (BASHARI *et al.*, 2017). Segundo Hallsteinsen *et al.* (2008), uma LPSD se distingue de uma LPS em pelo menos duas características: reconfiguração em tempo de execução e autogerenciamento. A reconfiguração em tempo de execução, representa a habilidade de configurar a variação em

tempo de execução durante todo o ciclo de vida do software, além da capacidade de lidar com mudanças inesperadas (BASHARI *et al.*, 2017). O autogerenciamento representa a habilidade do produto de ser sensível ao contexto e de tomar as decisões de forma automática (BASHARI *et al.*, 2017).

Segundo Tang e Leung (2017), a variabilidade numa família de produtos é definida pelo seu conjunto de *features*, analisadas no domínio da aplicação e descritas a partir das relações e restrições existentes entre si. Essas *features* podem ser representadas em um MF: um diagrama que organiza todo conjunto de *features* de uma linha de produto em formato de árvore, mapeando suas dependências e restrições (TANG; LEUNG, 2017). O MF também auxilia na geração e validação de configurações individuais de um produto, fornecendo uma forte contribuição para o processo de análise do domínio (DAVRIL *et al.*, 2013).

No desenvolvimento de sistemas autoadaptativos e sensíveis ao contexto, alguns estudos foram realizados a fim de se propor abordagens compatíveis com as características de LPSD. Bashari *et al.* (2017) compararam sete metodologias originais de desenvolvimento para a engenharia de LPSD que suportam os processos de adaptação em tempo real. Os autores identificaram que o processo de desenvolvimento de sistemas adaptativos é complexo e bastante desafiador, e que esse campo de conhecimento requer que mais pesquisas sejam realizadas (BASHARI *et al.*, 2017). Horcas *et al.* (2019) realizaram uma análise empírica de ferramentas para identificar se elas suportam o processo completo de gerenciamento de LPSs complexas. Os autores identificaram que não há uma abordagem integrada que compreenda todas as fases do gerenciamento de uma LPS.

Em Weckesser *et al.* (2018), foi definido uma abordagem para reconfiguração dinâmica em tempo de execução que usa informações de desempenho no processo de definição de uma configuração ideal para o contexto. Apesar de realizar a adaptação em tempo de execução, o modelo não considera as restrições entre *features*, o que pode tornar mais complexo a utilização da abordagem em LPSDs mais complexas. Em Sawyer *et al.* (2012) também foi definido uma abordagem para gerenciamento de sistemas autoadaptativos, porém utilizando-se da programação de restrições e do modelo de objetivos baseado no "*Keep All Objectives Satisfied*" (KAOS) (LAMSWEERDE, 2009). A abordagem dos autores encontra a configuração ideal do produto para o contexto ativo pela execução de um solucionador de restrições.

Em Bezerra *et al.* (2016) foi desenvolvida uma ferramenta *desktop* para modelagem de MF de LPSD, chamada DyMMer. A DyMMer permite a edição de *features* e o gerenciamento

de restrições entre *features*. A ferramenta também dispõe de um conjunto de métricas de manutenibilidade que podem ser aplicadas aos MFs para avaliação da qualidade. Posteriormente, a ferramenta foi estendida para uma versão *web* [1] para corrigir problemas da versão anterior. Com a extensão, a ferramenta ganhou novas funcionalidades, porém, apesar de modelar os MFs de LPSD, a ferramenta não possui suporte a modelagem de agentes de contexto e o gerenciamento da variabilidade dinâmica.

Neste cenário, este estudo tem como objetivo o desenvolvimento de um método de adaptação para MFs de LPSD, que em tempo de execução, priorize configurações que melhor atendam as *features* do sistema. Além disso, o método será implementado como extensão da DyMMer, contribuindo para que a ferramenta possa oferecer aos engenheiros de aplicação e domínio, desenvolvedores e gestores de software um instrumento de gerenciamento, modelagem e manutenção de MFs de LPSD. Com relação a engenharia da LPSD, esse estudo fornecerá um meio consistente de representação da variabilidade em MF, com suporte a modelagem de agentes de contexto para gerenciamento da adaptação em tempo de execução e escolha da melhor configuração para o contexto ativo.

[1] https://dymmerufc.github.io/#/home

2 FUNDAMENTAÇÃO TEÓRICA

Nesta seção serão apresentados os principais conceitos que fundamentam a proposta do projeto e posterior desenvolvimento da pesquisa. Na Seção 2.1 é apresentada uma visão geral das LPS, compreendendo seus conceitos e a relevância no tocante a manutenibilidade do software. Na Seção 2.2, os conceitos referentes a LPSD são apresentados e nesse ponto é discutido como a LPSD estende a LPS e quais problemas ela resolve. A Seção 2.3 apresenta o MF como uma notação utilizada para modelar LPS e LPSD. A Seção 2.4 apresenta as principais abordagens utilizadas para gerenciamento de variabilidade em LPSD, incluindo uma visão sobre como o processo MAPE-K pode contribuir com a reconfiguração dinâmica em tempo de execução. Finalmente, a Seção 2.5 apresenta a ferramenta DyMMer *web*, suas propriedades, arquitetura e funcionalidades.

2.1 Linhas de Produtos de Softwares

As LPS são uma estratégia de desenvolvimento aplicada a uma família de produtos de software, cujas características e funcionalidades são semelhantes. Desse modo, aumentando o suporte ao reuso, permitindo o desenvolvimento de softwares com menor tempo de comercialização, custos inferiores e melhor qualidade (LEE *et al.*, 2012). As LPSs estão presentes nos mais diversos domínios e numa ampla variedade de empresas, desde grandes corporações à negócios com poucos desenvolvedores (HALLSTEINSEN *et al.*, 2008). Segundo Hinchey *et al.* (2012), a abordagem orientada a reuso fez da LPS uma tecnologia amplamente conhecida, tendo sido incorporada na indústria como um fator de sucesso no desenvolvimento de ativos reutilizáveis em uma gama de produtos de software.

Pohl *et al.* (2005) definiram a Engenharia de LPS como um "paradigma para desenvolvimento de aplicações de software usando plataformas e customização em massa". O uso de plataformas diz respeito à construção proativa de componentes de software voltados para o reúso, já a customização em massa indica o gerenciamento dos aspectos em comum e as diferenças da aplicação com relação aos seus requisitos, arquitetura, componentes e demais artefatos de software (POHL *et al.*, 2005). Esse gerenciamento ajuda a compreender como um software funciona, tornando mais fácil de implementar mudanças, adaptações e inclusões de novas funcionalidades, contribuindo para a manutenibilidade do sistema (POHL *et al.*, 2005).

O gerenciamento da variabilidade é apontado por Hallsteinsen *et al.* (2008) como o

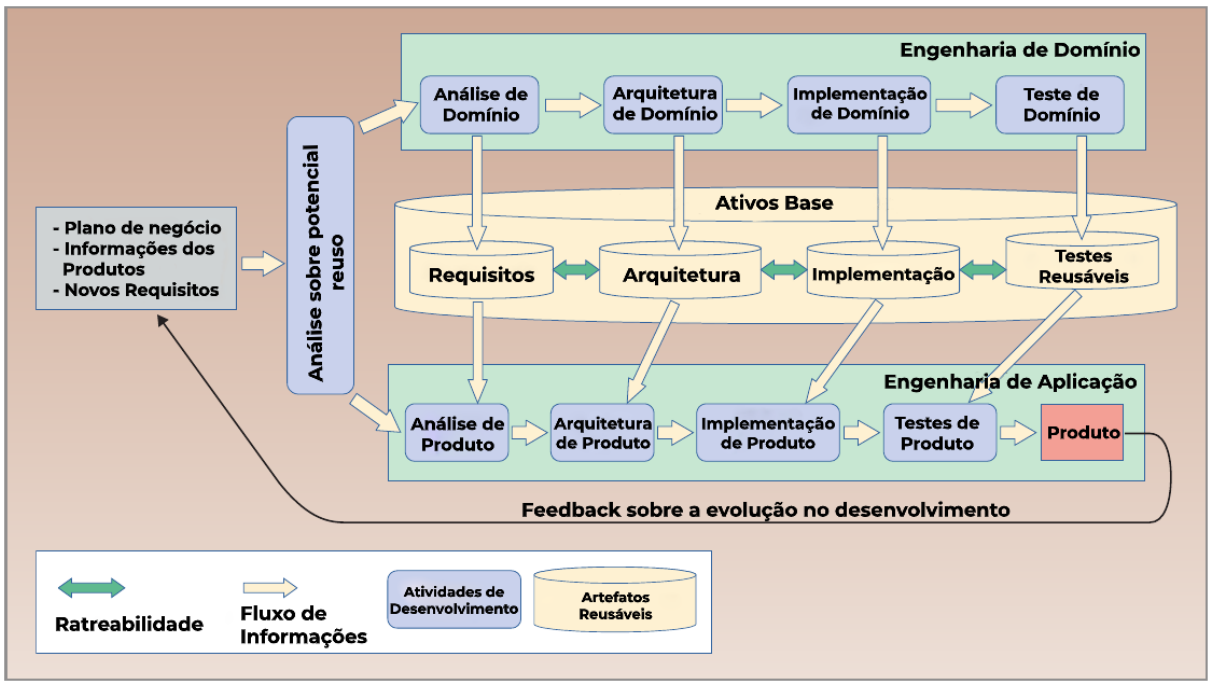
princípio fundamental de uma LPS. De acordo com os autores, esse gerenciamento envolve a separação de uma LPS em três partes: componentes comuns a todos os produtos, partes comuns em alguns produtos e o produto individual com seus requisitos específicos. Desse modo, os desenvolvedores podem identificar e evitar manter componentes que serão usados uma única vez, maximizando o reúso das partes do sistema. Hinchey *et al.* (2012) complementam apresentando quatro características importantes de uma LPS:

1. **Modelagem da variabilidade:** descreve de forma explícita as diferenças entre os produtos de uma LPS. Essa descrição inclui, por exemplo, as restrições que determinam quais características não podem estar presentes simultaneamente.
2. **Arquitetura de referência:** diferente de uma arquitetura de software padrão, a arquitetura de referência fornece personalizações explícitas que cobrem toda a LPS, além de suportar as variações descritas pelo modelo de variabilidade.
3. **Escopo do negócio:** O desenvolvimento de uma LPS exige o entendimento de que a arquitetura não diz respeito somente a um cliente ou projeto, mas a um domínio ou campo de negócios inteiro.
4. **Dois ciclos de vida:** diz respeito a distinção conceitual entre o *design* de componentes reusáveis, que sejam genéricos e facilmente acopláveis para diferentes produtos, e o desenvolvimento com reutilização, que descreve o desenvolvimento de produtos a partir dos ativos genéricos criados para reuso.

A Figura 1 apresenta um diagrama com a representação dos dois ciclos de vida de uma LPS, a engenharia do domínio e da aplicação. A **Engenharia do domínio** diz respeito a análise de toda a linha de produto, preocupando-se com a construção de componentes comuns e reutilizáveis, definindo o escopo geral de uma família de produtos de software. **A Engenharia de aplicação** está relacionado a derivação de produtos específicos a partir das definições da engenharia de domínio, fazendo-se uso dos componentes comuns e preocupando-se com a integração de todas as partes do produto. Cada ciclo de vida pode ter processos diferentes de acordo com as necessidades da organização e os objetivos dos produtos desenvolvidos (HALLSTEINSEN *et al.*, 2008).

Segundo Hinchey *et al.* (2012), a ideia principal de uma LPS é a de construir uma arquitetura de referência comum, com um conjunto bem projetado de componentes que se encaixem conforme o modelo de variabilidade. Nesse sentido, na engenharia de domínio, o modelo de variabilidade assume um papel de liderança ao descrever a faixa potencial de variações;

Figura 1 – Representação dos ciclos de vida de uma LPS.



Fonte: Adaptado de Hallsteinsen *et al.* (2008).

na engenharia de aplicação, fornece uma base para selecionar as características específicas do produto pretendido. Com isso, a arquitetura de referência garante que todas as configurações derivadas a partir de uma LPS correspondam a produtos que estão funcionando corretamente, além de fornecer mecanismos para compor adequadamente as várias partes do produto em diferentes momentos durante o desenvolvimento (HINCHEY *et al.*, 2012).

Ao longo dos anos, o surgimento de novos domínios, tais como computação ubíqua, serviços robóticos, exploração espacial e diversos outros, tornaram os softwares mais complexos devido a grande quantidade de variações, requisitos e restrições de recursos requeridos (HALLSTEINSEN *et al.*, 2008). A computação moderna começou a demandar sistemas com alto grau de adaptabilidade e fez surgir a necessidade de construir softwares capazes de mudar dinamicamente a disponibilidade de seus recursos em tempo de execução (HALLSTEINSEN *et al.*, 2008). Como as LPSs gerenciam a variabilidade em tempo de desenvolvimento, foi necessário desenvolver novas abordagens que suportassem a variabilidade das *features* durante a execução do sistema (BASHARI *et al.*, 2017), surgindo então o conceito de LPSD como uma extensão da LPS comum.

2.2 Linha de Produtos de Software Dinâmicas

O conceito de LPSD surge como uma extensão da LPS tradicional projetada para modelar sistemas com características especiais, tais como os sensíveis ao contexto, que realizam um processo de reconfiguração ou adaptação em tempo de execução (BASHARI *et al.*, 2017). Segundo Hinchey *et al.* (2012), ao ampliar as abordagens existentes da engenharia de linhas de produtos, as LPSD estendem também práticas e métodos já testados em diversas aplicações, fundamentando-se em engenharia sólida, praticável e comprovada. Segundo Sousa *et al.* (2019), as LPSD exigem mais atenção no processo de definição e modelagem das *features*, uma vez que as mudanças ocorrem durante a execução do software e problemas em tempo real podem impactar significativamente na disponibilidade e na qualidade do serviço do software.

Para Hallsteinsen *et al.* (2008), uma LPSD distingue-se de uma LPS em pelo menos duas características: reconfiguração em tempo de execução e autogerenciamento. A reconfiguração em tempo de execução permite que o software mude de acordo com as novas características do ambiente, já o autogerenciamento indica a habilidade de tomar as decisões de forma automática, sem interferência do usuário ou do desenvolvedor (HALLSTEINSEN *et al.*, 2008). De forma geral, produtos derivados a partir de uma LPS tem configurações únicas para mercados ou indivíduos específicos, mas o ponto principal é que as mudanças não ocorrem depois do desenvolvimento, já uma LPSD pode ser reconfigurada durante a execução do produto de software a partir de um conjunto de regras estabelecidas, em outras palavras em tempo de execução ocorre o processo de derivação de produtos, semelhantes a LPS em tempo de desenvolvimento (BENCOMO *et al.*, 2012).

Muitas técnicas de reuso, gerenciamento da variabilidade e modelagem de LPS são aplicáveis as LPSD (BENCOMO *et al.*, 2012). A variabilidade de uma LPSD é definida pela quantidade de recursos opcionais ativos e embora esse número possa ser grande, é um valor fixo (OLAECHEA *et al.*, 2018). Sistemas baseados em LPSD precisam lidar com as mudanças nas necessidades dos usuários e do ambiente de execução de maneiras não previstas no desenvolvimento (BENCOMO *et al.*, 2012). Como o ambiente em que uma LPSD está executando alterações, a configuração também pode ser alterada, a fim de manter a melhor qualidade de serviço (OLAECHEA *et al.*, 2018). Nesse sentido, suportar o gerenciamento em tempo de execução implica em saber lidar com as mudanças inesperadas do ambiente (BASHARI *et al.*, 2017), o que inclui o gerenciamento de falhas e todo o conjunto de fatores que impactam diretamente na qualidade do serviço oferecido. Para garantir a satisfação dos requisitos do

contexto, uma LPSD precisa encontrar a melhor variação e então adaptar o sistema, ativando e desativando os recursos durante a execução do sistema (OLAECHEA *et al.*, 2018).

Segundo Bashari *et al.* (2017), as propriedades de reconfiguração em tempo de execução são obrigatórias para um produto no contexto da LPSD. A reconfiguração em tempo de execução diz respeito à capacidade do produto de ativar e desativar recursos, de acordo com o modelo de variabilidade, em tempo de execução (BASHARI *et al.*, 2017). Esse processo de reconfiguração ocorre durante toda a vida útil do sistema e esse ponto evidencia a necessidade do sistema de lidar com situações inesperadas (BASHARI *et al.*, 2017). Bashari *et al.* (2017), pontuam que a capacidade de um sistema ser autogerenciável ou "autoadaptável" e funcionar como um "tomador de decisão automático" não é uma obrigatoriedade de uma LPSD. Em outras palavras, um humano poderia tomar uma decisão no nível abstrato para executar uma reconfiguração – seja mudando as necessidades do sistema, ou disparando o processo de adaptação – desde que todos os aspectos da realização da reconfiguração (ativação e desativação de *features*) em tempo de execução, sejam realizadas pela LPSD (HINCHEY *et al.*, 2012).

Embora o comportamento autônomo não seja necessário para uma LPSD, a maioria das LPSD abordam também o monitoramento e a tomada de decisão (HINCHEY *et al.*, 2012). Neste sentido, monitorar o estado atual e controlar a adaptação torna-se uma tarefa central em uma LPSD (HALLSTEINSEN *et al.*, 2008). Desse modo, as LPSDs passam a ser uma abordagem sistemática de engenharia para realizar sistemas autoadaptativos (HINCHEY *et al.*, 2012). Essas características sugerem o uso de extensos modelos e mecanismos de variabilidade para o gerenciamento da adaptação em tempo de execução (BASHARI *et al.*, 2017). Com base nisso, essa pesquisa visa contribuir com novas soluções para o gerenciamento do processo de adaptação em LPSD, com ênfase no MF, uma notação para modelagem de LPS e LPSD que será melhor discutida na próxima Seção.

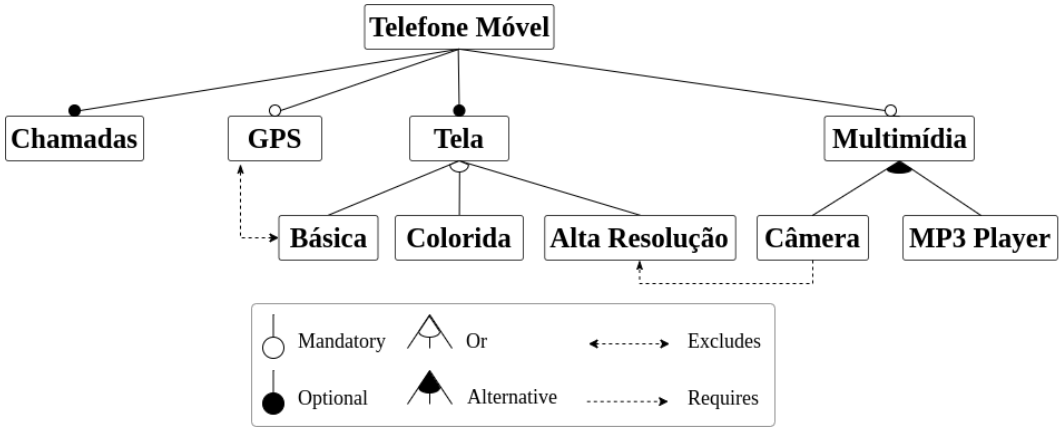
2.3 Modelo de *Features*

O MF vem sendo usado para representar e modelar a variabilidade de produtos de software desde o surgimento da LPS nos anos 1990 (CAPILLA; BOSCH, 2011). A adoção do MF na modelagem da LPS se dá pelo fato da abordagem auxiliar no gerenciamento e na validação individual da configuração dos produtos da LPS. Além disso, dependendo do nível de abstração, um MF pode permitir a visualização das características mais proeminentes do produto, contribuindo para as melhorias do software e para a análise de domínio (DAVRIL *et al.*, 2013).

De acordo com Tang e Leung (2017), ao contribuir com a análise de uma família de softwares e permitir o gerenciamento da variabilidade, o MF torna-se uma ferramenta fundamental para pequenas e grandes organizações evoluírem seus sistemas com menores custos, aumentando o suporte a manutenibilidade e encurtando o tempo de desenvolvimento.

De forma geral, o MF é um diagrama em formato de árvore que representa todo o conjunto de *features* hierarquicamente, representando também os relacionamentos e restrições entre as *features* do modelo (OLAECHEA *et al.*, 2018). A Figura 2 ilustra um MF simples inspirado na indústria de telefones móveis. No modelo é retratado como as *features* são usadas para especificar e construir softwares para telefones móveis. A partir do modelo é possível observar que todos os telefones móveis precisam incluir a *feature* **chamadas** e exibir informações em uma tela **básica**, **colorida** ou de **alta resolução**. Além disso, o produto pode incluir opcionalmente suporte a **GPS** e a **multimídia**, como **câmera**, **MP3 Player** ou ambos.

Figura 2 – Modelo de *features* de um Telefone Móvel.



Fonte: Adaptado de Benavides *et al.* (2010).

Com base na hierarquia do modelo, as *features* subjacentes só podem ser escolhidas se as *features* ancestrais estiverem incluídas no produto (BENAVIDES *et al.*, 2010). Além disso, o modelo inclui os seguintes relacionamentos:

1. *Mandatory* (Obrigatório) - quando uma *feature* filha está incorporada em todos os produtos no qual a *feature* pai aparece, dizemos que a *feature* filha tem um relacionamento obrigatório com a *feature* pai. Por exemplo, no modelo de *features* dos telefones móveis a *feature* **chamadas** estará presente em todos os produtos.
2. *Optional* (Opcional) - quando uma *feature* filha está inclusa opcionalmente nos produtos que o pai aparece, dizemos que elas têm um relacionamento opcional.

No modelo de *features* de exemplo, a *feature* **GPS** estará presente em alguns produtos derivados do modelo.

3. *Alternative/Xor* (Alternativa/Exclusiva) - quando há um conjunto de *features* filhas e somente uma pode ser selecionada em um produto no qual o pai aparece, dizemos que as *features* tem um relacionamento mutuamente exclusivo. Por exemplo, no modelo de *features* dos telefones móveis, um produto pode incluir suporte a uma tela **básica**, **colorida** ou de **alta resolução**, porém somente um dos tipos de tela pode ser incluído.
4. *Alternative/Or* - quando há um conjunto de *features* filhas e uma ou mais *features* podem ser selecionadas em um produto no qual o pai aparece, dizemos que as *features* tem um relacionamento alternativo. Por exemplo, na Figura 2 ao incluir **multimídia** em um produto é possível selecionar **câmera** ou **MP3 Player**, ou ambos.

Um MF também tem restrições entre os elementos da árvore (*cross-tree constraints*) que são especificados como restrições de inclusão (*require*) e de exclusão (*exclude*). No relacionamento de inclusão, uma *feature* A inclui uma *feature* B, quando a *feature* B precisa ser incluída no produto ao incluir a *feature* A. Na Figura 2 se um telefone móvel incluir a **câmera** é requerido a inclusão de uma tela de **alta resolução**. No relacionamento de exclusão, uma *feature* A exclui uma *feature* B, quando ambas as *features* não podem ser incluídas no mesmo produto de software. No MF dos telefones móveis, ao incluir **GPS** a opção de tela **básica** não pode ser incluída, pois ambas são incompatíveis.

O termo modelo de *features* foi inventado por Kang *et al.* (1990) no relatório *Feature Oriented Domain Analysis* (FODA) em 1990, e foi um dos principais tópicos na pesquisa em LPS desde então (PFANNEMULLER *et al.*, 2017). A notação do MF descrita é a base definida no FODA, mas ao longo do tempo diversas pesquisas estenderam ou incluíram novos relacionamentos ao modelo inicial de acordo com a necessidade de modelar novas situações do domínio ou de inserir novas informações sobre as *features* (BENAVIDES *et al.*, 2010). Os conceitos de **cardinalidade de *feature*** e **cardinalidade de grupo** são exemplos de extensões ao MF, a partir dos conceitos de multiplicidade da linguagem UML, motivados pela necessidade de suporte a modelagem de aplicações com novas características (BENAVIDES *et al.*, 2010).

– A cardinalidade de *feature* diz respeito a uma sequência de intervalos denotada por [n,..,m] no qual **n** é o limite inferior e **m** o limite superior. Em outras palavras, esse intervalo

determina o número de *features* que podem fazer parte do produto. Esse relacionamento pode ser usado na generalização dos relacionamentos obrigatórios ([1, 1]) e opcionais ([0,1]) da notação original definida no FODA (BENAVIDES *et al.*, 2010).

– A cardinalidade de grupo também diz respeito a uma sequência de intervalos denotada por <n,..,m> no qual **n** é o limite inferior e **m** o limite superior, porém esse relacionamento é usado para limitar o número de features filhas que podem ser incluídas no produto quando a feature pai é selecionada. Desse modo, o **relacionamento alternativo** é equivalente a cardinalidade de grupo <1, 1>, enquanto o **relacionamento "ou"** é equivalente a cardinalidade de grupo <1, n> (BENAVIDES *et al.*, 2010).

Segundo Benavides *et al.* (2010), não há um consenso sobre a notação dos atributos adicionados aos MF, mas a maioria das propostas realizadas nesse sentido concordam que os atributos precisam ter um nome, um domínio e um valor. Os atributos também podem receber um tipo (por exemplo, número inteiro, *string* ou uma enumeração predefinida), para que os respectivos valores possam ser originários apenas dos domínios dos tipos (MAURO *et al.*, 2018). Além disso, os MF estendidos podem incluir restrições mais complexas ao sobre a inclusão de *features* com base no valor de determinados atributos de outras *features* do modelo.

Na modelagem tradicional, um conjunto de funcionalidades de uma família de produtos de software é agrupado de acordo com suas características e em pequenas unidades do sistema, de modo que as variações sejam gerenciadas de forma mais eficiente e a identificação de variabilidade se torne mais precisa (CAPILLA; BOSCH, 2011). No contexto da LPS, identificar configurações válidas é extremamente importante para permitir a customização em massa (CAPILLA; BOSCH, 2011). Uma configuração é válida se não contradizer nenhuma das restrições impostas pelo MF, incluindo as restrições estruturais (MAURO *et al.*, 2018). Em tempo de desenvolvimento, ferramentas de análise automatizada podem ajudar os desenvolvedores a encontrar configurações inválidas, porém, essa análise não é possível em tempo de execução (CAPILLA; BOSCH, 2011).

O MF também pode se utilizado para modelar LPSD, porém algumas adaptações podem ser necessárias, tendo em vista que LPS e LPSD têm diferenças importantes com relação ao gerenciamento de variabilidade. Ao representar uma LPSD o MF precisa suportar a reconfiguração em tempo de execução, ou seja, a ativação e desativação de recursos de forma autônoma e em tempo real durante todo o ciclo de vida do produto (BENCOMO *et al.*, 2012). Esse gerenciamento pode ser um desafio, considerando que o MF de uma LPS grande pode conter

centenas de restrições (BERGER *et al.*, 2013) e um número elevado de variações se tiver muitas *features* opcionais (OLAECHEA *et al.*, 2018). Desse modo, se escolher a melhor configuração pode ser uma tarefa complexa em tempo de desenvolvimento, em tempo de execução, pode impactar significativamente na qualidade do serviço (TARTLER *et al.*, 2011).

As restrições podem impactar no desempenho do modelo se ambíguas ou em um número muito elevado, mas a definição delas é importante para guiar o modelo durante o processo de adaptação (PFANNEMULLER *et al.*, 2017). Como um modelo pode conter muitas variações, a definição de restrições entre *features* e os demais recursos do sistema funcionam como um mapeamento entre os contextos e o processo de reconfiguração, permitindo que os mecanismos de adaptação mudem a configuração do sistema em tempo de execução (PFANNEMULLER *et al.*, 2017).

Segundo Baresi *et al.* (2012), se as necessidades do sistema mudam em tempo de execução, alterar a configuração do sistema pode não ser suficiente para atender as restrições ou as novas necessidades do contexto, uma vez que o sistema pode precisar lidar com situações imprevistas, tendo que alterar as funcionalidades ou atributos do MF para acomodar a nova variabilidade. Nesse sentido, para garantir que a nova configuração satisfaça as necessidades do sistema é preciso identificar e monitorar as variáveis de contexto, determinar quando acionar o processo de adaptação e verificar como o sistema se comporta após a adaptação (SAWYER *et al.*, 2012). Desse modo, o MF torna-se uma entidade de tempo de execução, capaz de monitorar todos os aspectos necessários à manutenção da qualidade do serviço (BARESI *et al.*, 2012).

Este trabalho também se propõe a estender o MF para representar atributos e informações adicionais sobre os relacionamentos e restrições entre *features*. O modelo proposto será apenas de uso lógico do método de adaptação, não se trata da criação de uma nova notação para o MF, mas de uma adequação que forneça ao método um conjunto de informações relevantes sobre o ambiente e as *features*. Para realização desse processo, será necessário analisar o MF na notação usual e identificar informações que podem ser adicionadas, tais como informações de contexto, restrições entre **features**, dentre outras. Essas informações serão usadas em tempo de execução no processo de reconfiguração dinâmica, cujo conceito será apresentado na seção à seguir.

2.4 Reconfiguração Dinâmica

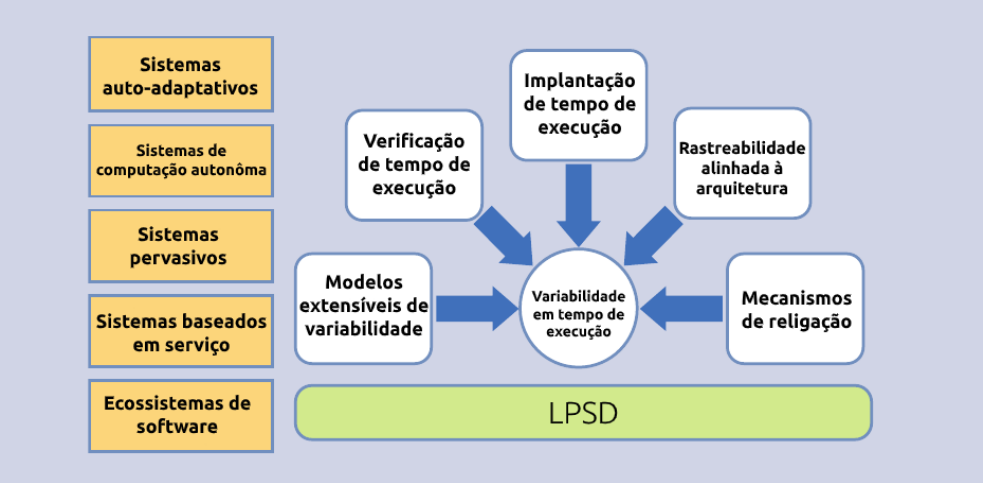
A computação e internet moderna passaram a demandar dos sistemas um alto grau de adaptabilidade, evidenciando a necessidade dos softwares alterarem a disponibilidade dos recursos em tempo de execução (HALLSTEINSEN *et al.*, 2008). Além disso, os sistemas precisam ter a habilidade de executar continuamente, mesmo em ambientes severos com condições adversas, falhas parciais e mudanças nas necessidades dos usuários (HINCHEY *et al.*, 2012). Para lidar com esses novos desafios, novas abordagens foram propostas, tais como "baseados em agentes, autônomos, sistemas auto-adaptáveis, emergentes e de inspiração biológica" (HINCHEY *et al.*, 2012). Diante disso, a adaptação em tempo de execução passou a ser essencial para o desenvolvimento da maioria dos sistemas de software (BARESI *et al.*, 2012).

Neste trabalho, a reconfiguração dinâmica será abordada como a capacidade de um sistema de gerenciar a variabilidade dinâmica durante a execução da aplicação. Segundo Capilla e Bosch (2011), a variabilidade dinâmica define as opções de design do produto visíveis para clientes e usuários do sistema, que podem selecionar entre as opções configuráveis disponíveis. Segundo Bashari *et al.* (2017), desenvolver sistemas adaptativos em tempo de execução é uma tarefa complexa e desafiadora. No mesmo caminho, Capilla e Bosch (2011) apresenta 3 desafios para a variabilidade dinâmica, a saber:

- representar a variabilidade dinâmica, alterando pontos de variação nas unidades de software durante a execução do sistema e automatizando a reconfiguração do sistema.
- validar e verificar as novas configurações de forma automatizada em tempo de execução de modo a manter a consistência e a estabilidade do sistema; e
- automatizar a implantação dos produtos reconfigurados dinamicamente em tempo de execução, evitando ao máximo as interrupções do sistema.

A Figura 3 ilustra quais são os mecanismos necessários para que uma LPSD possa suportar a variabilidade dinâmica em tempo de execução e algumas das abordagens que implementam essas características (CAPILLA; BOSCH, 2011). Segundo Capilla e Bosch (2011), as variações realizadas em tempo de execução são limitadas a ativação e desativação de recursos. Se for necessário adicionar *features* ao modelo, é preciso realizar uma análise para determinar o local a se inserir esse novo recurso, além disso, quaisquer alterações de adição ou remoção vão exigir uma verificação de compatibilidade das configurações do produto. Nesse sentido, construir um modelo puramente autônomo é um completo desafio, tendo em vista que algumas etapas podem exigir intervenção humana (CAPILLA; BOSCH, 2011).

Figura 3 – Mecanismos de variabilidade em tempo de execução necessários para LPSD.



Fonte: Adaptado de Capilla e Bosch (2011).

Com relação ao gerenciamento da variabilidade, já foi bastante discutido que a identificação e o monitoramento das variações do contexto e das demais informações computacionais, são o caminho para compreender as necessidades do ambiente e o estado do produto, para conduzir os processo de adaptação do sistema (BENCOMO *et al.*, 2012). Todos esses desafios da variabilidade dinâmica em tempo de execução contribuíram para que diversas pesquisas começassem a combinar ferramentas, técnicas e metodologias para se atingir os objetivos dos sistemas auto-adaptativos (BENCOMO *et al.*, 2012). O uso de técnicas como o **desenvolvimento baseado em componentes** (CBD, do inglês *Component-based Development*) e as **arquiteturas orientadas** a **serviços** (SOA, do inglês *Service-oriented Architectures*) são exemplos de combinações aplicadas a LPSD para gerenciamento de variabilidade (BENCOMO *et al.*, 2012).

2.4.1 Abordagens para Reconfiguração

Como apresentado anteriormente, os desafios para gerenciamento da variabilidade dinâmica motivaram muitos pesquisadores a combinar as práticas em LPSD com mecanismos de variabilidade em tempo de execução (BENCOMO *et al.*, 2012). Ao combinar as abordagens, uma LPSD adquire características importantes, que variam desde a capacidade de reconfigurar por mudanças do contexto, a possibilidade de adicionar ou remover features, recuperação de falhas e readaptação baseada na mudanças nos recursos do modelo (BENCOMO *et al.*, 2012). A seguir são descritas algumas das principais abordagens identificadas na literatura:

– **Abordagens baseadas em regras**: em tempo de desenvolvimento é realizado uma análise e planeamento das tarefas, a fim de se obter um conjunto de regras e adaptações na forma

evento-condição-ação (ECA) (BENCOMO *et al.*, 2012). Um conjunto de regras ECA compreende um conjunto de etapas de configuração com base em eventos de disparo, condições de proteção e as configurações atuais do sistema (BENCOMO *et al.*, 2012). Quando ocorrem eventos de disparo, uma condição de proteção é mantida enquanto o sistema executa as etapas de reconfiguração. De acordo com (BENCOMO *et al.*, 2012), os benefícios dessa implementação incluem a análise e validação do comportamento da adaptação em tempo de desenvolvimento para ganhar em performance durante a execução do sistema. Porém, a abordagem requer que todas as configurações e adaptações sejam identificadas em tempo de desenvolvimento, além disso o número de regras pode ser muito grande o que pode dificultar a análise em tempo de execução (BENCOMO *et al.*, 2012).

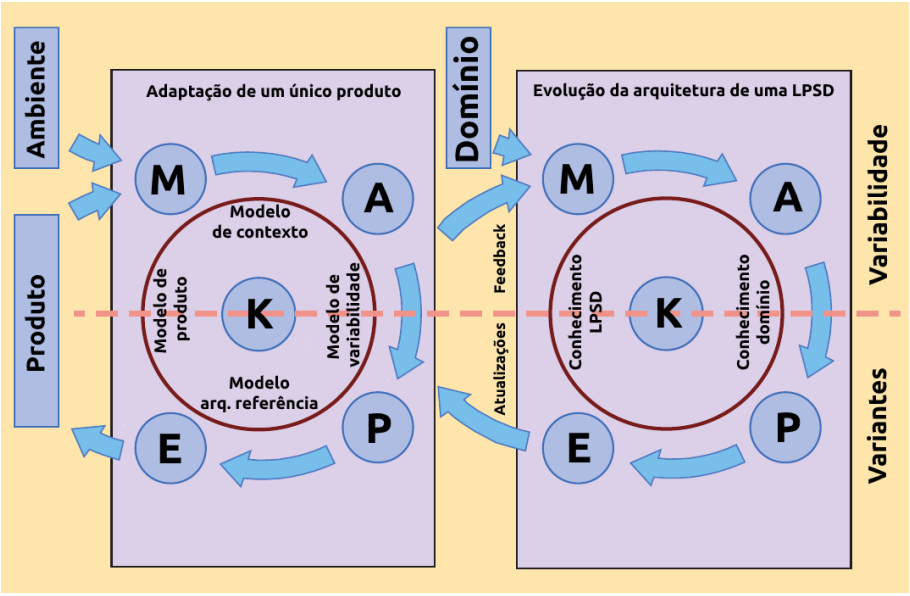
- **Abordagem baseadas em objetivos**: o modelo de decisão é mais abstrato, a análise e planejamento ocorrem em tempo de execução. Esse tipo de abordagem foca na maximização do objetivo, utilizando restrições explícitas para descartar configurações inválidas (BENCOMO *et al.*, 2012). Apesar de evitar o problema relacionado a enumeração de variáveis em tempo de desenvolvimento, acaba por elevar o custo em tempo de execução pela sobrecarga do sistema para analisar qual configuração melhor atende ao objetivo do contexto (BENCOMO *et al.*, 2012).
- **Abordagens baseadas em serviços**: segundo Lee *et al.* (2012), a natureza dinâmica dessa abordagem, de suportar as necessidades e expectativas do usuário em um ambiente de constante mudanças, auxilia a LPSD no processo de reconfiguração. Além disso, LPSD que utilizam essa abordagem pode combinar várias configurações e contextos, o que simplifica a implantação de variantes do produto a partir das necessidades do usuário (LEE *et al.*, 2012). Sistemas baseados em serviços auxiliam projetistas a entender quais qualidades seriam mais benéficas ao sistema (BARESI *et al.*, 2012), porém precisam evoluir dinamicamente em um ambiente altamente auto-adaptativo (LEE *et al.*, 2012).

O método a ser implementando nesse estudo realizará o mapeamento das restrições entre *features* e as correlações entre as *features* e agentes de contexto, utilizando essas informações para realizar o processo de reconfiguração. Como o mapeamento condiciona a ativação das *features* a um conjunto de critérios, conclui-se que o método está dentro do grupo das abordagens baseadas em regras.

2.4.2 MAPE-K

O processo de adaptação proposto nesse trabalho permeia a abordagem baseada em regras, com a inclusão do ciclo MAPE-K para monitoramento e tomada de decisões do modelo. A Figura 4 ilustra a utilização do MAPE-K em uma LPSD. O lado superior da Figura 4 ilustra como os produtos em uma LPSD podem ser reconfigurados dinamicamente no tempo de execução depois da sua derivação inicial estimulado pelo *loop* MAPE-K (BENCOMO *et al.*, 2012). A reconfiguração é baseada em modelos (o elemento K) derivados e representados de acordo com as práticas clássicas de engenharia da LPS, o que significa que partes da infraestrutura da LPS estão presentes no tempo de execução (BENCOMO *et al.*, 2012).

Figura 4 – Modelo conceitual de uma LPSD, que combina a LPS com o modelo MAPE-k para computação autonôma.



Fonte: Adaptado de Bencomo *et al.* (2012).

O lado direito da Figura 4 representa o processo de engenharia de domínio de uma LPSD, segundo Bencomo *et al.* (2012), esse processo é uma instanciação adicional ao *loop* MAPE-K que se concentra na evolução da LPSD, a partir do *feedback* do produto implantado e do domínio geral da aplicação. O lado direito representa a adaptabilidade aberta, que trata da variação de contexto não prevista em tempo de desenvolvimento e que requer uma extensão do modelo, uma evolução a longo prazo. O lado esquerdo da Figura ilustra a adaptabilidade limitada, que diz respeito à variação do contexto prevista no desenvolvimento acomodada pelos modelos LPSD (BENCOMO *et al.*, 2012). Abaixo é apresentado como o *loop* MAPE-K pode ser implementado para monitorar e gerenciar o processo de adaptação em LPSD:

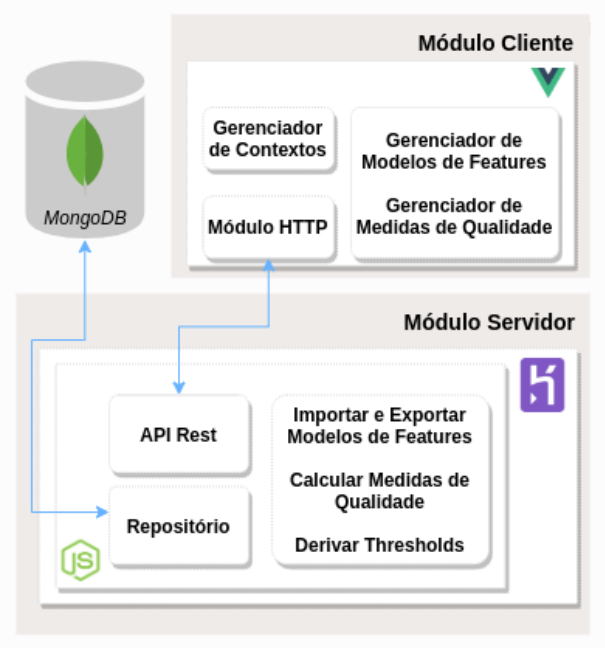
- **Monitoramento**: obtém os dados brutos dos recursos, variáveis de contexto, sensores, informações computacionais externas e de todos os recursos gerenciados. Os dados coletados são preparados e enviados ao **analisador**.
- **Analisador**: após receber os dados do monitoramento, o analisar realiza uma série de procedimentos e agrupamento dos dados de acordo com regras ou outros parâmetros pré-estabelecidos. Nesse ponto, relacionamentos entre as informações de contexto e recursos de contexto podem ser usados para selecionar atributos que representem o estado do sistema naquele momento. Informações do módulo de **conhecimento** podem ser usadas nessa etapa. Esses atributos são então enviados ao **planejador**.
- **Planejador**: faz uso dos atributos e demais informações do sistema para selecionar configurações válidas que atendam as necessidades do sistema. Se muitas configurações forem encontradas, atributos dos recursos, como informações de prioridade (PFANNEMULLER *et al.*, 2017), poderão ser considerados no momento da seleção de um recurso. O planejador também precisa saber lidar com a possibilidade de não haver configurações válidas. O resultado dessa etapa é uma lista de recursos do sistema que serão gerenciados, ou seja, a configuração selecionada que será enviada ao módulo de **execução**.
- **Executor**: responsável por realizar as mudanças no componente gerenciado, ou seja, realiza a ativação ou desativação dos recursos com base na configuração recebida do módulo de **planejamento**.
- **Conhecimento**: conjunto de informações do sistema e das etapas do *loop* que são armazenadas e que representam uma base de conhecimento do sistema sobre si mesmo. Esses dados podem ser utilizados para realizar melhorias em todas as etapas, devendo ser atualizada durante todo o ciclo de vida do software.

2.5 Ferramenta DyMMer Web

O presente trabalho tem como objetivo incorporar o método de adaptação como uma extensão da ferramenta DyMMer Web. A ferramenta foi inicialmente desenvolvida como uma aplicação *desktop*, utilizando a plataforma J2SE (*Java 2 Standard Edition*) para modelagem e avaliação da manutenibilidade de LPS e LPSD (BEZERRA *et al.*, 2016). Posteriormente foi realizada uma implementação da ferramenta para ser utilizada na Web. A ferramenta DyMMer Web é produto do Programa Institucional de Bolsas de Iniciação em Desenvolvimento Tecnológico e Inovação (PIBITI).

Segundo Bezerra *et al.* (2016), a versão da DyMMer para *desktop* suportava as seguintes funcionalidades: a) Importação do modelo de *features* construídos e exportados a partir do S.P.L.O.T.; b) Aplicação de medidas de qualidade para avaliação da manutenibilidade do modelo; c) visualização e edição de contextos, com a ativação e desativação de *features* para LPSD; d) Gerenciamento de restrições do modelo de *features* para evitar incompatibilidades entre as *features* na escolha de uma configuração; e) Exportação dos resultados da aplicação das métricas de qualidade para posterior análise.

Figura 5 – Arquitetura da DyMMer Web.



Fonte: Elaborado pelo autor.

A versão da DyMMer Web suporta todas as funcionalidades da versão *desktop*, algumas características foram melhoradas e novas funcionalidades foram adicionadas, tais como: i) Criação do modelo de *features*; ii) visualização detalhada dos contextos e gerenciamento avançado de ativação e desativação de *features*; iii) aplicação de métricas com análise via thresholds; iv) persistir os modelos de *features* criados ou editados na nuvem.

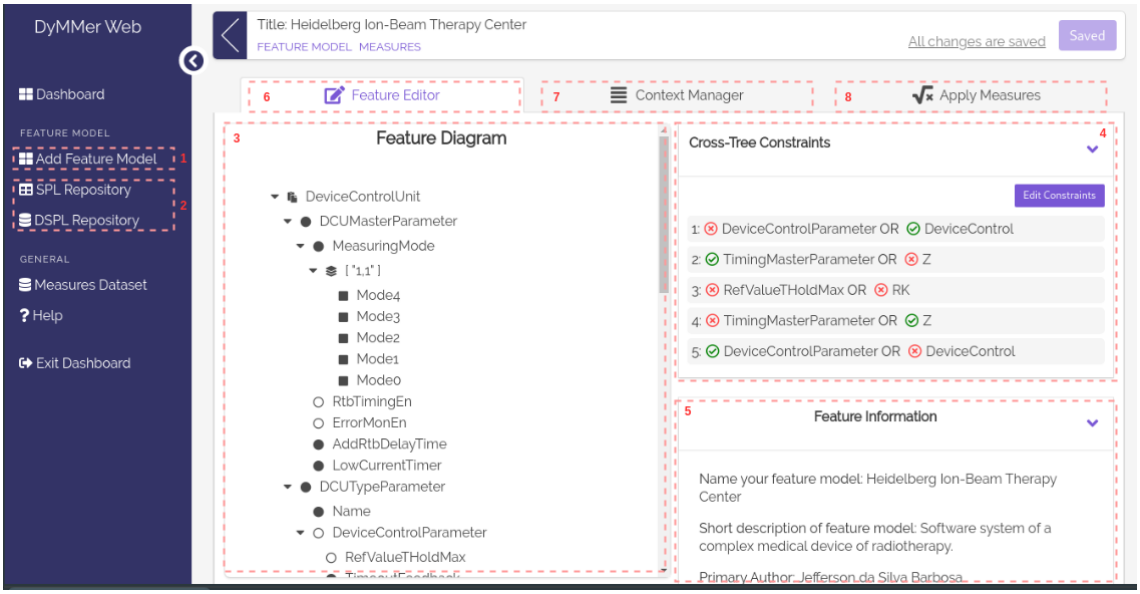
A Figura 5 ilustra a arquitetura da DyMMer Web. Diferente da versão *desktop* que foi construída para uso offline e em uma única aplicação, a versão web foi construída utilizando a arquitetura cliente-servidor, além de um banco de dados para salvar os modelos de *features* e métricas computadas. A camada do cliente foi desenvolvida utilizando-se o *framework* JavaScript VueJs com *framework* CSS Bulma, essa camada é responsável pela:

1. Criação e edição de modelos de *features* LPS e LPSD.

2. Importação de modelos de *features* LPS e LPSD.
3. Criação e gerenciamento de contextos de modelos de *features* LPSD.
4. Escolha de métricas para aplicação ao modelo de *features* do editor.

A camada do servidor foi desenvolvida em JavaScript utilizando-se o *framework* Node.Js. A camada de servidor é responsável por receber as requisições do cliente para processamento de fluxo de dados mais complexos, tais como a importação e exportação de modelos de *features* a partir do formato XML (formato de exportação da ferramenta S.P.L.O.T), a computação das métricas de manutenibilidade e a derivação dos *thresholds* a partir do repositório de métricas e modelos de *features*. Os modelos de *features* são persistidos na nuvem em um banco de dados não-relacional gerenciado pela tecnologia MongoDB.

Figura 6 – Visão geral das funcionalidades da DyMMer Web.



Fonte: Elaborado pelo autor.

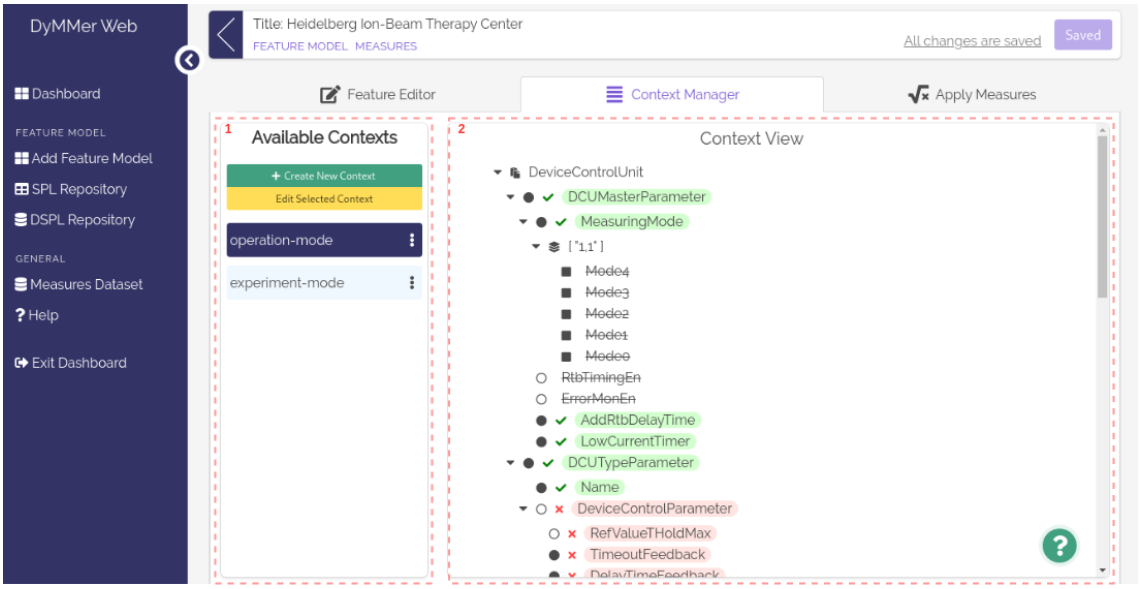
A Figura 6 apresenta a ferramenta DyMMer Web com as principais funcionalidades destacadas:

1. Adicionar MF: o modelo pode ser adicionado de duas formas, a primeira é importando um arquivo XML com a estrutura definida no S.P.L.O.T.. A outra maneira é criando um modelo do zero, então o desenvolvedor pode criar seu próprio MF e depois salvá-lo no repositório.
2. Repositório de LPS e LPSD: uma lista de MF de LPS e LPSD cadastrados no banco de dados da ferramenta.
3. Diagrama de *features*: exibe a árvore de *features* de uma LPS ou LPSD acordo

com a notação do MF definida no FODA, incluindo também informações de cardinalidade entre os grupos de *features* . A partir do diagrama é possível adicionar, remover ou editar as *features* e seus atributos.

4. Restrições do modelo: exibe uma lista de restrições aplicadas às *features* do modelo. Esse módulo permite a edição das restrições, podendo adicionar novas ou remover as existentes.
5. Informações do MF: informações adicionais sobre o modelo. Inclui os dados da instituição e do desenvolvedor responsável por criar o modelo de *features*.
6. Aba de edição do MF: nesse módulo é possível realizar as alterações em termos de estrutura de uma LPS e de uma LPSD.
7. Aba de gerenciamento do contexto: módulo de edição dos contextos de uma LPSD, é possível inserir informações de contexto, editar restrições e indicar quais *features* estarão ativas ou desativadas para cada contexto.
8. Aba de aplicação e visualização das métricas do MF: nesse módulo é realizado a aplicação de um conjunto de 40 métricas de manutenibilidade. Após a computação das métricas, é exibido o resultado do processo de modo que o desenvolvedor possa analisá-lo.

Figura 7 – Visão geral do gerenciamento de contextos.

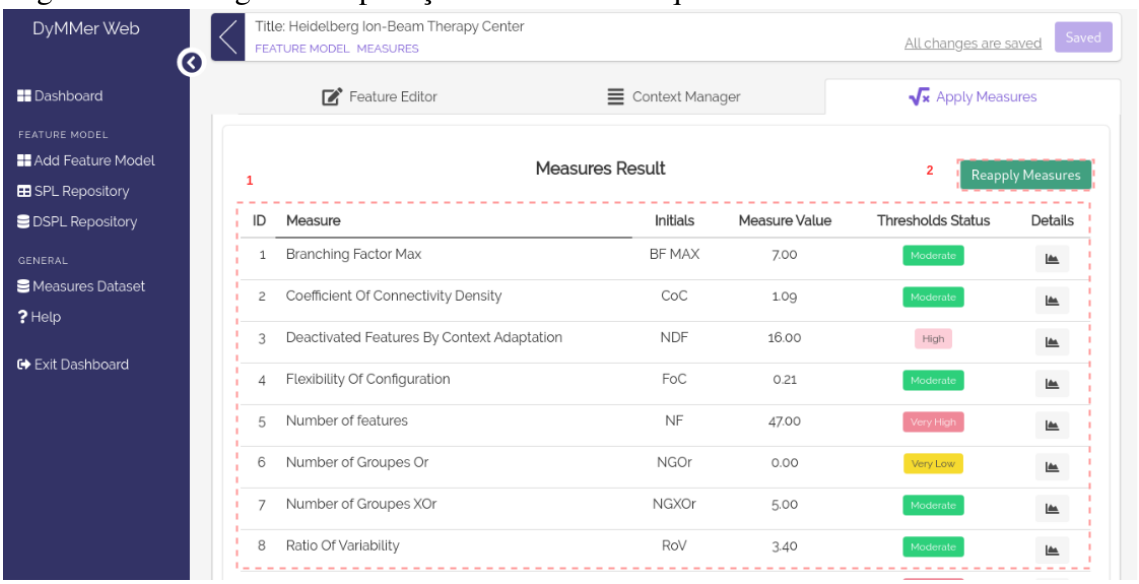


Fonte: Elaborado pelo autor.

Na Figura 7 esta em destaque o módulo de gerenciamento dos contextos do MF. Esse módulo é subdividido em 2 blocos: 1) Gerenciamento de contexto: neste bloco é realizado o processo de edição de contextos, incluindo a adição de novos ou exclusão de contextos existente;

2) Gerenciamento de variabilidade: neste bloco é possível indicar quais *features* do modelo estarão ativas ou desativadas para um determinado contexto, além de ser possível excluir as *features* que não farão parte do contexto.

Figura 8 – Visão geral da aplicação de métricas de qualidade ao modelo.



Fonte: Elaborado pelo autor.

A Figura 8 ilustra o módulo de aplicação das métricas de manutenibilidade presentes na ferramenta 1) Exibe a lista de medidas de qualidade computadas, com nome da métrica, abreviação, o resultado absoluto da métrica e o *status* de acordo com um conjunto de *thresholds* definidos e gerenciados pela ferramenta; 2) Botão para solicitar ao módulo a computação das métricas, a ferramenta espera para exibir o resultado na tela.

3 TRABALHOS RELACIONADOS

Esta seção apresenta os trabalhos relacionados ao estudo, que foram selecionados a partir de uma revisão bibliográfica. Nas seções a seguir, cada trabalho é apresentado e discutido.

3.1 Abordagem para reconfiguração em tempo de execução utilizando informações de desempenho

Weckesser *et al.* (2018) desenvolveu uma abordagem para a reconfiguração dinâmica de LPSD que, em tempo de execução, utiliza informações de desempenho no processo de escolha da melhor configuração para o contexto atual. Essas informações de desempenho são coletadas por meio de um treinamento do modelo em sistemas reais, as simulações permitem que em tempo de execução, os dados coletados auxiliem no entendimento do contexto e das necessidades atuais do sistema e então combine essas informações com outros dados (especialistas do domínio da aplicação, propriedades não-funcionais) para gerar uma configuração ótima do modelo (WECKESSER *et al.*, 2018).

O presente estudo pretende desenvolver uma abordagem para realização do processo de reconfiguração dinâmica em tempo de execução, se assemelhando ao trabalho de Weckesser *et al.* (2018). Porém, a abordagem definida nesse estudo não utilizará dados de simulações para treinar o modelo, e considerará informações de restrição entre *features*, configurações e recursos no processo de escolha de uma configuração adequada ao contexto atual do ambiente de execução do software.

3.2 Abordagem para monitoramento e verificação de configurações em sistemas dinamicamente adaptativos

Santos *et al.* (2019) propõem uma abordagem para monitoramento de Sistema Dinamicamente Adaptativos (SDA). O objetivo principal da ferramenta é de procurar falhas durante o processo de adaptação por meio da análise dos valores comportamentais das propriedades do sistema. A abordagem permite também a execução de simulações do contexto enquanto o ambiente de execução real não está pronto. Para análise das propriedades, a abordagem usa uma versão estendida do MF que mapeia a influência dos contextos nos recursos do sistema. Com isso, é possível especificar regras de adaptação e um conjunto de configurações relevantes definidas pelo engenheiro de software.

O trabalho de Santos *et al.* (2019) se relaciona com este estudo na análise da influência dos contextos nos recursos do sistema e na definição de regras de adaptação. Embora o trabalho de Santos *et al.* (2019) explore o processo de reconfiguração dinâmica, o objetivo da abordagem não é disparar a adaptação, mas o de fazer um processo contínuo de verificação das propriedades e auxiliar na escolha da melhor configuração do sistema. Já a abordagem proposta no presente trabalho, tem como objetivo realizar o processo de adaptação a partir de um conjunto de premissas que serão definidas ao longo da pesquisa.

3.3 Utilizando programação por restrições e *softgoals* para gerenciamento de sistemas auto-adaptativos

Sawyer *et al.* (2012) propôs uma abordagem para gerenciamento de sistemas auto-adaptativos, utilizando-se da programação de restrições e do modelo de objetivos baseado no KAOS. Esse modelo de objetivos foi utilizado na descrição do processo de escolha das configurações. A abordagem dos autores encontra a configuração ideal do produto para o contexto ativo pela execução de um solucionador de restrições. A abordagem também permite a verificação estática do modelo de objetivos, os dados coletados são armazenados para auxiliar no monitoramento em tempo de execução e fornecer uma base de conhecimento do modelo.

Diferente da abordagem apresentada neste trabalho, a de Sawyer *et al.* (2012) utilizam *softgoals* aliado a outras técnicas na modelagem de uma LPSD e representação dos atributos relevantes para o processo decisório em tempo de execução. O presente estudo utilizará o MF na modelagem de uma LPSD e seus atributos. Ambas as abordagens utilizam a programação por restrições para encontrar a configuração ideal. Por fim, a abordagem proposta no estudo utilizará o ciclo MAPE-K para monitoramento das mudanças, execução dos processos decisórios e construção de uma base de conhecimento do modelo.

3.4 ProDSPL, uma abordagem proativa que antecipa as variações e escolhe a melhor configuração

Ayala *et al.* (2021) propôs uma abordagem proativa, chamada ProDSPL, que antecipa as futuras variações do sistema para gerar a melhor configuração da LPSD. A abordagem utiliza um controlador proativo que que gera configurações válidas em tempo de execução. Segundo Ayala *et al.* (2021), antecipar as variações futuras promove adaptações que são boas para o

contexto por mais tempo, reduzindo o número de reconfigurações necessárias, tornando o sistema mais estável.

A desvantagem do modelo proativo adotado pela ProDSPL, é que muitas vezes as configurações escolhidas tem qualidade inferior em comparação a uma solução ideal, dentre as possibilidades de reconfiguração. O método apresentado nesse estudo utiliza a técnica da reatividade, não faz previsões das reconfigurações e gera a melhor configuração para o contexto com base em um conjunto de regras pré-definidas.

3.5 Metodologia que utiliza lógica para raciocinar formalmente sobre as decisões de reconfiguração

Göttmann *et al.* (2020) desenvolveu uma metodologia baseada em modelo para especificar e analisar automaticamente restrições, em tempo real, de decisões de reconfiguração de uma forma composicional e orientada a *feature*. A metodologia utiliza autômatos cronometrados para traduzir especificações da LPSD em tempo real, para raciocinar formalmente sobre a consistência e os comportamentos de tempo de execução das sequências de pior e melhor caso da decisão de reconfiguração. Segundo Göttmann *et al.* (2020), a metodologia foi implementada em uma ferramenta como protótipo para experimentação e validação.

Semelhante ao método proposto neste estudo, a abordagem de Göttmann *et al.* (2020) realiza a reconfiguração em tempo de execução e foi implementada como ferramenta para apoiar a validação. O método desenvolvido neste estudo não utiliza formalismo para escolher entre as configurações possíveis, realizando esse processo por meio de um conjunto de regras anotadas nas *features*.

3.6 ReMINDER, uma abordagem para modelagem de NFP e cenários de adaptação de contexto

Em Uchôa *et al.* (2017a) foi apresentado a abordagem ReMINDER para modelagem de *Non-Functional Properties* / Propriedades não Funcionais (NFP) de LPSD. A abordagem fornece uma maneira sistemática de identificar NFP, cenários de adaptação de contexto e restrições como forma de apoiar a representação de NFP da LPSD. Segundo Uchôa *et al.* (2017a), a abordagem se mostrou como um guia para identificar NFP e cenários de adaptação ao contexto para MF de LPSD.

A abordagem ReMINDER foi acoplada a uma ferramenta de modelagem (UCHÔA *et al.*, 2017b), porém, diferente do método apresentado nesse estudo, a abordagem centrou-se na modelagem de NFP e cenários da adaptação de contexto, com pouco aprofundamento sobre os processos de reconfiguração em tempo de execução.

3.7 Comparação dos Trabalhos Relacionados

O Quadro 1 apresenta um comparativo entre as abordagens, destacando quais os diferenciais do método proposto neste estudo. Cada abordagem foi enumerada para auxiliar na visualização do Quadro. Os trabalhos foram analisados com base nos seguintes critérios: Implementar o MAPE-K, Utiliza modelo de *features* para modelar LPSD; Realiza a reconfiguração em tempo de execução; Utiliza modelo de restrições; É acoplado a uma ferramenta de modelagem de LPSD e Realiza a verificação da adaptação. Para cada trabalho relacionado foram atribuídos números para melhor visualização: **1** - Weckesser *et al.* (2018); **2** - Santos *et al.* (2019); **3** - Sawyer *et al.* (2012); **4** - Ayala *et al.* (2021); **5** - Göttmann *et al.* (2020) e **6** - Uchôa *et al.* (2017a).

Quadro 1 – Comparativo entre os trabalhos relacionados e o presente estudo.

Características	**1**	**2**	**3**	**4**	**5**	**6**	**Presente estudo**
Implementa o MAPE-K	X	X					X
Utiliza modelo de features para modelar LPSD	X	X		X		X	X
Realiza a reconfiguração em tempo de execução	X		X	X	X	X	X
Utiliza modelo de restrições			X		X		X
É acoplado a uma ferramenta de modelagem de LPSD					X	X	X
Realiza a verificação da adaptação		X					

Fonte: elaborado pelo autor.

4 DESENVOLVIMENTO DE UM MÉTODO DE ADAPTAÇÃO PARA O MF DE LPSD

Este estudo visa desenvolver um método de adaptação para MFs de LPSD, que em tempo de execução priorize configurações que melhor atendam as *features* do sistema. O método será implementado como extensão da ferramenta DyMMer *web*, contribuindo para que a ferramenta possa oferecer aos engenheiros de domínio e de aplicação, um instrumento de gerenciamento, modelagem e manutenção dos MFs de LPSD.

A partir do estudo das principais técnicas de gerenciamento da variabilidade em *MF* de *LPSD*, e analisando diversas abordagens propostas para execução do processo de variabilidade dinâmica, foi proposto um conjunto de requisitos para guiar a implementação do objeto de estudo desse trabalho. Nesse sentido, os requisitos a seguir foram definidos para construção do método de adaptação:

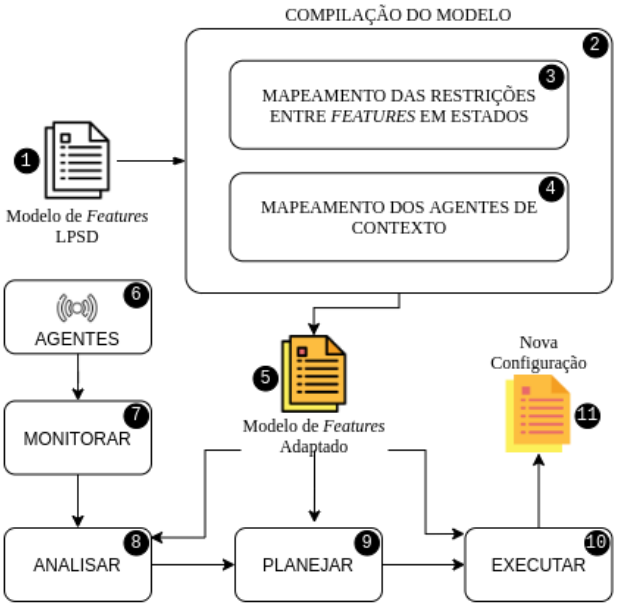
1. **Identificação das restrições entre *features***: capacidade do método de identificar as restrições entre as *features*.
2. **Mapeamento das restrições nos estados das *features***: após identificadas, as restrições devem ser mapeadas e anotadas nos estados das *features*. Esse mapeamento deve indicar que a ativação da *feature* está condicionada a um conjunto de restrições.
3. **Identificação dos agentes de contexto**: está relacionando a modelagem dos agentes de contexto.
4. **Mapeamento da influência dos agentes nas *features* do modelo**: após modelagem dos agentes e seus contextos, deve-se mapear como as *features* são afetadas pelas mudanças desses contextos.
5. **Implementação do processo de adaptação com base no MAPE-K**: realização da adaptação a partir do monitoramento das mudanças, análise dos dados, planejamento da nova reconfiguração e execução.
6. **Implementação do método de adaptação como extensão da DyMMer web**: estender a ferramenta DyMMer para modelar agentes de contexto e realizar o processo de reconfiguração dinâmica de MF de LPSD.

Um detalhamento do método de adaptação será apresentado nas seções seguintes.

4.1 Método de Adaptação: Visão Geral

O método de adaptação foi construído com base no modelo MAPE-K, conforme ilustra a Figura 9. O método foi implementado na linguagem de programação *JavaScript*, cuja escolha se deu pela familiaridade do pesquisador, e por ser a mesma linguagem utilizada na DyMMer *web*. O método recebe como entrada um modelo de *features*, que é compilado para um novo formato, com novos atributos e suporta a anotação de regras de gerenciamento da variabilidade. O novo MF é chamado de modelo de *features* adaptado e serve de insumo para as fases seguintes do modelo.

Figura 9 – Visão geral do método.



Fonte: elaborado pelo autor.

A Figura 9 apresenta uma visão geral do método de adaptação, no qual pode-se destacar duas etapas importantes. A primeira etapa representa a compilação do MF (**1**) para um MF adaptado (**5**). O MF adaptado é o principal insumo da segunda etapa, que consiste no processo de reconfiguração propriamente dito, disparado pelos agentes de contexto (**6**), resultando no produto que é a nova configuração. Como forma de ampliar o entendimento do método, a seguir são descritos os pontos destacados na Figura 9:

1. **MF de LPSD**: insumo do processo de compilação.
2. **Compilação do modelo**: processo de adequação do MF ao método de adaptação.
3. **Mapeamento das restrições**: as restrições são mapeadas, transformadas em estados e anotados nas *features*. Em outros termos, se uma ***feature* a** requer

uma ***feature* b**, a ***feature* a** recebe uma anotação indicando que o estado **ativo** depende da ativação da ***feature* b**. Esse processo ocorre de forma automática.

4. **Mapeamento dos agentes**: o mapeamento dos agentes exige a identificação manual dos agentes pelo Engenheiro de Domíno. Após a identificação, as *features* são anotadas com os agentes. Supondo que uma ***feature* a** requer a ativação do ***driver* a**, após a modelagem do *driver*, a ***feature* a** recebe uma anotação indicando que o estado **ativo** depende da ativação do ***driver* a**, um agente de contexto. Nesse caso, o processo de anotação é chamado de *link* e é feito manualmente pelo Engenheiro de Domínio.
5. **MF adaptado**: produto do processo de compilação, que contém todas as anotações necessárias para ativação e desativação de *features*, segundo as alterações nos agentes de contexto.
6. **Agentes**: os agentes mapeados na etapa de compilação (**2**) são monitorados e disparam o processo de reconfiguração quando ocorre mudança de contexto.
7. **Monitorar**: ponto de entrada do processo de reconfiguração, é responsável por monitorar os agentes, agregando as informações e passando para a etapa de análise.
8. **Analisar**: identifica e mapeia as *features* afetadas pela mudança nos agentes e envia para o planejamento.
9. **Planejar**: identifica os novos valores das *features* e agrega os dados para gerar uma lista de alterações que são enviadas para execução.
10. **Executar**: realiza as alterações necessárias nas *features*.
11. **Nova Configuração**: produto final do processo de reconfiguração dinâmica.

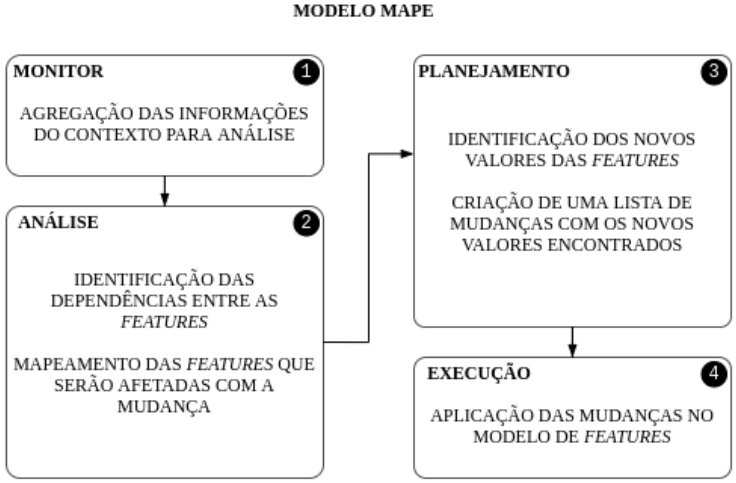
A Figura 9 não deixa explícito como o elemento K, o **Conhecimento** do MAPE-K, está presente no método. A base de **Conhecimento** está sendo usada para coletar informações para melhoria do próprio método. A versão atual ainda não utiliza o **Conhecimento** para reaproveitar informações nas demais fases do MAPE-K. Na implementação inicial, o monitoramento, análise, planejamento e execução foram priorizados.

A Figura 10 detalha o processo de adaptação utilizando o modelo MAPE-K. O **Monitor** (**1**) é o responsável por supervisionar os agentes e detectar mudanças no contexto. Quando ocorre uma mudança, o processo de adaptação é disparado. O **Monitor** agrega todas as informações relacionadas ao contexto e envia para o processo de **Análise** (**2**). Durante a etapa de

Análise, dois processos principais são executados:

1. Identificação das dependências entre as *features*: a partir do MF adaptado serão realizadas verificações para identificar as dependências entre as *features*. Nesse ponto, a análise identifica se uma ***feature* a** depende de ***feature* b**, que por sua vez, depende de uma ***feature* c**. Desse modo, é possível descobrir quais *features* serão afetadas pelas mudanças.
2. Mapeamento das *features* que serão afetadas com a mudança: Depois de detectar as dependências entre as *features*, é realizado um mapeamento para identificar como a mudança no contexto afetará o modelo. Nesse ponto, a análise gera uma lista contendo o caminho da mudança, por exemplo, a mudança de um ***driver* a** implicará na mudança de estado da ***feature* a**, que implicará na mudança da ***feature* d**, e assim por diante. As *features* não afetadas, são ignoradas e o resultado desse processo é passado para a etapa de planejamento.

Figura 10 – Detalhamento da utilização do modelo MAPE.



Fonte: elaborado pelo autor.

Com as informações recebidas da análise, os processo de **Planejamento** (**3**) são iniciados. Os dados recebidos contém todas as informações necessárias para construção de um plano de mudanças. Nesse sentido, o planejamento é realizado em duas etapas:

1. Identificação dos novos valores das *features*: cada *feature* afetada traz consigo um conjunto de informações sobre seus estados e as mudanças que estão acontecendo. Desse modo, os novos valores das *features* são identificados e passados para a etapa seguinte.
2. Criação de uma lista de mudanças com os novos valores encontrados: nesse

ponto, uma lista de mudanças é criada, organizada na ordem em que as mudanças devem ocorrer. Por fim, a lista de mudanças é enviada para a etapa de Execução.

Com a lista de mudanças, o processo de **Execução** (**4**) é iniciado. Esse processo é o mais simples, pois apenas aplica as alterações indicadas, gerando um produto final que é a nova configuração do MF.

O **Conhecimento** também é omitido na Figura 10, mas é presente na implementação do método como forma de guardar um conjunto de informações sobre o processo de reconfiguração, como explicado anteriormente. No momento, essas informações estão sendo usadas para aperfeiçoar o método, corrigir eventuais erros e ajustar as etapas para melhorar o desempenho. Em versões futuras, a base de **Conhecimento** deverá compartilhar informações que auxiliem o método durante a execução dos processos de reconfiguração. Já foi identificado, que a etapa 1 de mapeamento do processo de análise pode ser movido para a base de conhecimento, considerando que esse processo só precisa ser executado uma vez.

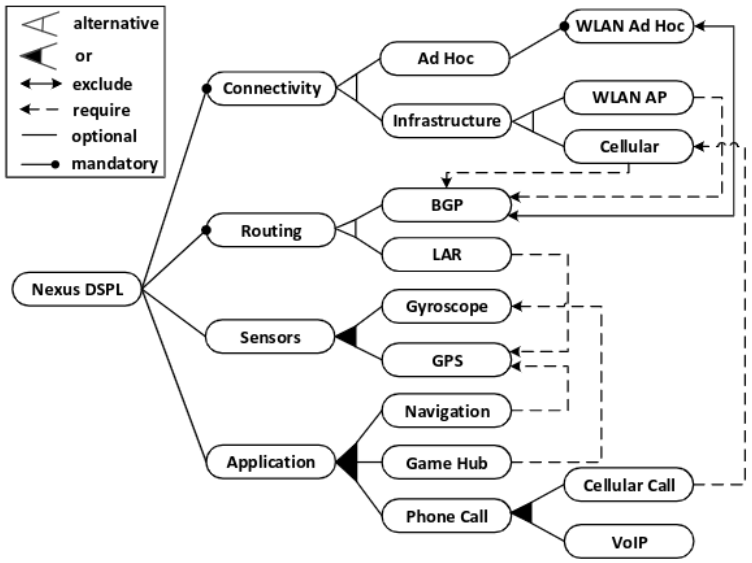
4.2 Método de Adaptação: Exemplo de Uso

Para ilustrar cada etapa do processo de adaptação, esta seção apresentará um exemplo de uso do método desenvolvido neste trabalho. A Figura 11 representa um MF da LPSD de um *smartphone* Google Nexus (SALLER, 2015). O referido modelo será utilizado para exemplificar o processo de adaptação. O modelo apresentado na Figura contém 19 *features*, relacionamentos de inclusão e exclusão, e cardinalidade de *features*. O MF descrito na Figura 11 está disponível no repositório de modelos de *features* LPSD da DyMMer *web* (ver Seção 2.5).

Conforme detalhado na seção anterior, o método de adaptação exige que os MF de LPSD da DyMMer *web* passem por um processo de adequação a um formato específico. Esse processo é extremamente relacionado ao código do método. Com o exemplo de uso, é possível melhorar o entendimento da necessidade de adequação do modelo. Nesse sentido, considere o MF apresentado na Figura 11. Algumas etapas são detalhadas para adequação do modelo:

1. A *feature* ***Connectivity*** tem duas *features* filhas alternativas (i.e., apenas uma das duas pode estar ativa em um dado momento). Para o método, as duas *features* ***Ad Hoc*** e ***Infrastructure*** são agregadoras (i.e., não são relevantes em termos de variabilidade). Considerando o caráter agregador das *features* ***Ad Hoc*** e ***Infrastructure***, a *feature* ***WLAN Ad Hoc*** acaba recebendo a informação de que não pode estar ativa caso alguma das *features* de ***Infrastructure*** esteja ativa. De

Figura 11 – Modelo de *features* LPSD de um Google Nexus.



Fonte: Adaptado de Saller (2015).

modo semelhante, as *features* ***WLAN AP*** e ***Cellular*** não podem ser ativas caso ***WLAN Ad Hoc*** esteja ativa. Essas informações são anotadas nas *features* como dependência para mudança de estados.

2. As *features* ***WLAN AP*** e ***Cellular*** são filhas de um relacionamento alternativo, apenas uma das duas pode estar ativa em um dado momento. Desse modo, é como se ***WLAN AP*** e ***Cellular*** fossem mutualmente exclusivas, e cada uma das *features* é anotada com essa informação. O mesmo ocorre com as *features* ***BGP*** e ***LAR***.
3. As restrições também são anotadas nas *features* relacionadas. Por exemplo, ***WLAN Ad Hoc*** e ***BGP*** são mutualmente exclusivas, as duas *features* não podem estar ativas ao mesmo tempo. Nesse caso, tanto ***WLAN Ad Hoc***, quanto ***BGP*** são anotadas com a informação de que uma só pode estar ativa quando a outra estiver desativada e vice-versa. No caso da *feature* ***Game Hub***, sua ativação está condicionada a ativação da *feature* ***Gyroscope***. Nesse caso, somente ***Game Hub*** é anotada com a informação de que depende da ativação da *feature* ***Gyroscope***.

Com as informações do modelo mapeadas e as restrições identificadas, cabe ao Engenheiro de Domínio adicionar as informações referentes ao contexto. Considerando o modelo de *features* Nexus da Figura 11, é possível indentificar o *driver* do ***GPS*** como um agente de contexto e anotar essa informação na *feature* ***GPS***. Alterações no sensor do ***GPS*** são identificadas pelo *driver* que está sendo monitorado, iniciando todo o processo de adaptação.

4.3 Método de Adaptação: Extensão da DyMMer *web*

Na Seção 2.5 foi apresentado a DyMMer *web*, uma ferramenta de modelagem e avaliação de LPS e LPSD. O objetivo desse estudo inclui a extensão da DyMMer *web* para suportar todo o processo de modelagem e gerenciamento da variabilidade de uma LPSD.

4.3.1 Notação do modelo de features

Os MFs são gerenciados e armazenados na DyMMer *web* utilizando uma notação específica codificada em JSON. A Figura 12 ilustra partes do JSON de um MF de uma LPSD. O objeto (**1**) contém informações de contato, endereço, informações acadêmicas e nome da pessoa responsável pela criação do modelo. No campo ***feature_tree*** são armazenadas as *features*, dispostas hierarquicamente em formato de árvore. O campo ***constraints*** contém as restrições entre as *features* do modelo. O método de adaptação está interessado somente nos campos ***feature_tree*** e ***constraints***.

Figura 12 – Notação doJSON utilizada pela DyMMer *web*.

Fonte: elaborado pelo autor.

A Figura 12 também representa de forma mais detalhada o formato das *features* e das restrições (**2**). Considere o campo ***feature_tree***, ele sempre terá um único elemento na lista, representando a *feature* principal (*root*) do modelo. Todas as demais *features* virão dentro do campo *children*, de acordo com a disposição hierárquica de cada uma. O campo *id* é o identificador da *feature* dentro da árvore, o campo *name* é usado para nomear a *feature* e o campo *type* define o tipo de *feature*, podendo ser **r** de ***root***, **m** de ***mandatory***, **o** de ***optional*** e **g** de ***grouped***.

A Figura 12 também ilustra como as restrições são definidas (**2**). O campo ***constraints*** contém uma lista de restrições entre as *features*. Cada restrição criada recebe um nome (*name*) e um valor (*value*), que contém a descrição da restrição. Considere as seguintes restrições: **1)** $\sim$ **_r_1_5_6_8 or** $\sim$ **_r_2_12_13**: o símbolo $\sim$ indica negação e os códigos **_r_1_5_6_8** e **_r_2_12_13** são identificadores de *features*. A negação antes dos identificadores indicam que as *features* são mutualmente exclusivas, em outras palavras, as duas *features* não podem estar ativas ao mesmo tempo; **2)** $\sim$ **_r_1_5_6_8 or _r_2_12_14**: neste caso, a negação em somente um dos identificadores indica uma relação de dependência, ou seja, a *feature* **_r_1_5_6_8** depende da ativação da *feature* **_r_2_12_14**.

4.3.2 Modelo de features adaptado

O MF adaptado inclui campos adicionais a notação padrão usada pela Dymmer *web*. A notação foi desenvolvida pelo autor do estudo para que as informações adicionais auxiliem o método durante o processo de reconfiguração dinâmica. A Figura 13 (**1**) apresenta os campos adicionados a notação original, dos quais: ***aggregator***: indica se uma *feature* deve ser ignorada pelo método, caso tenha função apenas de agregar um conjunto de *features*; ***multiplicity***: indica se um conjunto de *features* é alternativa ou exclusiva, sendo usado durante os processos iniciais de compilação do modelo; ***value***: valor da *feature*, em tempo de execução indica se a *feature* está ativa ou inativa; ***states***: são os possíveis estados que uma *feature* pode alcançar, o valor do estado determina o valor da *features* e a escolha entre os estados depende das regras de cada um. Na versão apresentada neste documento, o método só considera os valores binários verdadeiro e falso para indicar se uma *feature* está ativa ou inativa. Em versões futuras, pretende-se adicionar novas faixas de valores.

Complementando o entendimento do MF adaptado, a Figura 13 (**2**) ilustra como as restrições são anotadas nos estados das *features*. Em geral, apenas os estados que **ativam** *features* recebem anotações. Essas anotações são inseridas no campo ***requires***, com um ***address*** (identificador ou endereço de uma *feature* na árvore) e um ***value***, que é o valor da *feature* requirida. No exemplo da Figura 13, há 3 condições que precisam ser atendidas para que a referida *feature* seja ativada. Resumidamente, 2 *features* precisam estar desativadas e 1 ativada, quando essas condições forem satisfeitas, a *feature* é ativada.

Figura 13 – Notação do MF adaptado.

Fonte: elaborado pelo autor.

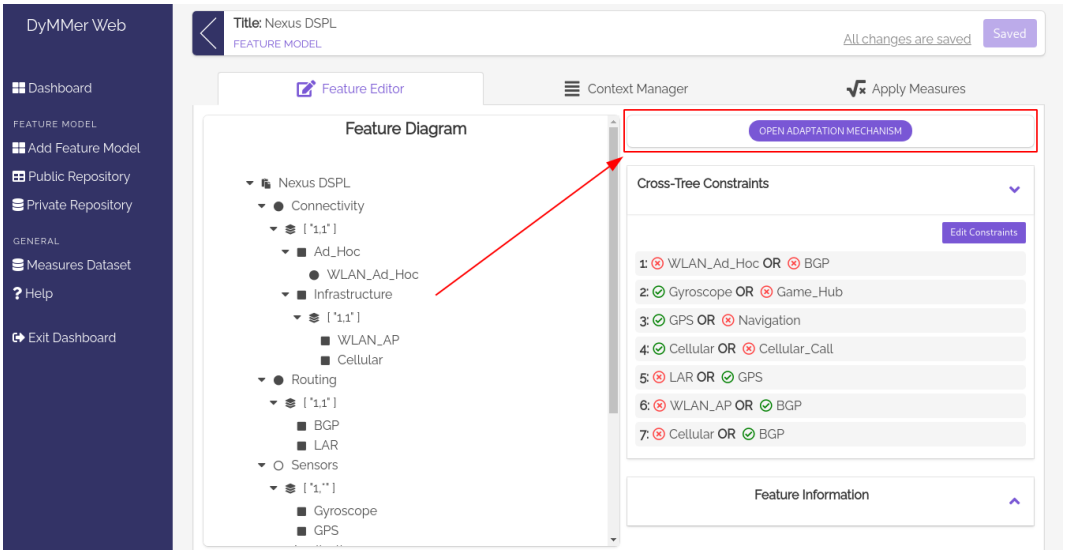
4.3.3 Layout do método implementado na DyMMer web

A Figura 14 apresenta o *Dashboard* da DyMMer *web* com a adição de um botão de acesso ao método de adaptação, destacado na imagem. O MF disposto no *Feature Diagram* é o mesmo do exemplo de uso da Seção 4.2. Além disso, a área *Cross-Tree Constraints* exibe todas as restrições entre as *features* do modelo. O *Feature Diagram* e as *Cross-Tree Constraints* foram destacadas por serem utilizadas no método de adaptação, a visualização auxilia na verificação manual do funcionamento do processo de adaptação. Os demais detalhes do *Dashboard* foram apresentados na Seção 2.5.

Antes de iniciar o método de adaptação, é importante que o Engenheiro de Domínio certifique-se de que todas as *features* estejam dispostas corretamente. Relacionamentos entre *features* e restrições também não devem conter problemas. Com essa verificação é possível evitar que determinadas *features* seja ativadas ou desativadas erroneamente durante a reconfiguração. A versão atual do método não valida se as relações e restrições do MF da LPSD estão adequadamente definidas.

A Figura 15 apresenta uma visão geral do método de adaptação que se divide em dois painéis: 1) *Custom Feature Model* - exibe o MF adaptado; 2) *Context Agents* - área de modelagem dos agentes de contexto. Antes da exibição dos painéis, o método realiza a compilação do MF da LPSD, gerando o MF adaptado que é exibido do lado esquerdo da figura. A versão do

Figura 14 – Botão de acesso ao método de adaptação.



Fonte: elaborado pelo autor.

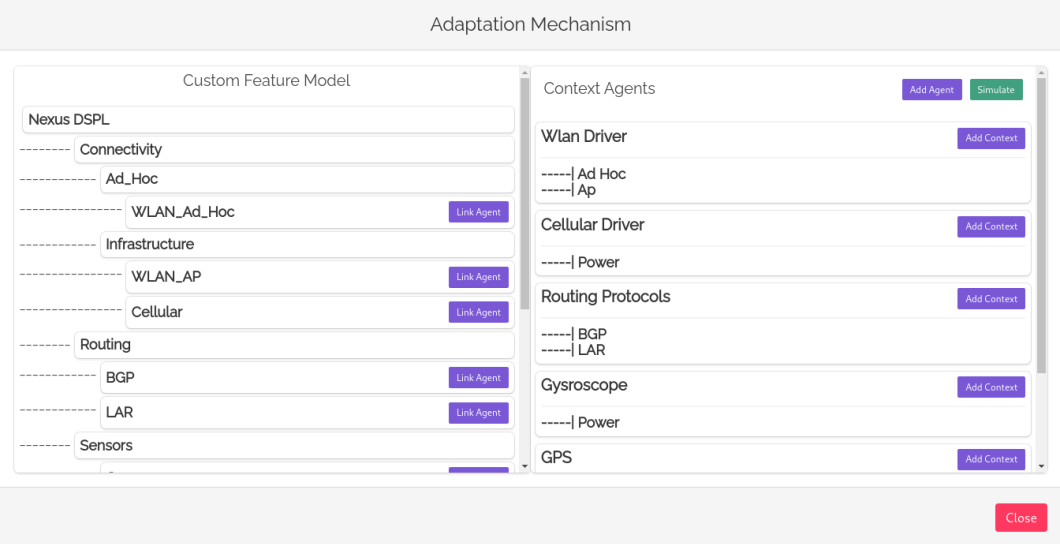
método apresentada neste documento omite visualmente as informações de restrições e os relacionamentos entre as *features* do MF. Porém, elas estão mapeadas nos estados das *features*, conforme já explicado anteriormente. Na Figura 15, no painel *Context Agents* já existem alguns agentes modelados, mas quando o método é aberto pela primeira vez, essa lista está vazia.

A modelagem dos agentes é simples, ao clicar em *Add Agent* no painel *Context Agents*, um campo de texto é aberto para inserção do nome do agente. Em seguida, é necessário adicionar os contextos do agente, esse procedimento é feito ao clicar no botão *Add Context* do agente e adicionar o nome do contexto na caixa de texto. Cada contexto adicionado recebe dois estados para indicar se o contexto está ativo ou inativo (*On* ou *Off*), por padrão, o contexto é inicializado inativo (*Off*).

No exemplo doa Figura 15, considere que o engenheiro inseriu o agente ***Wlan Driver*** e em seguida inseriu ***Ad Hoc*** e ***Ap*** como seus respectivos contextos. Cada um dos contextos recebeu dois estados e foram inicializados no estado inativo. O mesmo processo foi executado para cada um dos agentes apresentados na figura. O último passo da modelagem consiste na vinculação de um contexto a uma determinada *feature*. Para executar essa ação, deve-se clicar no botão *Link Agent* e executar os passos correspondentes.

O processo de vinculação é ilustrado na Figura 16. Ao clicar no botão *Link Agent* de uma *feature*, uma caixa é aberta e três informações importantes devem ser inseridas. De cima para baixo, na primeira caixa deve-se selecionar um agente. No exemplo da Figura 16, o botão *Link Agent* da *feature* ***WLAN_Ad_Hoc*** foi clicado e o agente ***Wlan Driver*** escolhido. Em seguida, foi escolhido o contexto do agente que seria vinculado na *feature*, no caso o contexto

Figura 15 – Visão Geral do Método.



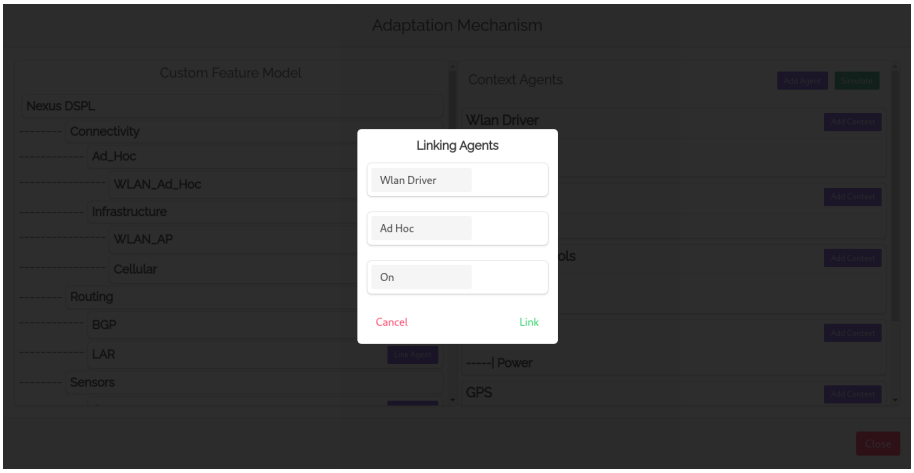
Fonte: elaborado pelo autor

.

Ad Hoc. Por fim, na última caixa de seleção, foi indicado que o valor do contexto seria ***On***. Em outras palavras, a *feature* ***WLAN_Ad_Hoc*** recebeu uma anotação indicando que só será ativada quando o contexto ***Ad Hoc*** do agente ***Wlan Driver*** tiver valor ***On***.

Figura 16 – Caixa de seleção de agentes vincular a *feature*.

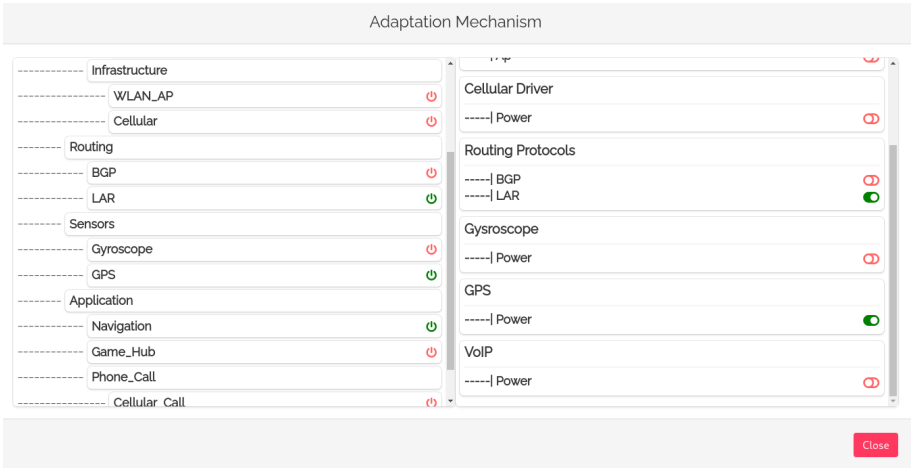
/



Fonte: elaborado pelo autor.

Com a conclusão do processo de modelagem e vinculação dos agentes, o próximo passo consiste na simulação das alterações de contexto, como ilustra a Figura 17. Para iniciar a simulação, o Engenheiro de Domínio deve clicar no botão *Simulate* no painel *Context Agents*. Ao iniciar a simulação, a *interface* do método muda, no painel *Custom Feature Model* as *features* são exibidas com um ícone que indica se estas estão ativadas ou desativadas. No painel de *Context Agents*, são exibidos botões para ativação e desativação dos contextos, desse modo, é possível ver como o método realiza o processo de reconfiguração dinâmica.

Figura 17 – Exemplo de contextos e *features* ativadas.



Fonte: elaborado pelo autor.

Na Figura 17, os contextos ***LAR***, do agente ***BGP***, e ***Power***, do agente ***GPS***, estão ativos. Analisando o modelo de *features* do Nexus LPSD apresentado anteriormente, nota-se que há 3 *features* ativas. A *feature* ***GPS*** foi ativa, por estar vinculada ao contexto ***Power***, que está ativo. As *features* ***Navigation*** e ***LAR*** estão ativas, pois requerem unicamente que a *feature* ***GPS*** esteja ativa.

Por fim, essa seção apresentou uma visão geral da implementação do método de adaptação como extensão da DyMMer *web*. A implementação ainda necessita de melhorias na visualização das *features*, restrições, agentes e contextos, além de alguns ajustes de usabilidade para minimizar possíveis erros no momento da modelagem.

4.4 Conclusão

Essa seção apresentou o método de adaptação desenvolvido para realizar o processo de reconfiguração de MF de LPSD em tempo de execução. Na Seção 3 foram apresentadas abordagens de reconfiguração semelhantes e um comparativo para embasar o desenvolvimento do objeto de estudo dessa pesquisa.

O método de adaptação proposto suporta a modelagem de MF da LPSD e a modelagem de agentes e contextos. Foi implementado com base no modelo MAPE-K, utiliza uma abordagem baseada em restrições para escolher as configurações e acoplado a uma ferramenta de modelagem de MF de LPSD. Desse modo, o método de adaptação contribui para fornecer uma ferramenta de modelagem LPSD com suporte a reconfiguração em tempo de execução.

5 AVALIAÇÃO

O método de adaptação apresentado na Seção 4, foi desenvolvido para apoiar a modelagem de MF de LPSD, permitindo a identificação dos agentes de contexto e a execução do processo de reconfiguração em tempo de execução. Esta Seção apresentará o processo de avaliação do método com relação a aplicabilidade e eficácia na realização da reconfiguração em tempo de execução. Desse modo, esta seção estão organizada da seguinte forma: Seção 5.1 define prova de conceito e detalha os objetivos da avaliação, a Seção 5.2 detalha as etapas do processo de avaliação, a Seção 5.3 apresenta os resultados da avaliação e por fim, a Seção 5.4 apresenta as considerações finais da avaliação.

5.1 Prova de Conceito

A avaliação foi realizada por meio de uma prova de conceito, definida por Glass *et al.* (2002) como uma estratégia para validação da viabilidade de um determinado conceito em uma perspectiva prática. Nesse sentido, o objetivo da avaliação é validar, por meio de uma prova de conceito, a aplicabilidade e eficácia do método de adaptação com relação ao processo de modelagem dos agentes e execução da reconfiguração em tempo real.

A avaliação contemplou aspectos como: o desempenho ao compilar os recursos e ao executar a configuração dinâmica; corretude na conversão de restrições em regras de adaptação; facilidade na identificação dos agentes e contextos do MF; facilidade de vinculação dos agentes às *features* relacionadas; e, capacidade de executar a reconfiguração em tempo de execução. Na seção à seguir serão descritas as etapas de avaliação do método.

5.2 Descrição

Para realizar a avaliação de desempenho do método, foi utilizado MFs de LPSD gerados pela ferramenta *S.P.L.O.T Feature Model Generator* (MENDONCA *et al.*, 2009). A ferramenta permite gerar MFs de diversos tamanhos e características, sendo possível selecionar a quantidade de restrições, *features* obrigatórias, opcionais ou agrupadas, dentre outras propriedades. Com a ferramenta foram gerados dois MFs com 100 e 1000 *features*, respectivamente.

Além disso, algumas ferramentas da linguagem de programação utilizada na implementação do método foram aplicadas para capturar o tempo decorrido durante a execução de cada processo. Antes do ponto de execução uma determinada operação, um *timer* foi definido

e iniciado, mantendo-se ativo até o fim da operação. Quando o processo finalizava, o *timer* encerrava a contagem e imprimia no *console* o tempo decorrido em milissegundos. A avaliação de desempenho foi feita com cada um dos MFs definidos e os resultados anotados para análise e discussão.

A etapa seguinte da avaliação contou com a participação de um especialista com pelo menos dois anos de experiência em modelagem de LPSD e MF. O participante tinha liberdade de decidir sobre sua colaboração no estudo conforme os termos definidos no Formulário de Consentimento (Apêndice A). O participante também foi submetido a um questionário de caracterização (Apêndice B), para definir o perfil e mensurar o seu nível de conhecimento sobre os conceitos da ferramenta. O processo de avaliação com especialista contemplou duas etapas importantes:

1. Treinamento para uso da ferramenta: o treinamento teve como objetivo ambientar o especialista sobre o uso do método. Como o método está acoplado à DyMMer *web* (apresentada na 2.5), o treinamento contemplou uma visão geral da DyMMer e posteriormente o processo de modelagem de agentes de contexto, vinculação de *features* e execução do processo de adaptação.
2. Execução das tarefas de avaliação: de modo geral, inclui a análise de um modelos de *feature* para identificação e modelagem dos agentes de contexto e execução da reconfiguração dinâmica.

A avaliação foi definida em um conjunto de tarefas, entregues ao participante para execução individual, com o experimentador observando, conforme o documento disponível no Apêndice C. As principais tarefas exigiram a modelagem dos agentes de contexto a partir da observação do modelo de *features*. O participante identifica o agente, define os contextos e em seguida, vincula-os às *features* correspondentes. O resultado esperado após essas tarefas incluem a identificação correta dos agentes de contexto, bem como, a correta vinculação dos contextos às *features*.

A última tarefa do estudo consistiu na execução do processo de reconfiguração dinâmica. Considerando os aspectos práticos, a reconfiguração é feita em modo de simulação, o sistema muda a *interface* para que o usuário possa ativar e desativar contextos e desse modo, observar a ativação e desativação das *features* relacionadas, tal qual ocorreria em um cenário real. Por fim, espera-se que o processo ocorra sem erros, que a reconfiguração obedeça as regras e que o participante reporte sua experiência e eventuais sugestões sobre o uso da ferramenta. A

coleta de dados referente a essa etapa da experimentação foi feita através de um questionário que está disponível Apêndice D. A seguir são apresentados os resultados e as análises dos dados obtidos junto durante a experimentação.

5.3 Resultados

Nesta seção são apresentados os resultados obtidos por meio da avaliação de desempenho e os dados obtidos junto a avaliação dos especialista.

5.3.1 Avaliação de Desempenho

A avaliação de desempenho foi realizado com 2 MF com características e tamanhos diferentes. O primeiro MF (MF1) tinha 100 *features*, das quais: 32 opcionais, 22 obrigatórias, 45 agrupadas. Entre as *features* agrupadas, 14 agrupamentos eram relacionamentos do tipo *or* exclusivo. Os grupos *or* exclusivo exigem mais computação, por isso estão representados em maior quantidade. O MF também continha 20 restrições entre *features*. Foram modelados 2 agentes, cada um com 10 contextos. O segundo MF (MF2) tinha 1000 *features*, das quais: 345 opcionais, 218 obrigatórias, 336 agrupadas. Entre as *features* agrupadas, 89 agrupamentos eram relacionamentos do tipo *or* exclusivo. O MF também continha 200 restrições entre *features* e foram modelados 4 agentes com 5 contextos cada.

O equipamento utilizado para a avaliação foi um *notebook* com a seguinte configuração: Sistema operacional: Manjaro Linux; Versão do KDE Plasma: 5.21.3; Versão do KDE Frameworks: 5.80.0; Versão da Qt: 5.15.2; Versão do kernel: 5.11.6-1-MANJARO; Tipo de sistema operacional: 64 bits; Plataforma gráfica: X11; Processadores: 4 × Intel® Core™ i5-7200U CPU @ 2.50GHz; Memória: 7,6 GiB de RAM; Processador gráfico: Mesa Intel® HD Graphics 620. O navgeador utilizado foi o Chromium versão 89.0.4389.90 (Versão oficial) Arch Linux 64 bits.

A execução do método conta com uma etapa preliminar de compilação que adapta o MF ao método e mapeia as restrições entre as *features* em estados. O MF1 levou 33 ms para fazer a compilação, já o MF2 precisou de 82 ms para finalizar o processo. Em seguida, foi feito a modelagem dos agentes e os contextos de ambos os modelos e a vinculação dos contextos a *features* do modelo. Posteriormente, foi realizada 5 simulações de adaptação com os seguintes resultados:

Tabela 1 – Resultados das simulações de adaptação do método.

MF	1ª Execução	2ª Execução	3ª Execução	4ª Execução	5ª Execução	Média
MF1	71 ms	79 ms	80 ms	37 ms	83 ms	70 ms
MF2	715 ms	620 ms	636 ms	654 ms	593 ms	643 ms

Fonte: elaborado pelo autor.

De acordo com a Tabela 1, a média de desempenho do adaptação no MF1 foi de 70 ms, enquanto no MF2 a média foi de 643 ms. De forma geral, os tempos foram bastante satisfatórios, mesmo o MF2 tendo bastante complexidade em função da quantidade de agrupamentos, número de restrições e de *features*, o tempo necessário para execução ficou dentro de um limite favorável. Com mais estudos será possível encontrar pontos de melhoria e aperfeiçoar cada vez mais o método para um desempenho ainda melhor. Em uma versão futura, as melhorias na base de conhecimento contribuirão para aumentar o desempenho do método durante as reconfigurações.

5.3.2 Prova de Conceito

A prova de conceito foi aplicada junto a um especialista em modelagem LPSD. O nome do participante foi preservado e para identificação será chamado apenas de P1. Antes da realização das atividades foi aplicado um teste de caracterização, cujos resultados indicaram que o especialista é graduando, tem entre 1 e 5 anos de experiência com LPS e LPSD na área acadêmica e entre 1 e 5 anos com modelagem de software na área acadêmica e na indústria. O especialista já aplicou a abordagem LPSD durante o desenvolvimento de softwares na área acadêmica, bem como já usou técnicas de modelagem de software na indústria. O participante tem pelo menos de 1 ano de experiência em reconfiguração dinâmica, porém, tem conhecimentos avançados em modelagem e manutenibilidade de MF de LPS e LPSD. O especialista atestou ter conhecimentos gerais em modelagem de agentes de contexto e em reconfiguração dinâmica.

Durante a realização do estudo foi medido o tempo em minutos para o participante para realização das tarefas propostas, na Seção 5.2. A contabilização do tempo indicou que o participante levou 29 minutos para concluir as tarefas. Posteriormente, foi aplicado o formulário de avaliação do método (Apêndice D), cujos os resultados serão apresentados a seguir.

A Tabela 2 apresenta as respostas do participante para as 4 primeiras questões. Com relação a **Q1** o participante P1 concordou que a utilização do método de adaptação auxiliou na modelagem de contexto e justificou dizendo que *"o método permite especificar os diversos agentes e cenários de adaptação e apresenta isso de forma visual."*. Nesse sentido, o participante

Tabela 2 – Respostas do participante para as 4 primeiras questões

Questão	Resposta
Q1 - Você concorda que a utilização do Método de Adaptação auxiliou na modelagem dos Agentes de Contexto?	Concordo Totalmente
Q2 - Você concorda que o Método de Adaptação permitiu a visualização do processo de reconfiguração após a mudança do contexto?	Concordo Totalmente
Q3 - Na sua percepção, qual foi o esforço para identificar os agentes relacionados ao modelo de *features*?	Médio
Q4 - Na sua percepção, qual foi o esforço para modelar os contextos de cada agente identificado?	Baixo

Fonte: elaborado pelo autor.

P1 destacou que a representação visual dos contextos facilitou o processo de modelagem dos cenários de adaptação.

O participante concordou com a **Q2** que afirma que o método de adaptação permite a visualização do processo de reconfiguração após a mudança no contexto. O participante complementou dizendo que *"a ativação e desativação das features afetadas pela mudança de contexto é apresentada de forma intuitiva"*. A representação do processo de reconfiguração é importante para que o responsável pela modelagem do MF possa compreender como as *features* serão ativadas em tempo de execução.

As questões **Q3** e **Q4** são relacionadas ao esforço na identificação e modelagem de agentes e contextos. O participante P1 considerou "médio" o esforço na identificação dos agentes e justificou afirmando que *"alguns agentes são facilmente identificados como é nos casos onde existem features que fazem referência clara a algum tipo de sensor"*. Uma *feature* chamada **GPS** é um exemplo de referência explícita a um agente de contexto. O participante complementou dizendo que *"em outros casos, a identificação dos agentes exige maior conhecimento do domínio da LPSD"*. Sobre a modelagem dos contextos, o participante considerou "baixo" o esforço empregado e ponderou que a compreensão do domínio da LPSD é importante no momento da modelagem.

A **Q5** buscou informações sobre o processo de simulação da adaptação. Segundo o participante P1, a simulação ajudou a entender quais configurações o modelo assumirá a partir de cada alteração do contexto, o mesmo concluiu que *"a ferramenta apresenta de forma visual e bem intuitiva a ativação e desativação das features nas alterações de contexto"*.

A **Q6** perguntou ao participante qual foi a principal dificuldade no uso da ferramenta, ele respondeu que sentiu *"dificuldade em entender o porquê de algumas features ficarem ativas ou inativas em algumas mudanças de contexto"*. O participante disse ainda que depois de algum

tempo, percebeu que a mudança nas *features "acontecia por causa de restrições ligadas a essas features"*. Algumas *features* mudam de acordo com as restrições existentes. Nesse sentido, o participante sugeriu que a ferramenta apresentasse algum *feedback* visual que ajudasse o engenheiro a entender a razão das mudanças nas *features*.

A **Q7** perguntou quais as vantagens e desvantagens do método de adaptação o participante P1 respondeu que *"Entre as vantagens estão a facilidade em ter uma visão geral dos agentes e contextos de adaptação existentes devido ao modo como a ferramenta exibe isso e fácil identificação das features que são afetadas por uma determinada mudanças de contexto"*. O participante afirmou não ter encontrado desvantagens no modelo.

Por fim, a **Q8** pediu sugestões de melhoria do método o participante fez uma sugestão: *"exibir na página do método de adaptação as restrições existentes, pois isso ajuda a identificar o porquê de algumas features ficarem ativas ou inativas em determinadas mudanças de contexto"*. As melhorias na visualização da restrições podem ajudar na compreensão das alterações nas *features* do MF.

5.4 Conclusões

Nesta Seção, foi apresentado uma prova de conceito do método de adaptação desenvolvido para realizar, em tempo de execução, o processo de reconfiguração dinâmica de MF de LPSD. A análise de desempenho revelou que na medida em que o MF cresce em complexidade, mais tempo é demandado para realização da reconfiguração. Porém, mesmo um modelo mais complexo teve um ótimo tempo de reconfiguração. Já a análise dos dados obtidos junto ao participante revelou que a ferramenta é intuitiva na na modelagem de agentes e de contextos. Além disso, revelou que o processo de simulação do processo de reconfiguração demonstra bem como as *features* do modelo são alteradas quando há mudanças no contexto.

De modo geral, a análise dos resultados obtidos na experimentação demonstrou que a o método cumpre bem o que foi proposto e executa todos os processos dentro de um tempo aceitável. Os dados obtidos durante a experimentação foram importantes para uma validação inicial da ferramenta. Mais estudos e outras etapas de avaliação devem ser realizados para que o método de adaptação cumpra seu papel no processo de reconfiguração em tempo de execução.

6 CONCLUSÕES E TRABALHOS FUTUROS

Esta Seção apresenta as considerações finais do trabalho e está organizada da seguinte forma: Seção 6.1 apresenta uma visão geral do estudo, Seção 6.2 apresenta as principais contribuições do estudo, Seção 6.3 descreve as limitações do estudo e por fim, a Seção 6.4 apresenta os trabalhos que podem derivar desse estudo.

6.1 Introdução

As novas demandas de software passaram a exigir o desenvolvimento de aplicações que executam continuamente, adaptando-se a mudanças de acordo com as necessidades do usuário (HINCHEY *et al.*, 2012). Para suprir essas demandas, abordagens como a LPS e LPSD começaram a ser utilizadas para gerenciamento de complexidade, customização de software e redução de custos e tempo de desenvolvimento (POHL *et al.*, 2005; BASHARI *et al.*, 2017). Nesse sentido, para modelagem dos softwares modernos são necessárias ferramentas que ofereçam aos engenheiros de aplicação e domínio, desenvolvedores e gestores de software, um instrumento de gerenciamento, modelagem e manutenção de MFs de LPSD. Com base nisso, esse trabalho teve como objetivo a criação de um método de adaptação para o MF de LPSD, que em tempo de execução, priorize configurações que melhor atendam as *features* do sistema.

Dessa forma, como principal contribuição desse trabalho foi desenvolvido um método de adaptação para MF de LPSD que permite a modelagem de agentes de contexto e a vinculação dos contextos as *features* do modelo para realização do processo de reconfiguração. Antes de iniciar o processo, o MF passa por uma adaptação no qual os relacionamentos e restrições entre *features* são mapeados em estados e anotados nas *features* correspondentes. A etapa seguinte consiste na modelagem dos agentes de contexto e posteriormente, a vinculação às *features* que são afetadas.

O método foi implementado como extensão da DyMMer *web*, para permitir que a ferramenta suporte o processo de modelagem de MFs de LPSDs e a realização da reconfiguração dinâmica desses modelos em tempo de execução. Para validar o método foram realizadas duas avaliações, uma de desempenho e uma prova de conceito com um especialista. O processo de validação de desempenho do método demonstrou que o tempo de compilação de modelos mais complexos é muito bom. Demonstrou também que o tempo de execução do processo de reconfiguração em tempo real é maior na medida em que é aplicado a MFs com muitas *features*.

Porém, apesar da alteração no desempenho, os tempos do MF mais complexo ainda foram ótimos. A validação junto ao especialista indicou que a ferramenta auxilia na modelagem dos agentes e contextos, além de apresentar visualmente o processo de reconfiguração de forma intuitiva, indicando quais *features* estarão ativas após as alterações do contexto.

Com a realização desse trabalho, espera-se fornecer aos engenheiros de domínio uma ferramenta completa de modelagem de MFs de LPSD, com suporte a modelagem de agentes e execução da variabilidade dinâmica.

6.2 Principais Contribuições

O método de adaptação foi implementado para auxiliar no processo de reconfiguração em tempo de execução em MF de LPSD. Ainda há algumas melhorias que precisam ser feitas para que a ferramenta esteja completa. Porém os resultados já demonstraram que o método cumpre bem o papel na modelagem de agentes, contextos e na execução do processo de reconfiguração. Nesse sentido, as principais contribuições identificadas com a realização deste trabalho foram:

- Proposição de uma método de adaptação para MF de LPSD.
- Extensão da ferramenta DyMMer *web* para modelar agentes e contextos para realização do processo de reconfiguração em tempo de execução.

6.3 Limitações do trabalho

Esta Seção discute as limitações dos resultados obtidos no trabalhos. A primeira limitação está relacionada a prova de conceito aplicada a um único especialista. Apesar do participante ter um ótimo conhecimento sobre os conceitos, é possível que algumas questões não tenham sido apontadas. Desse modo, a melhor forma de resolver essa limitação é a validação com mais especialistas.

A segunda limitação está relacionada ao método não estar 100% concluído visualmente, falta alguns elementos importantes como a exclusão e edição de agentes e contextos, visualização das restrições entre *features* e dos estados mapeados. Essa limitação pode ser resolvida com a implementação das funcionalidades restantes.

A terceira limitação está relacionada a implementação discreta do **Conhecimento** do modelo MAPE-K no método de adaptação, o que afeta o desempenho e impossibilita que um conjunto de informações coletadas sejam reaproveitadas durante a reconfiguração. Essa

limitação pode ser eliminada com a implementação da base de conhecimento e reaproveitamento das informações coletadas nas demais fases do processo.

6.4 Trabalhos Futuros

Como trabalhos futuros, pretende-se fazer melhorias nos processos do método, alguns algoritmos não precisam ser executados a cada reconfiguração. Desse modo, depois da primeira execução, as informações geradas podem ser armazenada na base de conhecimento do MAPE-K para serem reutilizadas nas reconfigurações seguintes. Esse processo pode ajudar a melhorar o desempenho do método de adaptação.

Em relação a extensão implementada na ferramenta DyMMer *web*, é possível realizar melhorias visuais como a exibição das restrições na tela do método, a possibilidade de excluir e renomear agentes e contextos e uma visão mais detalhada dos estados das *features*. Além disso, pode-se acrescentar uma visualização de como cada *feature* é afetada pelos contextos e demais *features*.

REFERÊNCIAS

AYALA, I.; PAPADOPOULOS, A. V.; AMOR, M.; FUENTES, L. Prodspl: Proactive self-adaptation based on dynamic software product lines. **Journal of Systems and Software**, [*S. l.*], v. 175, p. 110909, 2021. ISSN 0164-1212. Disponível em: https://www.sciencedirect.com/science/article/pii/S0164121221000066. Acesso em: 17 mar. 2021.

BARESI, L.; GUINEA, S.; PASQUALE, L. Service-oriented dynamic software product lines. **Computer**, IEEE, [*S. l.*], v. 45, n. 10, p. 42–48, 2012.

BASHARI, M.; BAGHERI, E.; DU, W. Dynamic software product line engineering: a reference framework. **International Journal of Software Engineering and Knowledge Engineering**, World Scientific, [*S. l.*], v. 27, n. 02, p. 191–234, 2017.

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

BENCOMO, N.; HALLSTEINSEN, S.; ALMEIDA, E. S. D. A view of the dynamic software product line landscape. **Computer**, IEEE, [*S. l.*], v. 45, n. 10, p. 36–41, 2012.

BERGER, T.; RUBLACK, R.; NAIR, D.; ATLEE, J. M.; BECKER, M.; CZARNECKI, K.; WĄSOWSKI, A. A survey of variability modeling in industrial practice. In: **Proceedings of the Seventh International Workshop on Variability Modelling of Software-intensive Systems**. [*S. l.*: *s. n.*], 2013. p. 1–8.

BEZERRA, C. I.; BARBOSA, J.; FREIRES, J. H.; ANDRADE, R. M.; MONTEIRO, J. M. Dymmer: a measurement-based tool to support quality evaluation of dspl feature models. In: **Proceedings of the 20th International Systems and Software Product Line Conference**. [*S. l.*: *s. n.*], 2016. p. 314–317.

CAPILLA, R.; BOSCH, J. The promise and challenge of runtime variability. **Computer**, IEEE, [*S. l.*], v. 44, n. 12, p. 93–95, 2011.

DAVRIL, J.-M.; DELFOSSE, E.; HARIRI, N.; ACHER, M.; CLELAND-HUANG, J.; HEYMANS, P. Feature model extraction from large collections of informal product descriptions. In: **Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering**. [*S. l.*: *s. n.*], 2013. p. 290–300.

GLASS, R.; VESSEY, I.; RAMESH, V. Research in software engineering: an analysis of the literature. **Information and Software Technology**, [*S. l.*], v. 44, n. 8, p. 491–506, 2002. ISSN 0950-5849. Disponível em: https://www.sciencedirect.com/science/article/pii/S0950584902000496. Acesso em: 10 mar. 2021.

GöTTMANN, H.; LUTHMANN, L.; LOCHAU, M.; SCHüRR, A. Real-time-aware reconfiguration decisions for dynamic software product lines. In: **Proceedings of the 24th ACM Conference on Systems and Software Product Line**. New York, NY, USA: Association for Computing Machinery, 2020. (SPLC '20). ISBN 9781450375696. Disponível em: https://doi.org/10.1145/3382025.3414945. Acesso em: 20 mar. 2021.

HALLSTEINSEN, S.; HINCHEY, M.; PARK, S.; SCHMID, K. Dynamic software product lines. **Computer**, IEEE, [*S. l.*], v. 41, n. 4, p. 93–95, 2008.

HINCHEY, M.; PARK, S.; SCHMID, K. Building dynamic software product lines. **Computer**, Citeseer, [*S. l.*], v. 45, n. 10, p. 22–26, 2012.

HORCAS, J.-M.; PINTO, M.; FUENTES, L. Software product line engineering: a practical experience. In: **Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A**. [*S. l.*: *s. n.*], 2019. p. 164–176.

KANG, K. C.; COHEN, S. G.; HESS, J. A.; NOVAK, W. E.; PETERSON, A. S. **Feature-oriented domain analysis (FODA) feasibility study**. [*S. l.*], 1990.

LAMSWEERDE, A. V. **Requirements engineering: From system goals to UML models to software**. [*S. l.*]: Chichester, UK: John Wiley & Sons, 2009. v. 10.

LEE, J.; KOTONYA, G.; ROBINSON, D. Engineering service-based dynamic software product lines. **IEEE Computer Architecture Letters**, IEEE Computer Society, [*S. l.*], v. 45, n. 10, p. 49–55, 2012.

MAURO, J.; NIEKE, M.; SEIDL, C.; YU, I. C. Context-aware reconfiguration in evolving software product lines. **Science of Computer Programming**, Elsevier, [*S. l.*], v. 163, p. 139–159, 2018.

MENDONCA, M.; WĄSOWSKI, A.; CZARNECKI, K. Sat-based analysis of feature models is easy. In: **Proceedings of the 13th International Software Product Line Conference**. [*S. l.*: *s. n.*], 2009. p. 231–240.

OLAECHEA, R.; ATLEE, J.; LEGAY, A.; FAHRENBERG, U. Trace checking for dynamic software product lines. In: **Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems**. [*S. l.*: *s. n.*], 2018. p. 69–75.

PFANNEMULLER, M.; KRUPITZER, C.; WECKESSER, M.; BECKER, C. A dynamic software product line approach for adaptation planning in autonomic computing systems. In: IEEE. **2017 IEEE International Conference on Autonomic Computing (ICAC)**. [*S. l.*], 2017. p. 247–254.

POHL, K.; BÖCKLE, G.; LINDEN, F. J. van D. **Software product line engineering: foundations, principles and techniques**. [*S. l.*]: Springer Science & Business Media, 2005.

SALLER, K. Model-based runtime adaptation of resource constrained devices. Universitäts-und Landesbibliothek Darmstadt, [*S. l.*], 2015.

SANTOS, E. B. dos; ANDRADE, R. M. de C.; SANTOS, I. de S. Runtime monitoring of behavioral properties in dynamically adaptive systems. In: **Proceedings of the XXXIII Brazilian Symposium on Software Engineering**. [*S. l.*: *s. n.*], 2019. p. 377–386.

SAWYER, P.; MAZO, R.; DIAZ, D.; SALINESI, C.; HUGHES, D. Using constraint programming to manage configurations in self-adaptive systems. **Computer**, IEEE, [*S. l.*], v. 45, n. 10, p. 56–63, 2012.

SOUSA, A.; UCHôA, A.; FERNANDES, E.; BEZERRA, C. I. M.; MONTEIRO, J. M.; ANDRADE, R. M. C. Rem4dspl: A requirements engineering method for dynamic software product lines. In: **Proceedings of the XVIII Brazilian Symposium on Software Quality**. New York, NY, USA: Association for Computing Machinery, 2019. (SBQS'19), p. 129–138. ISBN 9781450372824. Disponível em: https://doi.org/10.1145/3364641.3364656. Acesso em: 7 out. 2020.

TANG, Y.; LEUNG, H. Constructing feature model by identifying variability-aware modules. In: IEEE. **2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)**. [*S. l.*], 2017. p. 263–274.

TARTLER, R.; LOHMANN, D.; SINCERO, J.; SCHRÖDER-PREIKSCHAT, W. Feature consistency in compile-time-configurable system software: Facing the linux 10,000 feature problem. In: **Proceedings of the sixth conference on Computer systems**. [*S. l.*: *s. n.*], 2011. p. 47–60.

UCHÔA, A. G.; BEZERRA, C. I.; MACHADO, I. C.; MONTEIRO, J. M.; ANDRADE, R. M. Reminder: an approach to modeling non-functional properties in dynamic software product lines. In: SPRINGER. **International Conference on Software Reuse**. [*S. l.*], 2017. p. 65–73.

UCHÔA, A. G.; LIMA, L. P.; BEZERRA, C. I.; MONTEIRO, J. M.; ANDRADE, R. M. Dymmer-nfp: Modeling non-functional properties and multiple context adaptation scenarios in software product lines. In: SPRINGER. **International Conference on Software Reuse**. [*S. l.*], 2017. p. 175–183.

WECKESSER, M.; KLUGE, R.; PFANNEMÜLLER, M.; MATTHÉ, M.; SCHÜRR, A.; BECKER, C. Optimal reconfiguration of dynamic software product lines based on performance-influence models. In: **Proceedings of the 22nd International Systems and Software Product Line Conference-Volume 1**. [*S. l.*: *s. n.*], 2018. p. 98–109.

APÊNDICE A – FORMULÁRIO DE CONSENTIMENTO

1. **Objetivo**: Avaliar a eficácia do método de adaptação no suporte a modelagem dos agentes de contexto e realização do processo de reconfiguração dinâmica em tempo real.
2. **Procedimento**: Para alcançar os objetivos do estudo serão executadas 4 etapas: 1) Aplicação de um questionário de caracterização dos participantes; 2) Treinamento dos participantes sobre o uso da ferramenta adequadamente; 3) Execução do estudo com os participantes desenvolvendo um conjunto de atividades relacionadas a identificação e modelagem de agentes de contexto, vinculação dos agentes e simulação do processo de adaptação em tempo de execução; 4) Aplicação de um formulário de avaliação sobre o estudo.
3. **Confidencialidade**: As informações coletadas no estudo são confidenciais. O nome dos participantes não será divulgado. O participante deve se comprometer a não divulgar os resultados enquanto o estudo não for finalizado, mantendo em sigilo as técnicas e documentos apresentados durante o experimento.
4. **Liberdade de Desistência**: O participante está ciente de que pode fazer perguntas a qualquer momento, bem como solicitar a exclusão das informações fornecidas ou comunicar a desistência na participação do estudo. Por fim, concorda que sua participação é livre e de espontânea vontade com o único intuito de contribuir com o avanço e desenvolvimento de técnicas e processos para a Engenharia de Software.
5. **Pesquisador Responsável**: Rafael Gonçalves Lima Universidade Federal do Ceará - Campus Quixadá
6. **Professora Responsável**: Profª Drª. Carla Ilane Moreira Bezerra Universidade Federal do Ceará - Campus Quixadá
7. **Aceite**: Declaro ter mais de 18 anos de idade e concordar em participar de um estudo conduzido por Rafael Gonçalves Lima na Universidade Federal do Ceará - Campus Quixadá.

APÊNDICE B – FORMULÁRIO DE CARACTERIZAÇÃO

Formulário de caracterização dos participante do experimento controlado em Modelagem de Linhas de Produtos de Software Dinâmicas. Este formulário contém perguntas sobre a experiência acadêmica e profissional do participante.

1. **Perfil do Especialista**

   Questão 1. Nome Completo.

   Questão 2. E-mail.

   Questão 3. Instituição.

   Questão 4. Qual o seu nível de formação acadêmica?

   a) Pós-Doutorado
   b) Doutorado
   c) Doutorando
   d) Mestrado
   e) Graduado
   f) Graduando

   Questão 5. Quantos anos desde a sua formação acadêmica?

   a) Menos de 1 ano
   b) Mais de 1 ano e até 5 anos
   c) Mais de 1 ano e menos de 10 anos
   d) Mais de 10 anos

2. **Conhecimento Técnico**: Nesta seção há questões relacionadas aos aspectos técnicos do estudo. O objetivo da seção é de compreender a familiaridade do participante e o domínio do conteúdo do estudo.

   Questão 1. Qual o seu tempo de experiência em Linhas de Produtos de Software?

   a) Menos de 1 ano
   b) Mais de 1 ano e até 5 anos
   c) Mais de 1 ano e menos de 10 anos
   d) Mais de 10 anos

   Questão 2. Você já aplicou a abordagem de Linhas de Produtos de Software durante o desenvolvimento de algum software?

   a) Sim, mas somente em pesquisas acadêmicas

b) Sim, mas somente no domínio da indústria

c) Sim, em ambos, pesquisa e indústria

Questão 3. Qual o seu tempo de experiência em Linhas de Produtos de Software Dinâmicas?

a) Menos de 1 ano

b) Mais de 1 ano e até 5 anos

c) Mais de 1 ano e menos de 10 anos

d) Mais de 10 anos

Questão 4. Você já aplicou a abordagem de Linhas de Produtos de Software Dinâmicas durante o desenvolvimento de algum software?

a) Sim, mas somente em pesquisas acadêmicas

b) Sim, mas somente no domínio da indústria

c) Sim, em ambos, pesquisa e indústria

Questão 5. Qual o seu tempo de experiência com modelagem de software?

a) Menos de 1 ano

b) Mais de 1 ano e até 5 anos

c) Mais de 1 ano e menos de 10 anos

d) Mais de 10 anos

Questão 6. Você já usou ou analisou uma técnica de modelagem para desenvolver software?

a) Sim, mas somente em pesquisas acadêmicas

b) Sim, mas somente no domínio da indústria

c) Sim, em ambos, pesquisa e indústria

Questão 7. Modelo de *Features*?

a) Eu nunca ouvi falar sobre

b) Já ouvi falar, mas nunca usei

c) Eu tenho algum conhecimento sobre, mas nunca usei

d) Eu tenho um conhecimento médio e as vezes uso

e) Sou um especialista no assunto e sempre uso

Questão 8. Manutenibilidade de Modelo de *Features*?

a) Eu nunca ouvi falar sobre

b) Já ouvi falar, mas nunca usei

c) Eu tenho algum conhecimento sobre, mas nunca usei

d) Eu tenho um conhecimento médio e as vezes uso

e) Sou um especialista no assunto e sempre uso

Questão 9. Qual o seu tempo de experiência em modelagem de software com foco em reconfiguração dinâmica?

a) Menos de 1 ano

b) Mais de 1 ano e até 5 anos

c) Mais de 1 ano e menos de 10 anos

d) Mais de 10 anos

Questão 10. Você já usou ou analisou alguma técnica de modelagem de softwares com foco em reconfiguração dinâmica?

a) Sim, mas somente em pesquisas acadêmicas

b) Sim, mas somente no domínio da indústria

c) Sim, em ambos, pesquisa e indústria

Questão 11. Por favor, indique seu grau de experiência nos tópicos abaixo, seguindo a escala de 5 pontos

	Excelente	Alto	Bom	Médio	Baixo
LPS					
LPSD					
Modelo de Features					
Modelagem de Agentes de Contexto					
Reconfiguração Dinâmica					
Modelagem para Reconfiguração Dinâmica					

APÊNDICE C – DESCRIÇÃO DAS TAREFAS

1. **Apresentação**: Este documento contém instruções sobre a execução das tarefas do estudo. Por se tratar de um estudo de observação, sempre que possível, exponha suas dúvidas, desse modo o pesquisador pode melhor avaliar os resultados obtidos. Questione e comente sobre o que achar necessário.
2. **Contextualização**: As Linhas de Produtos de Softwares (LPS) são uma estratégia de desenvolvimento aplicada a uma família de produtos de software, cujas características e funcionalidades são semelhantes, desse modo, aumentando o suporte ao reúso, permitindo o desenvolvimento de softwares com menor tempo de comercialização, custos inferiores e melhor qualidade.

   As Linhas de Produtos de Software Dinâmicas (LPSD) são baseadas em LPS, porém adotam um conjunto de técnicas que permitam a adaptação dos recursos dinamicamente, com a variabilidade sendo gerenciada em tempo de execução.

   O Modelo de *features* auxilia no gerenciamento e na validação individual da configuração dos produtos da LPS. Além disso, dependendo do nível de abstração, um modelo de *features* pode permitir a visualização das características mais proeminentes do produto, contribuindo para as melhorias do software e para a análise de domínio.

   Por fim, a reconfiguração dinâmica é a capacidade de um sistema de gerenciar a variabilidade em tempo de execução. A variabilidade do tempo de execução define as opções de design do produto visíveis para clientes e usuários do sistema, que podem selecionar entre as opções configuráveis disponíveis.
3. **Tarefas**: O modelo de *features* utilizado para essa experimentação é o Nexus DSPL. O modelo pode ser encontrado no repositório da DyMMer *web* acessando https://dymmerufc.github.io/#/fmodel-manager/5cb66afc0268200004948a7e. Após abrir o modelo de *features* indicado, siga as instruções abaixo para realiazação do experimento:
   a) Analisando o modelo de *features*: A tarefa inicial consite na análide do modelo de *features*, na sua estrutura e organização, e no conjunto de restrições existentes. Após essa análise inciail, localize o botão chamado "***OPEN ADAPTATION MECHANISM***"e clique para abrir o método.
   b) Identificar agentes de contexto: A identificação dos agentes é realizada com a análise do modelo de *features*. Você deve observar quais *features* estão relacinadas a agentes como sensores, drivers e etc. Após identificar um novo agente, clique no botão

"***Add Agent***"e insirao-o como indicado. Após criar um agente, você pode definir contextos para ele, caso não existam contextos aparentes, você deve criar um chamado "***Power***"para indicar que o agente está ligado ou desligado. Esse processo pode ser realizado clicando-se no botão "***Add Context***"dentro do *card* do agente.

c) Vinculação dos contextos à *features*: Uma vez finalizado o processo de criação de agentes e contextos, você deverá vinculá-los às *features* correspondentes. Para fazer isso localize a *feature* no qual se deseja vincular a um contexto, clique em "***Link Agent***", uma caixa de seleção será aberta. Em seguida, selecione o agente, o contexto e o valor do contexto. Repita o processo até a vinculação de todos os contextos.

d) Simulação da adaptação: Para realização da simulação, clique no botão "***Simulate***". Ao inciar a simulação, a *interface* mudará e novos botões serão exibidos. No painel dos agentes de contexto, voce deve ligar e desligar os contextos e observar se as *features* do modelo são ativadas ou desativadas.

APÊNDICE D – *FORMULÁRIO DE AVALIAÇÃO DO MÉTODO*

Este formulário tem como objetivo coletar informações de cada participante sobre o uso do método de adaptação.

Questão 1. Você concorda que a utilização do Método de Adaptação auxiliou na modelagem dos Agentes de Contexto? Justifique sua resposta.

(a) Concordo totalmente
(b) Concordo parcialmente
(c) Indeciso
(d) Discordo em parte
(e) Discordo totalmente

Questão 2. Você concorda que o Método de Adaptação permitiu a visualização do processo de reconfiguração após a mudança do contexto? Justifique sua resposta.

(a) Concordo totalmente
(b) Concordo parcialmente
(c) Indeciso
(d) Discordo em parte
(e) Discordo totalmente

Questão 3. Na sua percepção, qual foi o esforço para identificar os agentes relacionados ao modelo de *features*? Justifique sua resposta.

(a) Alto
(b) Médio
(c) Baixo

Questão 4. Na sua percepção, qual foi o esforço para modelar os contextos de cada agente identificado? Justifique sua resposta.

(a) Alto
(b) Médio
(c) Baixo

Questão 5. Em relação ao processo de simulação da adaptação, você acredita que: Justifique sua resposta.

(a) Ajudou a entender quais configurações o modelo assumirá a partir de cada alteração do contexto

(b) Dificultou no entendimento, algumas simulações foram confusas ou as informações insuficientes.

Questão 6. Qual foi a sua principal dificuldade no uso da ferramenta?

Questão 7. Quais são as vantagens e desvantagens do Método de Adaptação?

Questão 8. Quais sugestões você teria para melhorar o Método de Adaptação?