Universidade Federal Rural do Semi-Árido
Universidade do Estado do Rio Grande do Norte

Programa de Pós-Graduação em
Ciência da Computação

Dissertação de Mestrado

Integração de LPS com Microsserviços para o Desenvolvimento de SaaS Multilocatário: proposta de diretrizes para o projeto arquitetural com variabilidades implementadas por meio de microsserviços

Mestrando
Manoel Marisergio Alves de Oliveira

Orientador
Paulo Gabriel Gadelha Queiroz

Mossoró - RN
agosto/2023

Manoel Marisergio Alves de Oliveira

Integração de LPS com Microsserviços para o Desenvolvimento de SaaS Multilocatário: proposta de diretrizes para o projeto arquitetural com variabilidades implementadas por meio de microsserviços

Dissertação apresentada ao Mestrado em Computação do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal Rural do Semi-Árido/Universidade Estadual do Rio Grande do Norte como requisito para obtenção do título de Mestre em Ciência da Computação.

Engenharia de Software: Engenharia de Software e Sistemas Computacionais

Mossoró - RN

agosto/2023

Dados Internacionais de Catalogação na Publicação (CIP)
Biblioteca Central Orlando Teixeira (BCOT)
Setor de Informação e Referência (SIR)

Autor
Integração de LPS com Microsserviços para o Desenvolvimento de SaaS Multilocatário: proposta de diretrizes para o projeto arquitetural com variabilidades implementadas por meio de microsserviços/ Manoel Marisergio Alves de Oliveira. Mossoró - RN, agosto/2023.
126 p.; 30 cm.

Paulo Gabriel Gadelha Queiroz

Dissertação de Mestrado – Universidade Federal Rural do Semiárido - Universidade Estadual do Rio Grande do Norte, Mossoró - RN, 10 de julho de 2023.

1. Linha de Produto de Software. 2. Reúso de Software. 3. IOT. I. Paulo Gabriel Gadelha Queiroz. II. Universidade Federal Rural do Semi-Árido. III. Integração de LPS com Microsserviços para o Desenvolvimento de SaaS Multilocatário: proposta de diretrizes para o projeto arquitetural com variabilidades implementadas por meio de microsserviços

CDU 02:141:005.7

Bibliotecário-Documentalista
Nome do profissional, Bib. Me. (CRB-15/10.000)

Manoel Marisergio Alves de Oliveira

Integração de LPS com Microsserviços para o Desenvolvimento de SaaS Multilocatário: proposta de diretrizes para o projeto arquitetural com variabilidades implementadas por meio de microsserviços

Dissertação apresentada ao Mestrado em Computação do Programa de Pós-Graduação em Ciência da Computação da Universidade Federal Rural do Semi-Árido/Universidade Estadual do Rio Grande do Norte como requisito para obtenção do título de Mestre em Ciência da Computação.

Linha de Pesquisa: Engenharia de Software e Sistemas Computacionais

Defendida em _____/____/_____.

BANCA EXAMINADORA

Paulo Gabriel Gadelha Queiroz, Prof. Dr. (UFERSA)
Presidente

Lenardo Chaves e Silva, Prof. Dr. (UFERSA)
Membro Examinador

Valdemar Vicente Graciano Neto, Prof. Dr. (UFG)
Membro Examinador

Mossoró - RN

agosto/2023

Agradecimentos

Ao senhor da minha vida, meu Pai Celestial, que me guia e me protege e que sem Ele e sem o Seu amor nada valeria a pena; sei que Ele não me abandonará, mesmo que os meus pais me abandonem, o Deus de Abraão, de Isac e Jacó.

Aos meus pais, Maria Vilani de Oliveira, Francisco Alves Feitosa, e aos meus irmãos, pelo amor, a confiança e o apoio incondicional.

Ao meu cunhado e amigo, Félix Mariano, pela sua solicitude e companheirismo.

Aos meus amigos, em especial Renato Alexandre, Thiago Felippe e Vaux Sandino, pelo apoio, companheirismo e pelos momentos de descontração.

Aos meus professores e, em especial, ao meu orientador, Paulo Gabriel, por me guiar sabiamente neste trabalho, por confiar na minha singela capacidade acadêmica, por compartilhar o conhecimento e fornecer oportunidade de crescimento.

A minha família, minha esposa, Renata Argélia Santiago Lima, e meus filhos, Sérgio Lima Oliveira e Maris Lima Oliveira, por todo o apoio, motivação e compreensão.

Resumo

Projetar sistemas para atender a uma grande quantidade de pessoas, que possuem demandas semelhantes, mas também apresentam necessidades variadas e geram um enorme volume de dados, exige uma arquitetura de software que possibilite uma evolução constante, seja fácil de manter e tenha a capacidade de escalar de forma inteligente. Embora a técnica de Linha de Produto de Software (LPS) em conjunto com uma arquitetura de microsserviços se mostre promissora para atender a esses requisitos, essa integração não é trivial. Dessa forma, foi planejada e executada uma revisão sistemática da literatura que identificou três arquiteturas construídas a partir da combinação dessas técnicas. Entretanto, as arquiteturas encontradas eram complexas e aumentavam o time-to-market, visto que propunham a implementação de todas as características da LPS por meio de microsserviços. Assim, de modo a reduzir a complexidade de desenvolvimento e consequentemente, reduzir o time-to-market, o principal resultado obtido a partir deste trabalho é um conjunto de diretrizes para guiar os engenheiros de software no projeto de uma arquitetura híbrida, por meio da combinação de monólitos e microsserviços. As diretrizes foram elaboradas com suporte do método pesquisa-ação, durante a realização de um estudo de caso, que consistiu na definição da arquitetura de uma LPS de clínicas médicas como um *Software as a Service* Multilocatário. Adicionalmente, essa arquitetura foi analisada e comparada com uma arquitetura construída a partir das diretrizes apresentadas em um dos trabalhos encontrados na RSL. Para avaliar a objetividade das diretrizes, foi realizado um experimento controlado com participação de representantes da indústria e da academia, no qual as diretrizes apresentaram 86% de assertividade.

palavras-chaves: Linha de Produto de Software; Microsserviços; Arquitetura de Software; Reúso de Software; SaaS Multilocatário.

Abstract

Designing systems to serve a large number of people, who have similar demands but also diverse needs and generate a vast volume of data, requires a software architecture that enables constant evolution, is easy to maintain, and has the capacity to scale intelligently. Although the Software Product Line (SPL) technique in conjunction with a microservices architecture proves are promising techniques to meet these requirements, this integration is not trivial. Therefore, we planned and executed a Systematic Literature Review (SLR) that identified 3 architectures built from the combination of these techniques. However, the found architectures were complex and increased the time-to-market, as they proposed implementing all SPL features through microservices. Thus, aiming to reduce the development complexity and consequently decrease the time-to-market, the main result obtained from this work is a set of guidelines to guide software engineers in designing a hybrid architecture through the combination of monoliths and microservices. The guidelines were elaborated with the support of the action research method during a case study, which consisted of defining the architecture of an SPL for medical clinics as a Multitenant Software as a Service. Additionally, this architecture was analyzed and compared with an architecture built from the guidelines presented in one of the works found in the SLR. To evaluate the objectivity of the guidelines, a controlled experiment was conducted with the participation of industry and academic representatives, in which the guidelines showed 86% of assertiveness.

Key-words: Software Product Line; Microservices; Software Architecture; Software Reuse; SaaS Multi-tenant.

Sumário

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	*Application Programming Interface*
BFF	*backend for frontend*
BPMN	*Business Process Model and Notation*
CRUD	*Create, Read, Update, Delete*
DTO	*Data Transfer Object*
DDD	*Domain Driven Design*
ERP	*Enterprise Resource Planning*
FODA	*Feature Oriented Domain Analysis*
HTTP	*Hypertext Transfer Protocol*
LOC	*Lines of Code*
LPS	Linha de Produto de Software
MS	Microsserviço
PLA	*Product Line Architecture*
REST	*Representational State Transfer*
RSL	Revisão Sistemática da Literatura
SPA	*Single Page Application*
SaaS	*Software as a Service*
UML	*Unified Modeling Language*
UML-DOP	*UML Delta-Oriented Programming*
URL	*Uniform Resource Locator*
WS	*Web Service*

1 Introdução

A dinamicidade dos processos de negócio e o rápido surgimento de novas tecnologias são aspectos marcantes no cenário atual de desenvolvimento de software. Com um mercado cada vez mais dinâmico, a demanda por software capaz de evoluir, escalar, oferecer segurança e resiliência tem aumentado significativamente. Assim, a capacidade de arquitetar software que possa se adaptar rapidamente a mudanças é um requisito essencial para garantir a qualidade de um produto. Além disso, há uma crescente exigência por software de alta qualidade e com menor tempo de entrega (VARAJAO, 2021).

Dessa forma, o desenvolvimento de software destinado a atender às exigências emergentes não pode ser abordado de qualquer forma. Portanto, uma análise aprofundada se faz necessária a fim de identificar as metodologias da engenharia de software que podem ser aplicadas de forma sistemática para conceber soluções que satisfaçam integralmente esses requisitos. Entre o conjunto de técnicas e abordagens já documentadas na literatura especializada, a combinação sinérgica de duas ou mais dessas técnicas frequentemente carece de diretrizes sistemáticas para auxiliar os engenheiros de software na otimização e na harmonização eficaz de cada técnica empregada.

Considerando esse cenário, a aplicação de sistemas distribuídos e computação em nuvem tem desempenhado um papel fundamental. Desde a década de 1990 (COULOURIS et al., 2013), essas tecnologias vêm crescendo consideravelmente e promovendo uma renovação no gerenciamento de armazenamento e tráfego de dados nas redes de computadores. Essa evolução tem possibilitado novas formas de arquitetar e desenvolver softwares. Atualmente, a maioria dos softwares são projetados desde sua concepção para funcionar de forma distribuída, seja utilizando *Applications Programming Interface* (API) e microsserviços ou mesmo através da replicação de instâncias de servidores e balanceamento de carga. Essa abordagem distribuída e baseada em nuvem oferece maior flexibilidade, escalabilidade e capacidade de adaptação às demandas do mercado em constante mudança.

Diante desse ambiente, a motivação inicial deste trabalho surgiu da demanda de uma empresa que precisava reestruturar o software desenvolvido para um hospital e adaptá-lo para atender às necessidades de diversas clínicas médicas, transformando-o em uma Linha de Produto de Software (LPS) orientada a serviço. As Linhas de Produtos têm o propósito de entregar software planejado para reúso, de modo a atender a um determinado domínio de mercado e que possa oferecer recursos customizáveis, de acordo com as necessidades de cada cliente (APEL et al., 2016; NGUYEN et al., 2019; LINDEN; SCHMID; ROMMES, 2007). Assim, essa técnica pode contribuir para modelar o domínio de clínicas e sistematizar o reúso do software desenvolvido para hospitais. Para atender aos

requisitos deste mercado dinâmico e se beneficiar das vantagens de sistemas distribuídos, optou-se por projetar uma LPS orientada a serviço para funcionar como um *Software as a Service* (SaaS) multilocatário.

Um relatório produzido pela empresa de pesquisa e consultoria de mercado *Grand View* (2022) concluiu que o mercado de SaaS gera um capital de bilhões de dólares por ano. Segundo Mell, Grance et al. (2011), SaaS é uma plataforma que fornece recursos para os clientes a partir de uma infraestrutura em nuvem, acessível por meio de interfaces, como é o caso de um navegador web ou uma interface de aplicativo. O cliente é um locatário que possui acesso aos recursos do SaaS, mas não controla a infraestrutura, como rede, servidores, sistemas operacionais e armazenamento.

Dessa forma, os recursos computacionais podem ser agrupados para atender a vários clientes usando o modelo de multilocatário, que permite a alocação desses recursos de acordo com a quantidade de clientes ativos. Nesse sentido, um SaaS oferece as seguintes vantagens: economia de escala, facilidade de manutenção e evolução, pois possui uma única base de código para todos os locatários e isolamento dos dados pertencentes a cada cliente. Adicionalmente, ao ser desenvolvido com a técnica de LPS, é possível fornecer alta capacidade de configurabilidade, ao passo que a LPS é projetada para permitir a customização dos recursos do software de acordo com as necessidades de cada locatário.

Além de fornecer um ambiente para a adoção do modelo SaaS, uma LPS orientada a serviço pode utilizar os *web services* para modelar e implementar as variabilidades da LPS, já que eles podem ser desenvolvidos de forma independente e podem ser combinados ou integrados facilmente para fornecer serviços às demandas de diversos clientes. Nesse sentido, alguns pesquisadores na área de LPS dedicaram atenção para investigar e propor trabalhos de Linha de Produto de Software orientada a serviço.

Queiroz (2009), por exemplo, propôs uma abordagem de desenvolvimento de linha de produtos com arquitetura orientada a serviços denominada SoProL-WS. Destaca-se nessa abordagem um conjunto de detalhes e uma delimitação clara de atividades que possuem o objetivo de construir uma LPS com arquitetura orientada a serviços. As atividades propostas iniciam pela eliciação e especificação dos requisitos, incluem a modelagem dos processos de negócio, definição da modelagem dos casos de uso e das características, passam por todas as etapas de análise de projeto, e finalizam com as atividades de geração dos produtos derivados da LPS.

Apesar da abordagem SoProL-WS ter sido a inspiração inicial para este trabalho, optou-se por investigar outras abordagens por dois motivos:

- a abordagem SoProL-WS se baseava na construção de uma arquitetura orientada a serviços utilizando serviços de alta granularidade. Atualmente, outros recursos como

API REST [1] e microsserviços são mais populares no mercado. Observa-se que essa diferença tecnológica pode exigir significativas mudanças na abordagem;

- o segundo fator está relacionado a algumas limitações dessa abordagem, tal como na identificação dos serviços, a partir do modelo de características, os recursos que ficam nas folhas são mapeados para operações de um único *web service*. Nos casos em que uma característica faz parte de um ponto de variação, a abordagem pode levar a criação de *Web Service* (WS) com alto acoplamento, pois um produto derivado da LPS terá acesso a um serviço que outros produtos terão, mas que não utilizam as mesmas operações. O ideal seria ter um WS para cada opção de um ponto de variação.

Assim, com o objetivo de identificar abordagens capazes de auxiliar na construção de uma LPS orientada a serviço que utilize recursos atuais, foi realizada uma revisão sistemática da literatura (RSL) entre junho de 2021 e dezembro de 2022, a partir da qual observou-se que nos últimos quatro anos houve uma crescente pesquisa sobre a integração de LPS e arquitetura de microsserviços como duas técnicas promissoras para atender aos requisitos do mercado atual. Os microsserviços são pequenos serviços independentes que possuem uma responsabilidade muito bem definida. A arquitetura de microsserviços pode oferecer vantagens, como a possibilidade de integração com heterogeneidade de tecnologias, alto grau de escalabilidade, maior disponibilidade, capacidade de redundância seletiva e econômica, como no caso de replicação apenas dos pontos de maior demanda, agilidade para alterar ou adicionar recursos, além de promover um maior desacoplamento e coesão, favorecendo a independência de equipes e o aumento da produtividade (LUZ et al., 2018; TAIBI; LENARDUZZI; PAHL, 2017).

Como resultado dessa revisão sistemática, pode-se destacar os seguintes trabalhos: (i) Benni et al. (2020), em seu trabalho, de forma periférica, propuseram uma solução para mapear recursos em microsserviços, tomando como base que cada microsserviço pode implementar vários recursos por meio de *endpoints* de API; (ii) Costa et al. (2019) propuseram um metamodelo para guiar arquitetos de software na construção da arquitetura da LPS. Eles utilizaram o modelo de características para extrair os microsserviços de acordo com os nós folha, que devem servir de entrada para o projeto de arquitetura da LPS; (iii) Becker e Lucrédio (2020) propuseram mapear os recursos em microsserviços, considerando que cada operação sobre uma entidade de registro deve gerar um microsserviço; e, (iv) Setyautami et al. (2020), em resposta a alguns desafios propostos por Assunção, Krüger e Mendonça (2020), apresentaram soluções para identificação de recursos, modelagem de

[1] API REST (*Application Programming Interface Representational State Transfer*) é um estilo de arquitetura de software que utiliza os princípios do protocolo HTTP para permitir a comunicação e interação entre sistemas distribuídos. As APIs REST usam métodos HTTP para realizar operações em recursos, os quais são representados por URLs (*Uniform Resource Locators*). Os dados são transferidos comumente em formato JSON (ZHOU et al., 2014).

variabilidade, arquitetura e reengenharia de linha de produtos com base nas funcionalidades de seis aplicativos de venda eletrônica e que foram desenvolvidos com microsserviços. Conforme analisado detalhadamente no Capítulo 3, os resultados selecionados na RSL podem ser categorizados em dois tipos de abordagens:

- abordagens cuja execução resulta em serviços muito grandes, de modo a perder as características de microsserviços e seus principais benefícios;
- abordagens que realmente propunham a modelagem de microsserviços, mas que pareciam utilizá-los de forma exagerada, e assim acrescentavam uma grande complexidade ao desenvolvimento e consequente aumento no tempo de entrega de produtos.

Diante dos resultados da RSL, nota-se que a integração de LPS com microsserviços ainda é algo desafiador para a engenharia de software, o que também é concluído por Ghofrani e Lübke (2018) e Jamshidi et al. (2018). Sistematizar essa integração não é uma tarefa simples, tendo em vista que existe uma diversidade de tipos de arquiteturas, além de diversos modelos e padrões de softwares que podem ser aplicados para o desenvolvimento de uma LPS implementada a partir de uma arquitetura de microsserviços. Além disso, os trabalhos publicados na literatura especializada são recentes e não apresentam estratégias bem definidas, o que é corroborado por Assunção, Krüger e Mendonça (2020), que descrevem seis desafios para integrar LPS e microsserviços. Dentre esses desafios, os quatro mais relevantes para esta pesquisa são:

- identificar os recursos dos sistemas baseados em microsserviços;
- definir um modelo de variabilidade que represente as semelhanças e a variabilidade de uma LPS baseado em microsserviços;
- definir uma arquitetura de linha de produto orientada a microsserviços que permita projetar e personalizar diferentes variantes de sistema;
- propor uma solução que permita intercambiar microsserviços de diferentes tecnologias em um sistema.

Face aos estudos realizados, por se tratar de um projeto em parceria com uma empresa do setor privado, mediando fatores como tempo de entrega e qualidade de software, a equipe participante tomou a decisão de desenvolver uma LPS do início, em vez de adaptar o software existente. Entretanto, alguns requisitos funcionais do software desenvolvido para hospitais foram reaproveitados.

Nesse contexto, desenvolver uma LPS no modelo SaaS demanda um projeto de arquitetura de software que inclui uma alta complexidade, se tratando da indispensabilidade de garantir que sejam atendidos diversos requisitos não funcionais, além das necessidades

particulares de diversos clientes da LPS. Adicionalmente, não foram encontradas na literatura, diretrizes sistemáticas capazes de facilitar o processo de definição da arquitetura da LPS, de modo a se beneficiar da arquitetura de microsserviços e ainda conseguir entregar software com agilidade. Observa-se que a arquitetura de microsserviços e baixo *time-to-market* parecem ser requisitos conflitantes (BAŠKARADA; NGUYEN; KORONIOS, 2020).

Conforme contextualizado anteriormente, o projeto e implementação de uma LPS utilizando microsserviços e a entrega de soluções como um SaaS multilocatário traz alguns desafios para os Engenheiros de Software. Assim, surgiu a seguinte questão de pesquisa:

- ***Como integrar LPS com microsserviços para construir um SaaS Multi-locatário, de forma a aproveitar os principais benefícios da arquitetura de microsserviços e, ao mesmo tempo, reduzir a complexidade inerente a essa arquitetura?***

Para responder essa questão deve-se explorar as melhores práticas e técnicas para a concepção de uma arquitetura que combine customização e reúso das LPS, flexibilidade dos microsserviços e escalabilidade do modelo SaaS, permitindo atender às demandas de múltiplos clientes do mesmo domínio de negócio. Para tanto, baseado nas lacunas identificadas na RSL, definiu-se as seguintes questões específicas:

- Como modelar características da LPS de forma que possa atender a requisitos da arquitetura de microsserviços?
- Como mapear características para microsserviços?
- Como projetar uma arquitetura híbrida para uma LPS que opere como um SaaS multilocátario de forma que possa atender aos principais requisitos da arquitetura de microsserviços?

Face a essas questões, a seguinte hipótese foi levantada: construir uma Linha de Produto de Software com uma arquitetura híbrida, que utilize microsserviços para implementar somente as variabilidades enquanto o núcleo pode ser composto de aplicações *backend*, [2] se torna mais vantajoso pois esta arquitetura conseguiria fornecer as principais vantagens de uma arquitetura de microsservicos, mas com redução da complexidade de desenvolvimento.

[2] O termo *backend* é utilizado para se referir à parte da aplicação responsável por implementar a lógica do software e fornecer uma interface de comunicação com outras aplicações, como, por exemplo, API REST.

1.1 Objetivos

O principal objetivo deste trabalho é definir um conjunto de diretrizes capaz de guiar os engenheiros de software a projetarem uma arquitetura híbrida para uma Linha de Produto de Software que opere como um SaaS multilocatário e comparar essa arquitetura com uma arquitetura de LPS projetada apenas com microsserviços. Essa arquitetura híbrida deve possuir componentes compostos por microsserviços e por aplicações *backend*.

Essa arquitetura deve atender aos requisitos de escalabilidade, flexibilidade, resiliência, entre outros e ainda, mitigar a complexidade inerente ao desenvolvimento de uma LPS composta puramente por microsserviços. Para alcançar o objetivo principal, foram estabelecidos os seguintes objetivos específicos:

- definir diretrizes para a construção do modelo de características de LPS's que possa contemplar a integração de microsserviços e APIs;
- definir diretrizes para o mapeamento de características para microsserviços e APIs. Esta meta consiste em definir os microsserviços ou APIs que devem ser implementados para atender às necessidades de cada característica da LPS, levando-se em consideração questões como granularidade e dependências entre as características, visto que esses pontos são importantes para uma modelagem de microsserviços que assegurem os princípios de reúso e independência;
- definir diretrizes para a modelagem da arquitetura de LPS's baseada em microsserviços e aplicações *backend*, de forma a fornecer parâmetros para a implementação dos seguintes requisitos: (i) comunicação entre microsserviços; (ii) autenticação e autorização; (iii) isolamento, consistência e sincronização dos dados de cada locatário.

1.2 Contribuições Científicas

As contribuições científicas deste trabalho visam proporcionar soluções para o desenvolvimento de aplicações SaaS multilocatário, utilizando a combinação de LPS e microsserviços. Nesse sentido, são três as contribuições: uma abordagem leve para a construção de LPS com uma arquitetura híbrida; um conjunto de diretrizes para mapeamento entre modelo de características e componentes arquiteturais; e, uma arquitetura de referência para LPS que operem como um SaaS multilocatário.

Uma das principais contribuições deste trabalho é a proposição de uma abordagem leve que integra a técnica de LPS com arquitetura de microsserviços, pois se baseia na análise de domínio por meio da observação de sistemas existentes, modelagem de características e estória dos usuários. Esses três artefatos são leves quando comparados com outras abordagens que utilizam vários diagramas UML (*Unified Modeling Language*),

como a abordagem de Gomaa (2004) e Queiroz (2009). Essa abordagem busca oferecer flexibilidade e reutilização no desenvolvimento de SaaS multilocatário por meio da identificação e encapsulamento de funcionalidades comuns e variabilidades em microsserviços independentes. Com isso, é possível obter uma arquitetura mais modular e escalável, permitindo o desenvolvimento ágil de novas funcionalidades e adaptação a diferentes demandas dos usuários.

Outra importante contribuição desta dissertação são as diretrizes propostas para o mapeamento eficiente entre o modelo de características e os componentes arquiteturais em uma abordagem de LPS com microsserviços. Essas diretrizes visam auxiliar os engenheiros de software na identificação e implementação das variabilidades da LPS, garantindo que as características definidas no modelo de características sejam adequadamente refletidas nos microsserviços correspondentes. Com esse mapeamento definido de forma coesa, torna-se possível manter a consistência e a integridade da arquitetura ao longo do desenvolvimento e evolução da LPS.

Por fim, este trabalho apresenta uma arquitetura de referência para uma LPS que opera como um SaaS multilocatário. A arquitetura proposta incorpora as diretrizes previamente definidas e permite a criação e gerenciamento eficiente de várias instâncias de software atendendo a diferentes locatários em um ambiente compartilhado. Essa arquitetura de referência fornece uma estrutura sólida para o desenvolvimento de aplicações SaaS multilocatário, considerando a flexibilidade, escalabilidade e manutenibilidade como pilares fundamentais.

As contribuições apresentadas neste trabalho visam ampliar a compreensão sobre a integração de LPS com microsserviços e fornecer diretrizes práticas e aplicáveis para o projeto arquitetural de SaaS multilocatário. Acreditamos que tais contribuições possam beneficiar a comunidade acadêmica e a indústria de desenvolvimento de software, abrindo caminho para a construção de sistemas mais flexíveis, eficientes e adaptáveis às demandas do mercado e dos usuários.

1.3 Procedimentos e Métodos

Este trabalho requer ações relacionadas tanto à pesquisa científica quanto à prática. Por esse motivo, escolheu-se adotar como forma de investigação, a pesquisa-ação. Esse método utiliza técnicas consagradas de pesquisa para definir as ações a serem decididas para melhorar a prática (TRIPP, 2005). A pesquisa-ação é um processo que obedece a um ciclo de tarefas que objetiva o aprimoramento da prática, ao passo que realiza uma permutação sistemática entre a investigação e a prática. Este ciclo prevê as seguintes etapas: planejamento, implementação, registro dos efeitos, e avaliação dos resultados. No decorrer do processo, estas etapas executadas em ciclo devem produzir aprendizado tanto

a respeito da prática quanto da investigação. Na Figura 1 apresenta-se as quatro etapas do ciclo básico da investigação-ação.

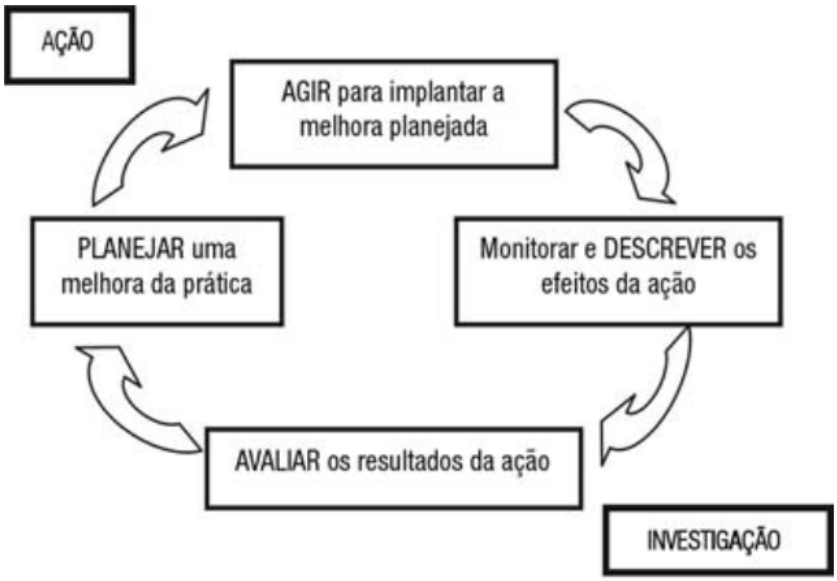

Figura 1 – Representação do ciclo básico da investigação-ação (TRIPP, 2005).

O planejamento consistiu na definição dos passos a serem seguidos nesta pesquisa: (i) estudo acerca dos fundamentos teóricos, abordagens e padrões relacionados a LPS e microsserviços; (ii) execução de uma RSL, cujo objetivo consistiu em identificar, analisar e correlacionar trabalhos que investigaram a integração de Linhas de Produtos de Software e Microsserviços; (iii) desenvolver um estudo de caso a partir da adaptação das abordagens estudadas; (iv) mapear diretrizes utilizadas para a construção da arquitetura proposta do estudo de caso; (v) abstrair o estudo de caso para o contexto geral de modo a definir uma arquitetura de referência da LPS; (vi) comparar a arquitetura proposta neste trabalho com uma arquitetura definida a partir das diretrizes encontradas no trabalho de Costa et al. (2019), a qual foi identificada como a que melhor atende aos requisitos da arquitetura de microsserviços; (vii) realizar um experimento controlado para avaliar melhor as diretrizes.

A resolução do estudo de caso se iniciou com um ciclo de reuniões semanais com *stakeholders* compostos por pesquisadores com experiência em LPS e representantes da empresa parceira que forneceu o projeto para o estudo desta pesquisa. Assim, os artefatos produzidos foram submetidos a validações e correções semanais com a participação e contribuição de toda a equipe envolvida.

As abordagens utilizadas como base para este trabalho são apresentadas no Capítulo 3 e, em suma, estabelecem que o projeto de arquitetura da LPS acontece após a especificação dos requisitos e modelagem das características. Seguindo esse princípio, após a observação dos requisitos do software da empresa parceira e de outros quatro softwares conhecidos no mercado, todos no domínio de clínicas médicas, produziu-se um documento que registra e organiza as características deles em dois grupos: no primeiro

grupo são catalogadas as características presentes em todos os softwares analisados; no segundo grupo estão as características presentes somente em alguns dos cinco softwares analisados e as características que apresentam diferenças no seu funcionamento em alguma das aplicações. Destaque-se que esta análise foi realizada com base na combinação da análise do documento de casos de uso do sistema hospitalar e da simulação de uso das funcionalidades dos quatro softwares de clínicas médicas em ambiente de amostra.

Na sequência foram confeccionadas as estórias de usuários a partir da catalogação dos requisitos e da organização das características dos cinco softwares analisados. Esse processo visou harmonizar as características semelhantes e fornecer um artefato que facilitasse a compreensão da equipe envolvida no projeto Rees (2002). Este documento proporcionou uma visão contextual do domínio e dos subdomínios praticados em softwares de clínica médica.

Uma vez que o domínio da LPS foi definido, o próximo passo foi modelar as suas características. Para alcançar esse objetivo, utilizou-se a principal técnica de modelagem de variabilidades, o *Feature Model* do método *Feature Oriented Domain Analysis* (FODA)(KANG et al., 1990), em conjunto com o princípio de *bound context* do *Domain Driven Design* (DDD)(EVANS, 2014). O foco do DDD é realizar uma modelagem com ênfase no domínio da aplicação. Segundo Evans (2014), essa técnica é recomendada para contextos que possuem algum nível de complexidade, sendo este o caso de LPS. O princípio de *bound context* fornece uma abordagem para organizar e dividir o domínio geral da aplicação com vistas a atingir a coerência das entidades modeladas.

O modelo de características foi a entrada principal para a definição da arquitetura da LPS e para o refinamento de uma das principais contribuições desta pesquisa, a definição de diretrizes para realizar o mapeamento do modelo de características para a arquitetura da LPS. Essas diretrizes foram definidas de forma cíclica e incremental, seguindo as etapas da pesquisa-ação. Na primeira iteração, decidiu-se pensar em uma abordagem que considerasse representar todas as variabilidades como microsserviços na arquitetura, enquanto os artefatos comuns deveriam ser modelados como um monólito. Durante a avaliação dessa primeira proposta, observou-se que existiam algumas variabilidades que gerariam microsserviços de alta granularidade enquanto outras gerariam um excessivo número de microsserviços. Além disso, os artefatos comuns deveriam ser subdivididos para aumentar a escalabilidade, disponibilidade e coesão da arquitetura proposta.

Esse processo de proposição de mapeamento do modelo de características para arquitetura passou por mais duas rodadas de propostas e avaliação, até que se iniciasse o processo de implementação da arquitetura para que fosse validada na prática.

Por fim, a arquitetura proposta no estudo de caso foi abstraída, para que pudesse ser representada como uma arquitetura de referência capaz de atender a diversas LPS de SaaS multilocatário que possam ser construídas para usufruir dos benefícios dos microsserviços,

mas também de modo a reduzir a complexidade de ter que manipular um número excessivo de microsserviços. Finalmente, com base em critérios definidos na Subseção 2.2.2, a arquitetura proposta foi avaliada.

Em alguns momentos de avaliação das ações foram percebidos pontos que necessitavam de melhoria. Dessa forma, foi necessário retroceder e aplicar as melhorias propostas. A seguir são listados os principais pontos identificados no decorrer do processo que sofreram alterações:

- seguindo as atividades propostas pela abordagem SoProl-WS, optou-se por uma estratégia mais leve ao trocar os casos de uso por estórias de usuários.

- na etapa de codificação dos primeiros microsserviços do estudo de caso, percebeu-se que os subdomínios da LPS projetados no modelo de características não estavam obedecendo aos princípios de coesão e baixo acoplamento. A primeira versão do modelo de características possuía um subdomínio que contemplava todos os cadastros base, que normalmente são usados por todos os demais subdomínios. Essa estratégia provocou um forte acoplamento entre os subdomínios, tornando inviável a utilização de microsserviços. Dessa forma, buscou-se na literatura conhecimento para aprimorar o modelo de características para que atendesse ao contexto de arquitetura de microsserviços. Como relatado anteriormente, escolheu-se a técnica de DDD que forneceu direcionamento para a modelagem de subdomínios contextualmente coesos e com baixo acoplamento. Assim, esse subdomínio de cadastros base foi desfeito e suas entidades foram distribuídas nos demais subdomínios, sendo necessário remodelar essas entidades para adequar as especificidades de cada subdomínio;

- a dificuldade mencionada no item anterior desencadeou uma correção nas seguintes etapas:

  I. nas diretrizes de mapeamento do modelo de características para a arquitetura, visto que foi necessário explicitar a recomendação em utilizar os princípios de DDD para se construir um modelo que contemple as demandas da arquitetura de microsserviços;

  II. houve alteração na arquitetura de referência da LPS, pois foi necessário incluir estratégias de sincronização entre as entidades de cadastros base que passaram a possuir cópias distribuídas em cada subdomínio. Por exemplo, no subdomínio atendimento, no qual o profissional de saúde pode ser um médico, mas no subdomínio financeiro, esse profissional passa a ser um funcionário. Isto provoca uma cópia não idêntica dos dados, pois ao invés de ter uma entidade com todos os possíveis dados para atender a todos os subdomínios, cada cópia possui apenas os dados necessários para aquele subdomínio a qual pertence, diminuindo

assim o acoplamento e aumentando a coesão. Para manter a integridade dos dados, tendo em vista que o profissional de saúde do setor de atendimento é a mesma pessoa que é funcionário no setor financeiro e que uma alteração do cadastro do profissional de saúde pode necessitar de atualização no cadastro desse funcionário, foi necessário adotar estratégias de serviços de mensageria para garantir a sincronia dos dados de forma não bloqueante;

III. uma vez que a arquitetura foi alterada, os modelos de dados e as implementações dos microsserviços também tiveram que ser alterados.

Por fim, conduziu-se um experimento controlado com o objetivo de avaliar a precisão das diretrizes, a conformidade dos componentes arquiteturais resultantes da aplicação dessas diretrizes com requisitos da arquitetura de microsserviços e a complexidade gerada por esses componentes.

A seguir são descritas em síntese, conforme ilustra-se na Figura 2, as etapas executadas na condução desta pesquisa.

1. Foi realizado um estudo sobre os fundamentos teóricos de LPS e microsserviços;
2. Realizou-se uma revisão sistemática da literatura sobre trabalhos publicados que abordam a integração de LPS e microsserviços. No Capítulo 3 registra-se o processo escolhido, o planejamento, a condução, os resultados e análises desta revisão;
3. Foram extraídas do estado da arte, definido no item anterior, as melhores técnicas e padrões utilizados na identificação, modelagem e implementação de variabilidades de LPS's por meio de microsserviços;
4. Com base nas lacunas identificadas, foram elaboradas as questões de pesquisa, a hipótese e os objetivos;
5. Para iniciar o estudo de caso, além do software para hospitais, optou-se por analisar outros softwares consolidados no mercado de clínicas médicas;
6. Seguindo a abordagem de Queiroz (2009), realizou-se o levantamento das características de quatro softwares de clínica e o software hospitalar;
7. Em seguida foram realizadas a análise das características e confecção das estórias de usuários. Conforme ilustra-se na Figura 2, a execução das etapas passam a ser cíclicas;
8. A partir da análise das características, definido na etapa anterior, construiu-se o modelo de características contemplando as variabilidades dos cinco softwares analisados. Como resultado dessa etapa, na Figura 20 ilustra-se uma parte do modelo construído;

9. Tomando como base a abordagem de Queiroz (2009) e o conhecimento adquirido na RSL, definiu-se diretrizes para mapear características para microsserviços;
10. De acordo com as diretrizes elaboradas na etapa anterior, foram identificados os microsserviços e as aplicações *backend*;
11. O conjunto de microsserviços identificados na etapa anterior foi utilizado como entrada para a definição de um processo para a modelagem da arquitetura;
12. Após a definição da arquitetura, foi implementado o *backend* da LPS para clínicas médicas;
13. Com a aplicação das diretrizes em um caso real, realizou-se um experimento controlado para validá-las;
14. Por fim, as diretrizes foram refinadas com base nas contribuições do experimento.

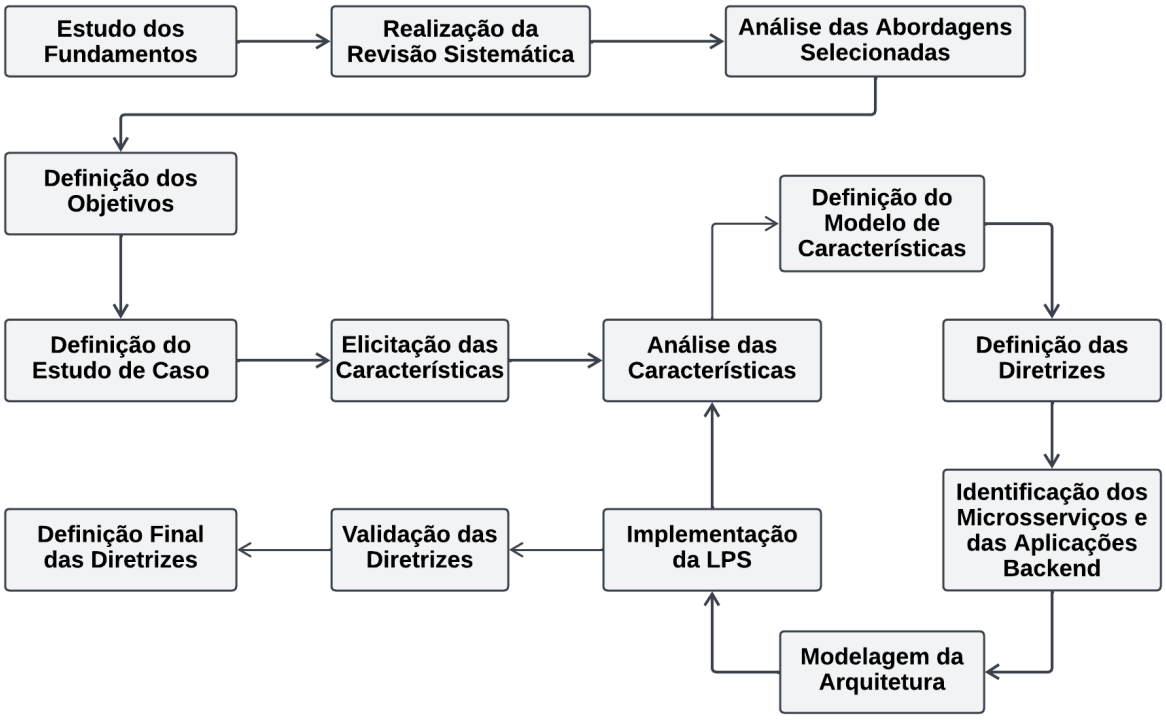

Figura 2 – Síntese das etapas executadas na condução desta pesquisa.

1.4 Organização

Este trabalho foi estruturado da seguinte forma. No Capítulo 2 são apresentados os conceitos fundamentais relacionados a esta pesquisa. No Capítulo 3, apresenta-se o planejamento e execução de uma revisão sistemática sobre o tema desta pesquisa. No Capítulo 4, apresenta-se a proposta de arquitetura híbrida para LPS. No Capítulo 5,

apresenta-se o estudo de caso que consistiu no desenvolvimento de uma LPS de clínicas médicas no formato de SaaS Multilocatário. No Capítulo 6 são abordados os resultados de um experimento controlado que buscou validar as diretrizes propostas por esse trabalho. Por fim, no Capítulo 7, são apresentadas as considerações finais desta dissertação.

2 Fundamentação Teórica

Neste capítulo são apresentados os conceitos fundamentais para esta pesquisa, tais como: LPS, Arquitetura de Microsserviços, Modelo C4 e requisitos que devem ser considerados para se projetar uma arquitetura de software.

O capítulo está organizado da seguinte forma: na Seção 2.1 é exposta a a técnica de LPS, destacando-se os seguintes conceitos correlacionados: engenharia de domínio, modelo de características e o método FODA. Na Seção 2.2 apresenta-se os seguintes conceitos relacionados a arquitetura de software: arquitetura de referência, alguns fatores importantes para se avaliar a qualidade de uma arquitetura de software, arquitetura de microsserviços e o Modelo C4 como ferramenta para modelar arquiteturas de software. Por fim, na Seção 2.3, apresenta-se as considerações finais sobre os tópicos apresentados neste capítulo.

2.1 Linha de Produto de Software

As soluções promovidas por softwares normalmente possuem muitos recursos semelhantes, inclusive em domínios diferentes. É fácil perceber que uma escola, uma fábrica de eletrodomésticos ou mesmo uma clínica, possuem setores em comum, tais como: recursos humanos, financeiro e estoque ou almoxarifado. Quando se trata de softwares do mesmo domínio de negócio, a maioria das funcionalidades são semelhantes e poucas são as diferenças, que normalmente se originam de regras de negócio. Tendo em vista as vantagens advindas da adoção de reúso, desenvolver uma família de softwares que possa atender a um determinado segmento do mercado e que também possa atender aos requisitos semelhantes e variáveis pode ser uma estratégia válida.

Essa é a estratégia da Linha de Produto de Software, uma técnica da engenharia de software que visa fornecer um catálogo de produtos que possuem um núcleo comum de funcionalidades, mas que cada produto é único por possuir alguma funcionalidade diferente dos demais. A técnica de Linha de Produtos busca justamente promover um reúso de forma planejada e eficiente, além de oferecer métodos para gerenciar as variabilidades (POHL; BÖCKLE; LINDEN, 2005). Segundo Becker e Lucrédio (2020), citando Cohen (2002), uma das principais definições de LPS é a seguinte:

> "Uma linha de produto de software é um conjunto de sistemas que usam software intensivamente, compartilhando um conjunto de características comuns e gerenciadas, que satisfazem as necessidades de um segmento particular de mercado ou missão, e que são desenvolvidos a partir de um conjunto comum de ativos principais e de uma forma preestabelecida." (CLEMENTS; NORTHROP, 2002, p. 563).

É comum empresas de software que atendem a um determinado domínio de negócio iniciarem com um cliente e, à medida que novos clientes são agregados, novos produtos são gerados a partir de uma cópia do original, com algumas modificações necessárias para atender aos novos requisitos. O problema dessa abordagem é que o reúso não é planejado e essas adaptações tendem a gerar graves instabilidades no software.

Para resolver esse problema, a técnica de LPS propõe que seja feita uma análise dos requisitos do domínio, e não apenas das necessidades de um único cliente. Essa análise é guiada por uma área da engenharia de software chamada engenharia de domínio (POHL; BOTTERWECK, 2012). Assim, a engenharia de domínio é responsável por modelar os recursos comuns e variáveis da LPS, permitindo a construção de produtos de software de alta qualidade de maneira mais rápida e econômica.

No contexto de uma LPS, os recursos comuns são aqueles que compõem o núcleo e fazem parte de todos os produtos da LPS. Enquanto que os recursos variáveis são utilizados para customizar os diversos produtos da LPS. Assim, as variabilidades da LPS referem-se à capacidade de personalizar ou configurar diferentes recursos, opções ou comportamentos de um software. Isso permite a criação de produtos diversos selecionando e combinando vários elementos configuráveis com base em requisitos específicos ou preferências dos clientes (CLEMENTS; NORTHROP, 2002).

Uma das abordagens para se realizar a engenharia de domínio e produzir a modelagem de variabilidades da LPS baseia-se na identificação e classificação das características dos softwares de um determinado domínio de negócio. As características são elementos distintos que podem ser incluídas no produto, dependendo das necessidades do cliente. Por exemplo, em um sistema de gestão de clínicas, as características podem incluir a possibilidade de fazer agendamento de consultas de forma presencial ou online, a capacidade de gerenciar prontuários, o suporte a diferentes tipos de dispositivos, entre outros.

A modelagem de características, do inglês *feature model*, vem do método FODA. No método FODA, uma *feature* equivale a uma característica do sistema que é visível ao usuário final (KANG et al., 1990). Em síntese, o modelo de características representa, em alto nível e de forma hierárquica, as características comuns e variáveis de uma LPS.

O método FODA possui uma notação para representar graficamente as características de uma LPS, e retrata o resultado do processo de análise de domínio. Além disso, o modelo de características contempla informações semânticas, pontos de variabilidade e regras de dependência entre as características. A notação FODA prevê três tipos de características: (I) obrigatória - representam as características que devem estar presentes em todos os membros da LPS; (II) opcional - características desse tipo podem ou não ser incluídas nos membros da LPS; (III) por fim, alternativa - são as características que representam mais de uma forma de realizar uma determinada operação e, portanto, permitem a possibilidade de se escolher múltiplas destas características para serem inclusas

em um determinado membro da LPS.

Na Figura 3 ilustra-se um modelo de característica. Neste modelo, cada característica é representada por um retângulo. As características são organizadas de forma hierárquica, sendo o primeiro nível sempre representado pelo domínio da LPS, exemplificado na Figura 3 como "Clínica". Os demais níveis são formados pelas características descendentes da característica de primeiro nível. Assim, o último nível de cada ramificação é formado pelas características folhas. As arestas com círculo preenchido em preto refletem as características obrigatórias. Já as arestas com círculo vazado simbolizam as características opcionais. As características alternativas são representadas por arcos, que podem ser vazados ou preenchidos. Os arcos vazados limitam a escolha de apenas uma característica dentre as alternativas. Já os arcos preenchidos representam a possibilidade de se escolher mais de uma alternativa.

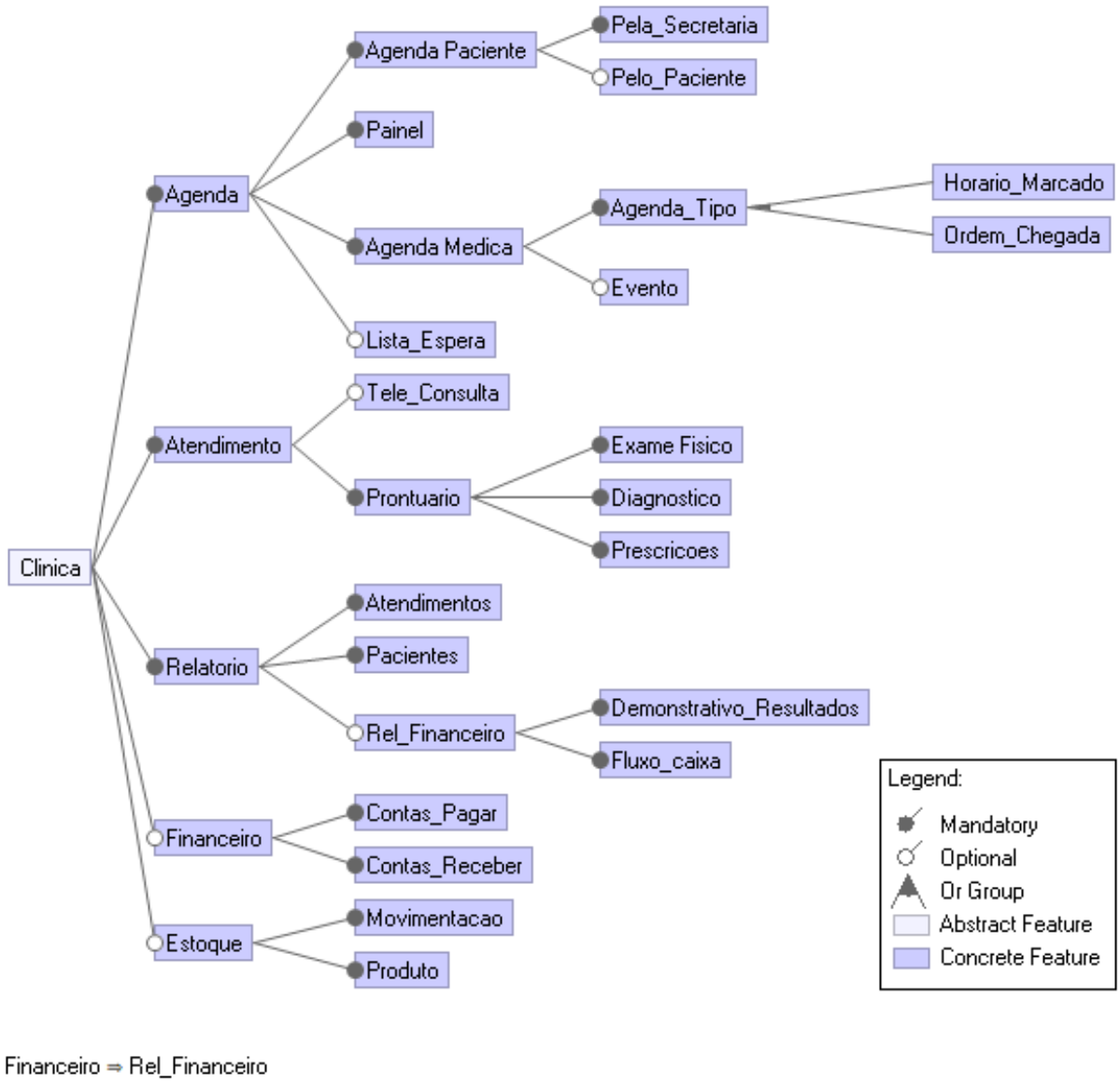

Figura 3 – Exemplo de modelo de características.

O resultado da engenharia de domínio é um conjunto de ativos-base que podem ser combinados para gerar um novo produto a partir de um subconjunto desses ativos. A ideia é que o software seja construído a partir do reúso de pequenas partes que podem ser combinadas de várias formas para que sejam gerados diversos produtos e assim poder atender a uma maior diversidade de clientes do mesmo domínio de negócio.

Um dos primeiros mecanismos para se construir um software dessa forma foi a utilização da modelagem de componentes. Nessa abordagem, o conjunto base de ativos é sistematizado com o objetivo de produzir componentes reutilizáveis. Ultimamente tem-se pesquisado o uso de microsserviços para substituir os componentes. Isso é possível devido os microsserviços serem conceituados como pequenas partes de um software, que funcionam de forma independentes e podem facilmente ser integrados por meio de diversos protocolos de comunicação.

A técnica de LPS oferece vantagens, tais como: (i) a melhoria na qualidade do software, visto que o reúso é planejado; (ii) a redução de custos e tempo de desenvolvimento, já que uma vez que o conjunto base de ativos esteja definido, poderá ser reutilizado para produzir rapidamente produtos personalizados; (iii) dessas vantagens anteriores, decorrem o aumento da produtividade e a facilidade de manutenção. Além disso, LPS permitem uma maior flexibilidade para atender às necessidades específicas de cada cliente (POHL; BÖCKLE; LINDEN, 2005).

2.2 Arquitetura de Software

Uma arquitetura de software representa a estrutura e organização de alto nível de um software. Ela abrange os principais componentes, a forma como esses componentes interagem, a distribuição de responsabilidades e os mecanismos de comunicação. Dessa forma, uma arquitetura de software fornece um esboço ou estrutura conceitual para a construção e evolução do software, orientando o processo de desenvolvimento e facilitando a comunicação entre as partes interessadas(SEVERO, 2021).

2.2.1 Arquitetura de Referência

O conceito de arquitetura de referência é importante para esta pesquisa, tendo em vista que um dos resultados principais deste trabalho é a proposta de uma arquitetura de referência híbrida para uma LPS que funcione como um SaaS Multilocatário. Arquitetura de referência (AR) é uma coleção de componentes arquiteturais, que podem servir de guia para gerar os produtos de uma LPS. Assim, a AR deve ser projetada para ser adaptável de modo a atender às necessidades específicas de cada produto (CLEMENTS; NORTHROP, 2002).

Nesse sentido, a AR é utilizada como modelo inicial de arquitetura para o desenvolvimento de produtos da família, fornecendo instrumentos para que as melhores práticas arquiteturais sejam aplicadas no desenvolvimento dos produtos. Dessa forma, a AR deve ser projetada para possibilitar uma reutilização eficaz e, com isso, seja possível o desenvolvimento de produtos que compartilham componentes arquiteturais comuns e padronizados. Existem algumas técnicas que podem ser aplicadas para se projetar a arquitetura de referência de LPSs, tais como:

- Análise de Requisitos: essa técnica envolve a identificação dos requisitos comuns a todos os produtos da linha e a análise das diferenças entre os produtos;
- Mapeamento de Características: essa técnica envolve a identificação das características dos produtos da linha e o mapeamento dessas características para componentes de software;
- Análise de Variabilidade: essa técnica envolve a identificação das variações entre os produtos da linha e a análise das implicações dessas variações para a arquitetura de referência.

2.2.2 Principais Requisitos Relacionados a Arquitetura de Software

Considerando que uma arquitetura de software é composta por componentes, para avaliá-la é necessário constatar se seus componentes possuem capacidade de funcionar e evoluir de maneira independente. Isso determinará o nível de acoplamento dos seus componentes. Arquiteturas que possuem componentes fracamente coesos, com forte acoplamento, difíceis de manter e evoluir, não conseguem atender as exigentes demandas do mercado.

Uma arquitetura de software deve atender além dos objetivos e restrições de negócio, os atributos de qualidade. Segundo Severo (2021), a avaliação de uma arquitetura deve levar em consideração os requisitos arquiteturalmente significativos, incluindo os seguintes atributos de qualidade:

- **Escalabilidade:** indica o potencial de um sistema em suportar a expansão das requisições e dos dados;
- **Desempenho:** fornece um indicativo de tempo para se executar um determinado procedimento;
- **Disponibilidade:** indica um quantitativo de requisições que um sistema consegue manter em um determinado nível de desempenho;
- **Segurança:** indica o quanto um software pode se manter e garantir a confidencialidade e a integridade em casos de ataques;

- **Manutenibilidade:** indica o nível de dificuldade para se manter o software;
- **Flexibilidade:** pode indicar a capacidade de fornecer alternativas para execução de um determinado processo;
- **Extensibilidade:** indica o suporte para permitir a adição de novos recursos;
- **Evolutividade:** indica a habilidade de um software de sobreviver a mudanças em seu domínio, requisitos e tecnologias de implementação.

Nos casos de sistemas distribuídos, outro fator importante a ser observado é a integração entre os componentes de software. Nesse sentido, três aspectos podem ser considerados:

- Comunicação: pode envolver diversos protocolos que podem ser executados de forma síncrona ou assíncrona;
- Consistência: pode ser tratada de duas formas genéricas, atomicidade ou eventualidade. A atomicidade pode provocar prejuízos para alguns atributos de qualidade, tais como: escalabilidade e desempenho;
- Coordenação da execução dos processos: pode ser realizada de duas formas: coreografia e orquestração.

Se tratando de LPS, outro requisito importante é o reúso dos artefatos de software que precisa ser planejado desde etapas iniciais do projeto. Vale salientar que o reúso tende a aumentar o acoplamento e a produzir unidades de software dependentes. Nesse sentido, é necessário projetar o reúso sem esquecer de considerar o acoplamento.

2.2.3 Microsserviços

No cenário atual, alguns softwares tratam de uma grande quantidade de dados e atendem a enormes quantidades de pessoas. Essa situação não seria possível sem que estes softwares funcionassem de forma distribuída. Sistemas distribuídos proporcionaram o avanço da capacidade de grandes plataformas de distribuição de conteúdo, *streams*, vendas online e redes sociais, como é o caso da Netflix, Mercado Livre, Facebook etc.

Tecnicamente, esse fato se torna possível devido ao processamento de requisições e o tratamento de dados poderem ser distribuídos entre várias máquinas, por meio de técnicas de balanceamento de carga e de otimização de escalonamento. Assim, projetar e arquitetar softwares para resolver problemas que envolvem essa complexidade exige técnicas e padrões específicos da engenharia de software.

Neste contexto, a arquitetura de microsserviços é uma das principais ferramentas que podem contribuir para uma solução distribuída. Segundo Lewis e Fowler (2014), a arquitetura de microsserviços é uma técnica que particiona o software em um conjunto de pequenos serviços. Ainda segundo Lewis e Fowler (2014), os microsserviço possuem os seguintes aspectos: (i) devem ser projetados conforme as regras de negócio, levando-se em consideração fatores como coesão e acoplamento; (ii) podem funcionar de forma heterogênea no que se refere à infraestrutura, tecnologia e equipe de desenvolvimento; (iii) possuem seu próprio processo e devem ser projetados para funcionar de forma independente, e assim, proporcionarem uma escalabilidade mais inteligente; (iv) e, a comunicação entre eles ocorre por meio de protocolos leves, como por exemplo, por uma *Application Program Interface (API)*, e *Hypertext Transfer Protocol (HTTP)*.

A adoção de microsserviços em projetos de software não é trivial, pois a complexidade associada a essa arquitetura exige o conhecimento de fatores e técnicas importantes, tais como:

- **Gerenciamento da consistência dos dados:** em um cenário em que os dados pertencentes a um sistema estão distribuídos em diversos servidores e sendo manipulados por diversas aplicações, a consistência precisa ser administrada de forma que eventualmente os dados estejam desatualizados em alguns microsserviços. Esse fato decorre da priorização da disponibilidade, que é um requisito almejado nos sistemas distribuídos. Do contrário, para assegurar a consistência atômica seria necessário bloquear os serviços enquanto as transações não fossem finalizadas, sendo este o cenário de aplicações não distribuídas.

- **Repositórios de Leitura:** a implementação da estratégia de repositórios para fornecer a leitura de dados consolidados é essencial para sistemas que implementam a arquitetura de microsserviços, pois o uso de junções entre tabelas do mesmo banco de dados é uma operação indesejada do ponto de vista do desempenho e quando se tem um cenário em que é preciso realizar junções entre tabelas que estão em bancos de dados e máquinas diferentes, esse tipo de operação passa a ser impraticável. Dessa forma, adotar a estratégia de armazenar uma cópia consolidada dos dados, normalmente em bancos NoSql, tem sido utilizada na indústria como solução para esse problema;

- **Banco de dados não compartilhado:** cada microsserviço deve possuir seu próprio banco de dados para que ele realmente funcione de forma independente dos demais. Assim, é possível garantir o isolamento dos dados, o escalonamento inteligente e a facilidade de manutenção;

- **Identificação de falhas e erros:** é preciso incluir no projeto, recursos para observar e monitorar o funcionamento dos microsserviços com o objetivo de identificar e

corrigir diversos tipos de falhas ou erros. Em um contexto em que dezenas ou centenas de microsserviços se comunicam entre si, pode ser impraticável identificar os erros. Dessa forma, existem diversas soluções de código aberto que podem facilitar o monitoramento de todo o sistema, tais como: Elastic Stack[1], Prometheus[2] e Grafana[3].

- **Comunicação assíncrona por meio de técnicas de mensageria:** esse recurso é bastante utilizado para enfileirar cargas de processamento de atividades marginais, como envio de emails ou outros serviços que envolvem disparo em massa. Também é utilizado para sincronizar dados entre microsserviços que utilizam entidades iguais e para alimentar os bancos de leitura, citado anteriormente no segundo item dessa lista. Dessa forma, essa estratégia pode ser utilizada para implementar a consistência eventual, pois faz uso de técnicas não bloqueantes. A ideia é que aplicativos enviem mensagens para "filas"e os aplicativos que desejam receber essas mensagens podem se inscrever nessas filas e receber mensagens quando estiverem disponíveis. Isso permite que os aplicativos se comuniquem de forma assíncrona e reduz a necessidade de acoplamento direto entre os aplicativos. Duas das principais ferramentas de código aberto que implementam serviços de mensagens são o Apache Kafka[4] e o RabbitMQ[5].

A arquitetura de microsserviços possui algumas vantagens e desvantagens com relação a arquitetura tradicional. Este trabalho nomeia de arquitetura tradicional, aquela em que o software não foi projetado para ser distribuído. Na Tabela 1 é descrita, de forma resumida, uma comparação entre estas arquiteturas.

2.2.4 Modelo C4

A utilização de modelos de arquitetura de software tem diminuído e por vezes utilizados sem padronização pelos adeptos de metodologias ágeis (SEVERO, 2021). O modelo C4 surge com uma proposta de notação para modelar arquiteturas de software, com o objetivo de ser uma ferramenta para atender as demandas das metodologias ágeis e, ao mesmo tempo, proporcionar a criação de diagramas claros e de fácil leitura. Para isso, a notação do modelo C4 recomenda não economizar na utilização de textos e legendas e assim, proporcionar modelos de arquitetura mais legíveis. Como seu próprio nome sugere, o modelo C4 é composto por quatro níveis hierárquicos de abstração: contexto, *containers*, componentes e código (BROWN, 2018).

[1] https://www.elastic.co/pt/elastic-stack/
[2] https://prometheus.io/
[3] https://grafana.com/
[4] https://kafka.apache.org/
[5] https://www.rabbitmq.com/

Aspecto	**Arquiteturas de Microsserviços**	**Arquiteturas Tradicionais**
Escalabilidade	Pensada para permitir um escalonamento inteligente, à medida em que é possível selecionar apenas as partes de maior carga para gerar novas instâncias.	É possível, mas não faz diferença entre as partes do sistema que possuem maior demanda. Assim, para escalonar, aumenta o número de recursos de infraestrutura, visto que é preciso replicar o sistema inteiro.
Manutenção	Como os microsserviços são independentes, a manutenção passa a ser mais fácil, pois pode ser realizada sem afetar as outras partes do sistema.	Mais complexa, visto que é necessário considerar o sistema como um todo e que uma mudança em uma parte pode ocasionar efeitos colaterais em outra parte do sistema.
Segurança	Expõe uma maior área de rede, porém uma falha em um determinado microsserviço não necessariamente atinge os demais.	Simplificada, todavia uma falha de segurança pode provocar indisponibilidade total do sistema.
Extensibilidade	Fácil, pois esse modelo de conectar novos serviços com funcionalidades específicas, que produz componentes de softwares coesos e desacoplados é uma das principais vantagens dos microsserviços.	Pode ser difícil, pois é preciso considerar o relacionamento dessa nova extensão com o restante do sistema. Isso envolve a avaliação e o entendimento do funcionamento do sistema como um todo.
Testes	Os testes globais são bem mais complexos e envolvem conhecimentos especializados.	Mais simples, pois não envolve fluxos externos ao processo e nem avaliação de tráfego de rede.
Desempenho	Possui vantagens quando a carga de processamento é alta, pois ela pode ser distribuída entre várias instâncias.	Tendem a ser melhores para cargas baixas, pois não envolvem acréscimos de comunicação entres as partes por meio da rede.

Tabela 1 – Comparação entre arquitetura de microsserviços e tradicional.

Neste trabalho, a arquitetura é apresentada no nível de *containers*, dessa forma, vale discorrer sobre esse diagrama. *Container* no modelo C4 pode representar um elemento de software em que códigos são executados ou dados são armazenados, como por exemplo um aplicativo móvel, um microsserviço, uma API *backend*, um aplicativo web *frontend*[6]

[6] *Frontend* refere-se à parte responsável por apresentar e interagir com os usuários, exibindo informações

ou um banco de dados. Segundo Brown (2018), idealizador do modelo C4, o diagrama de contêineres tem o objetivo de expor em alto nível a arquitetura do software, estabelecendo a responsabilidade de cada *container*. O diagrama de contêineres também exibe as tecnologias e as formas como os contêineres se comunicam. Esse diagrama possui o foco nas tecnologias de alto nível e pode ser utilizado pelos desenvolvedores e arquitetos.

Na Figura 4 exemplifica-se o diagrama de *container* do modelo C4. Nesta Figura estão representados os seguintes *containers: Web Application, Single-Page Application, Mobile App, Database, API Application, E-mail System* e *Mainframe Bankinfg System*. Observa-se ainda neste diagrama um nível alto de detalhamento dos elementos textuais, sendo este um dos alvos do modelo C4.

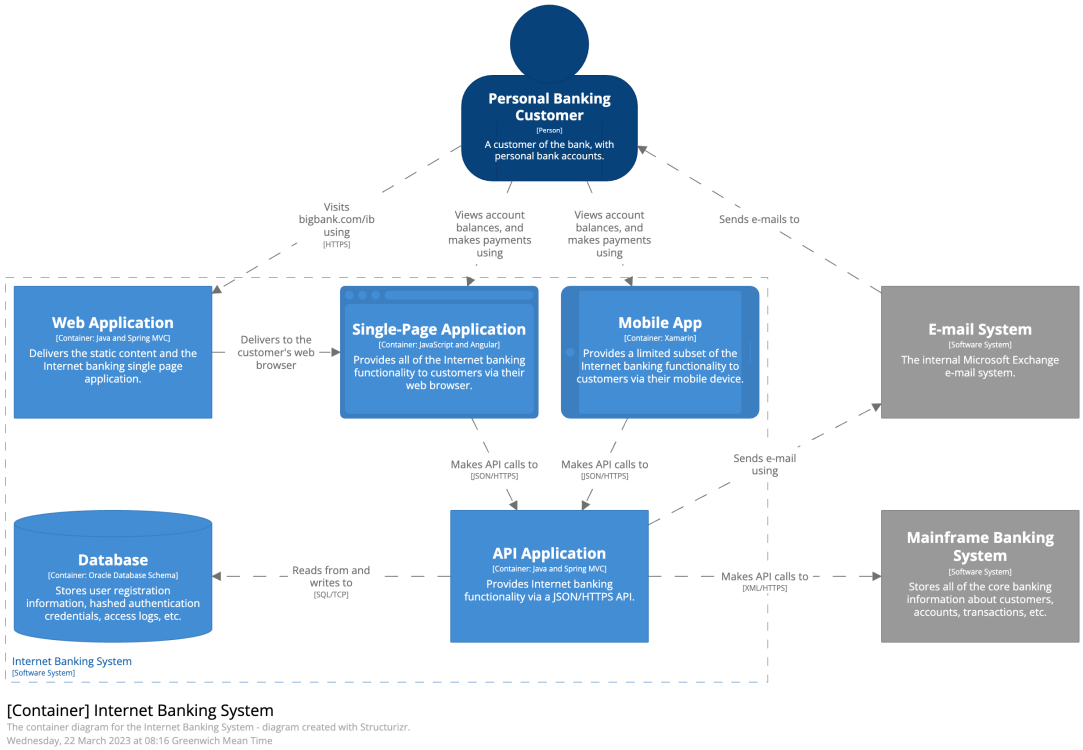

Figura 4 – Exemplo do diagrama de *container* retirado do site oficial do modelo C4.

2.3 Considerações Finais do Capítulo

Nesta seção foram contextualizados os fatores que possuem relação direta com a qualidade de uma arquitetura de software, para que se possa realizar uma avaliação criteriosa da arquitetura de referência de uma LPS que utiliza microsserviços para implementar as variabilidades.

A integração de LPS com microsserviços é uma estratégia promissora, pois, ao combinar essas duas técnicas, a primeira pode ser aplicada para analisar, definir e modelar

e interfaces gráficas de forma visualmente acessível, facilitando a interação com o sistema.

as características comuns e variáveis de um domínio de negócio; e, a segunda pode ser adotada para implementar a variabilidade definida pela primeira. Assim, ao integrar LPS com microsserviços, o objetivo é criar um conjunto de serviços que possam ser facilmente configurados para atender às necessidades comuns e específicas de cada produto da LPS. No entanto, de acordo com a RSL, por se tratar de uma estratégia ainda pouco explorada, essa integração pode apresentar desafios expressivos, como por exemplo, a falta de diretrizes e padrões que possam auxiliar nas decisões relacionadas à análise e projeto da arquitetura da LPS.

Ainda neste capítulo, foram apresentados os principais conceitos relacionados a esta pesquisa e foram contextualizados os fatores que possuem relação direta com a qualidade de uma arquitetura de software, para que se possa realizar uma avaliação criteriosa da arquitetura de referência de uma LPS que utiliza microsserviços para implementar as variabilidades.

3 Revisão Sistemática

Neste capítulo é apresentada a revisão sistemática realizada com o objetivo de identificar trabalhos que abordam LPS e microsserviços. Assim, esta revisão buscou conhecer e analisar as melhores técnicas e abordagens de modelagem, arquitetura e padrões aplicados na integração de LPS e microsserviços.

Para isso, definiu-se o protocolo dessa revisão sistemática de acordo com Biolchini et al. (2005), tendo em vista que aplicar métodos já estabelecidos na literatura são necessários para diminuir os erros durante a condução dos trabalhos e para que se possa realizar uma pesquisa auditável e replicável. Optou-se por Biolchini et al. (2005) como referência principal para esta RSL porque seu objetivo é fornecer um protocolo para RSL em engenharia de software e esta revisão foi pensada sob a perspectiva desta área. Além disso, esses autores se fundamentam em processos reconhecidos na literatura e aplicados em outras áreas de conhecimento, como a medicina.

Dessa forma, protocolo desta RSL estabelece três etapas: planejamento, execução e análise dos resultados. A primeira etapa, o planejamento, compreende a definição dos objetivos e a formulação do protocolo de pesquisa. A segunda etapa, a execução, é realizada seguindo as atividades de identificação, seleção e avaliação dos estudos encontrados. Na terceira e última etapa, são analisados os resultados a patir da síntese dos dados extraídos. Na Figura 5 ilustra-se o processo adotado nesta pesquisa.

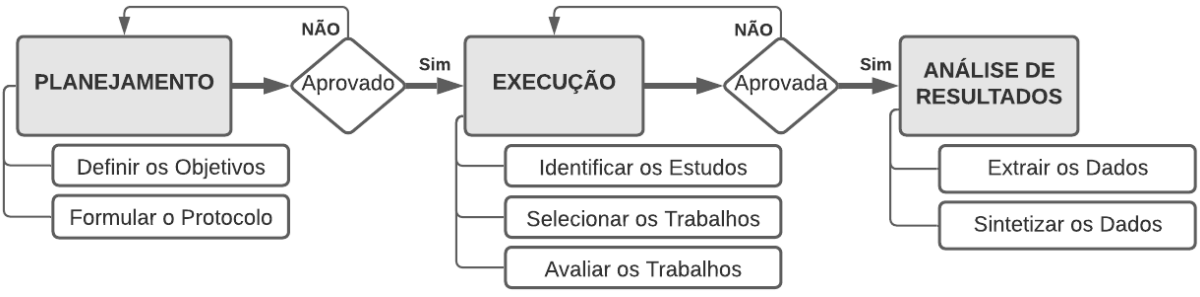

Figura 5 – Etapas da revisão sistemática (adaptada de Biolchini et al. (2005)).

Este capítulo está organizado da seguinte forma: na Seção 3.1, apresenta-se o planejamento da RSL a partir da definição dos objetivos e formulação do protocolo. Na Seção 3.2, apresenta-se a condução da revisão. Na Seção 3.3, os resultados são apresentados e discutidos. Na Seção 3.4 são descritas as aplicações da RSL. Na Seção 3.5 se discorre sobre as ameaças à validade do estudo e as medidas para atenuá-las. Por fim, na Seção 3.6 são apresentadas as considerações finais deste capítulo.

3.1 Planejamento

Nesta seção é detalhada a etapa de planejamento, realizada antes da condução da pesquisa, que é composta das seguintes atividades intermediárias: definição dos objetivos da pesquisa; formulação da questão de pesquisa; estratégia de busca para seleção de estudos primários; critérios e procedimentos para seleção dos estudos; processo de seleção dos estudos primários; e por fim, estratégias de extração e sumarização dos resultados.

3.1.1 Objetivos da Pesquisa

O objetivo principal desta revissão sistemática é encontrar arquiteturas, padrões, abordagens e ferramentas utilizadas na construção de LPS orientada a microsserviços. Por consequência, também busca-se identificar técnicas de modelagem e de implementação de variabilidades por meio de microsserviços. Este objetivo pode ser subdividido nos seguintes objetivos específicos.

- **Objetivo 1:** identificar arquiteturas para microsserviços aplicadas na construção de LPS;
- **Objetivo 2:** identificar os padrões e técnicas utilizadas para modelagem e implementação de variabilidades com microsserviços;
- **Objetivo 3:** identificar como as abordagens, técnicas, linguagens e padrões propostos nos trabalhos encontrados foram validados.

3.1.2 Formulação da Questão de Pesquisa

Quais arquiteturas, técnicas, padrões e linguagens de modelagem tem sido utilizados para construção de linhas de produto com arquitetura de microsserviços?

Essa é a questão principal definida para guiar a condução dos trabalhos e atingir os objetivos da pesquisa, e que foi subdividida em questões secundárias, de acordo com os objetivos específicos, da seguinte forma:

- **Questão Secundária (SQ1)**: Quais arquiteturas para microsserviços são aplicadas na construção de LPS?
- **Questão Secundária (SQ2):** Quais os padrões e técnicas são utilizados para modelagem e implementação de variabilidades com microsserviços?
- **Questão Secundária (SQ3):** Quais as metodologias e experimentos utilizados para validar as abordagens, técnicas, linguagens e padrões propostos nos trabalhados?

A qualidade e a abrangência da pesquisa depende de uma especificação semântica e de uma definição clara do contexto que gerou as questões de pesquisa. Para tanto, Biolchini et al. (2005) definiram, entre outros, os seguintes itens para avaliar a qualidade das questões de pesquisa: (1) Intervenção: delimita o escopo a ser examinado na pesquisa; (2) Controle: registra o contexto inicial e conhecimentos prévios do pesquisador; (3) População: indica o grupo populacional dos trabalhos que devem ser examinados de acordo com a intervenção; (4) Resultado: estabelece as métricas aplicadas para verificar o efeito da pesquisa; e (5) Aplicação: definição dos setores beneficiados com os resultados da revisão sistemática. Portanto, a seguir apresenta-se a definição desses itens de acordo com as questões de pesquisa levantadas.

- Intervenção: arquiteturas, técnicas, ferramentas, abordagens e padrões aplicados na modelagem e implementação de variabilidades por meio de microsserviços.

- Controle:

  - ASSUNÇÃO, Wesley KG; KRÜGER, Jacob; MENDONÇA, Willian DF. Variability management meets microservices: six challenges of re-engineering microservice-based webshops. In: Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A. 2020. p. 205-210.

  - BENNI, Benjamin et al. Can microservice-based online-retailers be used as an SPL? a study of six reference architectures. In: Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A. 2020. p. 217-222.

  - TIZZEI, Leonardo P. et al. Using microservices and software product line engineering to support reuse of evolving multi-tenant saas. In: Proceedings of the 21st International Systems and Software Product Line Conference-Volume A. 2017. p. 205-214.

- População: trabalhos primários que utilizam técnicas, abordagens ou ferramentas para modelar e implementar LPS utilizando microserviços.

- Resultados: apresentação e comparação dos padrões arquiteturais e de projeto, das abordagens e técnicas e das ferramentas utilizadas no desenvolvimento de LPS implementadas por meio de API's web e microsserviços.

- Aplicação: engenheiros de software, pesquisadores de LPS, pesquisadores e desenvolvedores de sistemas distribuídos baseados em microsserviços e *startups* com foco em software.

3.1.3 Estratégia de Busca para Seleção de Estudos Primários

Nesta seção, as estratégias de busca adotadas nessa pesquisa são descritas a partir da definição dos seguintes itens:

- **Critério de seleção das fontes:** escolheu-se realizar consultas nas cinco principais bibliotecas digitais que possuem forte ênfase em computação (MARQUES et al., 2019; CHEN; BABAR; ZHANG, 2010) e armazenam publicações avaliadas por pares, de modo que as pesquisas puderam ser feitas a partir da submissão de palavras-chave ou *strings* de busca nos seus motores de busca. Observa-se que apesar do Google Scholar retornar uma quantidade maior de trabalhos, optou-se por não utilizá-lo como base, pois muitos dos trabalhos retornados não são avaliados por pares e/ou podem ser publicados em veículos de natureza duvidosa, de modo que poderia comprometer o resultado desta pesquisa.

- **Métodos de busca de fontes:** submeter uma *string* de busca ou palavras-chave às buscas avançadas dos motores de busca de cada base de dados escolhida.

- **Palavras-chave:** as palavras-chave foram definidas buscando-se a amplipute adequada para atingir o objetivo de relacionar LPS e microsserviços. Dessa forma foram estabelecidas as seguintes palavras-chave: *software product line* e *microservices*. Os seus respectivos sinônimos são descritos na Tabela 2.

Palavras-chave	**Sinônimos**
software product line	*product line, software product family, systems family, SPL, software factory, variability, product variants*
microservices	*service, web services*, API REST, REST API, RESTful, RESTful API, API RESTful

Tabela 2 – Palavras-chave e sinônimos

- ***String* de busca:** após definir as palavras-chave e seus sinônimos, a *string* de busca foi formulada, ligando os sinônimos por meio do conectivo OR e ligando os grupos de sinônimos por meio do conectivo AND. Logo a seguir é descrito a *string* de busca:

  (("product lines"OR "product line"OR "software product line"OR "software product family"OR "systems family"OR "spl"OR "software factory"OR "variability"OR "product variants"OR "product variant") AND ("API REST"OR "REST API"OR "API RESTful"OR "RESTful API"OR "service"OR "web services"OR "microservices"))

- **Listagem de fontes:** de acordo com os critérios de seleção de fontes e métodos de busca definidos nos itens anteriores, as seguintes bases de dados foram selecionadas:

  - IEEE Xplorer Digital Library: http://ieeexplore.org
  - ACM Digital Library: http://dl.acm.org
  - Science Direct: http://www.sciencedirect.com
  - Scopus: http://www.scopus.com
  - Directory of Open Access Journals (DOAJ) https://doaj.org

- **Tipo dos estudos primários:** todos os tipos de estudos relacionados ao tema de pesquisa serão selecionados.
- **Idioma dos estudos primários:** inglês.

3.1.4 Critérios e Procedimentos para Seleção dos Estudos

Tendo em vista a importância de selecionar apenas os trabalhos que realmente possuem relevância, que respondem pelo menos a uma das questões de pesquisa, e consequentemente atendem, ainda que parcialmente, aos objetivos deste trabalho, é que são definidos os seguintes critérios de inclusão e exclusão de trabalhos.

3.1.4.1 Critérios de Inclusão

Esta pesquisa incluiu trabalhos que se enquadram em algum dos critérios de inclusão definidos a seguir.

- Critério de Inclusão (IC1): o artigo define ou apresenta padrões ou estilos arquiteturais aplicados à construção de família de produtos cujas variabilidades são implementadas por meio de microsserviços;
- Critério de Inclusão (IC2): o trabalho aborda técnicas, padões ou ferramentas para modelagem e implementação de variabilidades por meio de microsserviços.

3.1.4.2 Critérios de Exclusão

Os trabalhos que não atendem a nenhum dos critérios de inclusão são excluídos da pesquisa com base nos critérios de exclusão apresentados a seguir.

- Critério de Exclusão (EC1): o artigo aborda LPS, mas não implementa as variabilidades por meio de microsserviços;
- Critério de Exclusão (EC2): o artigo aborda microsserviços, mas não trata de variabilidades e nem de linhas de produtos;
- Critério de Exclusão (EC3): trabalhos publicados antes de 2011;

- Critério de Exclusão (EC4): trabalhos escritos em idiomas diferentes do inglês e do português.

Em relação ao critério de exclusão EC3, definiu-se o ano de 2011 como ano inicial desta pesquisa, pois foi o ano de surgimento de publicações relacionadas a arquitetura de microsserviços.

3.1.4.3 Critérios de Qualidade

Com o objetivo de ranquear as contribuições dos trabalhos encontrados, foram definidos três critérios de qualidade que adicionam pontos de qualidade ao trabalho. A escolha das pontuações adicionadas foi arbitrária, segundo elementos considerados importantes, pelos pesquisadores, durante a leitura e análise dos trabalhos selecionados. Os seguintes critérios de qualidade foram definidos:

- Critério de Qualidade (QC1): considerar as pontuações apresentadas na Tabela 3 para artigos publicados em periódicos e as pontuações apresentadas na Tabela 4 para artigos publicados em conferências.

- Critério de qualidade (QC2): quando a pesquisa apresentada no artigo foi utilizada para resolver problemas reais, optou-se por adicionar 2 pontos de qualidade ao artigo;

- Critério de qualidade (QC3): os trabalhos que fornecem artefatos construídos durante a pesquisa recebem 1 ponto adicional de qualidade.

Tabela 3 – Pontuações de artigos publicados em periódicos

Fator de Impacto	**Pontuação**
entre 1,0 e 2,0	1
entre 2,1 e 3,0	2
entre 3,1 e 5,0	3
maior que 5,0	5

Tabela 4 – Pontuações de artigos publicados em conferências

Qualis	**Pontuação**
A1	5
A2	3
A3	2
A4	1

3.1.5 Processo de Seleção dos Estudos Primários

Nesta seção são descritos os quatro passos que devem ser seguidos para selecionar os trabalhos primários, conforme detalha-se a seguir:

- Processo de seleção preliminar: buscar e coletar os artigos nas bases de dados, excluindo-se os repetidos;
- Processo de seleção inicial: leitura dos títulos e resumos dos artigos retornados na seleção preliminar. Incluir ou excluir trabalhos de acordo com os critérios definidos na Seção 2.4;
- Processo de seleção final: leitura completa dos trabalhos selecionados a partir da seleção inicial. Após a leitura, novamente são aplicados os critérios de inclusão e exclusão estabelecidos na Seção 2.4 para incluir ou excluir trabalhos neste estudo;
- Avaliação de qualidade dos estudos primários: cada artigo selecionado, com os seus respectivos metadados, é registrado em uma tabela ordenada por maior pontuação, de acordo com os critérios de qualidade definidos na Seção 3.1.4.

É importante destacar que esse processo foi realizado por dois pesquisadores e que em caso de dúvida, os trabalhos foram incluídos ou excluídos apenas em caso de consenso entre os pesquisadores.

3.1.6 Estratégia de Extração e Sumarização dos Resultados

Para extrair as informações de cada estudo selecionado, foi definido um formulário para registrar os seguintes dados:

- *Título;*
- *Autores;*
- *Resumo;*
- *Objetivos do artigo;*
- *Fases das abordagens propostas;*
- *Ano de publicação;*
- *Bases de dados: IEEE, ACM, SCOPUS, Science Direct;*
- *Tipo de publicação;*

- *Questões de pesquisa: arquitetura proposta; abordagem de modelagem e implementação; e métodos de validação.*
- *Limitações;*
- *Desafios propostos.*

Quanto à sumarização dos resultados, a seguir são apresentados, por meio de tabelas e gráficos, a título de comparação e de idenficação de respostas às perguntas da pesquisa, uma síntese dos dados extraídos dos trabalhos encontrados. Por último, utilizando o aplicativo VOSViewer[1] (ECK; WALTMAN, 2010), pode-se constatar que as palavras-chave para esta revisão foram definidas de acordo com a rede de palavras-chave dos estudos primários selecionados.

3.2 Condução da Revisão Sistemática

A revisão sistemática foi conduzida de acordo com o planejamento apresentado nas seções anteriores. Ao todo, foram recuperados 1243 trabalhos, que foram submetidos às etapas de seleção preliminar, seleção final e extração de resultados. Nas próximas seções são apresentados mais detalhes das atividades realizadas, incluindo os resultados das buscas para cada uma das fontes selecionadas.

3.2.1 Seleção Preliminar

A seleção preliminar foi conduzida em cinco etapas, sendo elas: (1) Validação da *string* de busca; (2) Adequação da *string* de busca; (3) Buscar e coletar; (4) Eliminação dos repetidos; e (5) Seleção inicial. Essas etapas são detalhadas nas próximas seções.

- **Validação da *String* de Busca**

  Para validar a *string* de busca planejada inicialmente, foi realizada uma busca na base IEEE no dia 19/10/2021. Após a análise dos 20 primeiros trabalhos retornados e da quantidade de trabalhos retornados, constatou-se que deveriam ser adicionados novos sinônimos na *string* de busca, tais como os relacionados a API REST, visto que a *string* inicial não continha esses sinônimos. Após a realização de uma nova busca, observou-se que foram retornados novos trabalhos relevantes.

- **Adequação da *String* de Busca**

  Como relatado anteriormente, a *string* foi adaptada para cada base e ficou formulada da seguinte forma:

[1] https://www.vosviewer.com/

  - **IEEE, ACM, SCOPUS e DOAJ:**

    ("product lines"OR "product line"OR "software product line"OR "software product family"OR "systems family"OR "spl"OR "software factory"OR "variability"OR "product variants"OR "product variant") AND ("API REST"OR "REST API"OR "API RESTful"OR "RESTful API"OR "service"OR "web services"OR "microservices")

  - **Science Direct:** devido a limitação da interface de busca relacionada ao quantitativo de termos e conectivos, foi necessário dividir a *string* em duas, o que levou a necessidade de coletar e juntar os resultados de duas buscas da mesma base de dados. Segue as *strings* executadas:

    * *String 1:* ("product line"OR "software product family"OR "systems family"OR "spl"OR "variability") AND ("REST API"OR "RESTful API"OR "service"OR "microservices")
    * *String 2:* ("product lines"OR "software product line"OR "software factory"OR "product variants"OR "product variant") AND ("REST API"OR "RESTful API"OR "service"OR "microservices")

  É importante destacar que, com exceção da base DOAJ, que teve a busca baseada em todos os campos, nas demais bases a busca foi filtrada pelos campos título, resumo e palavras-chave.

- **Buscar e coletar**: ao realizar as buscas em cada base de dados, os resultados foram coletados e organizados em uma planilha. Na Figura 6 ilustra-se os resultados de cada base.

- **Eliminação dos repetidos:** para eliminar os trabalhos repetidos, foi utilizado o software Mendeley que encontrou um total de 12 artigos duplicados.

- **Seleção inicial:** a partir da leitura do título e do resumo de cada artigo, seguindo os critérios de inclusão e exclusão, definidos na Seção 3.1.4, foram selecionados 18 artigos. Na Figura 6 são apresentados os resultados dessa primeira etapa, contendo os quantitativos de artigos incluídos e excluídos em cada etapa.

3.2.2 Seleção Final

Após a leitura completa dos 18 artigos selecionados inicialmente, os critérios de inclusão e exclusão foram reaplicados, restando apenas seis trabalhos, conforme é ilustrado na Figura 6. É importante destacar que os trabalhos referenciados nos artigos selecionados foram analisados, mas não foram encontradas contribuições relevantes para esta pesquisa, de modo que nenhum novo trabalho foi adicionado.

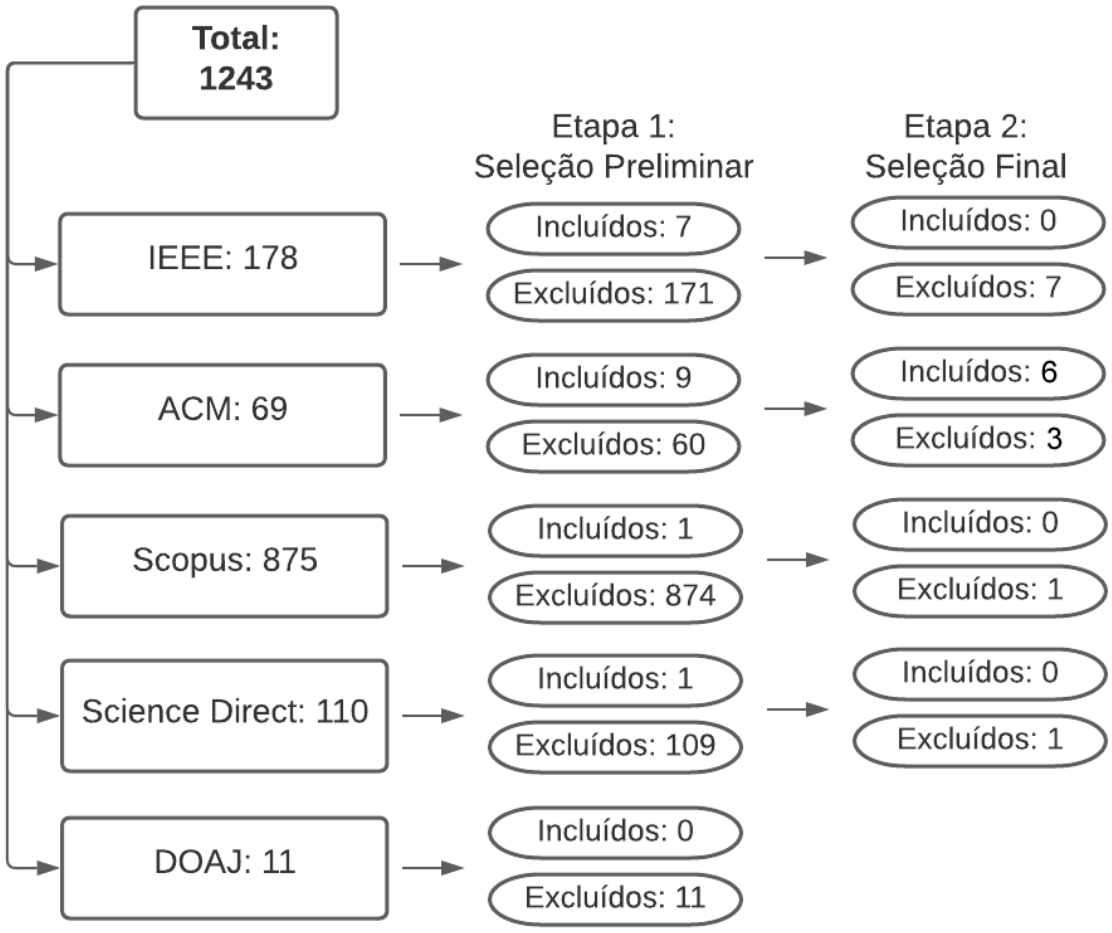

Figura 6 – Resultado do processo de avaliação dos artigos da RSL

Nesta etapa, apenas artigos indexados pela plataforma ACM foram selecionados. No entanto, os mesmos trabalhos podem ser indexados por diferentes plataformas, fazendo com que a mesma quantidade de artigos possa ser encontrada em mais de uma plataforma.

A aplicação dos critérios de qualidade foi a última atividade executada nesse processo de avaliação dos artigos selecionados. Na Tabela 5, composta pelos campos: identificador (ID) do artigo, autor, pontos atribuídos em cada critério de qualidade e total de pontos, é descrito a qualificação de cada estudo selecionado.

ID	Autor	CQ1	CQ2	CQ3	Total
1	Benni et al. (2020)	2	0	0	2
2	Costa et al. (2019)	0	2	1	3
3	Becker e Lucrédio (2020)	0	2	1	3
4	Setyautami et al. (2020)	2	0	1	3
5	Tizzei et al. (2017)	2	2	0	4
6	Assunção, Krüger e Mendonça (2020)	2	0	0	2

Tabela 5 – Ranking dos artigos selecionados com base nos critérios de qualidade

3.3 Resultados e Discussões

Ao final do processo de condução da revisão sistemática, foram selecionados seis trabalhos que apresentaram abordagens e métodos para o desenvolvimento de uma LPS que utiliza microsserviços para implementar as variabilidades. Esses trabalhos enfatizaram a modelagem e arquitetura de microsserviços no contexto de LPS. Considerando as questões de pesquisa, três artigos não propuseram uma arquitetura para microsserviços aplicada a LPS. Cinco apresentaram técnicas para modelagem e mapeamento de características para microsserviços. Por fim, os seis trabalhos validaram suas propostas por meio da aplicação em estudos de caso.

As palavras-chave de cada artigo foram inseridas no VOSViewer para gerar o gráfico de visualização da rede. Na Figura 7 é possível perceber que *software product line* e *microservice* possuem a maior quantidade de ocorrências. Também pode-se constatar que existe uma grande correlação entre esses dois termos, o que pode refletir justamente o foco desses trabalhos em integrar LPS e microsserviços. Além disso, é importante destacar palavras-chaves que estão relacionadas tanto aos desafios de fazer tal integração, como também palavras-chave que representam a necessidade de estudar e aprofundar o assunto, inclusive para a criação de abordagens e metamodelos que possam contribuir para auxiliar os engenheiros de software nas atividades responsáveis pelo desenvolvimento de uma LPS baseada em microsserviços, principalmente atividades relacionadas a definição da arquitetura da LPS. As obras selecionadas são discutidas a seguir.

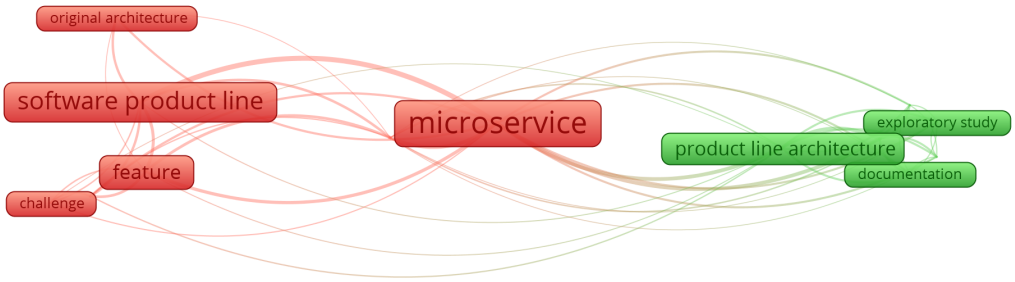

Figura 7 – Rede de palavras-chave dos artigos selecionados.

Benni et al. (2020), propuseram um estudo de seis arquiteturas para responder ao quarto desafio dos seis desafios propostos por Assunção, Krüger e Mendonça (2020). Esse quarto desafio refere-se a encontrar uma solução que permita intercambiar microsserviços de diferentes tecnologias no contexto de LPS. Apesar desse desafio fugir do escopo dessa pesquisa, de forma marginal, os autores propuseram uma solução para o primeiro desafio, que trata-se de uma proposta de mapeamento de recursos para microsserviços, sendo este o motivo desse trabalho ter sido selecionado para esta pesquisa. Para mapear recursos para microsserviços, os autores propuseram que cada microsserviço pode implementar vários recursos por meio de *endpoints* de API. Contudo, não fica claro de que forma as características devem ser agrupadas para fazer parte de um determinado microsserviço.

Como também não é apresentado as vantagens dessa abordagem e nem um comparativo com outras formas de mapeamento.

Costa et al. (2019) propuseram um metamodelo baseado no metamodelo proposto por Colanzi et al. (2014) para guiar arquitetos de software na construção da arquitetura da LPS. Com base nos microsserviços extraídos do modelo de características, os autores propuseram a construção da arquitetura da LPS. A extração dos microsserviços foi baseada nos recursos folha do diagrama de característica, ou seja, cada nó folha gera um microsserviço. A crítica a esse trabalho está relacionada ao fato de que, em alguns casos, o mesmo microsserviço poderia ser usado por mais de uma característica. Quando isso acontece o reúso é fragilizado, pois o mapeamento direto de cada nó folha do modelo de características para um microsserviço não leva em consideração os relacionamentos e as dependências entre as características modeladas. Com relação ao metamodelo proposto, ilustrado na Figura 8, o próprio trabalho aponta alguns pontos a melhorar, como por exemplo a falta de elementos que representem os repositórios e suas respectivas conexões com os microsserviços.

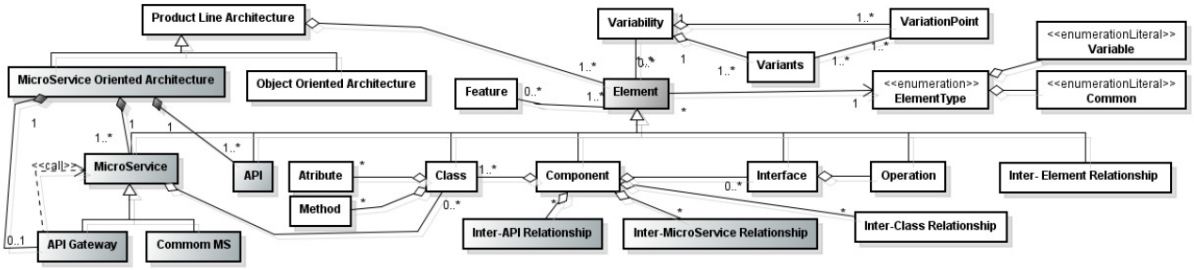

Figura 8 – Proposta de Metamodelo de Arquitetura de LPS orientadas a Microsserviços (COSTA et al., 2019).

No trabalho de Becker e Lucrédio (2020) foi apresentado um estudo empírico que buscou identificar as vantagens e desvantagens da arquitetura de microsserviços para a fase de evolução de uma LPS. Os resultados mostraram que nas tarefas adaptativas os microsserviços trazem benefícios à LPS. No entanto, nas tarefas evolutivas houve um aumento de esforço de codificação. Para realizar o experimento foi feito um comparativo entre dois cenários de uma mesma LPS. O primeiro cenário retratando uma aplicação monolítica e o segundo cenário baseado em uma arquitetura de microsserviços. A LPS monolítica foi modificada para gerar a LPS orientada a microsserviços. Para fazer esse mapeamento, os autores consideraram que cada operação sobre uma entidade de registro deveria gerar um microsserviço, por exemplo: cada operação CRUD da entidade "Pessoa", salvar, excluir, alterar e pesquisar registros de pessoas, gerou um microsserviço. Os autores, ao afirmarem que no contexto de LPS e microsserviços, o principal desafio está na granularidade das *features*, reforçam a necessidade de pesquisas semelhantes. É importante destacar que apesar de ter sido implementado os microsserviços no segundo cenário, os autores utilizaram um único repositório para atender a todos os microsserviços, o que

evidencia um alto acoplamento, visto que se o servidor de banco de dados ficar inoperante, todos os microsserviços passam a deixar de funcionar.

Em resposta a quatro desafios de seis propostos por Assunção, Krüger e Mendonça (2020), Setyautami et al. (2020), apresentaram soluções para identificação de recursos, modelagem de variabilidade, arquitetura e reengenharia de linha de produtos com base nas funcionalidades de seis aplicativos de venda eletrônica e que foram desenvolvidos com microsserviços. A identificação de recursos foi realizada a partir da simulação do uso das funcionalidades, da revisão das configurações de implantação e da análise do código fonte dos microsserviços. Como resultado dessas atividades, foi construído uma matriz, conforme ilustrado na Figura 9, que relaciona cada recurso identificado ao seus respectivos aplicativos.

Features	Subject Systems			
	HipsterS	SockS	eShop	SRobotS
Front-end	✓	✓	✓	✓
User		✓	✓	✓
Payment	✓	✓	✓	✓
Catalog	✓	✓	✓	✓
Cart	✓	✓	✓	✓
Order	✓	✓	✓	✓*
Product	✓*	✓*	✓*	✓*
Shipping	✓	✓		✓
Email Contact	✓			
Marketing	✓		✓	
Recommendation	✓			
Audit				
Currency	✓			
Location			✓	

Figura 9 – Matriz de rastreabilidade de recursos de cada aplicativo analisado (SETYAUTAMI et al., 2020).

Essa matriz serviu de entrada para construção do modelo de características. Com o diagrama de recursos modelado, os autores utilizaram a estratégia de mapear cada recurso filho do recurso raiz para um microsserviço e as características folha foram mapeadas para *endpoints* dos seus respectivos microsserviços. Esse mapeamento foi baseado em uma abordagem de modelagem de características multinível (RABISER et al., 2018). O diagrama de recursos também foi usado como entrada para projetar a arquitetura da linha de produtos, que foi modelada usando um perfil UML-DOP estendido.

Apesar do artigo de Setyautami et al. (2020) ser detalhado, apresentar as etapas e os artefatos gerados, alguns pontos importantes não foram contemplados e outros, reconhecidos pelos próprios autores, necessitam de aprimoramento, tais como: (i) "Atualmente, não temos um mecanismo de reutilização para compartilhar a implementação de recursos que possuem serviços e terminais semelhantes" (SETYAUTAMI et al., 2020). Isso é uma dificuldade realmente considerável, pois compromete o reúso, requisito caro tanto para LPS como para microsserviços; (ii) um segundo ponto, decorrente do primeiro, refere-se aos relatórios, requisito importante para aplicações reais, e que não foi tratado pelos autores. Como os relatórios podem envolver dados de vários recursos e esses dados podem estar em repositórios diferentes, implementá-los neste contexto demonstra ser uma tarefa complexa. Dessa forma, fica aberto um campo de pesquisa para estudar a modelagem de uma arquitetura que contemple a implementação dos relatórios; (iii) por fim, pelo seu propósito de responder a outro trabalho, não apresenta as soluções de forma generalista, como fizeram Costa et al. (2019), transparecendo assim, uma visão mais restrita ao seu estudo de caso e menos contributiva para a engenharia de software.

Tizzei et al. (2017) desenvolveu um experimento para investigar o uso integrado de LPS e microsserviços em uma aplicação Software as a Service (SaaS) multilocatário. Este estudo analisou principalmente o comportamento da reutilização de software e do uso de microsserviços. Os autores concluíram que houve uma reutilização média de software de 62% das linhas de código entre os locatários e que a utilização de microsserviços apresentou ganhos de escalabilidade e manutenção. Apesar dos resultados apresentados, observa-se que não houve uma implementação real da arquitetura de microsserviços, visto que foi modelado um único serviço com muitas responsabilidades, caracterizando-se assim um alta nível de granularidade, ou mesmo, descaracterizando a arquitetura de microsserviços. Vale a pena destacar que os próprios autores reconhecem a necessidade de particionar tal serviço, gerando assim, de fato, uma arquitetura de microsserviços.

No trabalho apresentado por Mendonça et al. (2020), os autores apresentaram uma abordagem para extrair modelos de características de um conjunto de sistemas baseados em microsserviços. O objetivo principal deste trabalho foi utilizar tais modelos para que os profissionais raciocinem sobre a reutilização e customização de funcionalidades. Para extrair os modelos de características, os autores utilizaram os algoritmos evolutivos multi-objetivos NSGA-II e SPEA2, com a pretensão de maximizar a (i) precisão (modelos que permitam derivar apenas sistemas desejados), (ii) recall (modelos que incluísse todos os sistemas desejados) e (iii) conformidade (modelos que atendam às dependências existentes entre microsserviços). Apesar dos modelos extraídos atenderem aos objetivos propostos pelos autores, analisando os modelos extraídos se percebe que as hierarquias construídas nas árvores de cada modelo se baseou apenas no aspecto das dependências entre os microsserviços, e portanto, desconsiderando uma organização lógica e natural das características advindas da análise de um domínio.

3.3.1 SQ1 - Quais arquiteturas para microsserviços são aplicadas na construção de LPS?

A arquitetura de uma LPS retrata de forma generalista as arquiteturas de todos os produtos que podem ser derivados da LPS, sendo assim um importante artefato da engenharia de domínio que estabelece as partes fundamentais da infraestrutura de reúso (POHL; BÖCKLE; LINDEN, 2005).

Inicialmente, observa-se que o trabalho de Benni et al. (2020) não propõe uma arquitetura, pois nesse artigo, um dos objetivos é identificar as arquiteturas e aplicações já existentes, ao invés de projetar uma nova arquitetura. Adicionalmente, o trabalho de Mendonça et al. (2020) também não propõe ou apresenta nenhuma arquitetura de LPS.

Costa et al. (2019) basiaram-se em uma arquitetura de um software utilizado no seu estudo de caso, conforme apresenta-se na Figura 10. Nela pode-se perceber as seguintes quatro macro divisões: (i) *Client Apps* - representa a camada cliente, responsável por fazer requisições, neste caso podendo ser aplicações web, mobile ou até mesmo outros microsserviços; (ii) *API Gateway* - gerencia fila de requisições recebidas e realiza o balanceamento de carga. O API *Gateway* mantém o registro de cada um dos microsserviços disponíveis para que possa chamá-los; (iii) *Microservices* - representa o conjunto de microsserviços que foram identificados. Destaca-se que eles podem ser heterogêneos na sua forma de se comunicar; e (iv) *Repositories* - representa a camada de persistência dos dados que são utilizados pelos microsserviços. Apesar dessa arquitetura ser baseada em práticas já utilizadas na indústria, algumas fragilidades podem ser identificadas, tais como o compartilhamento do banco de dados entre os microsserviços, o que leva a um acoplamento entre eles, e a centralidade da *API Gateway*, o que pode provocar a inoperância de todas as aplicações caso esta venha a falhar.

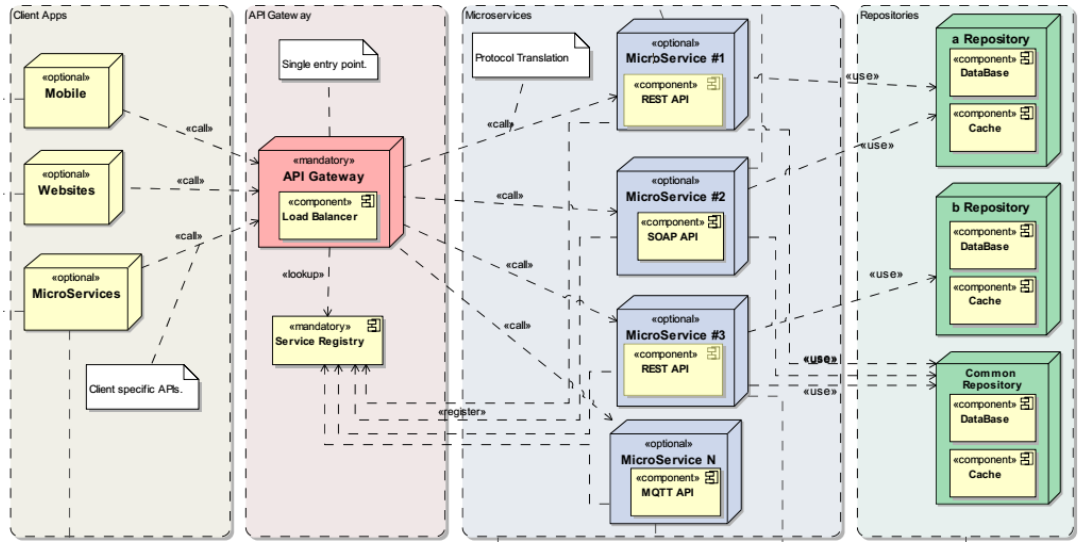

Figura 10 – Arquitetura utilizada no estudo de caso de Costa et al. (2019) (adaptada de Costa et al. (2019)).

Em seu estudo, Becker e Lucrédio (2020) evoluíram uma LPS monolítica, desenvolvida em Java, para uma arquitetura descentralizada, baseada em microsserviços. Na Figura 11 é apresentada a nova arquitetura, nela é possível perceber as seguintes camadas: (i) visualização, representada por uma única aplicação. Essa camada possui o papel de realizar as chamadas aos microsserviços utilizados por cada cliente. Portanto, é nesta camada de visualização que as configurações de variabilidade de cada produto da LPS são definidas; (ii) microsserviços, responsável por implementar as variabilidades da LPS; (iii) e por fim, a camada de banco de dados, que é responsável por todos os dados gerenciados pelos microsserviços. É importante mencionar que essa arquitetura foi projetada para evoluir o contexto inicial da LPS, e portanto, alguns pontos como o banco de dados e as tecnologias utilizadas foram mantidos pelos autores para uma melhor adapção da implantação da nova LPS. Assim, é possível perceber que ainda existe um nível de centralização e aclopamento, principalmente pelo fato de todos os microsserviços utilizarem o mesmo banco de dados.

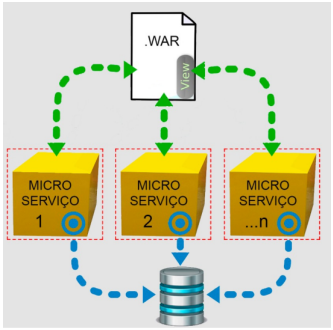

Figura 11 – Arquitetura proposta por Becker e Lucrédio (2020).

Setyautami et al. (2020) em seu artigo exemplificou, conforme a Figura 12, a arquitetura de uma aplicação gerada a partir da LPS. É possível perceber que essa proposta é semelhante a de Costa et al. (2019), todavia percebe-se um desaclopamento da comunicação entre os microsserviços e os repositórios, pois neste caso, cada microsserviço possui seu próprio repositório. A seleção dos recursos, e por conseguinte, dos *endpoints* dos microsserviços é definido com base em arquivos de configuração que são processados por uma ferramenta de implantação produzida pelos autores. Uma vez definido os recursos de cada aplicativo, os recursos não selecionados inicialmente ficam inacessíveis. Para fazer isso, foi utilizado o módulo delta que ajusta a configuração de roteamento com base nos recursos escolhidos.

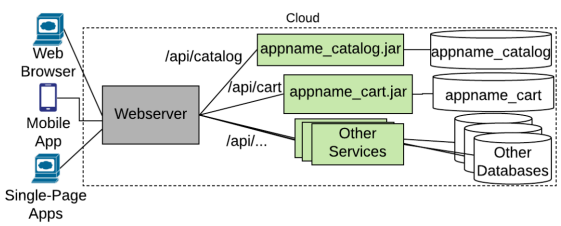

Figura 12 – Arquitetura proposta por Setyautami et al. (2020).

No trabalho de Tizzei et al. (2017) a arquitetura proposta é projetada para atender a escalabilidade e monitoramento, entretanto não contempla elementos de microsserviços e nem de variabilidade da LPS, visto que todos os elementos arquitetônicos são obrigatórios, e a variabilidade do software é implementada por meio das operações da API RESTful, de acordo com as características específicas de cada locatário.

Em resumo, dos seis artigos selecionados, Benni et al. (2020) e Mendonça et al. (2020) não propuseram arquitetura; Tizzei et al. (2017) propõe uma arquitetura que não representa uma arquitetura de microsserviços e nem contempla as variabilidades da LPS; dois artigos, Costa et al. (2019) e Setyautami et al. [19] propuseram arquiteturas semelhantes, levando-se em consideração os seguintes pontos: (i) utilização de API *Gateway* para controlar e balancear as requisições aos microsserviços; e (ii) aplicação de repositórios dedicados para cada microsserviço. De outra forma, na proposta de Becker e Lucrédio (2020) as aplicações cliente se comunicam diretamente com os microsserviços e o repositório é compartilhado entre todos os microsserviços. Na Tabela 6, apresenta-se uma síntiese sobre as arquiteturas propostas e, na Tabela 7, são descritas as vantagens e desvantagens de cada uma delas.

Tabela 6 – Resumo das arquiteturas propostas

Autor	**API Gateway**	**Repositórios**
Costa et al. (2019)	Utilizou.	Existência, ainda que de baixo nível, de compartilhamento entre os microsserviços.
Becker e Lucrédio (2020)	Não utilizou.	Único repositório para todos os microsserviços.
)setyautami2020	Utilizou	Cada microsserviço possui o seu próprio repositório.

3.3.2 SQ2 - Quais padrões e técnicas são utilizados para modelagem e implementação de variabilidades com microsserviços?

Comparando as três questões de pesquisa, a SQ2 foi a que se mostrou com maior heterogeneidade em suas respostas, conforme é descrito a seguir.

No trabalho de Benni et al. (2020), como já mencionado anteriormente, o foco não foi responder as questões desta pesquisa, todavia, como pré-requisito para atingir seu objetivo, os autores propuseram uma técnica para realizar o mapeamento de recursos para microsserviços. Assim, tal técnica define que cada microsserviço pode implementar vários recursos por meio de *endpoints* de API. Sendo sucinto em suas declarações acerca da escolha adotada para modelar os microsserviços, os autores apenas apresentaram uma tabela que exemplifica o uso da aplicação dessa técnica em um casos de uso específico.

Na Figura 13 é ilustrada a estrutura utilizada para modelagem dos microsserviços proposta por Costa et al. (2019). Nela, pode-se perceber que cada microsserviço possui

Tabela 7 – Vantagens e desvantagens das arquiteturas propostas

Autor	Vantagens	Desvantagens
Costa et al. (2019)	- Facilidade para gerenciar o controle de requisições e balanceamento de carga; - Baixo nível de complexidade para gerenciar a integração dos dados dos repositórios.	- Dependência entre os microsserviços devido ao compartilhamento de repositórios.
Becker e Lucrédio (2020)	- Em um contexto de migração de uma arquitetura monolítica, esta proposta pode representar um passo intermediário para facilitar o processo de mudança de arquitetura; - Facilidade para manter a consistência e integração dos dados; - Maior independência entre as aplicações cliente e os microsserviços.	- Dificuldade para gerenciar o roteamento e balanceamento de carga; - Algo grau de acoplamento entre os microsserviços e o repositório.
Setyautami et al. (2020)	- Facilidade para gerenciar o controle de requisições e balanceamento de carga; - Maior desacoplamento e independência entre os microsserviços e os repositórios.	- Toda comunicação fica dependente da API *Gateway*; - Maior complexidade para gerenciar a sincronização e integração dos dados nos repositórios.

uma classe *Content* que representa o conteúdo e a classe *ContentBuilder*, responsável por construir a classe *Content*. Também nota-se a classe *Rule* que, basicamente, classifica os microsserviços em obrigatórios ou opcionais. Essa é uma classe importante para gerenciar as variabilidades da LPS. É possível ainda perceber a classe *ContentEndPoint* que é utilizada para trocar mensagens com a *API Gateway*.

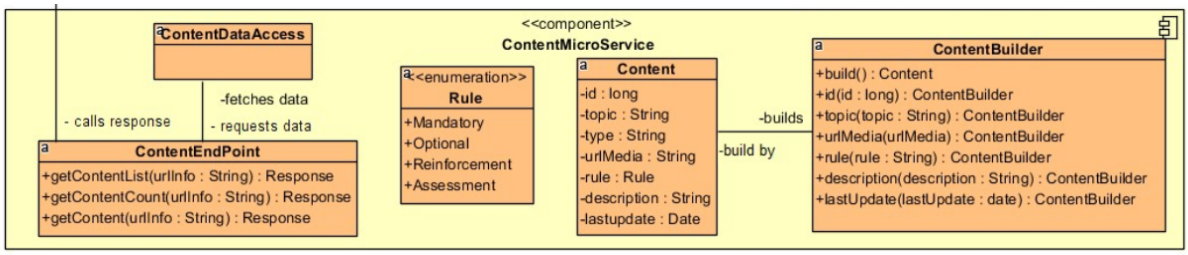

Figura 13 – Modelo de cada microsserviço proposto por Costa et al. (2019) (adaptado de Costa et al. (2019)).

Becker e Lucrédio (2020) em seu artigo definiram que cada funcionalidade deve ser implementada por seu respectivo microsserviço, conforme é apresentado na Figura 14 que representa uma parte dos microsserviços desenvolvidos para uma linha de produtos real, do domínio ERP (*Enterprise Resource Planning*). Observe que na Figura 14 ilustra-se três *web services* que são descritos a seguir: (i) WS Container Tributário, que agrupa os microsserviços nas *features* Tributação e Natureza; (ii) WS Container Cadastro; e (iii) WS

Container Financeiro. Em relação às funcionalidades das *features* Tributação e Natureza foram criados os microsserviços responsáveis pelos gerenciamento de registros e de regras essenciais para a manipulação das suas respectivas funcionalidades.

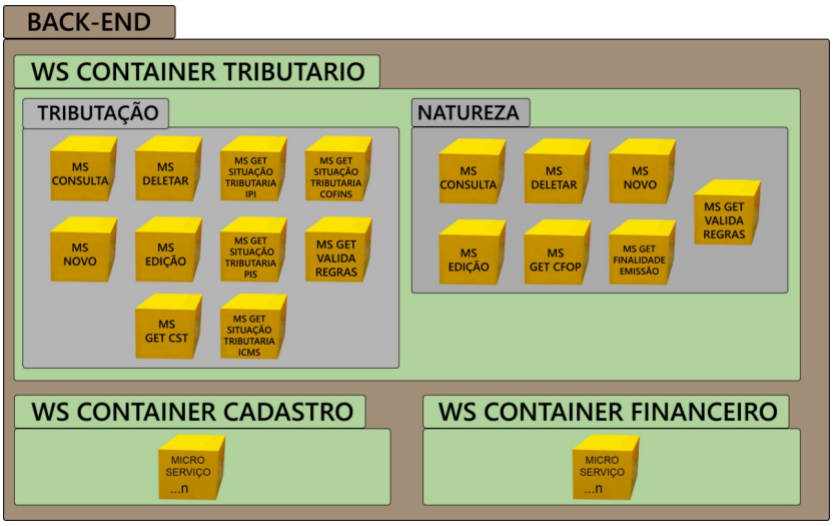

Figura 14 – Excerto dos microsserviços modelados por Becker e Lucrédio (2020).

Setyautami et al. (2020) utilizaram um perfil UML para programação orientada a delta (UML-DOP) que suporta engenharia de LPS (SETYAUTAMI et al., 2016). Assim, adaptando-se o perfil UML-DOP através da adição de alguns estereótipos UML para representar as variabilidades, os autores exemplificaram a aplicação de sua técnica de modelagem arquitetônica por meio da Figura 15. Nela foi modelada apenas a característica Item de Catálogo, pertencente ao estudo de caso trabalhado no artigo, que se refere a aplicativos de lojas virtuais. Tal técnica utilizou o diagrama de classe, que deve ser construído a partir do modelo de características, para analisar a variabilidade de forma estrutural e em seguida confeccionar um diagrama de classes UML com o perfil UML-DOP. Observe na Figura 15 que a partir do perfil UML-DOP é possível representar os recursos obrigatórios e opcionais por meio dos estereótipos *core* e *delta* respectivamente. Dessa forma, observe que todos os atributos e métodos comuns são inseridos no módulo obrigatório, conforme ilustrado na classe *CatalogItem*. Já as variantes requeridas por cada produto são implementadas no módulo *delta*. Isto é apresentado na Figura 15 por meio da classe *CatalogItem* com estereótipo «*modificadoClass*» no módulo *delta DTagCatalog*. Esta classe consiste em três novos atributos e um novo método *getTags*. Para fazer uso do módulo *delta* é preciso incluir no produto seus respectivos *endpoints*, que neste caso são representados pelo componente UML *DisplayTags*. Na Figura 15 representa-se o produto *SockShop* que requer os *endpoints GetItemById* e *DisplayTags*, que de acordo com o diagrama de recursos modelado pelos autores, é, respectivamente obrigatório e opcional.

Tizzei et al. (2017) não fizeram uso do modelo de características para modelar as variabilidades. Apenas um serviço responsável por implementar todas as característi-

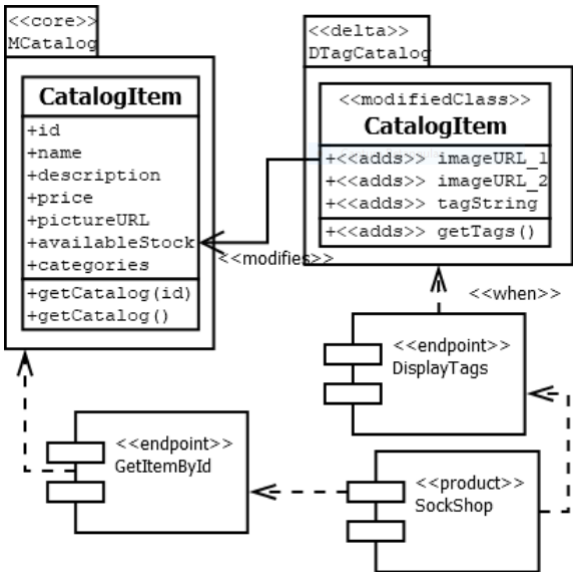

Figura 15 – Excerto dos microsserviços modelados por Setyautami et al. (2020).

cas foi proposto e as variabilidades foram gerenciadas por meio de padrões de projeto, principalmente o padrão de projeto *strategy*, e parametrização nos *endpoints* da API.

Diferentemente das demais abordagens, Mendonça et al. (2020) trabalharam em um método para mapear microsserviços em características. Dessa forma, para construir o modelo de características, tomaram como base, principalmente, as dependências entre os microsserviços identificados nos softwares utilizados no estudo de caso. Portanto, ao analisar os modelos construídos, é possível perceber que eles não representam uma abstração lógica de uma análise de domínio.

Conforme a discussão de cada um dos trabalhos selecionados no início desta sessão, na Tabela 8 são sintetizadas as técnicas propostas para mapeamento entre as características e os microsserviços. Ainda nesta tabela, são apresentadas as implicações em termos de reúso e implementação de variabilidades de cada uma das abordagens propostas.

3.3.3 SQ3 - Quais as metodologias e experimentos utilizados para validar as abordagens, técnicas, linguagens e padrões propostos nos trabalhados?

Para validar suas propostas, cada um dos trabalhos, baseou-se principalmente em estudos de caso, como é relatado a seguir.

Os trabalhos de Benni et al. (2020) e Setyautami et al. (2020) são propostas de soluções aos desafios apresentados por Assunção, Krüger e Mendonça (2020), dessa forma, foram trabalhados os próprios estudos de caso propostos pelos desafiantes, que no caso é uma LPS composta de seis lojas virtuais que foram desenvolvidas com base em arquitetura de microsserviços.

Costa et al. (2019) realizaram um estudo exploratório composto pelas seguintes

Tabela 8 – Resumo das abordagens de mapeamento de características para microsserviços

Autor	**Abordagem**	**Implicações**
Benni et al. (2020)	Cada microsserviço pode implementar várias características por meio de endpoints.	Apesar desta abordagem não ser clara, é possível perceber que microsserviços foram usados para implementar características agrupadas por subdomínio da aplicação. Neste caso, há um elemento integrador com LPS, visto que LPS utilizam a análise de domínio. Todavia, um nível maior de detalhamento seria necessário para um entendimento mais concreto.
Costa et al. (2019)	Cada nó folha do modelo de características gera um microsserviço.	A granularidade dessa proposta trata as características mandatórias e opcionais de forma semelhante, visto que os microsserviços seriam utilizados para implementar tanto as variabilidades como o núcleo. Nesse caso, o reúso poderia atingir o seu nível máximo, mas se faz necessário avaliar os relacionamentos entre as características para que se mantenha a independência entre os microsserviços e a consistência dos dados nos repositórios.
Becker e Lucrédio (2020)	Cada operação CRUD de uma característica gera um microsserviço.	Não viabiliza a garantia do princípio da arquitetura de microsserviços que define um repositório de dados para cada microsserviço, o que torna essa abordagem inviável.
Setyautami et al. (2020)	Cada característica filha da característica raiz gera um microsserviço e as características folha devem ser mapeadas para *endpoints*.	Nos casos em que o modelo de características possua mais de dois níveis, pode possibilitar a modelagem de serviços relativamente grandes. Isso implicaria em sacrificar o reúso e a descaracterização de microsserviço.
Tizzei et al. (2017)	Todas as características foram mapeadas para endpoints de um único serviço e as variabilidades foram implementadas por meio de padrões de projeto e parametrização.	Esta abordagem não faz uso de microsserviços para implementar as variabilidades.

etapas: (i) inicialmente, foi realizado um *survey* com profissionais da área de LPS e microsserviços para identificar padrões e técnicas relacionadas a microsserviços; (ii) na sequência foi proposto um metamodelo de arquitetura para LPS baseada em microsserviços; (iii) para validar o metamodelo elaborado na segunda etapa, como estudo de caso real, foi instanciado uma LPS para criação de aplicações *m-learning* e em seguida um novo *survey* foi realizado e desenvolvedores puderam avaliar a aplicação do metamodelo; (iv) por

último, a título de generalização de domínio, o metamodelo proposto foi instanciado em um outro domínio por alunos de mestrado, que também puderam avaliar as adaptações dos elementos relacionados a microsserviços que foram adicionados ao metamodelo proposto por Colanzi et al. (2014).

Becker e Lucrédio (2020) realizaram um estudo de caso em parceria com uma empresa que possui uma LPS voltada para o domínio de ERP. Neste estudo, os autores realizaram uma comparação entre um cenário com arquitetura centralizada e um segundo cenário com arquitetura de microsserviços. Vale destacar a importância do estudo ter sido realizado em um domínio complexo e de representatividade relevante, visto que ERP normalmente possui uma diversidade considerável de usuários e por conseguinte de demandas variantes.

No trabalho de Tizzei et al. (2017), os autores desenvolveram um estudo empírico usando um SaaS multilocatário de aplicação real com objetivo de migrar para uma arquitetura descentralizada e avaliar o reúso de software e a capacidade de escalabilidade. Para isso, os autores definiram um *pipeline* que compreende três etapas progressivas: construção, teste e implantação. Na infraestrutura foram utilizados contêineres para evitar problemas com dependências e ferramentas para fornecer observabilidade da aplicação. Para verificar o nível de reúso, os autores utilizaram a técnica de contagem de Linhas de Código(LOC), fazendo uso misto de ferramentas automatizadas e auxílio humano. O aumento de LOCs reutilizados foi cerca de 60%, enquanto houve um aumento de 200% da parte não reaproveitada. Dessa forma, após avaliar várias LPS de indústrias, constatou-se a faixa de nível de reutilização entre 50% a 90%. Assim, os autores consideraram que seu trabalho proporcionou um ganho no reúso de software na média do valor esperado.

Mendonça et al. (2020) aplicaram sua abordagem a partir da avaliação de seis sistemas baseados em microsserviços de código aberto. Os seis sistemas são do domínio de lojas virtuais. Como já mencionado, os autores utilizaram os algoritmos evolutivos multi-objetivo NSGA-II e SPEA2 para empregar um processo evolutivo para encontrar um conjunto de modelos de características com compensação que melhor representasse os sistemas escolhidos para o estudo de caso. Neste trabalho, os autores detalharam uma série de passos e regras da sua abordagem e tem como base os seguintes artefatos: (i) entrada: matriz de sistemas e microsserviços e gráfico de dependências entre microsserviços; e (ii) saída: um conjunto de modelos de características. Como os próprios autores reconhecem, faltou considerar os aspectos relacionados ao domínio, ao invés de atender apenas as dependências entre os microsserviços.

Em síntese, a maioria dos trabalhos selecionados aplicaram estratégias que possibilitam uma forma eficiente de validar suas respectivas propostas, principalmente os trabalhos de Tizzei et al. (2017), Becker e Lucrédio (2020) e Costa et al. (2019) que optaram por validar suas abordagens por meio de estudo de caso em softwares reais. Um ponto negativo

que vale ser mencionado é a indisponibilidade da maioria dos documentos completos que foram desenvolvidos em cada estudo.

Diferentemente dos trabalhos apresentados nesta seção, o presente estudo propõe um conjunto de diretrizes que valoriza a análise de domínio, ao priorizar requisitos importantes para as técnicas de LPS e microsserviços, como é o caso do reúso, coesão e baixo acoplamento. Tendo em vista que o reúso e o baixo acoplamento podem ser conflitantes, dado que o reúso tende a enfraquecer a independência dos componentes de software, enquanto o baixo acoplamento tende a dificultar o reúso(RICHARDS; FORD, 2020), a aplicação de princípios de DDD foi relevante para balancear esses requisitos e, assim, oferecer suporte para a modelagem de subdomínios coesos, independentes e que contemplem o reúso, por meio do núcleo da LPS e da reutilização de variabilidades em diversas aplicações multilocatárias. Dessa forma, ao buscar usufruir das vantagens de aplicações tradicionais e de microsserviços, propõe-se um conjunto de diretrizes capaz de produzir uma arquitetura híbrida em que a maioria das variabilidades é implementada por meio de microsserviços e o núcleo da LPS é implementado como aplicações *backend* que podem funcionar no formato de *API´s REST*.

3.4 Aplicações

O processo de consolidação de uma abordagem ou método científico pode e deve passar por aprimoramentos. As abordagens apresentadas nos trabalhos selecionados ainda se caracterizam como propostas incipientes. A comunidade acadêmica deve tomar estas pesquisas como um incentivo à busca por abordagens mais elaboradas e que forneçam melhores diretrizes para os engenheiros de softwares.

Diante deste contexto, pode-se aplicar o conhecimento deste trabalho em projetos de software que tenham os seguintes requisitos: (i) necessidade de atender a um determinado domínio de mercado, como educação, medicina ou qualquer outra área; (ii) que tenha a necessidade de oferecer customização e atender as variabilidades decorrentes dos processos de negócio dos clientes; (iii) e que busque atingir as exigentes demandas de um mercado volátil. Projetar e arquitetar softwares que atendam a estes requisitos não é uma tarefa simples. Assim, a equipe desses projetos de softwares pode conhecer os resultados dos trabalhos apresentados nesta RSL para iniciar pesquisas que possam adaptar, validar e consolidar esses métodos.

Dessa forma, os pesquisadores da área devem oferecer soluções aos engenheiros de softwares, que necessitarão de:

- diretrizes para modelar o domínio de forma a produzir unidades de software desacopladas, que possibilite a aplicação em ambiente distribuído, que por sua vez exige

componentes de softwares independentes e de fácil integração por meio de protocolos de comunicação síncrona e assíncrona;

- abordagens que os auxiliem em decisões mais específicas, como a escolha entre projetar ou não um determinado microsserviço.

- abordagens que proponham diretrizes para um projeto de arquitetura que integre LPS com microsserviços, mas com menor complexidade do que as que estabelecem o mapeamento direto de característica para microsserviço.

Nesse sentido, pode-se tomar como exemplo uma empresa que precise desenvolver um software para atender a centenas ou milhares de clínicas médicas, que apesar de possuírem necessidades em comum, também apresentam algumas diferenças de regras de negócio. Além disso, essas clínicas não possuem infraestrutura suficiente para suportar a disponibilidade, segurança, desempenho e capacidade de escalabilidade que são comumente demandados pelo mercado. Para simular uma situação mais realista, imagine ainda que os requisitos apresentados por essas clínicas estão em constante evolução. Diante deste cenário, a equipe responsável por desenvolver esse software deve ter ciência que um SaaS multilocatário que utilize a técnica de LPS integrada com microsserviços pode ser uma boa estratégia a ser adotada em seu projeto. Portanto, outras equipes, com problemas semelhantes, podem aplicar esta pesquisa como fundamentação para tomar decisões de modelo, projeto e arquitetura de software. E assim aprimorar as abordagens apresentadas nesta RSL ou desenvolver sua própria abordagem sem o risco de incorrer nas desvantagens exploradas neste trabalho.

3.5 Ameaças à Validade do Estudo desta RSL

Buscou-se executar uma pesquisa da forma mais ampla possível, mas sem perder o foco, com o objetivo de sistematizar o estado da arte da integração de LPS com microsserviços. Entretanto, algumas questões relevantes podem ameaçar os resultados deste trabalho, conforme listado a seguir:

- a maioria dos trabalhos que foram descartados, logo na fase inicial de leitura do título e do resumo, abordavam LPS integrada com *web services*, sem tratar de microsserviços. Suspeita-se que a leitura total desses trabalhos poderia fornecer uma fundamentação mais ampla e consistente para a contribuição deste trabalho. A escolha de manter o foco apenas nos artigos que abordavam obrigatoriamente LPS e microsserviços se deu devido à dificuldade de conciliar os objetivos específicos desta pesquisa com toda uma massa de conhecimento que antecederam o contexto da integração de LPS com microsserviços. Além disso, um dos membros desta pesquisa

possui experiência em LPS, inclusive com trabalhos publicados justamente no tema dos artigos descartados. Assim, espera-se atenuar esta ameaça.

- a possibilidade da existência de artigos de conferências e de periódicos não indexadas nas bases de dados definidas para esta pesquisa representa uma outra ameaça à contribuição deste trabalho. Para mitigar essa ameaça, as bases de dados escolhidas indexam os periódicos e conferências mais importantes relacionados à LPS e engenharia de software. Além disso, foram realizadas buscas manuais nas principais comunidades relacionadas a LPS e os artigos encontrados já estavam entre os encontrados nas bases de dados.

- devido a arquitetura de microsserviços ter sido inicialmente concebida na indústria e as pesquisas ainda serem recentes na academia sobre essa tecnologia (CARVALHO et al., 2019), os trabalhos publicados podem não representar o estado da arte que reflita a realidade empregada no mercado. No entanto, os autores deste trabalho fazem parte de uma equipe maior, a qual possui representantes da indústria, visto que os conhecimentos adquiridos por meio desta RSL serão aplicados em um estudo de caso de LPS real. Dessa forma, muitas implicações e discussões relatadas na seção 3.3 são baseadas em experiências da indústria.

- a maioria das documentações completas utilizadas nos trabalhos selecionados não estão disponíveis para uma análise mais aprofundada. Apesar desses estudos serem ricos em detalhes, os resultados e a aplicação das abordagens propostas podem ter sido pouco explorados ou mal interpretados. Tendo em vista que não foi encontrado na literatura nenhuma RSL que trate do problema desta pesquisa, e assim não foi possível comparar e validar, esta RSL se apresenta como trabalho pioneiro. Nesse sentido, este trabalho pode alavancar a discussão sobre a temática abordada e ser bastante útil em pesquisas futuras que poderão aprová-la ou reprová-la.

3.6 Considerações Finais do Capítulo

Esta revisão sistemática apresenta o estado da arte da integração de LPS e microsserviços, levando-se em consideração os trabalhos publicados entre os anos de 2011 a dezembro/2022 nos principais veículos de divulgação científica. Ressalta-se que apesar dos trabalhos terem sido pesquisados a partir de 2011, apenas trabalhos de 2017 em diante foram selecionados, tendo como consequência um baixo número de trabalhos publicados, o que evidencia uma falta de consolidação sobre as arquiteturas, padrões e técnicas de modelagem aplicados na construção de LPS integrada com microsserviços . Nesse sentido, foram identificadas e apresentadas algumas lacunas de pesquisa. Essa RSL seguiu a abordagem proposta por Biolchini et al. (2005) e alguns princípios definidos por Kitchenham (2004). Para isso, foi estabelecido um protocolo de pesquisa de forma detalhada.

Em relação às arquiteturas, técnicas de modelagem e métodos de validação identificados, este trabalho detalha cada proposta, destacando-se os pontos mais relevantes e os mais fracos. Vale destacar a inexistência de trabalhos relacionados que registrem, de forma sistemática, as soluções publicadas.

Assim, foram analisados seis estudos relacionados à integração de LPS com microsserviços. A metade dos artigos selecionados propuseram soluções para as três questões desta pesquisa. Em relação a primeira pergunta, referente a arquitetura, as propostas são semelhantes. Em relação a segunda pergunta, referente as técnicas de modelagem e implementação, as propostas são diversificadas. Por fim, em relação a terceira pergunta, que trata dos métodos de validação das propostas, observou-se que algumas utilizaram técnicas de consulta a especialistas, além de fazer uso de estudo de caso com aplicações reais.

Após uma análise refinada, é possível destacar algumas lacunas de pesquisa relacionadas às abordagens de mapeamento de características para microsserviços, principalmente quando se trata de características que compartilham os mesmos dados ou quando os microsserviços envolvidos podem fazer parte de várias características, como é o caso de características relacionadas relatórios. Uma outra lacuna refere-se às variabilidades nos dados, pois não foi abordado nos trabalhos encontrados. Também ainda são escassas propostas mais generalistas, sendo a maioria delas voltadas para resolver os problemas específicos de seus respectivos estudos de caso.

Além das lacunas já citadas, outro ponto muito importante e que não foi abordado de forma evidencial, foi o reúso, pois não foi apresentado uma relação direta entre o uso dos microsserviços para implementação das variabilidades e o aproveitamento de recursos de software. Visto que essa última lacuna é essencial para LPS, pode-se constatar que ainda há muito o que se investigar sobre este assunto. Desta forma, trabalhos futuros podem se concentrar em uma abordagem que detalhe melhor a aplicação dos microsserviços para implementar as variabilidades e possa mostrar metricamente os ganhos relacionados à reúso de software.

Por fim, é relevante mencionar que pesquisas posteriores podem se basear no protocolo desta revisão sistemática com o objetivo de melhorar as soluções e diagnosticar o progresso da área no decorrer do tempo. Também considera-se importante trabalhos futuros relacionados a engenharia de aplicação a fim de complementar este trabalho que concentrou-se em responder questões ligadas, principalmente, a engenharia de domínio.

4 Diretrizes para Produzir uma Arquitetura Híbrida para LPS

Neste capítulo são apresentadas as diretrizes para a modelagem da arquitetura a partir do modelo de característica. Seguindo-se essas diretrizes, apresenta-se uma proposta de arquitetura de referência para uma linha de produtos que utilize o modelo de SaaS Multilocatário. No final deste capítulo é apresentada uma análise dessa arquitetura em termos dos fatores de qualidade definidos na Seção 2.2.2.

4.1 Diretrizes para mapeamento do modelo de características para a arquitetura de referência da LPS

Embora tenham sido encontrados trabalhos na literatura com o objetivo de auxiliar na definição de uma arquitetura de microsserviços para implementar as variabilidades de LPS, esses trabalhos revelaram uma diversidade de ideias e lacunas que suscitam a necessidade de, ainda mais, esforços de pesquisa para se obter um resultado com maior contribuição para a engenharia de software.

Para contribuir com o tema e complementar os trabalhos abordados na Seção 3.3, após a construção do modelo de característica, este trabalho propõe um conjunto de diretrizes para que, a partir desse modelo, engenheiros de software sejam capazes de gerar uma arquitetura híbrida. A arquitetura deve prever a utilização de microsserviços, que podem se comunicar com artefatos de maior granularidade, chamados neste trabalho de aplicações *backend*. Essas aplicações devem contemplar a implementação de várias características.

As diretrizes foram definidas com o propósito de guiar os engenheiros de softwares na modelagem de elementos arquiteturais que atendam aos requisitos da arquitetura de microsserviços, relatadas na Subseção 2.2.3, mas que tenham a complexidade de desenvolvimento atenuada, ao passo que objetivam fornecer um guia para implementar as variabilidades de uma LPS que funcione como um SaaS multilocatário. Tais diretrizes baseiam-se nas abordagens selecionadas na RSL e em requisitos relevantes para a modelagem de microsserviços, como é o caso da definição de contextos bem definidos, coesos e com baixo acoplamento. Para atingir esses requisitos, utilizou-se de princípios de contexto delimitado da técnica de DDD.

As diretrizes para realizar o mapeamento do modelo de características para arquitetura da LPS de SaaS Multilocatário são elencadas a seguir:

4.1.1 Diretriz - 1

O modelo de características deve ser construído levando-se em consideração os subdomínios da LPS.

a) Cada subdomínio deve ser mapeado para uma característica de segundo nível;

b) Em seguida, as características dos demais níveis devem ser organizadas em seus respectivos subdomínios.

Segundo Lewis e Fowler (2014), os microsserviços devem ser projetados conforme as regras de negócio e possuir um contexto bem definido. Dessa forma, a Diretriz 1 objetiva produzir um modelo de característica que possa produzir aplicações *backend's* ou microsserviços coesos e com baixo acoplamento. Assim, essa diretriz prepara o ambiente para que a variabilidade da LPS possa ser implementada por microsserviço. Um modelo que não siga essa diretriz não deve auxiliar no projeto de componentes arquiteturais independentes, fáceis de evoluir, extender e manter.

4.1.2 Diretriz - 2

Cada característica de segundo nível deve gerar uma aplicação *backend* (API).

a) Para cada característica de segundo nível, deve ser gerado um microsserviço para prover os relatórios relacionados a todas as respectivas características descendentes;

b) As características obrigatórias descendentes devem ser mapeadas para *endpoints* das suas respectivas aplicações.

As diretrizes 2 e 2.B foram baseadas na abordagem de Setyautami et al. (2020), que utiliza como fundamentação os princípios da Diretriz 1.

Com a finalidade de melhorar a escalabilidade e o desempenho, visto que essa proposta promove o armazenamento de dados de forma distribuída, a Diretriz 2.A visa agregar em uma única base os dados que estão distribuídos em diversos bancos de dados para otimizar o tempo de geração de relatórios. Essa diretriz utiliza princípios do padrão CQRS (*Command Query Responsibility Segregation*), que separa as operações de escrita das operações de leitura, permitindo assim otimizações específicas para cada tipo de operação (PACHECO, 2018).

4.1.3 Diretriz - 3

A exceção à Diretriz 2 ocorre nas características que não estão relacionadas ao domínio do negócio e que são de utilidade para todo o sistema, como é o caso

de autenticação e observabilidade. Nesses casos, cada característica deve gerar um microsserviço. De forma específica, recomenda-se:

a) projetar um microsserviço para prover os recursos relacionados a autenticação e autorização. Esse serviço deve agrupar os dados de acesso e de permissões de todos os locatários. Dessa forma, esse microsserviço deve gerenciar as configurações de acesso aos recursos opcionais e alternativos da LPS;

b) projetar um microsserviço para prover recursos relacionados à observabilidade da LPS.

A Diretriz 3 também é consequência dos fundamentos da Diretriz 1, visto que existem características em uma LPS que estão mais relacionadas às soluções técnicas de software do que ao problema do domínio do negócio; e que por essa razão, sua implementação pode ser reutilizada por todo o sistema.

A Diretriz 3.A visa centralizar o serviço de autenticação e autorização para facilitar o processo de validação de credenciais em todo o sistema (DIAS; SIRIWARDENA, 2020).

Conforme relatado na Subseção 2.2.3, ao utilizar microsserviços, é recomendado implantar ferramentas que facilitem a identificação de erros e falhas (LI et al., 2022). Sendo assim, a Diretriz 3.B visa atingir esse requisito.

4.1.4 Diretriz - 4

Cada característica opcional ou alternativa pertencente aos demais níveis, que origina entidades[1] de domínio do negócio e não se restringe apenas a variações nos campos/atributos de entidades já existentes, deve ser mapeada para microsserviço.

a) Caso tal característica represente apenas variações nos campos/atributos das entidades de domínio já existentes, ela deve ser administrada por meio de parâmetros nos *endpoints* e não deve gerar microsserviço.

A Diretriz 4 busca implementar as variabilidades de LPS por meio de microsserviços, conforme é proposto de diferentes formas nos trabalhos selecionados na RSL, não existe ainda uma recomendação padronizada para essa finalidade. Nesse caso, a fundamentação advém de implicações dos conhecimentos adquiridos durante essa pesquisa e segue a lógica de que se apenas alguns dos produtos da LPS podem incluir essas características, elas podem ser desacopladas das aplicações *backend's*, podendo ser escaladas de forma independente e serem reutilizadas apenas pelos membros da LPS que as inclua.

[1] Observa-se que uma entidade de domínio do negócio é uma representação abstrata de um objeto ou conceito do mundo real, que possui atributos e relacionamentos.

Buscando projetar microsserviços independentes, os bancos de dados não podem ser compartilhados entre diferentes microsserviços (LEWIS; FOWLER, 2014). Dessa forma, a Diretriz 4.A recomenda que não seja projetado um microsserviço somente para resolver variabilidades de campos/atributos de entidades de domínio que já tenham sido contempladas. Essa estratégia também diminui a complexidade de desenvolvimento, tendo em vista não ser necessário a utilização de estratégias de sincronização dos dados entre os microsserviços que compõem um determinado ponto de variabilidade.

4.1.5 Diretriz - 5

Devem ser projetados BFF's [2] de acordo com os perfis de usuários da LPS.

a) Nesse caso a relação entre BFF e perfil de usuário não é direta, devendo ser agrupado em apenas um BFF os perfis que possuem necessidades comuns, sendo as diferenças apenas nos níveis de permissão.

A Diretriz 5 foi definida para guiar os engenheiros de software a projetarem componentes de software para melhor atender a camada *frontend*, além de buscar contornar o gargalo de ter um único nó de acesso para todos os serviços, o que poderia causar a inoperância de todo o sistema caso esse nó venha a parar de atender as requisições (COSTA et al., 2019).

A ideia de utilizar uma arquitetura híbrida visa contemplar os benefícios da arquitetura de microsserviços na implementação das variabilidades, mas também busca reduzir a complexidade de implementação gerada por esse tipo de arquitetura, ao passo que também faz uso de elementos de arquiteturas que mais se assemelham a monolíticos e são mais simples e rápidos de serem implementados. Desta forma, é possível se beneficiar de requisitos considerados conflitantes em sistemas distribuídos, como é o caso de consistência atômica dos dados versus desempenho, e ainda promover uma redução de custos, redução da complexidade de desenvolvimento e entrega mais rápida do software por meio da simplificação de alguns componentes dessa arquitetura.

Adicionalmente, essa proposta de arquitetura híbrida deve ser adotada como um passo intermediário para uma entrega mais rápida da LPS. Após a organização desenvolvedora começar a lucrar com os produtos entregues, com o amadurecimento da LPS, as aplicações *backend* podem ser desmembradas e a LPS pode evoluir para uma arquitetura puramente de microsserviços. Dessa forma, o impacto negativo da

[2] O conceito de BFF (*Backend for Front-end*) se baseia na dificuldade de lidar com diferentes e numerosos microsserviços ao desenvolver APIs e integrá-los. O BFF age como uma camada intermediária que auxilia na organização de arquiteturas orientadas a microsserviços, coordenando as funcionalidades entre os microsserviços. Quando uma aplicação precisa de dados, a solicitação é traduzida através do BFF, que se comunica com os microsserviços (COSTA et al., 2019).

complexidade da arquitetura de microsserviços é suavizado e as decisões arquiteturais podem ser tomadas a posteriori com maior embasamento.

Com base nas sugestões fornecidas por um dos participantes do experimento controlado, cujos resultados são relatados no Capítulo 6, foi elaborado um (*Business Process Model and Notation*) que ilustra o processo de aplicação das diretrizes, com exceção da primeira e da última diretriz, conforme pode ser observado na Figura 16.

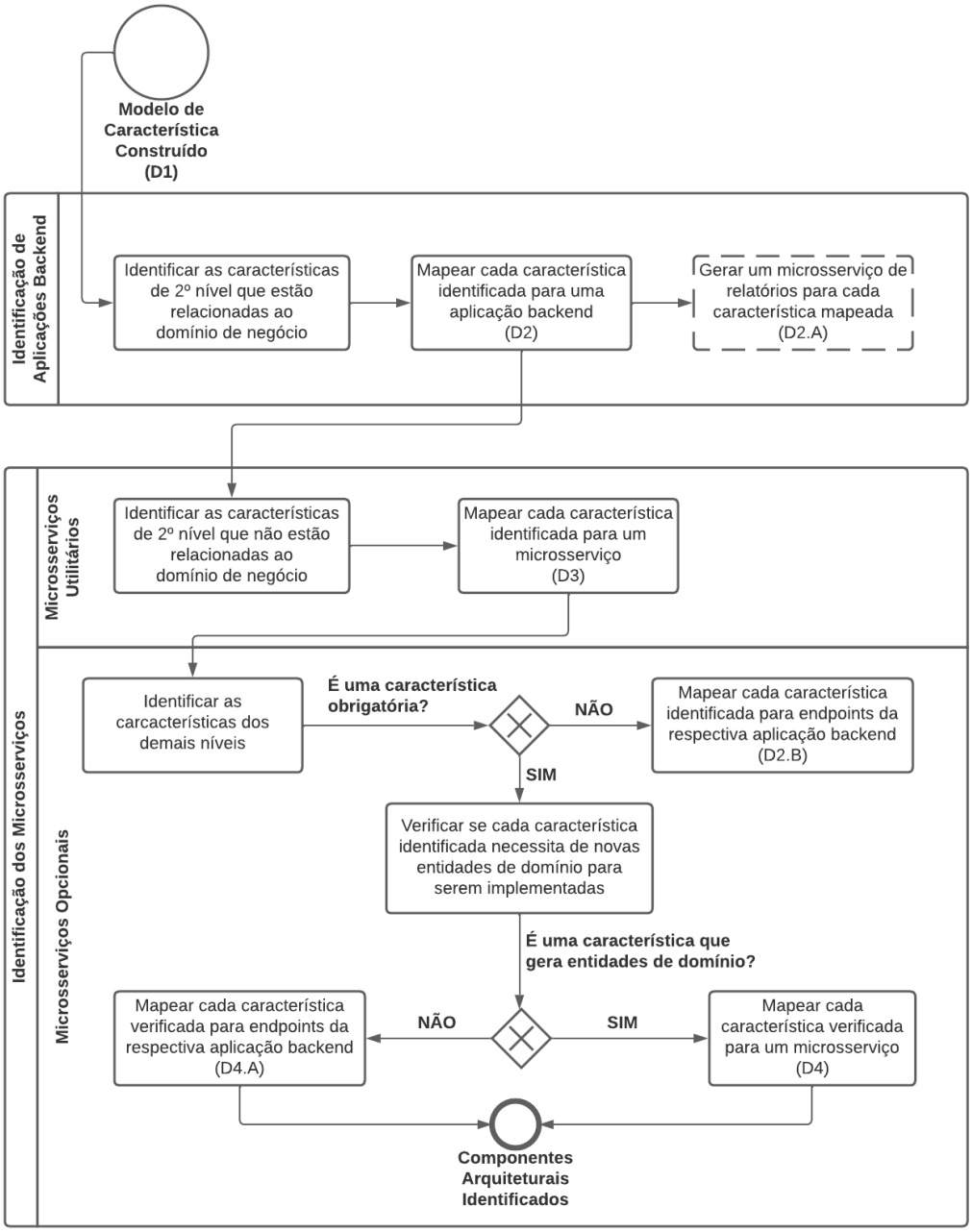

Figura 16 – Representação das diretrizes em fluxograma

Um exemplo prático de mapeamento do modelo de características para componentes arquiteturais utilizado em um estudo de caso real, a partir da aplicação desse conjunto de diretrizes, é apresentado e descrito nas Seções 5.3 e 5.4.

4.2 Camadas e elementos que compõem a Arquitetura

A partir das diretrizes propostas e apresentadas na seção anterior, foram definidos os elementos para a arquitetura de referência da LPS. Os modelos arquiteturais apresentados neste trabalho seguem o modelo C4.

Os elementos arquiteturais propostos neste trabalho podem ser categorizados em *person* e *container*, conforme apresenta-se o diagrama de *containers* na Figura 17. Enquanto os *persons* representam os usuários do sistema, os *containers* foram utilizados para representar cinco tipos de elementos de software descritos a seguir:

- *frontend* (*containers* na cor vermelha): os usuários podem interagir com o software por meio desta camada. Ela pode ser composta por *Single Page Applications* (SPAs) ou aplicativos móveis para atender as necessidades dos diversos perfis de usuários. Também é proposta uma *single page application* (SPA) para disponibilizar recursos de configuração e monitoramento aos administradores da LPS. Cada uma dessas aplicações usa seu respectivo *backend for frontend* (BFF) por meio do protocolo HTTP;

- API's *gateway* (*containers* na cor roxa): essa camada é formada por BFF's que são responsáveis por receber as requisições dos seus respectivos *frontend's* e direcionar para aplicações *backend's* ou para os microsserviços. Dessa forma, ela realiza o balanceamento de carga e assume a posição de orquestrador;

- Aplicações *backend's* (*containers* na cor amarela): essa camada é formada por *API's REST*, e elas devem ter os seus escopos modulados de acordo com os subdomínios da LPS, seguindo princípios de DDD que recomenda implementar uma aplicação *backend* para cada *bound context*. Além disso, essa camada pode ser dividida em componentes de dois tipos: (i) núcleo, responsável por implementar as características obrigatórias e (ii) opcional, responsável por implementar as características opcionais ou alternativas. Neste caso, como proposta inicial de projeto, as aplicações *backend's* também podem ser utilizadas para implementar as variabilidades da LPS;

- Microsserviços (MS) (*containers* na cor verde): nessa camada são implementadas as características opcionais e que estejam a partir do terceiro nível do modelo de características. Assim, como proposta inicial de projeto, os microsserviços são utilizados para implementar as variabilidades da LPS. Além disso, é importante destacar três microsserviços obrigatórios para todos os membros da LPS: MS Autenticação, responsável por realizar o controle de autenticação e autorização da LPS; MS Relatório, responsável por registrar dados integrados em repositório *NoSQL* de forma assíncrona, por meio de mensageria, e disponibilizar esses dados em consultas e relatórios; MS Monitor, responsável por registrar e disponibilizar, também de forma

assíncrona, *logs* e dados de rastreabilidade, a fim de fornecer condições para se realizar observabilidade do sistema;

- Repositórios (*containers* na cor azul): por último, na camada de repositórios são armazenados e disponibilizados os dados para aplicações *backend* ou para os microsserviços. É relevante mencionar que cada aplicação *backend* ou microsserviço possui o seu próprio repositório. Além disso, também é importante destacar dois pontos: por se tratar de um SaaS multilocatário, cada locatário possui seus próprios repositórios, podendo assim, ser garantido o isolamento dos dados e um escalonamento mais eficiente ao permitir analisar a carga de cada inquilino para se realizar um possível escalonamento; e, existe a possibilidade de se escolher de forma heterogênea as tecnologias de banco de dados e ainda poder garantir o desempenho e consistência dos dados.

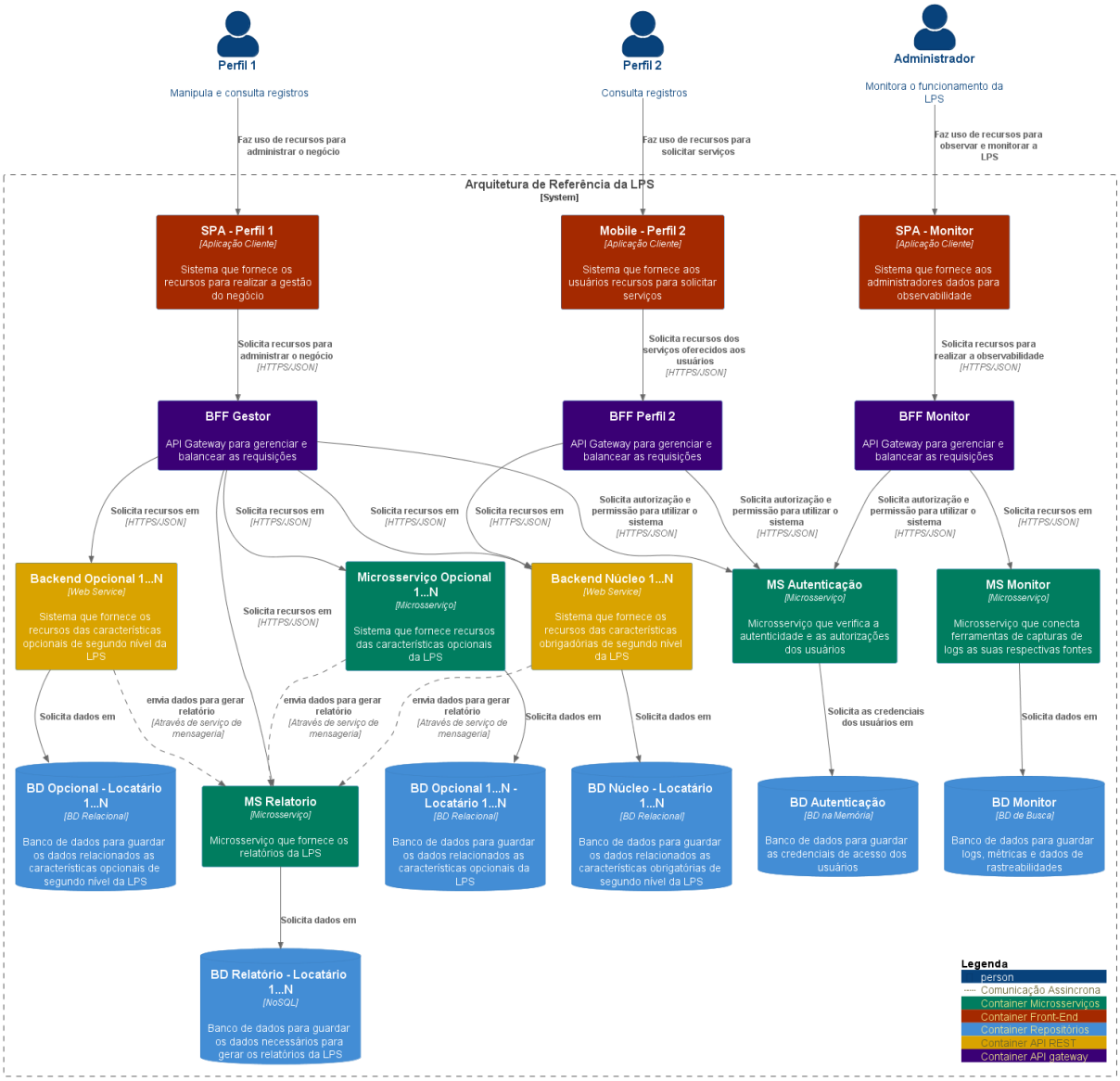

Figura 17 – Arquitetura de referência da LPS

4.3 Análise dos Fatores de Qualidade da Arquitetura

Em termos de atributos de qualidade, as implicações das estratégias adotadas para modelagem da arquitetura proposta neste trabalho são detalhadas a seguir.

- Escalabilidade: O escalonamento, normalmente, adiciona complexidade, e por muitas vezes o banco de dados é um dos elementos que pode ser responsável por restringir o potencial da escalabilidade. Por esse motivo, propõe-se que cada locatário tenha o seu próprio banco de dados. Dessa forma, é possível customizar o escalonamento de acordo com as demandas de cada locatário. Essa estratégia garante o isolamento dos dados de cada inquilino. Além disso, em termos de serviços, como foi proposto uma arquitetura híbrida, o desmembramento das aplicações *backend* em microsserviços será facilitado, já que a arquitetura proposta, pelo menos em parte, já faz uso dessa estratégia. Adicionalmente, o escalonamento pode ser facilmente aplicado por meio dos microsserviços que implementam as variabilidades da LPS.

- Desempenho: Para atender a esse atributo, além de uma perspectiva de escalonamento, tendo em vista que normalmente o número de consultas é maior que os demais tipos de requisição, propõe-se uma estratégia de projetar microsserviços específicos para consultas e relatórios, em conjunto com bancos de dados NoSQL, que tende a responder melhor nesses casos. É importante destacar que a alimentação dessas bases de dados NoSQL devem ser implementadas de forma assíncrona por meio de serviços de mensageria. Outro ponto importante, é o uso de *cache*, por meio de um banco em memória, proposto principalmente para o microsserviço responsável pela autenticação e autorização do sistema.

- Disponibilidade: Projetos de arquitetura que incluem um *gateway* como único elo entre o *front e back* possuem um gargalo, visto que caso o *gateway* venha a ficar inoperante, nenhuma requisição poderá ser atendida. A arquitetura proposta neste trabalho pretende atender ao atributo de disponibilidade por meio da divisão das requisições em BFF's, ao invés de deixar a cargo de apenas uma *API Gateway* para realizar o balanceamento de carga. Dessa forma, além de cada BFF balancear as requisições específicas dos seus contextos, o escalonamento por meio de novas instâncias dos microsserviços e das aplicações *backend* pode ser orquestrado.

- Segurança: Por se tratar de uma arquitetura distribuída este trabalho sugere que os diversos componentes sejam implantados em ambientes de infraestrutura diferentes. Outro fator que pode contribuir para garantir a disponibilidade, em casos de ataque, é a possibilidade da instanciação dinâmica das aplicações e microsserviços. Por fim, o microsserviço de observabilidade oferece recursos para melhor identificar e anular ataques.

- Manutenibilidade: Por se tratar de uma arquitetura baseada em microsserviços, pode-se contar com componentes de baixo acoplamento e independentes. Desta forma, podem existir equipes diferentes e dedicadas para cada microsserviço ou componente de software.

- Flexibilidade: Um dos objetivos principais de uma LPS é oferecer alternativas para a execução de um determinado requisito de negócio, que eventualmente pode possuir restrições ou comportamentos diferentes para cada locatário. Essa customização pode ser atendida por meio dos microsserviços opcionais, propostos na arquitetura deste trabalho.

- Extensibilidade: A arquitetura de microsserviços facilita o incremento de novas funcionalidades sem interferir na integração com as funcionalidades existentes, visto que o baixo acoplamento e a independência é um princípio desta arquitetura.

- Evolutividade: Novamente, a arquitetura de microsserviços pode oferecer independência de tecnologia e componentes, o que resulta na possibilidade do sistema ser bastante heterogêneo em termos de tecnologias e seus componentes possam ser evoluídos de forma independente.

Em termos de estratégias de integração, a seguir são descritos os fatores e as respectivas técnicas sugeridas por esta pesquisa:

- Comunicação: sugere-se a utilização de comunicação síncrona nas requisições que alimentam os bancos de dados para garantir a consistência, e comunicação assíncrona por meio de tecnologias de mensageria para alimentar os repositórios de consulta. Assim, é possível dispor de consistência dos dados e um melhor desempenho nas requisições de consultas mais complexas.

- Consistência: propõe-se a consistência atômica nas operações que modificam os dados, e eventual nas operações que alimentam os microsserviços responsáveis por disponibilizar operações de consultas e emissão de relatórios. Dessa forma, as operações de maior demanda podem ter uma fonte de dados que oferece maior desempenho, ainda que não ofereçam uma consistência atômica.

- Coordenação: este trabalho propõe a orquestração por meio dos BFFs que atuam na coordenação do balanceamento de carga, seja por meio de uma distribuição de requisições mais eficiente entre as instâncias dos serviços ou por meio da instanciação de novos microsserviços. Pode-se advogar contra essa estratégia o fato de possuir um único ponto de acesso. Entretanto, o trabalho do *gateway* demanda um processamento leve e além disso, é proposto um BFF para cada contexto de usuário de modo a amenizar esse ponto negativo.

Por fim, é necessário analisar o reúso. Nesse sentido, é possível constatar que a arquitetura de microsserviços promove um baixo acoplamento, o que reduz o reúso nos casos em que entidades são replicadas em diversos microsserviços. Apesar disso, o sistema ganha em coesão, e o reúso é contemplado a partir da utilização dos componentes comuns em todos os membros da LPS, e na reutilização dos microsserviços que implementam variabilidades e podem ser usados por diversos produtos. Além disso, é vantajoso que os componentes opcionais e alternativos sejam mais independentes e com contextos muito bem definidos, pois isso evita uma série de problemas relacionados a manutenção, extensibilidade e evolução do software.

4.4 Considerações Finais do Capítulo

Neste capítulo foram apresentadas as diretrizes propostas por este trabalho. A arquitetura de referência da LPS obtida como resultado da aplicação dessas diretrizes foi analisada critério a critério de acordo com os requisitos definidos na Subseção 2.2.2. Diferente dos trabalhos selecionados na RSL, pode-se perceber que os requisitos da arquitetura de microsserviços foram atendidos, ao mesmo tempo em que a complexidade inerente a essa arquitetura foi reduzida ao se utilizar a combinação com aplicações de maior granularidade. Assim, observa-se que o objetivo proposto por este trabalho foi atingido, ao apresentar diversas vantagens em comparação com as demais abordagens encontradas na literatura.

5 Estudo de Caso

Neste capítulo são apresentados os documentos e artefatos que foram produzidos no desenvolvimento do estudo de caso. Este capítulo está organizado da seguinte forma: na Seção 5.1 é apresentado o contexto do software; na Seção 5.2 constam a listagem dos requisitos e as estórias de usuários; na Seção 5.3 é apresentado o modelo de características; na Seção 5.4 a arquitetura instanciada no estudo de caso é apresentada e descrita; na Seção 5.6 são apresentados os projetos de banco de dados; na Seção 5.7 são descritos alguns detalhes da implementação; na Seção 5.8 é apresentado uma comparação com uma abordagem de arquitetura puramente de microsserviços; e, por último, na Seção 5.9 são relatadas as considerações finais deste capítulo.

5.1 Contextualização

Este trabalho de pesquisa foi conduzido utilizando-se um SaaS multilocatário do mundo real para atender ao domínio de clínicas médicas. Este software automatiza os processos dos setores de atendimento ao paciente, consultório, almoxarifado, tesouraria e financeiro, além de atender aos principais atores envolvidos em uma clínica, tais como:

- Paciente, por meio de um aplicativo *mobile* para agendamento de atendimento e acompanhamento dos serviços utilizados;

- Secretária, por meio de um aplicativo web que oferece recursos para agendar atendimento, manter uma comunicação com os pacientes, controlar o estoque de produtos, emitir diversos relatórios, realizar recebimentos e pagamentos de fornecedores;

- Profissional de saúde, por meio de um aplicativo web que possibilita o controle de atendimentos, o registro completo e personalizado do prontuário.

Para atender a esses recursos, este estudo desenvolveu uma LPS que utiliza microsserviços para implementar as variabilidades e operar no modelo SaaS. Dessa forma, os recursos comuns e a maioria dos recursos específicos de cada clínica podem ser atendidos. Para isso, foi realizada a análise de domínio, tomando como base o software para hospitais da empresa parceira e quatro softwares concorrentes já consolidados no mercado.

5.2 Requisitos da LPS

5.2.1 Eliciação dos requisitos de cinco softwares

Ao analisar os documentos de casos de uso do software da empresa parceira desta pesquisa - (o Softmedical) e fazer uso das funcionalidades dos demais softwares em ambiente de demonstração gratuita, pôde-se catalogar os principais requisitos do domínio de clínicas médicas. Na Figura 18 são apresentadas as principais funcionalidades dos softwares Softmedical, S1, S2, S3 e S4. Nessa tabela, a coluna núcleo é sinalizada quando a funcionalidade está presente em todos os softwares avaliados. Dessa forma, esta coluna foi utilizada para definir as características que fazem parte do núcleo da LPS.

Elicitação de Requisitos						
Funcionalidades	Softmedical	S1	S2	S3	S4	Núcleo
Agenda						
Agendar Paciente	✓	✓	✓	✓	✓	✓
Cadastro de dias/horários de atendimento de cada médico	✓	✓	✓	✓	✓	✓
Lista de espera (encaixe)	✓	✓				
Agendamento pelo paciente		✓		✓	✓	
Cadastro de Evento (falta de um profissional, manutenção de equipamento etc)			✓		✓	
Atendimento						
Cadastro de paciente (geral, endereço, dados complementares, dados familiares, outras informações, convênio)	✓	✓	✓	✓	✓	✓
Iniciar atendimento	✓	✓	✓	✓	✓	✓
Prontuário (resumo, anamnese, exame físico, resultados de exames, hipótese diagnóstica, condutas, exames e procedimentos, prescrições, documentos e atestados, Imagens e anexos)	✓	✓	✓	✓	✓	✓
Cadastro de Colaborador(médico, recepcionista etc)	✓	✓	✓	✓	✓	✓
Enviar lembrede de atendimento (sms, whatsApp, email)		✓		✓	✓	
Teleconsulta		✓	✓		✓	
Relatórios						
Atendimentos	✓	✓	✓	✓	✓	✓
Pacientes	✓	✓	✓	✓	✓	✓
Financeiro - Repasse médico		✓	✓			
Financeiro - Demonstrativo de resultados		✓	✓	✓		
Financeiro - Fluxo de Caixa		✓		✓	✓	

Figura 18 – Eliciação dos Requisitos da LPS

5.2.2 Estórias de Usuários

Com base nos requisitos elicitados e na experiência da equipe da empresa parceira, foram elaboradas as estórias dos usuários. Na Figura 19, encontram-se descritas as principais estórias de usuários, que, de forma semelhante aos requisitos, foram categorizadas em núcleo, quando a estória é comum a todos os softwares avaliados.

Estórias de Usuário	
Funcionalidades	Núcleo
1 - Agenda	
3.013 - Agendar Atendimento pela secretária Como secretária, quero agendar, confirmar ou cancelar um atendimento médico para que os atendimentos possam ser organizados.	TRUE
3.014 - Agendar Atendimento pelo paciente Como paciente, quero agendar atendimento médico para que eu possa ter maior facilidade e agilidade.	FALSE
3.015 - Cadastro de dias/horários de atendimento de cada médico Como secretária, quero definir os dias da semana, os horários do dia e as configurações de atendimento para que possa ser agendado atendimento somente nos dias/horários predefinidos.	TRUE
3.016 - Cadastro de Evento (falta de um profissional, manutenção de equipamento etc) Como secretário, quero poder inserir um evento de impedimento da agenda médica para que não seja agendado um horário que apesar de ter sido configurado para atendimento, mas que aconteceu um imprevisto.	FALSE
3.017 - Lista de espera (encaixe) Como secretário, quero gerenciar uma lista de pacientes que esperam vaga de atendimento, para que eu possa encaixá-los em caso de desistência dos pacientes já agendados.	FALSE
3.018 - Agenda por Horário Marcado Como secretário, quero poder definir a sequência de atendimento por hora pré definida, para que os atendimentos sejam organizados dessa forma.	Alternativa
3.019 - Agenda por Ordem de Chegada Como secretário, quero poder definir a sequência de atendimento pela ordem de chegada do paciente, para que os atendimentos sejam organizados dessa forma.	Alternativa
5 - Relatório	
5.037 - Agendamentos por status Como secretária, quero gerar relatórios de atendimentos por status para que se possa conhecer a quantidade de atendimentos distribuída em cada status.	TRUE
5.039 - Atendimentos por período Como secretária, quero gerar relatórios de atendimentos para que se possa conhecer a quantidade de atendimentos por período.	TRUE
5.045 - Financeiro - Demonstrativo de resultados Como secretária, quero gerar relatório de entradas e saídas financeiras por plano de contas para que eu possa avaliar o plano de contas e o resultado final.	FALSE

Figura 19 – Estórias de Usuários

5.3 Modelagem das Características

Seguindo-se a Diretriz 1 proposta por este trabalho, e de acordo com a abordagem FODA, o modelo de características foi produzido buscando-se organizar os subdomínios da LPS em características de segundo nível. Na Figura 20 ilustra-se um extrato do modelo de características da LPS do estudo de caso desta pesquisa. Nessa imagem é possível identificar os subdomínios agenda, atendimento, financeiro e estoque, e suas respectivas características.

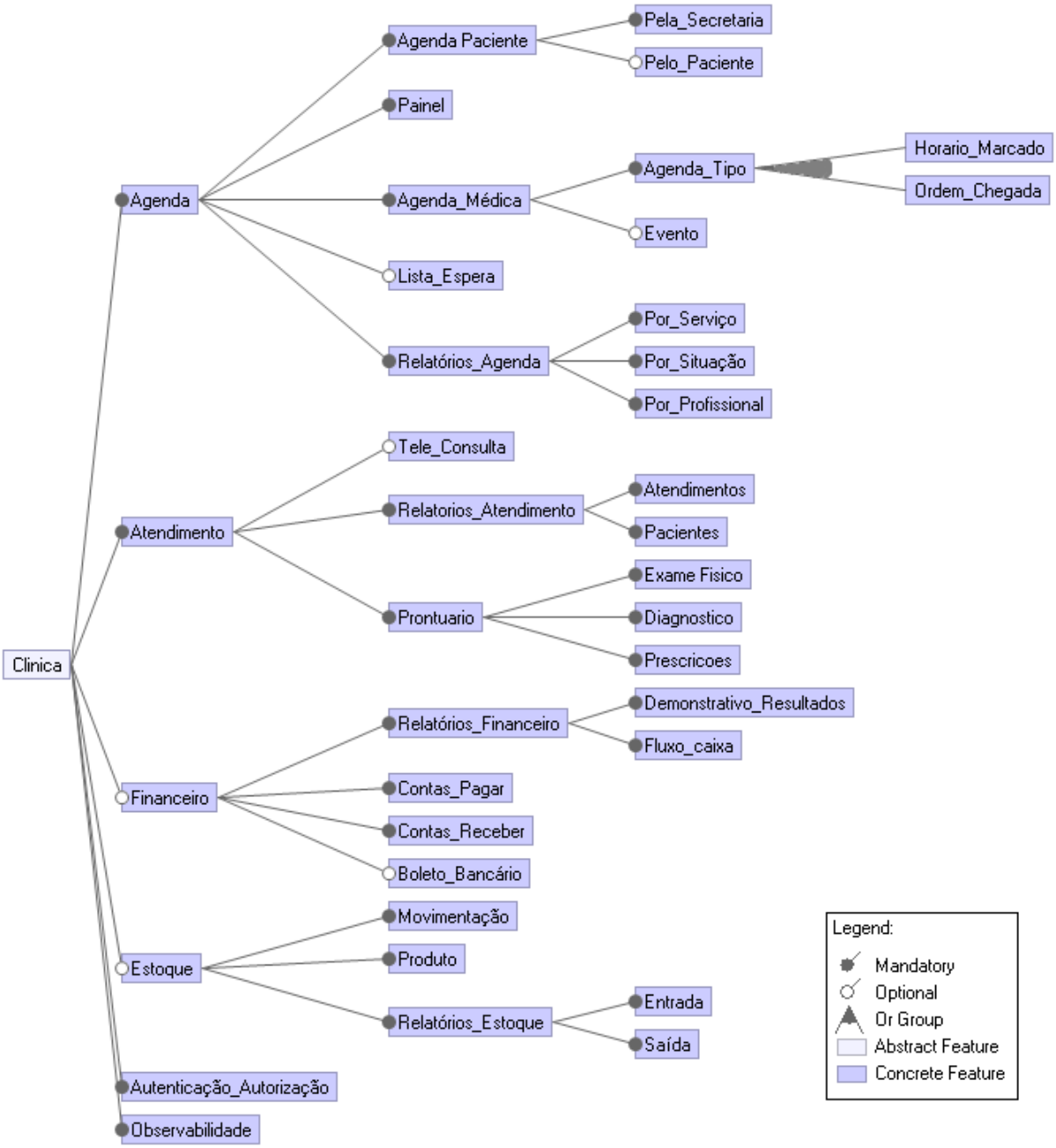

Figura 20 – Extrato do modelo de características da LPS

5.4 Instância da Arquitetura do Estudo de Caso

De modo a exemplificar uma aplicação das diretrizes descritas na Subseção 4.1, foi derivada a arquitetura da LPS de clínicas médicas, levando-se em consideração as características do subdomínio "agenda"representadas na Figura 20. Apresenta-se na Tabela 9, as aplicações *backend's* e os microsserviços identificados.

Tabela 9 – Exemplo da aplicação das diretrizes propostas no extrato do modelo de características do estudo de caso desta pesquisa.

Aplicações backend		**Microsserviços**	
Característica	**Diretriz**	**Característica**	**Diretriz**
Agenda	2	Lista Espera	4
Atendimento	2	Tele Consulta	4
Financeiro	2	Boleto Bancário	4
Estoque	2	Relatórios Agenda	2.b
		Relatórios Atendimento	2.b
		Relatórios Financeiro	2.b
		Autenticação Autorização	3.a
		Observabilidade	3.b

Observa-se que as características que não foram mapeadas para aplicações *backend* ou para microsserviço, devem ser mapeadas para *endpoints* de suas respectivas aplicações backend's.

Na Figura 21 apresenta-se a arquitetura desta LPS. De forma específica para o presente estudo de caso, diferentemente da arquitetura de referência da LPS, os *persons* são tipificados em secretário, médico, paciente e administrador do sistema. Os *containers* também receberam nomenclaturas específicas, conforme descrito a seguir:

- *frontend* (*containers* na cor vermelha): foi modelada uma SPA para atender as necessidades dos funcionários e médicos e outro para disponibilizar recursos de configuração e monitoramento da LPS aos administradores. Para atender as necessidades de interação com os pacientes foi modelado um adaptador *mobile*;
- *API's gateway* (*containers* na cor roxa): para atender a camada de *frontend*, foram modelados os seguintes BFF's: gestor, paciente e monitor.
- *Aplicações backend's* (*containers* na cor amarela): foi modelada uma aplicação *backend* agenda que contempla a implementação das características do subdomínio agenda;
- Microsserviços (MS) (*containers* na cor verde): nesta camada foram incluídos os seguintes microsserviços: MS Autenticação, MS Relatórios Agenda e MS Monitor. Supondo que a característica opcional Lista de Espera, descendente da característica

Agenda, tenha sido selecionada para compor o produto em questão, foi incluído o microsserviço MS Lista de Espera;

- Repositórios (*containers* na cor azul): na camada de repositórios foram incluídos bancos de dados para cada aplicação *backend* e para cada microsserviço. É importante destacar que a nomenclatura "1...N" representa a necessidade de instanciar N's repositórios de acordo com o número de locatários.

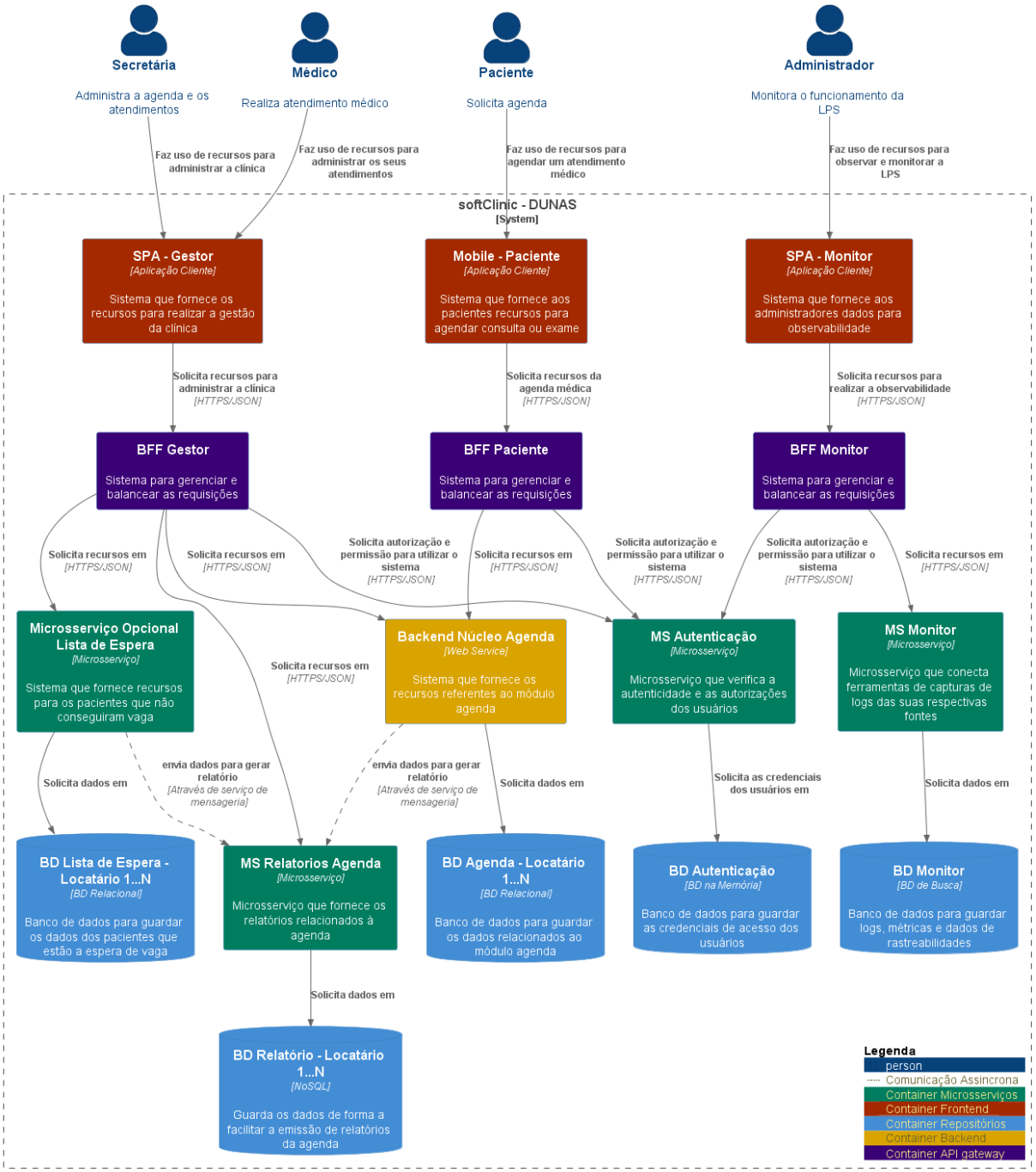

Figura 21 – Exemplo de arquitetura construída a partir das características apresentadas na Figura 20.

Destaca-se que foram implementados, pelo menos, um elemento representativo de cada um dos tipos descritos na seção 4.2. A seguir, as aplicações *backend* e os microsserviços arquitetados para este estudo de caso são apresentados.

5.4.1 API *Gateway*

Uma API *Gateway* foi desenvolvida para orquestrar os serviços implementados neste estudo de caso. Além disso, o balanceamento de carga também foi contemplado neste *Gateway*, que foi desenvolvido utilizando o *framework open source Spring Cloud*.

5.4.2 Aplicação *Backend* Agenda

Esta aplicação automatiza os processos de agendamento de procedimentos médicos de uma clínica, tais como registro de dias e horários disponíveis para cada profissional de saúde, agendamento de consulta e exames, além das configurações de lembretes de confirmação.

É importante destacar que as operações de consulta e relatórios desta aplicação são fornecidas pelo microsserviço de relatórios. Assim, por exemplo, cada vez que é agendada uma nova consulta médica, deverá haver uma comunicação entre a aplicação da agenda e o microsserviço de relatório, para que seja realizada a sincronização de dados entre as bases dos dois serviços. Este processo foi desenvolvido por meio de comunicação assíncrona, utilizando o *broker* de mensageria RabitMQ.

5.4.3 Microsserviço de Autenticação e Autorização

Este microsserviço controla as permissões dos usuários de cada locatário da LPS. Trata-se de um microsserviço obrigatório, pertencente a todos os produtos da LPS, que permite a criação de usuários e a definição de permissões para cada um destes, tornando possível o acesso ao sistema, de forma que cada usuário possa acessar somente os recursos do sistema que lhe forem autorizados.

5.4.4 Microsserviço de Lista de Espera

Este microsserviço foi escolhido para ser desenvolvido de modo a contemplar a implementação de uma variabilidade da LPS neste estudo de caso. O objetivo deste microsserviço é colher os dados de contato do paciente e formar uma fila de espera para, caso surja uma vaga, ele possa ser contatado.

5.5 Microsserviço de Relatórios

O microsserviço de relatório é responsável por fornecer os relatórios da aplicação *backend* agenda. Esta estratégia foi adotada para prover maior desempenho nas operações de consulta ao banco de dados. Dessa forma, os dados de agendamento são persistidos de forma consolidada por meio de repositório NoSQL baseado em documentos.

5.6 Definição do Modelo de Dados

Após a definição da instância da arquitetura do estudo de caso, levando-se em consideração um conjunto mínimo de aplicações que pudessem representar todos os tipos de containers projetados na arquitetura de referência da LPS, pode-se definir os projetos de banco de dados. A seguir são ilustrados os modelos de banco de dados da aplicação *backend* agenda, do microsserviço de autenticação e do microsserviço de lista de espera, respectivamente Figura 22, Figura 23 e Figura 24.

Observa-se na Figura 22 que a entidade principal é a entidade **pessoa**. Nesta entidade são armazenados os dados dos pacientes. Optou-se por utilizar pessoa em vez de paciente para aumentar o reúso e possibilitar a integração com os bancos de dados dos demais microsserviços. Visto que cada microsserviço que necessite dos cadastros dos atores do sistema (paciente, médico, secretária etc) devem referenciar um único identificador em todo o sistema. As demais entidades contemplam os dados necessários para atender ao subdomínio agenda, com exceção das características que devem ser segregadas para um microsserviço, como é o caso da característica Lista de Espera.

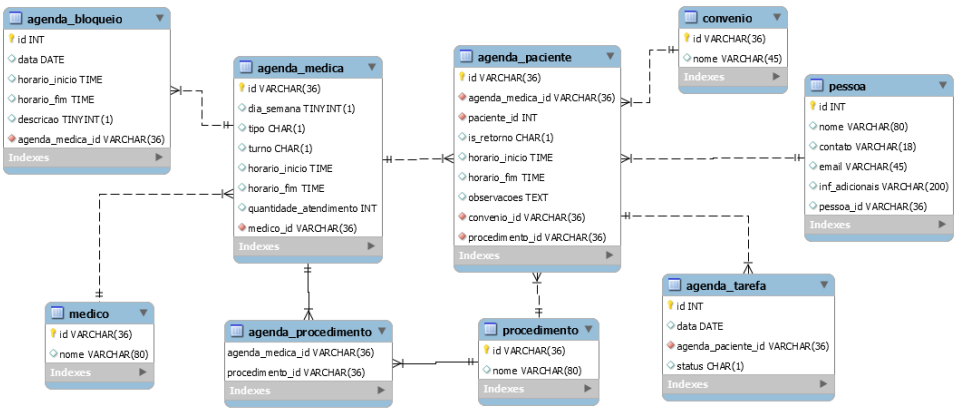

Figura 22 – Modelo do banco de dados da aplicação backend Agenda.

Na Figura 23 pode-se observar o modelo de dados para atender aos requisitos relacionados as autorizações de cada locatário de acordo com o conjunto de características variáveis que foi contratado. Assim, a entidade **Característica** registra todas as características opcionais e alternativas da LPS com objetivo de relacioná-las a cada locatário que é registrado na entidade **Clinica**. Dessa forma, os usuários (entidade **Usuário**) pertencentes

a cada **Clínica** pode possuir um grupo de permissões (entidade **Permissao**), que por sua vez, possui autorizações para um grupo de **Características**.

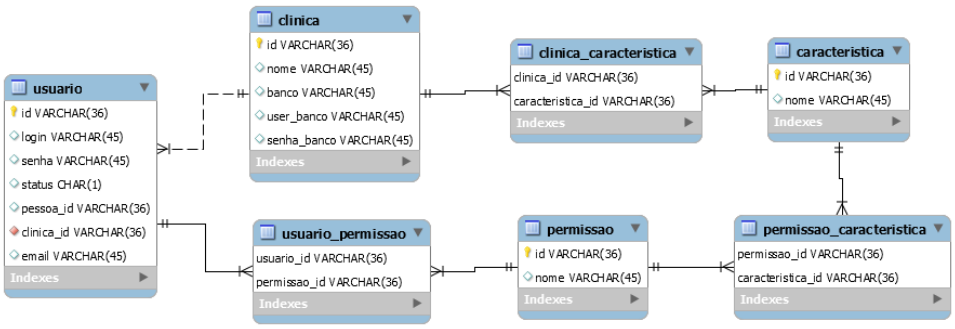

Figura 23 – Modelo do banco de dados do Microsserviço Autenticação.

Na Figura 24 representa-se o modelo de dados do microsserviço de **Lista de Espera**. Destaca-se que as entidades **paciente** e **medico** possuem dados simplificados, pois representam uma cópia dessas entidades que contém apenas os campos necessários para o contexto desse microsserviço. Vale a pena mencionar que os respectivos id's dessas entidades são referências únicas para todo o sistema.

Figura 24 – Modelo do banco de dados do Microsserviço Lista de Espera.

5.7 Implementação

Apesar da arquitetura de microsserviços permitir a heterogeneidade de tecnologias e padrões entre os microsserviços, todas as aplicações foram desenvolvidas com o *Framework* Java Spring Boot. Dessa forma, a estrutura do microsserviço de autenticação é descrita na Figura 25 como modelo das demais aplicações implementadas, pois este microsserviço conta com alguns pacotes relativos ao RabbitMQ, além dos que estão presentes nos demais microsserviços. A escolha dos pacotes e das camadas da arquitetura dos microsserviços segue alguns dos padrões discutidos por Robert (2017), tais como: *Entities, Controller, Data Transfer Object, Service, Repository.*

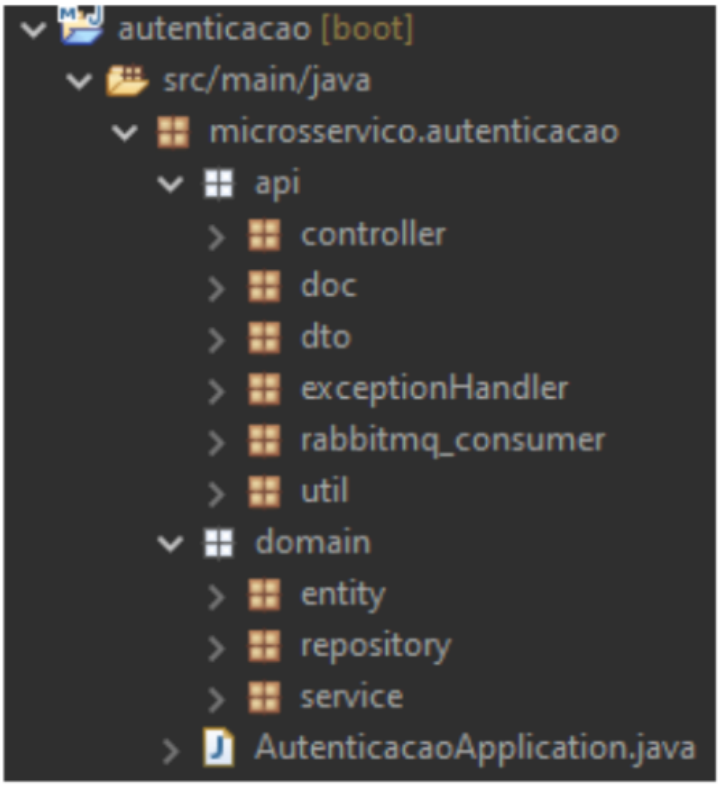

Figura 25 – Estrutura dos pacotes do microsserviço de Autenticação.

Na Figura 25 é apresentada a estrutura de pacotes do microsserviço de autenticação e suas respectivas camadas e subcamadas. As duas camadas principais são API e *domain*. A primeira abrange os pacotes e classes voltados à parte web da aplicação, ou seja, classes que não estão propriamente ligadas ao domínio da aplicação. A segunda divisão engloba as classes diretamente ligadas ao domínio da aplicação, ou seja, aquelas ligadas ao ramo de negócio do sistema.

No pacote ***domain*** estão contidos os seguintes subpacotes: pacote entity que representa as entidades do sistema, no caso do Spring Boot, também são definidas nestas classes as tabelas da base de dados. O pacote ***repository*** é responsável por prover acesso diretamente ao banco de dados. O pacote ***service*** tem como responsabilidade gerenciar as regras de negócios específicas do sistema.

No pacote **api** estão contidos os seguintes subpacotes: pacote ***controller***, que gerencia as rotas do sistema; o pacote ***doc*** engloba as classes de documentação das rotas do sistema. A ferramenta de documentação escolhida foi o Swagger [1]. O pacote **DTO** armazena as classes de transporte de dados, que implementam o padrão *Data Transfer Object* (DTO) e geralmente contém atributos de uma ou mais entidades. Normalmente o padrão DTO é usado para criar diversas visões do sistema. O pacote ***rabbitmq-consumer*** envelopa as classes de configuração do RabbitMQ. Por último, o pacote **util** engloba classes utilitárias para diversos fins genéricos; especificamente neste microsserviço, foi utilizado

[1] https://swagger.io/

para armazenar classes de criptografia.

5.7.1 Comunicação entre microsserviços utilizando fila de mensagens

A forma de comunicação escolhida foi a utilização de serviços de mensageria, pois trata-se de uma forma de comunicação assíncrona que se adapta melhor ao cenário distribuído. A utilização da comunicação no estilo REST prejudicaria a confiabilidade, uma vez que caso um serviço estivesse incapaz de receber a requisição HTTP, assim que ele voltasse a funcionar, continuaria a executar com a base de dados dessincronizada, uma vez que não haveria o reenvio da requisição responsável por atualizar do banco de dados.

Os benefícios da utilização da mensageria no ambiente distribuído contornam a situação descrita acima, pois a mensagem de atualização das bases de dados seria persistida, ficando disponível até que o microsserviço com conexão defeituosa voltasse a funcionar normalmente, permitindo que eventuais indisponibilidades não atrapalhassem o funcionamento do sistema como um todo.

Um fluxo de dados presente no sistema e que ilustra o uso de mensageria acontece quando o usuário do sistema cadastra um médico na API principal; a API principal então envia uma mensagem para o *broker* de mensageria, que armazena a mensagem no *buffer*. Quando o microsserviço de autenticação estiver conectado ao *broker* de mensageria, ele receberá a mensagem solicitando atualização da base de dados para que tanto a API principal quanto o serviço de autenticação fiquem com as bases de dados sincronizadas, como é exemplificado na Figura 26.

Na Figura 26 são apresentados quatro momentos. No primeiro momento, a máquina cliente realiza um POST comunicando-se com a API principal do sistema para criação de um médico. Um exemplo ocorre no contexto da atendente cadastrando um médico. No segundo momento, a API principal atualiza sua base de dados e envia uma mensagem para a fila de criação de usuários, gerenciada pelo *broker* RabbitMQ. No terceiro momento, o *broker* armazena a mensagem na fila. No quarto momento, o *broker* entrega a mensagem de criação do usuário (referente ao médico criado no primeiro momento) ao remetente, microsserviço de autenticação, para que as duas bases de dados fiquem sincronizadas, sem que haja comunicação direta entre a API principal e o microsserviço de autenticação, mantendo um baixo nível de acoplamento.

No exemplo apresentado anteriormente, a comunicação entre cliente e API principal utiliza o estilo REST. O método POST indica a criação de um objeto. Após a operação desejada ser realizada na base de dados do cliente, uma mensagem assíncrona solicitando atualização ou criação de dados será enviada ao *broker*. Assim que o *broker* receber a mensagem, estas serão enfileiradas e armazenadas em um *buffer*. Posteriormente, uma mensagem assíncrona é enviada ao microsserviço de autenticação para que a operação

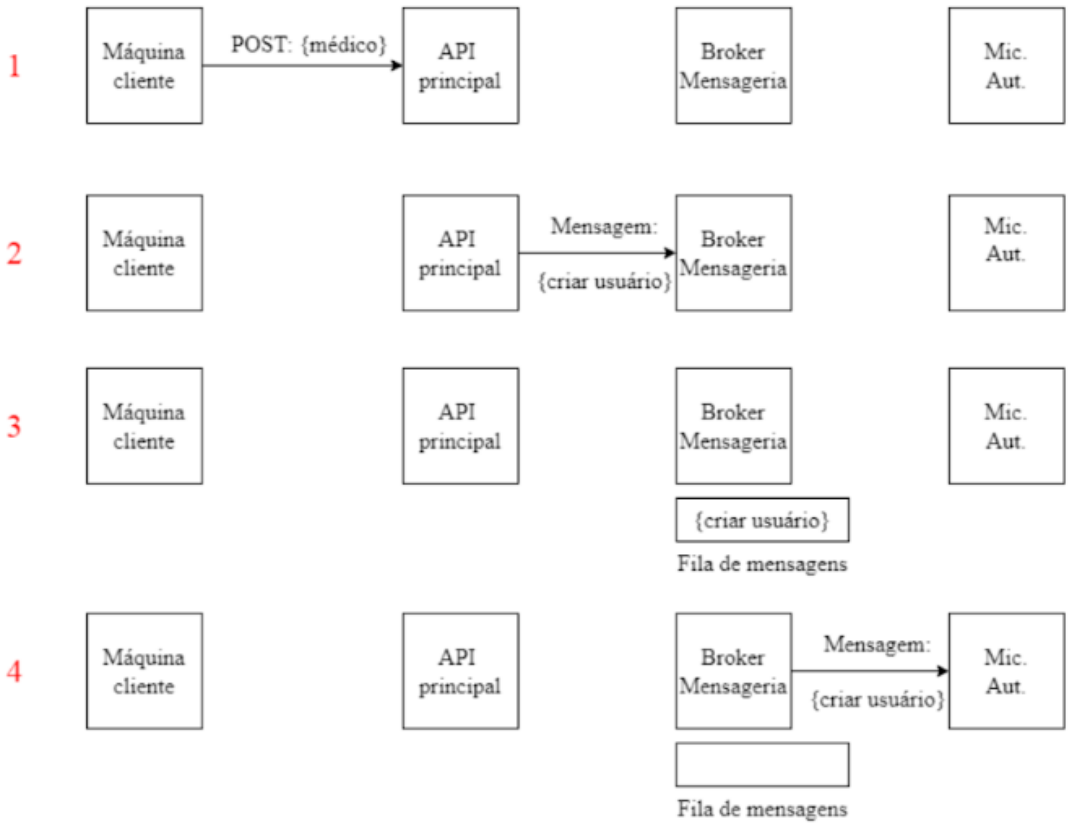

Figura 26 – Esquema que ilustra a comunicação de microsserviços por meio de mensageria.

armazenada no *buffer* seja executada na base de dados do serviço gerenciador de autenticação. A escolha de fila para gerenciar a comunicação ocorreu devido ao fato de existir apenas um destinatário das mensagens (microsserviço de autenticação), evitando a necessidade de utilizar comunicação por grupo ou em tópicos.

5.7.2 Conteinerização dos microsserviços

O objetivo de empacotar um microsserviço em um *container* é para facilitar o processo de implantação e replicação do mesmo, de forma que a sua execução fique independente do sistema operacional do servidor. Isto permite que a execução seja facilmente transportada de um servidor baseado em Linux para um outro servidor que execute um sistema operacional diferente, como o Windows. Além disso, o tamanho diminuto do contêiner facilita no ato de transferência entre servidores.

Devido ao fato de não existir uma imagem Docker [2] proprietária do Java ou do Spring Boot facilmente disponível no Docker Hub (repositório oficial de imagens Docker), a imagem Docker escolhida para rodar o JRE (ambiente de execução do Java) foi a eclipse-temurin em sua versão 17 montada sob a versão Alpine do Linux, conhecida por seu pequeno tamanho de armazenamento. A versão escolhida fornece suporte a versão 17

[2] https://www.docker.com/

do Java, utilizada no desenvolvimento dos microsserviços.

5.7.3 Orquestração de microsserviços

O *framework* Spring possui um conjunto de recursos para o desenvolvimento de sistemas distribuídos. Um desses recursos é o Spring Cloud, que é uma ferramenta utilizada para facilitar a implementação da orquestração dos microsserviços. Visto que, neste projeto, os microsserviços foram desenvolvidos com o Spring Boot, a escolha pelo Spring Cloud para implementar a orquestração dos microsserviços se deu pela facilidade apresentada ao se utilizar uma ferramenta nativa do ecosistema Spring. Além dessa facilidade de integrar com projetos Spring Boot, o *Spring Cloud Netflix* [3] foi utilizado para a implementação de *service discovery*. *Service discovery* é um padrão arquitetural para sistemas distribuídos em que os serviços registram suas informações em um registro central, permitindo que outros serviços os encontrem e se comuniquem com eles de maneira dinâmica.

O *Spring Cloud Netflix* pode trazer benefícios significativos para a construção de sistemas distribuídos escaláveis e resilientes, permitindo que as aplicações se comuniquem de maneira dinâmica, com facilidade de configuração e balanceamento de carga, além de uma melhor visibilidade sobre o estado dos serviços.

5.7.4 Testes

Esta pesquisa não tem o objetivo de atingir todas as etapas do desenvolvimento de software. Assim, os testes se resumiram à execução de chamadas aos *endpoints* disponibilizados pelo API *Gateway* e à verificação dos estados dos bancos de dados de cada serviço que foi implementado.

5.8 Comparação com a abordagem de Costa et al. (2019)

Para comparar a arquitetura proposta neste trabalho com uma abordagem que utilize apenas a arquitetura de microsserviços foi escolhida a abordagem proposta por Costa et al. (2019), pois atende aos critérios de comparação e é uma abordagem de fácil utilização.

A arquitetura ilustrada pela Figura 27 foi projetada com base no modelo de características do estudo de caso, conforme a Figura 20. Para este exemplo, levou-se em consideração apenas o subdomínio Agenda. Assim, de acordo com Costa et al. (2019), cada nó folha do modelo de características deve gerar um microsserviço. Devido ao aumento da quantidade de *containers*, optou-se por esconder as camadas relacionadas a *person* e a interface com o usuário, visto que essa mudança não interfere nos requisitos a

[3] https://cloud.spring.io/spring-cloud-netflix/reference/html/

serem analisados. As demais camadas permanecem com as mesmas padronizações que são descritas na Subseção 4.2.

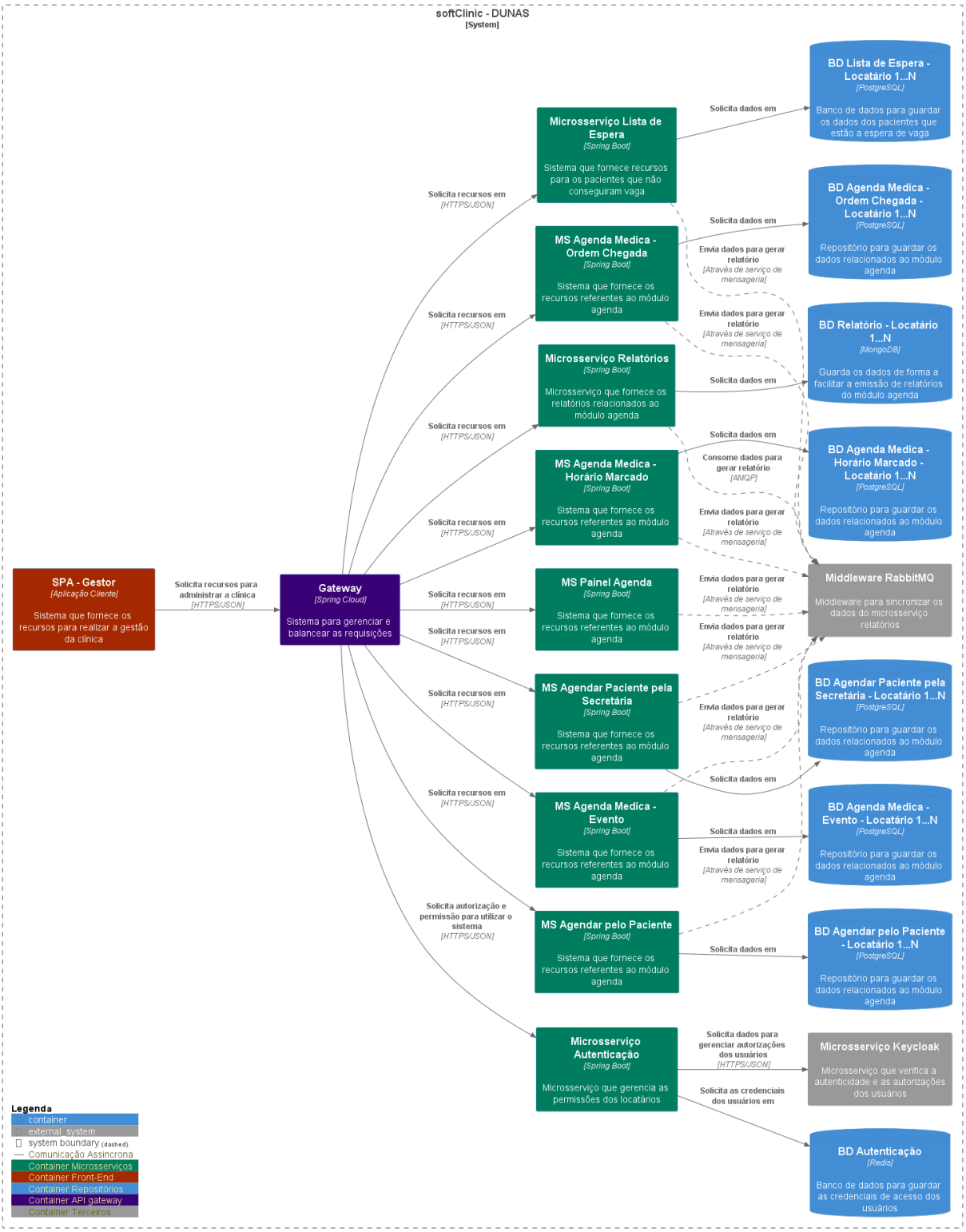

Figura 27 – Arquitetura de microsserviços do subdomínio Agenda da LPS.

Isto posto, destaca-se a complexidade originada pelo aumento de microsserviços para atender apenas a implementação de um subdomínio da LPS. Enquanto a abordagem proposta neste trabalho modela apenas uma aplicação *backend* e um microsserviço para

este subdomínio, a abordagem de Costa et al. (2019) sugere a implementação de nove microsserviços, como ilustrado na Figura 27. Essa complexidade gera dificuldades para sincronizar os dados comuns às características do subdomínio agenda, pois esses dados devem ser armazenados em até sete cópias, de acordo com o repositório de cada microsserviço. Outro ponto negativo dessa proposta é a dificuldade de atender aos relacionamentos entre as características para manter a independência entre os nove microsserviços arquitetados.

Diferentemente, esta pesquisa propôs implementar como microsserviços, pelo menos para a criação da primeira versão da LPS, apenas os pontos de variabilidades que venham a produzir dados além dos já produzidos pelas características do núcleo. Dessa forma, os repositórios pertencem a apenas um microsserviço e assim, a consistência poderá ser tratada com menos dificuldade.

5.9 Considerações Finais do Capítulo

Este estudo de caso contribuiu para o desenvolvimento da pesquisa, pois forneceu as condições necessárias para que as teorias, conhecimentos e conclusões oriundas desta pesquisa fossem aplicadas em um projeto real, com uma extensibilidade e complexidade suficientemente necessária para validar os resultados deste trabalho. Vale ressaltar a importância da equipe da empresa que concedeu o *case* para estudo de caso, pois o conhecimento do domínio do negócio contribuiu significativamente para o desempenho das atividades práticas deste trabalho.

Assim, foi desenvolvido um produto mínimo viável que forneceu uma API para gerenciar uma agenda médica e que funciona como um SaaS multilocatário. Esse produto foi construído seguindo as diretrizes propostas por esse trabalho e pode ser continuado para que sejam implementados os demais subdomínios da LPS.

6 Resultados do Experimento

Para avaliar as diretrizes propostas por esse trabalho, foi realizado um experimento controlado, envolvendo 13 participantes, sendo oito representantes da indústria e cinco da academia. O experimento buscou investigar três pontos: (i) assertividade na aplicação das diretrizes para realizar o mapeamento do modelo de características para componentes arquiteturais; (ii) conformidade com requisitos da arquitetura de microsserviços; e (iii) complexidade da arquitetura produzida. Para cada um desses pontos foi definido, no formulário do experimento, pelo menos uma questão. O método utilizado foi uma análise qualitativa e estatística a partir da transformação e quantificação dos dados coletados.

6.1 Perfil dos Participantes

A seleção dos sujeitos participantes do experimento foi realizada das seguintes formas: (i) os participantes da indústria foram selecionados entre egressos e alunos de mestrado em Ciência da Computação da UFERSA, que possuíam experiência na indústria como engenheiro/arquiteto de software; (ii) os representantes da academia foram selecionados entre os professores do IFCE, que tivessem conhecimento em LPS ou microsserviços. No primeiro caso, foram selecionados cinco participantes com, pelo menos, um ano de experiência na indústria. No segundo caso, foi selecionado um pesquisador com experiência em microsserviços e um pesquisador com experiência em LPS. Nos dois casos, utilizou-se o método *snowball* para selecionar os demais participantes.

Na Figura 28 pode-se perceber o perfil dos participantes em relação ao tempo de experiência, sendo possível observar que a maioria tem mais de três anos de experiência de atuação na indústria ou na academia. Ao analisar a correlação entre o tempo de experiência e a média de precisão na aplicação das diretrizes propostas por cada participante, observa-se que as menores médias estão entre aqueles com menos experiência, resultando em uma média de assertividade de 79,5%.

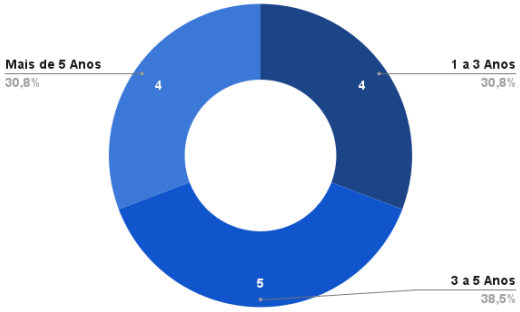

Figura 28 – Tempo de experiência dos participantes.

Na Figura 29 é apresentado um gráfico que classifica os participantes em quatro níveis de formação, sendo a maioria formada por graduados ou especialistas. Nesse ponto é relevante mencionar que essa maioria é devido aos estudantes de mestrado, que compõem o experimento por estarem na indústria.

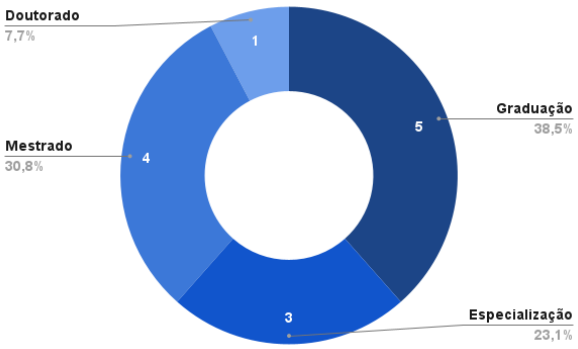

Figura 29 – Nível de formação acadêmica dos participantes.

Com relação aos conhecimentos e habilidades dos participantes, foram fornecidas três opções relevantes para esta pesquisa: LPS, microsserviços e sistemas distribuídos. Os participantes poderiam marcar mais de uma opção. Na Figura 30 é apresentado um gráfico que representa a distribuição dos participantes entre as técnicas mencionadas anteriormente. Neste gráfico, é possível observar que: (i) quatro participantes tinham conhecimento de LPS; (i) quatro participantes tinham conhecimento de Microsserviços; (ii) e apenas um participante tinha conhecimento de LPS e microsserviços. Destaca-se que cinco participantes não tinham conhecimento de LPS e de microsserviços.

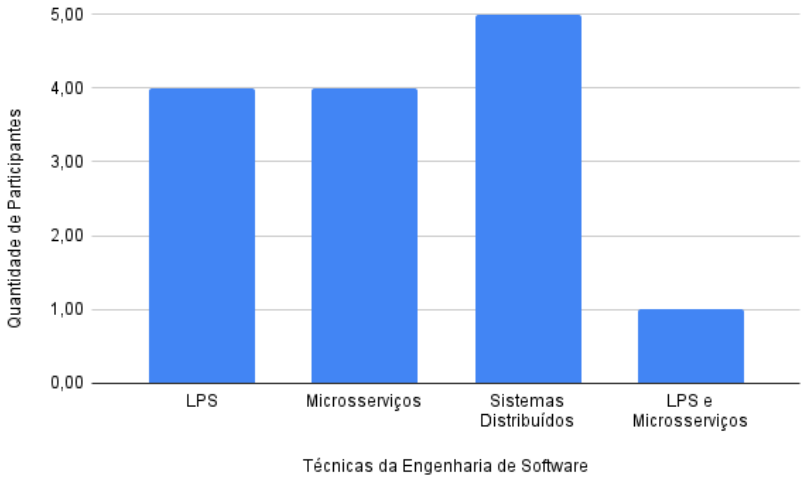

Figura 30 – Conhecimentos e habilidades dos participantes.

6.2 Instrumentalização e Aplicação

O experimento foi realizado de forma presencial no laboratório de informática da UFERSA para os participantes do perfil da indústria, enquanto para os participantes do perfil de pesquisador, optou-se por conduzi-lo remotamente. Essa decisão foi tomada devido à dificuldade de reunir os professores de diversos campi do IFCE em um único local físico. No entanto, é importante ressaltar que todos os participantes receberam os mesmos instrumentos, para garantir a igualdade de condições.

Foram criados dois instrumentos para a realização do experimento. O primeiro consistia em um documento contendo os fundamentos teóricos relacionados ao experimento, abrangendo conceitos de LPS, FODA, modelo de características e microsserviços - ANEXO A. Esse documento também incluía as diretrizes propostas pelo trabalho, um modelo de características e um exemplo de mapeamento do modelo para aplicações *backend* e microsserviços.

O segundo instrumento - ANEXO B, além de questões relacionadas ao perfil dos participantes, continha quatro questões principais:

1. Os participantes foram solicitados a preencher uma tabela com as aplicações *backend* e os microsserviços identificados a partir das diretrizes e do modelo de características fornecidos. Eles também deveriam relacioná-los às respectivas diretrizes que os originaram;

2. A segunda questão solicitava que os participantes apontassem falhas nas diretrizes e oferecessem sugestões para melhorá-las;

3. Na terceira questão, buscava-se a opinião dos participantes sobre a conformidade dos componentes arquiteturais identificados na primeira questão com a arquitetura de microsserviços;

4. A última questão tinha o objetivo de avaliar a complexidade de uma arquitetura híbrida, composta por aplicações *backend* e microsserviços.

Esses dois instrumentos permitiram uma abordagem completa para coletar dados relevantes e analisar os resultados de forma detalhada em relação ao experimento conduzido.

Antes da aplicação do experimento, foi realizada uma execução piloto para verificar possíveis problemas nos instrumentos e na coleta dos dados. Essa execução foi realizada com um analista de sistema que tem mais de 10 anos de atuação na indústria e atualmente trabalha em projetos de softwares relacionados a sistemas distribuídos e arquitetura de microsserviços. Após a execução piloto, foram realizadas pequenas alterações nas diretrizes, em especial, na ordem das diretrizes e no enunciado da terceira questão.

A aplicação do experimento seguiu-se com a leitura do instrumento teórico com os participantes, que depois ficaram livres por até 40 minutos para preencher os questionários.

6.2.1 Coleta e Análise dos Resultados

Buscando agrupar e classificar os dados coletados, nesta subseção são descritas as respostas às quatro questões:

Questão 1: Preencher a tabela com as aplicações backend's e microsserviços identificados a partir da aplicação das diretrizes no modelo de característica fornecido.

Para responder essa primeira pergunta foi fornecido o mesmo modelo de característica utilizado no estudo de caso desta pesquisa. Além disso, visando auxiliar na aplicação da diretriz 4, disponibilizamos um extrato de estória de usuário. Esse recurso foi fornecido com o propósito de servir de suporte durante a análise de características que possam exigir apenas campos ou atributos adicionais no modelo do domínio, bem como para identificar se seria necessário projetar novas entidades de domínio.

A média de acertos de cada participante foi calculada ao comparar o respectivo preenchimento da tabela com o gabarito produzido pelos autores desta pesquisa. Para se produzir uma média mais justa, buscou-se não apenas quantificar a quantidade de acertos entre as aplicações backend's e microsserviços, mas também quantificar os componentes mapeados erroneamente. A seguir, são apresentadas duas equações para calcular o valor da média de acertos de cada participante:

$$\textit{Equação 1 (E1): } MP = NA \;/\; (NA + NE)$$

$$\textit{Equação 2 (E2): } MP = NA \;/\; NC$$

Onde:

- **MP** representa o valor da média de acertos de cada participante;
- **NA** é a quantidade de acertos obtidos;
- Em **Equação 1 (E1)**, **NE** é a quantidade de erros;
- Em **Equação 2 (E2)**, **NC** é a quantidade máxima de componentes que deveriam ser mapeados para o exemplo fornecido.

A escolha entre as equações E1 e E2 é feita com base na quantidade de componentes identificados por cada participante. Se o resultado da soma de **NA** com **NE** for maior ou igual a **NC**, a Equação E1 é aplicada. Caso contrário, a Equação E2 é utilizada. Essas equações são amplamente utilizadas em *machine learning* para calcular uma métrica fundamental, conhecida como precisão (GOUTTE; GAUSSIER, 2005). Neste experimento, foi

aplicado tais equações com o objetivo de obter uma avaliação quantitativa do desempenho dos participantes.

Em seguida foi calculada a média aritmética entre todos os participantes, obtendo-se um valor médio de 86% de acertos e uma mediana próxima à média, com valor de 85% de acertos, significando que os valores extremos não tiveram um impacto relevante na média. A menor média de acerto foi 71% e a maior foi 100%.

Ao analisar as médias de acertos separadamente por grupo de participantes, indústria e academia, calcula-se 82% e 91%, respectivamente. Observa-se que essa diferença de 9% entre os grupos participantes pode ser atribuída ao fato de que os participantes da academia, além de possuírem maior experiência com LPS, também tinham maior nível de formação e mais tempo de experiência, o que resultou em uma média maior do que a do outro grupo. Todavia, o fato de uma nova abordagem ser experimentada por profissionais em início de carreira é relevante para que se possa produzir um trabalho capaz de ser aplicável por profissionais com diversos níveis de experiência.

Questão 2: Registre os pontos que precisam melhorar nas diretrizes para torná-las mais objetivas e claras quanto à identificação das aplicações *backend* e dos microsserviços a partir do modelo de características.

Para essa questão, a maioria relatou a necessidade de melhorar a Diretriz 4, como foi o caso de um dos participantes que disse o seguinte: "Os itens 4 e 4.a poderiam ser melhor detalhados para facilitar a decisão de quando uma característica deve ou não gerar um microsserviço, explicitando como identificar se a característica gera ou não um domínio para o item 4. Já o item 4.a poderia descrever como identificar se a característica representa apenas variações de campus/atributos." Essa resposta representa a maior parte do que foi respondido. Além disso, foi sugerido a criação de um fluxograma que representasse as diretrizes, de modo a facilitar seu entendimento e aplicação. Os demais participantes declaram não necessitar de melhorias nas diretrizes.

Ao analisar os erros de mapeamento, percebe-se claramente que a aplicação da Diretriz 4 foi a causa de 100% dos erros. Esse fato ocorre principalmente porque a correta aplicação dessa diretriz exige o conhecimento do domínio do negócio em que se pretende aplicar as diretrizes. Como não se pôde selecionar os participantes que atendessem a esse critério, foi apresentado um extrato de estórias de usuários para auxiliá-los, o que não foi suficiente para extinguir tais dúvidas. Todavia, acredita-se que a aplicação das diretrizes em um caso real, por um sujeito que entenda do domínio do negócio, seja mais assertiva do que os resultados do experimento.

Questão 3: Na sua visão, os componentes que foram identificados atendem aos requisitos da arquitetura de microsserviços, tais como: foram mapeados com base no domínio do negócio; são coesos e possuem baixo acoplamento; e podem funcionar de forma

independente? Justifique sua resposta.

Essa questão aborda a viabilidade de implementar a variabilidade de LPS utilizando microsserviços sem comprometer os princípios fundamentais dessa arquitetura. Nesse sentido, três participantes apontaram que os microsserviços referentes aos relatórios poderiam gerar um alto acoplamento, visto que os dados necessários para gerar os relatórios dependeriam de diversas aplicações. Corroborando com esse entendimento, um participante respondeu "sim"a essa questão com relação a produzir componentes coesos, porém relatou que os componentes arquiteturais poderiam ou não ser acoplados, dependendo da forma como eles fossem implementados.

As análises desses quatro participantes podem ser justificadas pelo fato de não terem sido apresentados a eles os fundamentos inerentes a cada uma das diretrizes propostas. Neste caso, o acoplamento é contornado por meio de redundância e agrupamento de dados no banco de dados específico de cada microsserviço de relatório, que devem ter seus dados alimentados por meio de comunicação assíncrona e serviços de mensageria. Dessa forma, qualquer relatório não precisaria integrar dados que estivessem além da sua própria base de dados, o que os torna independentes.

Os demais participantes, no caso 9, que representam a maioria, responderam "sim"a essa questão e justificaram, conforme pode ser exemplificado pelo único participante que possui conhecimento em conjunto de LPS e microsserviços: "Sim, concordo que os microsserviços identificados são componentes altamente específicos e independentes que oferecem a vantagem de reutilização de funcionalidades em diferentes aplicações. Além disso, eles podem ser substituídos por microsserviços mais atualizados e adequados dentro da própria aplicação. Essa modularidade dos microsserviços permite um maior grau de flexibilidade e escalabilidade, tornando mais fácil adaptar e evoluir o sistema ao longo do tempo.".

Questão 4: Os componentes arquiteturais mapeados pelas diretrizes devem produzir uma arquitetura híbrida ao unir aplicações backend's e microsserviços. Na sua opinião, essa arquitetura híbrida é menos complexa do que uma arquitetura puramente de microsserviços? Justifique sua resposta.

A Questão 4 buscou avaliar a complexidade da arquitetura produzida, da perspectiva de que uma arquitetura híbrida é menos complexa de manter e de desenvolver do que uma arquitetura puramente de microsserviços. As respostas a essa pergunta foram unânimes, pois todos responderam sim, tendo suas justificativas representadas pela seguinte resposta: "Sim, esta arquitetura é menos complexa e de fácil gerenciamento. Os microsserviços possuem inúmeros benefícios, mas seu gerenciamento sempre foi um problema, principalmente quando a aplicação escala muito de tamanho. O uso de API "monolíticas" para implementar aquilo que é comum dentro de um domínio, facilita o gerenciamento e os microsserviços, para os subdomínios opcionais, trazem a flexibilidade e

adaptabilidade necessária para a personalização e escalabilidade da aplicação.".

6.3 Ameaças à Validade do Experimento

Destaca-se que a principal ameaça à validade do experimento é o tamanho da amostra. Apesar disso, as amostras são heterogêneas já que os participantes representam ambientes diferentes de aplicação do conhecimento e têm diferentes tempos de experiência.

Para mitigar possíveis problemas com os questionários, foi conduzido um estudo piloto que nos permitiu realizar modificações na instrumentação utilizada. Isso garantiu a consistência e o entendimento do questionário antes de coletar as respostas. Outra ameaça enfrentada diz respeito ao domínio do modelo de características no qual foi aplicados o mapeamento. Para superar essa dificuldade, foi utilizado um exemplo no instrumento de fundamentação teórica proveniente de um domínio inquestionavelmente diferente do estudo de caso. Além disso, buscou-se representar nos dois modelos a aplicação de todas as diretrizes.

A experiência dos participantes pode influenciar nos resultados coletados, especialmente porque profissionais mais experientes tendem a possuir maior conhecimento. No entanto, para permitir uma maior generalização dos resultados, buscou-se uma heterogeneidade de perfis entre os participantes, o que incluiu diferenças no nível de conhecimento. Nesse sentido, a disparidade no conhecimento não foi considerada uma ameaça.

É importante mencionar que foi desafiador encontrar pessoas com conhecimento em microsserviços e LPS simultaneamente, o que pode ter limitado a representatividade de certos perfis. Contudo, a expectativa é que participantes com esse perfil tendem a entender melhor do assunto e consequentemente teriam menos dificuldades. Entendimento este que pode ser corroborado pelo único participante deste perfil que acertou 100% da aplicação das diretrizes.

Finalmente, no que diz respeito aos efeitos de fadiga, não foram considerados relevantes, uma vez que se estima que o tempo médio para responder ao formulário tenha sido de aproximadamente 45 minutos. Dessa forma, acredita-se que o tempo tenha sido razoável para minimizar tais efeitos.

6.4 Considerações Finais do Capítulo

Foi conduzido um experimento com 13 participantes, sendo cinco pesquisadores do IFCE e oito graduados e estudantes de mestrado em Ciência da Computação da UFERSA, com, pelo menos, um ano de experiência profissional. Destaca-se a heterogeneidade dos participantes em relação ao tempo de experiência, níveis de formação e conhecimento das

técnicas necessárias para o experimento. Além disso, os instrumentos foram refinados durante a execução do piloto.

Diante dos resultados coletados, verificou-se que a aplicação das diretrizes propostas por esse trabalho teve uma média de 86% de assertividade, e que os 14% de erros se devem apenas à aplicação da Diretriz 4, que exige um conhecimento mais refinado do domínio do negócio para que seja aplicada corretamente. Nesse sentido, acredita-se que o percentual de assertividade aumente em casos reais.

O experimento mostrou que as diretrizes produzem componentes arquiteturais coesos, de baixo acoplamento e independentes, conforme a resposta de 69% dos participantes. Acredita-se que os 31% dos demais participantes que afirmaram que os microsserviços de relatórios causam acoplamento na arquitetura o fizeram por desconhecer os fundamentos inerentes às diretrizes, conforme foi relatado anteriormente. Adicionalmente, todos os participantes declararam que a arquitetura híbrida produzida pela aplicação das diretrizes é menos complexa do que uma arquitetura composta somente de microsserviços.

Vale ressaltar outras contribuições advindas do experimento: a sugestão para a confecção de um fluxograma que represente a aplicação das diretrizes; adequação no texto de algumas diretrizes para facilitar a clareza do entendimento do leitor; e, por último, a definição de alguns conceitos mais rudimentares para melhor contextualização de alguns termos.

Finalmente, pôde-se concluir que as diretrizes propostas por este trabalho podem auxiliar no projeto de uma arquitetura híbrida para LPS, ao fornecer um guia para definição de componentes arquiteturais coesos e com baixo acoplamento, capazes de escalar e evoluir de forma independente. Assim, além das diretrizes produzirem componentes que atendem aos requisitos da arquitetura de microsserviços, fornecem uma arquitetura menos complexa em comparação com arquiteturas puramente de microsserviços.

7 Considerações Finais

Foi Conduzido um estudo usando um SaaS multilocatário do mundo real do domínio de clínica médica. Este estudo iniciou-se por meio de um estudo dos fundamentos relacionados a LPS e microsserviços, seguindo-se pela a realização de uma RSL e prosseguiu com uma análise do domínio através de cinco softwares já consolidados no mercado. A principal contribuição desse trabalho é um conjunto de diretrizes que tem como objetivo guiar os engenheiros de software no mapeamento entre o diagrama de características e a arquitetura de LPS. Como resultado da aplicação dessas diretrizes, destaca-se a elaboração de uma arquitetura de referência híbrida para LPS, na qual as variabilidades são implementadas por meio de microsserviços que são combinados com elementos de maior granularidade para respeitar o requisito de redução de complexidade de desenvolvimento.

Como forma de avaliar, qualitativamente, a arquitetura de referência proposta, ela foi analisada com base em critérios de qualidade definidos na literatura, de modo que suas vantagens e desvantagens foram elucidadas. Adicionalmente, como forma de aplicar e aprimorar as diretrizes propostas neste trabalho foi apresentado um estudo de caso do mundo real, no domínio de clínicas médicas. O estudo de caso permitiu mostrar como utilizar as diretrizes para derivar a arquitetura da LPS a partir do modelo de características. Além disso, por meio do estudo de caso, foi possível comparar a abordagem proposta neste trabalho com uma abordagem encontrada na literatura. A partir dessa comparação, verificou-se que as diretrizes propostas neste trabalho, auxiliam na definição de uma arquitetura capaz de ser implementada com menor complexidade.

Para validar as diretrizes propostas por este trabalho, foi realizado um experimento controlado com 13 participantes entre profissionais da indústria e pesquisadores. Os resultados mostraram uma média de 86% de assertividade na aplicação das diretrizes. A maioria dos participantes considerou que as diretrizes produziram componentes arquiteturais coesos e independentes. O experimento também trouxe sugestões, como a criação de um fluxograma que representasse a aplicação das diretrizes e algumas adequações no texto para facilitar a compreensão. Dessa forma, pôde-se concluir que as diretrizes podem auxiliar no projeto de uma arquitetura híbrida para LPS, que é composta de componentes coesos, com baixo acoplamento e menor complexidade em comparação com arquiteturas puramente de microsserviços.

Esta abordagem possui um impacto inovador com relação aos trabalhos encontrados na literatura, ao propor um conjunto de diretrizes capaz de produzir uma arquitetura híbrida, e assim proporcionar a união da simplicidade da arquitetura monolítica com as vantagens da arquitetura de sistemas distribuídos. Dessa forma, fatores que são

considerados pontos fracos em sistemas distribuídos podem ser atenuados, como é o caso da consistência dos dados e do desempenho. Essas características tornam esta proposta como uma alternativa viável, para se iniciar o projeto e desenvolvimento de um SaaS multilocatário. A medida que a LPS evolui, com o amadurecimento das equipes envolvidas e do próprio sistema, pode-se gradativamente ir diminuindo a granularidade das APIs. Nesse sentido, o impacto negativo da complexidade da arquitetura de microsserviços é suavizado e as decisões arquiteturais podem ser tomadas a posteriori com maior embasamento. Assim, é importante destacar a necessidade da arquitetura inicial ter sido planejada para possibilitar essa evolução. Isso foi alcançado ao utilizar princípios de DDD na modelagem de subdomínios coesos e com baixo acoplamento.

O conjunto de diretrizes descrito na Subseção 4.1 pode ser replicado em qualquer domínio para se produzir uma arquitetura para uma LPS que integre microsserviços e funcione como um SaaS multilocatário. Isso posto, esta abordagem passa a ter uma conotação generalista. A exceção está nas duas primeiras camadas da arquitetura, a camada de *frontend* e a camada de *API Gateway*, pois elas podem variar de acordo com as especificidades do domínio.

7.1 Ameaças à Validade Desta Pesquisa

Em relação aos possíveis riscos de validade desta pesquisa, destaca-se o fato da arquitetura de referência da LPS ter sido derivada a partir do estudo de um único domínio de aplicação. Como forma de mitigar esse risco, o domínio escolhido para esta pesquisa é amplo, e contempla subdomínios comuns a diversos domínios de aplicação.

A RSL também enfrentou algumas ameaças à validade. Primeiramente, ao excluir trabalhos que abordavam LPS integrada com *web services* e não com microsserviços, a fundamentação pode ter sido limitada. Para minimizar esse risco, optou-se por focar apenas em artigos que tratavam obrigatoriamente de LPS e microsserviços. Além disso, a possibilidade de existirem artigos não indexados nas bases de dados escolhidas foi mitigada pela busca manual em comunidades relacionadas a LPS. A natureza recente das pesquisas sobre arquitetura de microsserviços pode ter limitado a representatividade dos resultados em relação ao mercado, mas a inclusão de membros da indústria na equipe permitiu incorporar experiências do mundo real nas análises. Por fim, a falta de documentações completas nos estudos selecionados foi reconhecida como uma limitação, mas, como esta RSL é pioneira na temática, ela pode servir de base para discussões futuras e validações mais aprofundadas.

Com relação ao experimento, a principal ameaça à validade é o tamanho da amostra, apesar de suas heterogeneidades representando diferentes ambientes de aplicação e níveis de experiência dos participantes. Para mitigar problemas com os questionários, realizou-se

um estudo piloto, e exemplos de domínios diferentes foram utilizados para fundamentar o modelo de características. A heterogeneidade de perfis dos participantes buscou permitir maior generalização dos resultados. A dificuldade em encontrar participantes com conhecimento em microsserviços e LPS simultaneamente pode ter limitado a representatividade, mas o único participante com esse perfil obteve 100% de acerto nas diretrizes.

7.2 Trabalhos futuros

Esta pesquisa focou apenas no *backend*. Desta forma, trabalhos futuros relacionados à pesquisa e ao desenvolvimento de um *frontend* baseado em composição de componentes dinâmicos poderiam contribuir de forma complementar. Essa abordagem pode oferecer um potencial significativo para criar interfaces de usuário altamente flexíveis e adaptáveis. Além disso, é essencial explorar técnicas e ferramentas que facilitem a composição eficiente desses componentes dinâmicos, garantindo a reusabilidade e manutenção do código.

Por fim, como trabalhos futuros, destaca-se a importância de submeter esta abordagem a outros domínios para que se possa avaliar os seus efeitos e assim, aprovar ou refinar a abordagem, pois podem existir características específicas que justifiquem um método diferente para a concepção da arquitetura da LPS. Ademais, podem ser conduzidas pesquisas para propor abordagens sistemáticas para evoluir a arquitetura híbrida e identificar quando e como desmembrar partes das aplicações *backend* em microsserviços.

7.3 Artigos Submetidos

Nesta seção, é apresentada a lista de artigos submetidos a eventos e periódicos científicos. Os artigos listados a seguir abrangem as principais contribuições científicas desta pesquisa.

- Integration of Software Product Line with Microservices: A Systematic Review; Revista HOLOS - Qualis A1;
- Building Multitenant SaaS by Combining Software Product Lines and Microservices; ACM Transactions on Software Engineering and Methodology - Qualis A2;
- SPL integrated with Microservices: a hybrid architectural proposal for multitenant SaaS; Simpósio Brasileiro de Componentes, Arquiteturas e Reutilização de Software (SBCARS 2023) - Qualis B1.

Referências

APEL, S. et al. *Feature-oriented software product lines*. [S.l.]: Springer, 2016. Citado na página 19.

ASSUNÇÃO, W. K.; KRÜGER, J.; MENDONÇA, W. D. Variability management meets microservices: six challenges of re-engineering microservice-based webshops. In: *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*. [S.l.: s.n.], 2020. p. 1–6. Citado 6 vezes nas páginas 21, 22, 54, 55, 57 e 64.

BAŠKARADA, S.; NGUYEN, V.; KORONIOS, A. Architecting microservices: Practical opportunities and challenges. *Journal of Computer Information Systems*, Taylor & Francis, v. 60, n. 5, p. 428–436, 2020. Citado na página 23.

BECKER, A. M.; LUCRÉDIO, D. The impact of microservices on the evolution of a software product line. In: *Proceedings of the 14th Brazilian Symposium on Software Components, Architectures, and Reuse*. [S.l.: s.n.], 2020. p. 51–60. Citado 11 vezes nas páginas 14, 21, 33, 54, 56, 60, 61, 62, 63, 65 e 66.

BENNI, B. et al. Can microservice-based online-retailers be used as an spl? a study of six reference architectures. In: *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*. [S.l.: s.n.], 2020. p. 1–6. Citado 7 vezes nas páginas 21, 54, 55, 59, 61, 64 e 65.

BIOLCHINI, J. et al. Systematic review in software engineering. *System engineering and computer science department COPPE/UFRJ, Technical Report ES*, v. 679, n. 05, p. 45, 2005. Citado 4 vezes nas páginas 14, 45, 47 e 69.

BROWN, S. *The C4 model for visualising software architecture*. <https://c4model.com/>. (Accessed on 06/18/2023). Nenhuma citação no texto.

BROWN, S. *Just Enough Software Architecture: A Risk-Driven Approach*. New York, NY: Marshall Press, 2018. ISBN 978-0-9845277-5-5. Citado 2 vezes nas páginas 40 e 42.

CARVALHO, L. et al. Extraction of configurable and reusable microservices from legacy systems: An exploratory study. In: *Proceedings of the 23rd International Systems and Software Product Line Conference-Volume A*. [S.l.: s.n.], 2019. p. 26–31. Citado na página 69.

CHEN, L.; BABAR, M. A.; ZHANG, H. Towards an evidence-based understanding of electronic data sources. In: *14th International conference on evaluation and assessment in software engineering (EASE)*. [S.l.: s.n.], 2010. p. 1–4. Citado na página 48.

CLEMENTS, P.; NORTHROP, L. *Software product lines*. [S.l.]: Addison-Wesley Boston, 2002. Citado 3 vezes nas páginas 33, 34 e 36.

COHEN, S. *Product line state of the practice report*. [S.l.], 2002. Citado na página 33.

COLANZI, T. E. et al. A search-based approach for software product line design. In: *Proceedings of the 18th International Software Product Line Conference-Volume 1*. [S.l.: s.n.], 2014. p. 237–241. Citado 2 vezes nas páginas 56 e 66.

COSTA, A. C. L. et al. Microservice-oriented product line architecture design: An exploratory study. In: *Proceedings of the XIII Brazilian Symposium on Software Components, Architectures, and Reuse*. [S.l.: s.n.], 2019. p. 113–122. Citado 17 vezes nas páginas 13, 14, 21, 26, 54, 56, 58, 59, 60, 61, 62, 64, 65, 66, 74, 93 e 95.

COULOURIS, G. et al. *Sistemas Distribuídos-: Conceitos e Projeto*. [S.l.]: Bookman Editora, 2013. Citado na página 19.

DIAS, W. K. A. N.; SIRIWARDENA, P. *Microservices security in action*. [S.l.]: Simon and Schuster, 2020. Citado na página 73.

ECK, N. V.; WALTMAN, L. Software survey: Vosviewer, a computer program for bibliometric mapping. *scientometrics*, Akadémiai Kiadó, co-published with Springer Science+ Business Media BV ..., v. 84, n. 2, p. 523–538, 2010. Citado na página 52.

EVANS, E. *Domain-Driven Design Reference: Definitions and Pattern Summaries*. 1st edition. ed. [S.l.]: Addison-Wesley Professional, 2014. ISBN 978-0321125217. Citado na página 27.

GHOFRANI, J.; LÜBKE, D. Challenges of microservices architecture: A survey on the state of the practice. *ZEUS*, v. 2018, p. 1–8, 2018. Citado na página 22.

GOMAA, H. *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*. USA: Addison Wesley Longman Publishing Co., Inc., 2004. ISBN 0201775956. Citado na página 25.

GOUTTE, C.; GAUSSIER, E. A probabilistic interpretation of precision, recall and f-score, with implication for evaluation. In: SPRINGER. *European conference on information retrieval*. [S.l.], 2005. p. 345–359. Citado na página 100.

JAMSHIDI, P. et al. Microservices: The journey so far and challenges ahead. *IEEE Software*, IEEE, v. 35, n. 3, p. 24–35, 2018. Citado na página 22.

KANG, K. C. et al. *Feature-Oriented Domain Analysis (FODA) Feasibility Study*. [S.l.], 1990. 115–116 p. Citado 2 vezes nas páginas 27 e 34.

KITCHENHAM, B. Procedures for performing systematic reviews. *Keele, UK, Keele University*, v. 33, n. 2004, p. 1–26, 2004. Citado na página 69.

LEWIS, J.; FOWLER, M. *Microservices: a definition of this new architectural term*. 2014. <https://martinfowler.com/articles/microservices.html>. Accessed: 2023-05-07. Citado 3 vezes nas páginas 39, 72 e 74.

LI, B. et al. Enjoy your observability: an industrial survey of microservice tracing and analysis. *Empirical Software Engineering*, Springer, v. 27, p. 1–28, 2022. Citado na página 73.

LINDEN, F. J. Van der; SCHMID, K.; ROMMES, E. *Software product lines in action: the best industrial practice in product line engineering*. [S.l.]: Springer Science & Business Media, 2007. Citado na página 19.

LUZ, W. et al. An experience report on the adoption of microservices in three brazilian government institutions. In: *Proceedings of the XXXII Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2018. p. 32–41. Citado na página 21.

MARQUES, M. et al. Software product line evolution: A systematic literature review. *Information and Software Technology*, Elsevier, v. 105, p. 190–208, 2019. Citado na página 48.

MELL, P.; GRANCE, T. et al. The nist definition of cloud computing. Computer Security Division, Information Technology Laboratory, National . . . , 2011. Citado na página 20.

MENDONÇA, W. D. et al. Towards a microservices-based product line with multi-objective evolutionary algorithms. In: IEEE. *2020 IEEE Congress on Evolutionary Computation (CEC)*. [S.l.], 2020. p. 1–8. Citado 5 vezes nas páginas 58, 59, 61, 64 e 66.

NGUYEN, P. H. et al. Using microservices for non-intrusive customization of multi-tenant saas. In: *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. [S.l.: s.n.], 2019. p. 905–915. Citado na página 19.

PACHECO, V. F. *Microservice Patterns and Best Practices: Explore patterns like CQRS and event sourcing to create scalable, maintainable, and testable microservices*. [S.l.]: Packt Publishing Ltd, 2018. Citado na página 72.

POHL, K.; BÖCKLE, G.; LINDEN, F. V. D. *Software product line engineering: foundations, principles, and techniques*. [S.l.]: Springer, 2005. Citado 3 vezes nas páginas 33, 36 e 59.

POHL, K.; BOTTERWECK, G. *Product Line Engineering: Foundations, Principles and Techniques*. Berlin, Germany: Springer Science & Business Media, 2012. Citado na página 34.

QUEIROZ, P. G. G. *Uma abordagem de desenvolvimento de linha de produtos com uma arquitetura orientada a serviços*. Tese (Doutorado) — Universidade de São Paulo, 2009. Citado 4 vezes nas páginas 20, 25, 29 e 30.

RABISER, D. et al. Multi-purpose, multi-level feature modeling of large-scale industrial software systems. *Software & Systems Modeling*, Springer, v. 17, n. 3, p. 913–938, 2018. Citado na página 57.

REES, M. J. A feasible user story tool for agile software development? In: IEEE. *Ninth Asia-Pacific Software Engineering Conference, 2002.* [S.l.], 2002. p. 22–30. Citado na página 27.

RESEARCH, G. V. *Cloud Computing Market Size, Share & Trends Report, 2030*. 2022. <https://www.grandviewresearch.com/industry-analysis/cloud-computing-industry>. (Accessed on 06/14/2023). Citado na página 20.

RICHARDS, M.; FORD, N. *Fundamentals of software architecture: an engineering approach*. [S.l.]: O'Reilly Media, 2020. Citado na página 67.

ROBERT, M. *Clean architecture: a craftsman's guide to software structure and design*. [S.l.]: Prentice Hall, 2017. Citado na página 89.

SETYAUTAMI, M. R. et al. Variability management: re-engineering microservices with delta-oriented software product lines. In: *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A-Volume A*. [S.l.: s.n.], 2020. p. 1–6. Citado 11 vezes nas páginas 14, 21, 54, 57, 58, 60, 62, 63, 64, 65 e 72.

SETYAUTAMI, M. R. et al. A uml profile for delta-oriented programming to support software product line engineering. In: *Proceedings of the 20th International Systems and Software Product Line Conference*. [S.l.: s.n.], 2016. p. 45–49. Citado na página 63.

SEVERO, E. R. *Manual do Arquiteto de Software*. Brasil: Arquitetura de Software, 2021. Disponível em: <https://arquiteturadesoftware.online/>. Citado 3 vezes nas páginas 36, 37 e 40.

TAIBI, D.; LENARDUZZI, V.; PAHL, C. Processes, motivations, and issues for migrating to microservices architectures: An empirical investigation. *IEEE Cloud Computing*, IEEE, v. 4, n. 5, p. 22–32, 2017. Citado na página 21.

TIZZEI, L. P. et al. Using microservices and software product line engineering to support reuse of evolving multi-tenant saas. In: *Proceedings of the 21st International Systems and Software Product Line Conference-Volume A*. [S.l.: s.n.], 2017. p. 205–214. Citado 6 vezes nas páginas 54, 58, 61, 63, 65 e 66.

TRIPP, D. Pesquisa-ação: uma introdução metodológica. *Educação e pesquisa*, Faculdade de Educação, v. 31, n. 03, p. 443–466, 2005. Citado 3 vezes nas páginas 14, 25 e 26.

VARAJAO, J. Software development in disruptive times. *Commun. ACM*, Association for Computing Machinery, New York, NY, USA, v. 64, n. 10, p. 32–35, sep 2021. ISSN 0001-0782. Disponível em: <https://doi.org/10.1145/3453932>. Citado na página 19.

ZHOU, W. et al. Rest api design patterns for sdn northbound api. In: IEEE. *2014 28th international conference on advanced information networking and applications workshops*. [S.l.], 2014. p. 358–365. Citado na página 21.

ANEXO A - Fundamentação Teórica para os Participantes do Experimento

1. Linha de Produto de Software

Linha de Produto de Software (LPS) é uma técnica da engenharia de software que visa fornecer um catálogo de produtos que possuem um núcleo comum de funcionalidades, mas que cada produto é único por possuir alguma funcionalidade diferente dos demais. A técnica de Linha de Produtos busca promover um reúso de forma planejada e eficiente, além de oferecer métodos para gerenciar as variabilidades (POHL; BÖCKLE; LINDEN, 2005).

A técnica de LPS propõe que seja feita uma análise dos requisitos do domínio, e não apenas das necessidades de um único cliente. Essa análise pode ser feita por meio da modelagem dos recursos comuns (pertencentes a todos os produtos da linha) e variáveis (pertencentes a alguns produtos ou com funcionalidades adaptadas para cada produto) da LPS.

Uma das abordagens para se realizar a engenharia de domínio e produzir a modelagem de variabilidades da LPS baseia-se na identificação e classificação das características dos softwares. As características são elementos distintos que podem ser incluídas no produto, dependendo das necessidades do cliente. Por exemplo, em um sistema de gestão de clínicas, as características podem incluir a possibilidade de fazer agendamento de consultas de forma presencial ou online, a capacidade de gerenciar diferentes prontuários, o suporte a diferentes tipos de dispositivos, entre outros.

A modelagem de características, do inglês feature model, vem do método FODA. No método FODA, uma feature equivale a uma característica do sistema que é visível ao usuário final (KANG et al., 1990). Em síntese, o modelo de característica representa, em alto nível e de forma hierárquica, as características comuns e variáveis de uma LPS.

O método FODA possui uma notação para representar graficamente as características de uma LPS, e retrata o resultado do processo de análise de domínio. A notação FODA prevê três tipos de características: (I) obrigatória - representam as características que devem estar presentes em todos os membros da LPS; (II) opcional - características desse tipo podem ou não ser incluídas nos membros da LPS; (III) por fim, alternativa - são as características que representam mais de uma forma de realizar uma determinada operação e, portanto, permitem a possibilidade de se escolher múltiplas destas características para serem inclusas em um determinado membro da LPS.

Na Figura abaixo ilustra-se um modelo de característica. Neste modelo, cada característica é representada por um retângulo. As características são organizadas de forma hierárquica, sendo o primeiro nível sempre representado pelo domínio da LPS, exemplificada na Figura 1 como **Escola**. Os demais níveis são formados pelas características descendentes da característica de primeiro nível. Assim, o último nível de cada ramificação é formado pelas características folhas. As arestas com círculo preenchido em preto refletem as características obrigatórias. Já as arestas com círculo vazado simbolizam as características opcionais. As características alternativas são representadas por arcos, que podem ser vazados ou preenchidos. Os arcos vazados limitam a escolha de apenas uma característica dentre as alternativas. Já os arcos preenchidos representam a possibilidade de se escolher mais de uma alternativa.

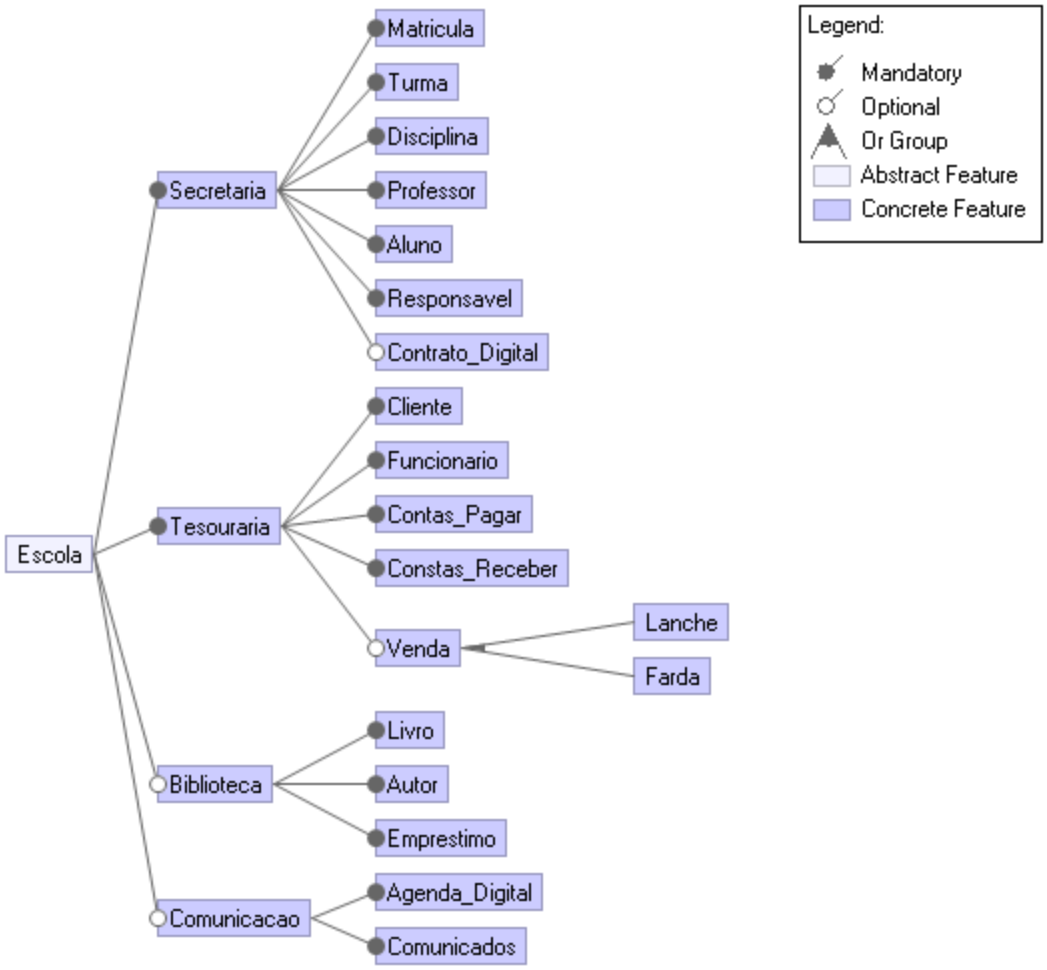

Figura 1: Exemplo do modelo de características.

A ideia é que o software seja construído a partir do reúso de pequenas partes que podem ser combinadas de várias formas para que sejam gerados diversos produtos e assim poder atender a uma maior diversidade de clientes do mesmo domínio de negócio.

Ultimamente tem-se pesquisado o uso de microsserviços para se construir um software dessa forma. Isso é possível devido os microsserviços serem conceituados como pequenas partes de um software, que funcionam de forma independentes e podem facilmente ser integrados por meio de diversos protocolos de comunicação.

2. Arquitetura de Microsserviços

Neste contexto, a arquitetura de microsserviços é uma das principais ferramentas que podem contribuir para uma solução distribuída. Segundo Lewis e Fowler (2014), a arquitetura de microsserviços é uma técnica que particiona o software em um conjunto de pequenos serviços. Ainda segundo Lewis e Fowler (2014), os microsserviço possuem os seguintes aspectos: (i) devem ser projetados conforme as regras de negócio, levando-se em consideração fatores como coesão e acoplamento; (ii) podem funcionar de forma heterogênea no que se refere à infraestrutura, tecnologia e equipe de desenvolvimento; (iii) possuem seu próprio processo e devem ser projetados para funcionar de forma independente, e assim, proporcionarem uma escalabilidade mais inteligente; (iv) e, a comunicação entre eles ocorre por meio de protocolos leves, como por exemplo, por uma Application Program Interface (API), e Hypertext Transfer Protocol (HTTP).

A adoção de microsserviços em projetos de software não é trivial, pois a complexidade associada a essa arquitetura exige o conhecimento de fatores e técnicas importantes, tais como: gerenciamento da consistência dos dados; repositórios de leitura; banco de dados não compartilhado; ferramentas para identificação de falhas e erros; comunicação assíncrona por meio de técnicas de mensageria.

Este experimento consiste em receber um modelo de características e, a partir das diretrizes apresentadas a seguir, identificar as aplicações back-end’s e os microsserviços que juntos formarão a arquitetura híbrida da LPS.

3. Diretrizes para mapeamento do modelo de características para a arquitetura de referência da LPS

1. O modelo de características foi construído levando-se em consideração os subdomínios da LPS;
    a. Cada subdomínio foi mapeado para uma característica de segundo nível;
    b. Em seguida, as características dos demais níveis foram organizadas em seus respectivos subdomínios;
2. Cada característica de segundo nível, sendo obrigatória ou opcional, deve gerar uma aplicação back-end (API);
    a. As características obrigatórias descendentes devem ser mapeadas para endpoints das suas respectivas aplicações.
    b. Para cada característica de segundo nível, deve ser gerado um microsserviço para prover os relatórios relacionados a todas as respectivas características descendentes.
3. A exceção para o item anterior ocorre nas características que não estão relacionadas ao domínio do negócio e que são de utilidade para todo o sistema, como é o caso de autenticação e observabilidade. Nesses casos, recomenda-se o seguinte:
    a. Projetar um microsserviço para prover os recursos relacionados a autenticação e autorização. Esse serviço deve agrupar os dados de acesso e de permissões de todos os locatários. Dessa forma, esse microsserviço deve gerenciar as configurações de acesso aos recursos opcionais e alternativos da LPS.
    b. Projetar um microsserviço para prover recursos relacionados à observabilidade da LPS.
4. Com relação às características opcionais e alternativas pertencentes aos demais níveis:
    a. que geram entidades de domínio do negócio devem gerar microsserviços;
    b. que representem apenas variações nos campos/atributos das entidades de domínio, **não** devem gerar microsserviços;
    c. caso tais características representem apenas variações nos campos/atributos das entidades de domínio já existentes, elas devem ser administradas por meio de parâmetros nos endpoints e não devem gerar microsserviços.

4. Exemplo da Aplicação das Diretrizes em uma LPS do Domínio de escolas.

Na Figura 2 apresenta-se um modelo de característica para uma LPS do domínio de escolas a ser utilizado como exemplo da aplicação das diretrizes descritas na seção 3. Ainda com essa mesma finalidade, na Tabela 1 são descritas algumas histórias de usuários para auxiliar na aplicação da diretriz 4.a.

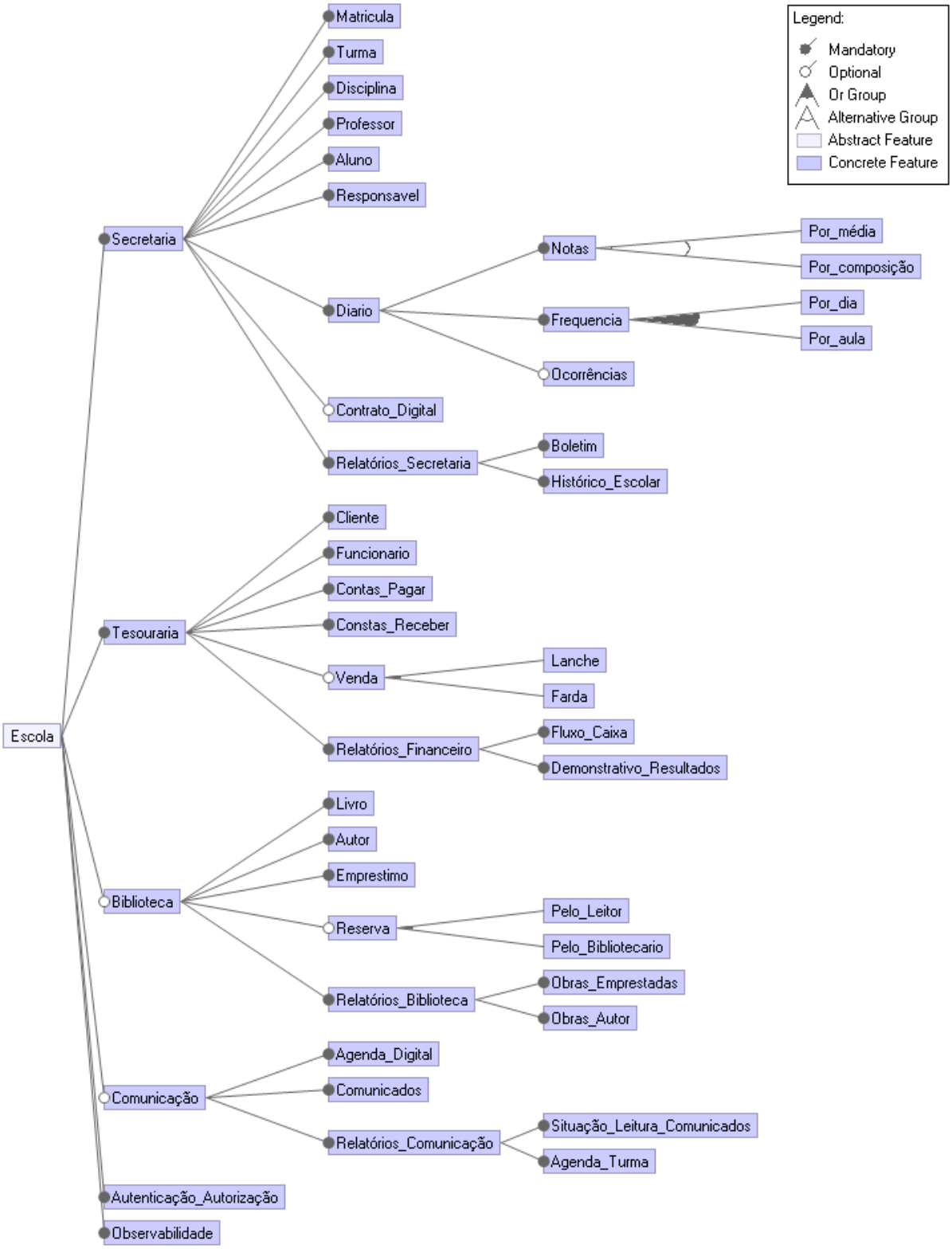

Figura 2: Modelo de características de uma LPS para escolas.

Tabela 1: Extrato de histórias de usuários de uma LPS de escolas.

Contrato Digital Como secretária escolar, quero celebrar contratos de matrícula de forma digital para facilitar a burocracia de registros em cartório e poder consultar de forma mais rápida os registros de contrato, inclusive mantendo um histórico de quem acessou os dados de um contrato.
Ocorrências Como coordenador pedagógico, quero poder manter um registro de ocorrências no decorrer do ano letivo para poder nas reuniões pedagógicas apresentar aos pais e também para direcionarmos medidas de correção do ensino. Essas ocorrências podem ser originadas de bom ou mau comportamento dos alunos.
Venda Como operador de caixa, quero poder registrar as vendas de lanches e fardas para poder administrar melhor as contas a receber.
Reserva Como bibliotecária, quero poder registrar no sistema reservas das obras da nossa biblioteca para que essas obras tenham um melhor uso e os usuários tenham mais chances de conseguir fazer um empréstimo.

Aplicando-se as diretrizes descritas na Seção 3, tomando como base o modelo de características apresentada na Figura 2 e as histórias de usuários descritas na Tabela 1, temos como resultado os componentes apresentados na Tabela 2.

Tabela 2: Mapeamento do Modelo de Características para Componentes Arquiteturais da LPS

Aplicações Back-end		**Microsserviços**	
Característica	**Diretriz**	**Característica**	**Diretriz**
Secretaria	2	Ocorrências	4.a
Tesouraria	2	Contrato_Digital	4.a
Biblioteca	2	Venda	4.a
Comunicação	2	Reserva	4.a
		Relatórios_Secretaria	2.b
		Relatórios_Tesouraria	2.b
		Relatórios_Biblioteca	2.b
		Relatórios_Comunicação	2.b
		Autenticação_Autorização	3.a
		Observabilidade	3.b

Observa-se que uma entidade de domínio do negócio é uma representação abstrata de um objeto ou conceito do mundo real, que possui atributos e relacionamentos.

5. Referências

- POHL, K.; BÖCKLE, G.; LINDEN, F. V. D. Software product line engineering: foundations, principles, and techniques. [S.l.]: Springer, 2005. Citado 3 vezes nas páginas 29, 32 e 55.
- POHL, K.; BOTTERWECK, G. Product Line Engineering: Foundations, Principles and Techniques. Berlin, Germany: Springer Science & Business Media, 2012. Citado na página 30.
- CLEMENTS, P.; NORTHROP, L. Software product lines. [S.l.]: Addison-Wesley Boston, 2002. Citado 3 vezes nas páginas 29, 30 e 32.
- KANG, K. C. et al. Feature-oriented domain analysis (FODA) feasibility study. [S.l.], 1990. Citado 2 vezes nas páginas 23 e 30.
- JUNIOR, E. R. S. Manual do Arquiteto de Software. Brasil: Arquitetura de Software, 2021. Disponível em: <https://arquiteturadesoftware.online/>. Citado 3 vezes nas páginas 32, 33 e 36.
- LEWIS, J.; FOWLER, M. Microservices: a definition of this new architectural term. 2014. <https://martinfowler.com/articles/microservices.html>. Accessed: 2023-05-07. Citado na página 34.

ANEXO B - Formulário de Aplicação do Experimento

Prezado(a),

Solicitamos a sua valiosa participação voluntária nesta pesquisa de mestrado acadêmico em ciência da computação, na área de engenharia de software.

O objetivo principal é analisar as diretrizes propostas por este trabalho de pesquisa com o propósito de avaliar com respeito à eficácia do ponto de vista da academia e da indústria no contexto de pesquisadores e profissionais ao aplicarem as diretrizes para mapear o modelo de característica para aplicações back-end e microsserviços.

Os dados coletados serão estritamente utilizados para fins acadêmicos, na elaboração da dissertação de mestrado do aluno Manoel Marisergio Alves de Oliveira da UFERSA, sob a orientação do Prof. Dr. Paulo Gabriel Gadelha Queiroz.

Ao preencher este formulário, você estará concordando em participar desta pesquisa. Salientamos que a sua participação é de extrema importância e contribuirá para o aprimoramento desta pesquisa.

Agradecemos pela sua disponibilidade.

Nome completo:	Idade:
Empresa/Instituição de afiliação:	
Cargo/Ocupação:	

Experiência e Formação:

1. Tempo de experiência na área de engenharia de software:
    a. Menos de 1 ano
    b. 1-3 anos
    c. 3-5 anos
    d. Mais de 5 anos

2. Nível de formação acadêmica:
    a. Graduação
    b. Pós-graduação (especificar o nível): _____
    c. Outro (especificar): _____

3. Informe o ano de formação, curso e área de especialização (se aplicável):

Experiência Prévia em Experimentos:

4. Participação prévia em experimentos científicos ou pesquisas?
    a. Sim
    b. Não

Conhecimento e Habilidades Técnicas:

5. Conhecimento de metodologias e Técnicas de engenharia de software:
    a. Scrum
    b. DevOps
    c. Linha de Produto de Software
    d. Sistemas Distribuídos
    e. Arquitetura de Microsserviços
    f. Arquitetura de Software
    g. Outras (especificar): _____

Participação no Experimento:

6. Concorda em participar voluntariamente do experimento?
    a. Sim
    b. Não
7. Está ciente dos objetivos e procedimentos do experimento?
    a. Sim
    b. Não
8. Possui alguma restrição física ou de saúde que possa afetar sua participação?
    a. Sim (especificar)
    b. Não

1. Diretrizes para mapeamento do modelo de características para a arquitetura de referência da LPS

1. O modelo de características foi construído levando-se em consideração os subdomínios da LPS;
    a. Cada subdomínio foi mapeado para uma característica de segundo nível;
    b. Em seguida, as características dos demais níveis foram organizadas em seus respectivos subdomínios;
2. Cada característica de segundo nível, sendo obrigatória ou opcional, deve gerar uma aplicação back-end (API);
    a. As características obrigatórias descendentes devem ser mapeadas para endpoints das suas respectivas aplicações.
    b. Para cada característica de segundo nível, deve ser gerado um microsserviço para prover os relatórios relacionados a todas as respectivas características descendentes.
3. A exceção para o item anterior ocorre nas características que não estão relacionadas ao domínio do negócio e que são de utilidade para todo o sistema, como é o caso de autenticação e observabilidade. Nesses casos, recomenda-se o seguinte:
    a. Projetar um microsserviço para prover os recursos relacionados a autenticação e autorização. Esse serviço deve agrupar os dados de acesso e de permissões de todos os locatários. Dessa forma, esse microsserviço deve gerenciar as configurações de acesso aos recursos opcionais e alternativos da LPS.
    b. Projetar um microsserviço para prover recursos relacionados à observabilidade da LPS.
4. Com relação às características opcionais e alternativas pertencentes aos demais níveis:
    a. que geram entidades de domínio do negócio devem gerar microsserviços;
    b. que representem apenas variações nos campos/atributos das entidades de domínio, **não** devem gerar microsserviços;
    c. caso tais características representem apenas variações nos campos/atributos das entidades de domínio já existentes, elas devem ser administradas por meio de parâmetros nos endpoints e não devem gerar microsserviços.

2. Modelo de Característica

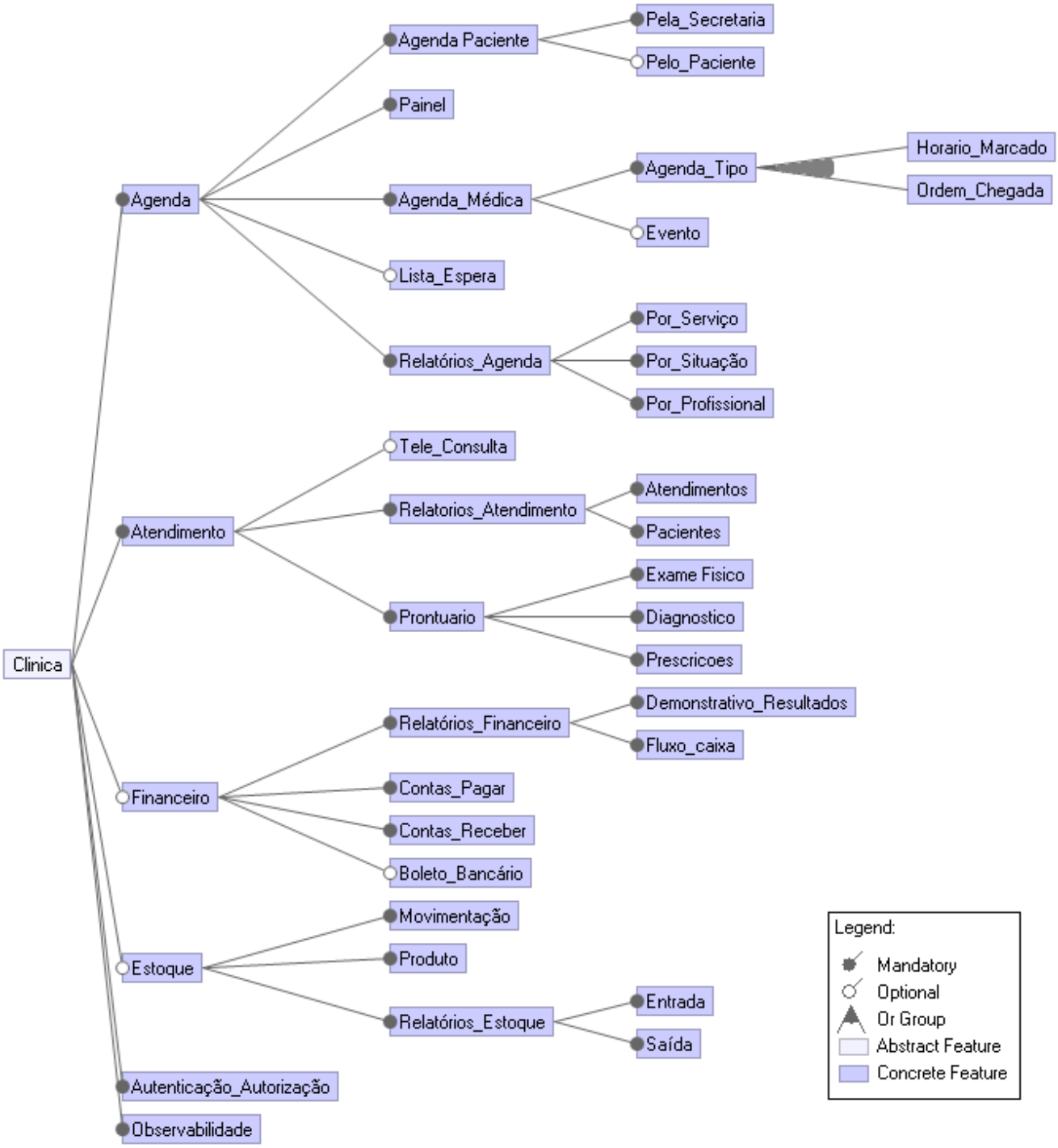

Figura 1: Extrato do Modelo de Característica de uma LPS para Clínica Médica.

3. Histórias de Usuário

Utilize a Tabela 1 para auxiliar na decisão de aplicar a diretriz 4.

Tabela 1: Extrato de Histórias de Usuários de uma LPS de Clínicas Médicas.

Histórias de Usuário
Agendar Atendimento pela secretária Como secretária, quero agendar, confirmar ou cancelar um atendimento médico para que os atendimentos possam ser organizados.
Agendar Atendimento pelo paciente Como paciente, quero agendar atendimento médico para que eu possa ter maior facilidade e agilidade.
Cadastro de Evento (falta de um profissional, manutenção de equipamento etc) Como secretário, quero poder inserir um evento de impedimento da agenda médica para que não seja agendado um horário que apesar de ter sido configurado para atendimento, mas que aconteceu um imprevisto.
Lista de espera (encaixe) Como secretário, quero gerenciar uma lista de pacientes que esperam vaga de atendimento, para que eu possa encaixá-los em caso de desistência dos pacientes já agendados.

4. Item Avaliativo:

4.1. Preencha a tabela abaixo para realizar o mapeamento do modelo de característica descrito na Figura 1, aplicando as diretrizes descritas na seção 1.

Tabela 2: Mapeamento do Modelo de Características para Componentes Arquiteturais da LPS

Aplicações Back-end		**Microsserviços**	
Característica	**Diretriz**	**Característica**	**Diretriz**

4.2. Registre os pontos que precisam melhorar nas diretrizes para torná-las mais objetivas e claras quanto a identificação das aplicações back-end e dos microsserviços a partir do modelo de características.

4.3. Na sua visão, os componentes que foram identificados atendem aos requisitos da arquitetura de microsserviços, tais como: foram mapeados com base no domínio do negócio; são coesos e possuem baixo acoplamento; e podem funcionar de forma independente? Justifique sua resposta.

4.4. Os componentes arquiteturais mapeados pelas diretrizes devem produzir uma arquitetura híbrida ao unir aplicações back-end's e microsserviços. Na sua opinião, essa arquitetura híbrida é menos complexa do que uma arquitetura puramente de microsserviços. Justifique sua resposta.