UNISINOS

Programa de Pós-Graduação em

Computação Aplicada

Mestrado Acadêmico

Maicon Azevedo da Luz

EVENTCHAIN: Uma proposta de estilo arquitetural para sistemas orientados a cadeia de eventos na área financeira

São Leopoldo, 2021

Maicon Azevedo da Luz

EVENTCHAIN:
Uma proposta de estilo arquitetural para sistemas orientados a cadeia de eventos na área financeira

Dissertação apresentada como requisito parcial para a obtenção do título de Mestre pelo Programa de Pós-Graduação em Computação Aplicada da Universidade do Vale do Rio dos Sinos — UNISINOS

Orientador:
Prof. Dr. Kleinner Silva Farias de Oliveira

São Leopoldo
2021

L979e Luz, Maicon Azevedo da

EventChain: uma proposta de estilo arquitetural para sistemas orientados a cadeia de eventos na área financeira / Maicon Azevedo da Luz — 2021.

123 f.: il.; 30 cm.

Dissertação (mestrado) — Universidade do Vale do Rio dos Sinos, Programa de Pós-Graduação em Computação Aplicada, São Leopoldo, 2021.

"Orientador: Prof. Dr. Kleinner Silva Farias de Oliveira, Unidade Acadêmica de Pesquisa e Pós-Graduação".

1. Arquitetura de Software. 2. Blockchain. 3. Microsserviços. 4. Estilo Arquitetural de Software. 5. Orientado a Eventos. I. Título.

CDU 004

Dados Internacionais de Catalogação na Publicação (CIP)
(Bibliotecária: Silvana Dornelles Studzinski – CRB 10/2524)

(Esta folha serve somente para guardar o lugar da verdadeira folha de aprovação, que é obtida após a defesa do trabalho. Este item é obrigatório, exceto no caso de TCCs.)

AGRADECIMENTOS

Agradeço aos meus pais, Neroi e Nicolau por todo o apoio, incentivo e educação que me deram ao longo da minha caminhada. A minha irmã Nicoli pelo incentivo na reta final de conclusão dessa dissertação.

Gostaria de agradecer ao Programa de Pós-Graduação em Computação Aplicada da Unisinos, pela oportunidade e por todo o conhecimento compartilhado.

Um agradecimento mais que especial ao meu orientador Prof. Kleinner Farias, que acreditou no meu trabalho, teve muita paciência, me incentivou em vários momentos e compartilhou como sempre muito do seu conhecimento.

E a minha esposa Camila que esteve ao meu lado em todos os momentos, agradeço a compreensão por minha ausência em diversos momentos. Te amo.

Agradeço a Oxalá e todas as entidades que me iluminam e protegem.

O presente trabalho foi realizado com apoio da Coordenação de Aperfeiçoamento de Pessoal de Nível Superior - Brasil (CAPES) - Código de Financiamento 001.

RESUMO

Estilos arquiteturais são importantes para engenharia pois são a ponte entre os requisitos e o *design* de implementação. Tem a função de expressar um conjunto de características de uma arquitetura de *software*, objetivando fornecer uma visão ampla da comunicação entre os componentes da arquitetura de *software*, facilitando o reuso e reduzindo a complexidade. Com o crescimento da área financeira, tais empresas empregaram diferentes estilos arquiteturais no desenvolvimento de *software* visando aumentar a reusabilidade, desempenho e a segurança. A literatura acerca do tópico, porém, carece de estudos que investiguem estilos arquiteturais modernos que possuam foco nas necessidades específicas das arquiteturas de *software* para o desenvolvimento de aplicações na área financeira, tais como escalabilidade, alta disponibilidade, consistência e integridade das informações. Além disso, em face do recente crescimento dessa área, torne o desenvolvimento de novas aplicações simples e robusto. Esta dissertação, portanto, apresenta o EventChain, o qual trata-se de um estilo arquitetural orientado a cadeia de eventos, que emprega o uso de comunicação assíncrona e Blockchain para o desenvolvimento de aplicações da área financeira. O estilo arquitetural proposto foi avaliado através de duas formas. A primeira sendo a construção de um protótipo com o objetivo de avaliar a viabilidade e demonstrar o seu funcionamento, e a segunda, a aplicação de um questionário de aceitação tecnológica para avaliar a aceitação do estilo arquitetural por profissionais da indústria. Os resultados obtidos mostram que o estilo arquitetural proposto é uma implementação viável, funcional e que atende aos requisitos de sistemas na área financeira. Por fim, conclui-se que o estilo arquitetural representa uma nova abordagem com grande potencial para facilitar o desenvolvimento de novos sistemas na área financeira, que endereça os requisitos específicos e torna flexível a implementação de novas aplicações.

Palavras-chave: Arquitetura de Software. Blockchain. Microsserviços. Estilo Arquitetural de Software. Orientado a Eventos.

ABSTRACT

Architectural styles are important for engineering as they bridge the gap between requirements and implementation design. Its function is to express a set of features of a software architecture, aiming to provide a broad view of the communication between the components of the software architecture, facilitating reuse and reducing complexity. With the growth of the financial area, such companies employed different architectural styles in software development aiming to increase reusability, performance and security. The literature on the topic, however, lacks studies that investigate modern architectural styles that focus on the specific needs of software architectures for the development of applications in the financial area, such as scalability, high availability, consistency and integrity of information. Also, given the recent growth in this area, make developing new applications simple and robust. This dissertation, therefore, presents the EventChain, which is an architectural style oriented to the chain of events, which employs the use of asynchronous communication and Blockchain for the development of applications in the financial area. The proposed architectural style was evaluated in two ways. The first is the construction of a prototype in order to assess the feasibility and demonstrate its operation, and the second, the application of a technological acceptance questionnaire to assess the acceptance of the architectural style by industry professionals. The results obtained show that the proposed architectural style is a viable, functional implementation that meets the requirements of systems in the financial area. Finally, it is concluded that the architectural style represents a new approach with great potential to facilitate the development of new systems in the financial area, which addresses specific requirements and makes the implementation of new applications flexible.

Keywords: Blockchain. Event Driven Architecture. Microservice. Software Architecture. Software Architectural Style.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS

fintech	Financial technology

LISTA DE SIGLAS

2PC	Two-Phase Commit
ABNT	Associação Brasileira de Normas Técnicas
ACID	Atomicity, Consistency, Isolation, Durability
API	Application Programming Interface
BFT	Byzantine Fault Tolerance
BOSE	Blockchain-oriented Software Engineering
CAPES	Coordenação de Aperfeiçoamento de Pessoal de Nível Superior
CC	Critério de Comparação
CE	Critério de Exclusão
C&C	Connectors and Components
CI	Critério de Inclusão
CQRS	Command and Query Responsibility Segregation
FAPERGS	Fundação de Amparo à Pesquisa do Estado do Rio Grande do Sul
GB	GigaByte
GHz	GigaHertz
HTTP	Hypertext Transfer Protocol
JAR	Java ARchive
JSON	JavaScript Object Notation
JWT	JSON Web Token
MB	Megabytes
MHz	MegaHertz
MVC	Model–view–controller
NoSQL	Non-structured Query Language
P2P	Peer-to-Peer
PBFT	Practical Byzantine Fault Tolerance
PoS	Proof of Stakes
PoW	Proof of Works

QP	Questão de Pesquisa
RC	Requisito Comum
REST	Representational state transfer
SSD	Solid-State Drive
SDK	Software Development Kit
SMS	Systematic Mapping Study
SOA	Service-oriented architecture
SOAP	Simple Object Access Protocol
SPOF	Single Point Of Failure
SQL	Structured Query Language
TPS	Transações por Segundo
TAM	Technology Acceptance Model
UI	User Interface
URI	Uniform Resource Identifier

SUMÁRIO

1 INTRODUÇÃO

O estilo arquitetural desempenha um papel fundamental no desenvolvimento de aplicações na área financeira, pois é através do conjunto coordenado de restrições arquiteturais que os papéis são delimitados ou as características dos elementos que compões a arquitetura são definidas. As relações permitidas entre os elementos dentro de qualquer implementação de um estilo de arquitetura devem estar em conformidade com tais restrições (FIELDING; TAYLOR, 2000).

Esse tema tem despertado o interesse de pesquisadores há muito tempo, Bass, Clements e Kazman (2003), Clements et al. (2002) e Hofmeister, Nord e Soni (1999) são exemplos de estudos que contribuíram para chamar a atenção para essa importante área. Sendo Shaw e Garlan (1996) um dos precursores do tema. Desde então, muitos outros estilos arquiteturais e padrões de arquitetura foram desenvolvidos e aprimorados.

A área financeira ao longo do tempo aplicou diferentes estilos arquiteturais, visando tornar os seus sistemas mais robustos, seguros e modernos. Sendo essa uma área que busca por padrões, Zhang, Wang e Zhang (2010) e Kamogawa e Okada (2008) apresentaram estudos de caso aplicando o estilo arquitetural SOA (*Service Oriented Architecture*). Ao longo do tempo novos estilos arquiteturais surgiram como o estilo de microsserviços (RICHARDSON, 2018), oferecendo novas possibilidades e ocasionando assim a migração de um estilo para outro e convivendo com múltiplas abordagens ao mesmo tempo (Bucchiarone et al., 2018). Recentemente, com o crescimento no número de transações, o estilo arquitetural orientado a eventos tem sido aplicado em conjunto ao estilo arquitetural de microsserviços na busca por maior escalabilidade, como demonstrando nos trabalhos de Soethout (2019) e Gil e Díaz-Heredero (2018).

Com o recente crescimento dessa área, observa-se um aumento na demanda por sistemas de *software* que proporcionem vantagem competitiva nas instituições e reduzam o tempo de criação de novos produtos. No entanto, as aplicações e soluções de *software* do mundo financeiro enfrentam desafios complexos, como forte dinamismo para responder às mudanças e que também devem suportar os requisitos não-funcionais, tais como alta disponibilidade, consistência e integridade das informações, acrescentando complexidade para o desenvolvimento de *software*. Nesse cenário, rotineiramente arquitetos e desenvolvedores precisam implementar novas funcionalidades tendo em vista os requisitos não-funcionais, tornando essa tarefa repetitiva e suscetível a erros, dado os

diversos estilos arquiteturais envolvidos.

Devido a isso, tais empresas têm buscado novas tecnologias não somente para atender à crescente demanda, mas também para aumentar a segurança e confiabilidade das informações armazenadas e mantidas por seus sistemas, dado que estas ganharam um valor inestimável. Uma das tecnologias mais promissoras para a área financeira é a Blockchain, um livro-razão distribuído que fornece apenas a possibilidade de adicionar novas informações tornando-as imutáveis. As informações dentro da Blockchain são separadas em blocos, ao final de cada bloco eles são assinados criptograficamente e, o próximo novo bloco possui a referência da assinatura do bloco anterior formando assim a cadeia de blocos. Por isso muitas instituições estão investindo na pesquisa e aplicação dessa tecnologia (DANEZIS; MEIKLEJOHN, 2015) (TSAI et al., 2016) (TAPSCOTT; TAPSCOTT, 2017). Tais ações têm apresentado resultados promissores, encorajando novas pesquisas e descobertas a fim de extrair o máximo do potencial e possibilidades oferecidas pela tecnologia e ferramentas desse ecossistema.

1.1 Problemática

De acordo com Bucchiarone et al. (2018) nas arquiteturas monolíticas, as abstrações de modularização dependem do compartilhamento de recursos da mesma máquina (memória, bancos de dados e/ou arquivos), e os componentes não são, portanto, executáveis de forma independente, notadamente um dos maiores problema desse estilo arquitetural é a escalabilidade. Além do mais, esse estilo arquitetural apresenta alto acoplamento e dificuldades relacionadas às mudanças. Como mencionado anteriormente, os estilos arquiteturais de microsserviços e orientado a eventos foram adicionados na busca por escalabilidade, desempenho e tolerância a falhas. Sendo a Blockchain a mais recente tecnologia agregada nas arquiteturas de *software*.

Porém, um dos maiores desafios na introdução e uso de Blockchain é a falta de estudos na criação e aplicação de estilos arquiteturais de *software* que envolvam tal tecnologia e que suportem os requisitos não-funcionais mencionados. Dada a diversidade de aplicações e estilos arquiteturais, o desenvolvedor de *software* precisa empregar um esforço na busca de conectar os sistemas já existentes com a Blockchain. Sobretudo no estilo arquitetural orientado a eventos, que possui características únicas, como transações distribuídas e a não-garantia de atomicidade entre os serviços, o que causa uma

série de desafios para a consistência e integridade das informações nesta abordagem.

Neste sentido, vários trabalhos foram propostos nos últimos anos (ALLEN, 1997), (LOULOU et al., 2006), (Loulou et al., 2006), (Schwanke, 2001), (Guozhen Tan et al., 2005), (Tran; Zdun, 2013), (Taylor et al., 1996), (Taylor; Medvidovic; Oreizy, 2009), buscando introduzir uma solução. Porém, tais estudos apresentam algumas limitações: (1) não endereçar no estilo arquitetural o suporte ao uso de Blockchain em aplicações orientadas a eventos, tampouco que os eventos gerados pelas aplicações não serão manipulados posteriormente; (2) não ser específico para aplicações financeiras; (3) não existência de um mecanismo que possa extrair informações dos eventos gerados pelas aplicações; (4) falta de avaliação empírica da solução e validação de aceitação da mesma; (5) possuir um histórico completo e imutável da sequência de eventos.

Neste sentido, a literatura atual ainda carece de estilos sensíveis a requisitos inerentes às aplicações financeiras e, principalmente, de um estilo arquitetural utilizando Blockchain em conjunto com os estilos arquiteturais e aplicações existentes.

Aplicações Financeiras. As aplicações financeiras se caracterizam por uma série de eventos de negócio, originados a partir de diferentes situações após o fim de uma transação ou de um processo. Essa implementação exige que o desenvolvedor implemente um mecanismo complexo de controle de transações, confirmando ou desfazendo as transações em caso de falhas durante a mesma. Diversas instituições financeiras (DANEZIS; MEIKLEJOHN, 2015) estão estudando e aplicando a Blockchain, buscando a melhoria de seus processos (TSAI et al., 2016) (TAPSCOTT; TAPSCOTT, 2017).

Observando-se a Figura 1 é possível visualizar os estilos arquiteturais utilizados atualmente e o fluxo de comunicação entre as aplicações, que pode ser dividido nas seguintes etapas (cada etapa possui o número correspondente na figura):

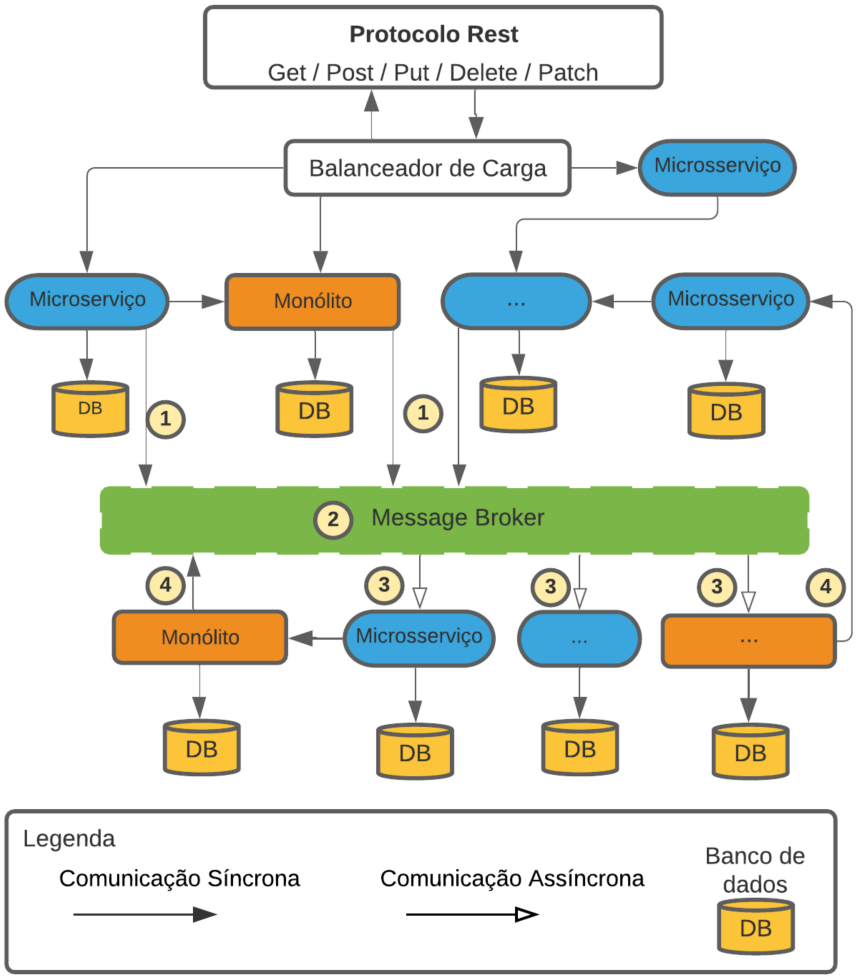
- **Etapa 1: Eventos de comunicação.** Ao receber uma requisição as aplicações monolíticas ou de microsserviços executam a sua lógica comunicando-se entre si e persistindo as informações no seu banco dados, colaborando para retornar a resposta. Os eventos entre as aplicações podem ser gerados durante ou ao final do processamento, e tem como objetivar sinalizar outras aplicações que uma ação ocorreu.

- **Etapa 2: *Message Broker*.** O *message broker* exerce o papel fundamental de receber o evento e rotear a entrega do mesmo para os consumidores que tem

interesse no mesmo. Recebendo e confirmando os eventos de forma síncrona, e entregando para os consumidores de assincronamente.

- **Etapa 3: Aplicações consumidoras.** São as aplicações clientes, monolíticas ou de microsserviços que possuem interesse em escutar pelos eventos realizados. Ao receber da etapa anterior o evento, tais aplicações podem tomar ações de forma assíncrona, colaborando entre si sem prejudicar a processamento realizado na Etapa 1.

Figura 1 – Cenário atual



Fonte: Elaborado pelo autor.

- **Etapa 4: Resultado dos eventos processados.** Similar ao realizado na etapa 1,

tais aplicações podem lançar novos eventos e continuar o fluxo de comunicação entre as mesmas.

As informações armazenadas se distribuem em diferentes bancos de dados, confiando a cada um dos componentes a responsabilidade de manter a consistência e integridade das informações.

A implementação da combinação desses dois estilos arquiteturais, de microsserviços e monólito, introduz alguns desafios: (1) necessidade de habilitar o uso de transações entre sistemas distribuídos e monolíticos; (2) informações e dados divididos entre múltiplos banco de dados; (3) difícil implementação de consistência e integridade dos dados; (4) difícil entendimento de quais eventos devem ser utilizado pelas aplicações; e (5) implementação de múltiplos eventos de negócio em forma de transação.

Dessa forma, é possível observar que atualmente é utilizado um cenário híbrido como demonstrado na Figura 1, tal abordagem emerge da necessidade de atender a demanda dos requisitos de negócio rapidamente e da necessidade de evoluir a arquitetura das aplicações utilizadas. A aplicação do estilo arquitetural de microsserviços se enquadra neste cenário, permitindo que novas funcionalidades possam ser adicionadas rapidamente sem a necessidade de alteração nos sistemas que já estão em execução, sejam eles legados monolíticos ou não.

Todavia, diversos sistemas não estão preparados para esse estilo arquitetural, dessa forma uma camada de mensageria assíncrona é adicionada para desacoplar a dependência entre os sistemas, criando um novo estilo arquitetural composto por microsserviços e orientado a eventos para comunicação entre as partes interessadas. É nesse cenário desafiador que a Blockchain surgiu, apresentando novas possibilidades para responder aos desafios complexos dessa área de negócio.

Portanto, partindo dessa problemática esse estudo propõe o *EventChain*, um estilo arquitetural orientado a cadeia de eventos para o desenvolvimento de aplicações financeiras. Ao utilizar o *EventChain*, desenvolvedores de *software* se beneficiarão de um estilo arquitetural sensível a preocupações inerentes a essa área, tais como segurança, performance, tolerância a falhas, alta disponibilidade, consistência e integridade dos dados. Para isso, o *EventChain* propõe um conjunto de decisões arquiteturais e conceitos que potencializam a produtividade dos desenvolvedores de *software*, tais como cadeia de eventos, resolvedor de eventos e eventos de domínio.

1.2 Questões de Pesquisa

Conforme mencionado, a introdução e uso de Blockchain em arquiteturais modernas carece de maiores estudos, principalmente em uma área de negócio tão complexa e crítica como a área financeira. Além disso, foi possível identificar que os novos estilos arquiteturais introduzidos não endereçam nativamente soluções que garantam a consistência e integridade das informações utilizando arquiteturas orientadas a evento.

Diante das necessidades expostas, este trabalho investigará as seguintes Questões de Pesquisa (**QPs**):

- **QP1**: Qual o estado-da-arte em relação ao uso de Blockchain na área financeira?
- **QP2**: Como implementar um estilo arquitetural orientado a eventos com suporte a Blockchain?
- **QP3**: Qual seria a aceitação do estilo arquitetural proposto por profissionais da indústria que trabalham na área financeira?

Dessa forma, a implementação dessa pesquisa se propõe a responder as questões de pesquisa apresentadas. Para isso foram os definidos alguns objetivos que serão apresentados na seção a seguir.

1.3 Objetivos

O objetivo principal desse trabalho é o de **propor um estilo arquitetural para uso de Blockchain em sistemas orientados a eventos na área financeira**. Para alcançar o objetivo geral foram definidos os seguintes objetivos específicos (OE):

- **OE1: Realizar o mapeamento sistemático da literatura**. Esse objetivo visa identificar os trabalhos realizados na área financeira detectando as lacunas e as oportunidades de pesquisa sobre uso de Blockchain. Bem como, criar um mapeamento sistemático dos trabalhos realizados identificando as principais tecnologias, as motivações para uso da Blockchain, quais áreas de atuação empregaram a Blockchain e métodos de pesquisa empregados. O mapeamento sistemático é explorado em detalhes na Seção 3.2 do Capítulo 3.

- **OE2: Propor um estilo arquitetural com suporte a Blockchain**. Tem o objetivo de propor um estilo arquitetural definindo um conjunto de restrições arquiteturais para uso e adoção de Blockchain em aplicações orientadas a eventos. E que seja um estilo arquitetural viável tecnicamente para uso pelas aplicações financeiras. Esse objetivo é explorado no Capítulo 4.

- **OE3: Produzir conhecimento empírico sobre o estilo arquitetural proposto**. Tem o objetivo de implementar um estudo de caso que permita avaliar viabilidade de implementação do estilo arquitetural proposto utilizando tecnologias atuais bem como avaliar a aceitação tecnológica por profissionais da área financeira. O Capítulo 5 explora em mais detalhes a implementação desse objetivo.

1.4 Metodologia

Essa seção tem o objetivo de apresentar os detalhes da metodologia utilizada nesse estudo. A metodologia utilizada possui as seguintes etapas:

- **Etapa 1: Estado-da-arte de estilos arquiteturais**. A primeira etapa se caracteriza pela revisão da literatura relacionada a estilos arquiteturais de *software* orientados a eventos.

- **Etapa 2: Mapeamento sistemático**. Nessa segunda etapa foi realizado uma revisão da literatura sobre o uso Blockchain na área financeira, utilizando mapeamento sistemático tendo como foco responder à QP1 (PETERSEN; VAKKALANKA; KUZNIARZ, 2015a). Além de revisar a literatura atual identificando as oportunidades de pesquisa, o mapeamento sistemático proporciona uma visão geral do aprendizado já realizado.

- **Etapa 3: Definição do estilo**. A terceira etapa consiste na definição e na formulação dos conceitos propostos pelo *EventChain*, o estilo arquitetural proposto mediante este trabalho. É nessa etapa que serão definidas as bases do estilo arquitetural para as etapas seguintes visando responder à QP2.

- **Etapa 4: Implementação**. Na quarta etapa, um estudo de caso implementa algumas funcionalidades comuns do contexto de aplicações financeiras utilizando o estilo arquitetural proposto.

- **Etapa 5: Resultados empíricos**. A última etapa desse estudo visar analisar os resultados obtidos através da implementação do caso de uso e do questionário de aceitação, respondendo assim à QP3.

1.5 Organização do Trabalho

Esta dissertação é estruturado da seguinte forma: a Fundamentação Teórica (Capítulo 2) apresenta os conceitos básicos para o entendimento do estudo; os Trabalhos Relacionados (Capítulo 3) enumera e discute os artigos que se relacionam com essa pesquisa; o Estilo Arquitetural Proposto (Capítulo 4) discorre sobre o estilo arquitetural proposto e seus detalhes; a Avaliação (Capítulo 5) apresenta uma avaliação do estilo arquitetural e a Conclusão (Capítulo 6) discorre sobre as conclusões do estudo, bem como os trabalhos que podem ser realizados futuramente.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo apresenta uma visão geral e os conceitos básicos sobre os temas abordados nesse estudo. A Seção 2.1 aborda os conceitos relacionados a estilo arquitetural de *softwares*. A Seção 2.2 apresenta os conceitos de arquitetura de software, enquanto que a Seção 2.3 discorre sobre os conceitos do estilo arquitetural de monólito. Na Seção 2.4 os conceitos do estilo arquitetural de microsserviços são apresentados. A Seção 2.5 apresenta os conceitos relacionados a *Event Sourcing* e a Seção 2.6 introduz o conceito de Blockchain.

2.1 Estilo Arquitetural

Estilo arquitetural de *software* é um conjunto coordenado de restrições arquiteturais que delimita os papéis ou características dos elementos da arquitetura, e as relações permitidas entre os elementos dentro de qualquer arquitetura que esteja em conformidade com esse estilo (FIELDING; TAYLOR, 2000). É um mecanismo para descrever e categorizar as arquiteturas, definindo as suas características comuns, servindo de base para que novas arquiteturas possam ser definidas a partir de instâncias de um estilo especifico (di Nitto; Rosenblum, 1999). Dessa forma, o estilo arquitetural fornece uma visão de alto nível da interação entre os componentes, não sendo direcionado a pontos específicos de uma arquitetura, ajudando na organização de uma aplicação de um contexto específico e refletindo as funcionalidades que são comuns à estrutura ou domínio (Capítulo 4).

O uso de estilos como veículo para caracterizar uma família de arquiteturas de *software* é motivado por uma série de benefícios. Os estilos fornecem um vocabulário comum para arquitetos, permitindo que os desenvolvedores entendam facilmente um projeto de arquitetura de rotina. Em muitos casos, novos estilos arquiteturais são definidos baseado em estilos já existentes, ao adaptar um estilo para funções específicas, um novo é criado (KIM; GARLAN, 2010).

Cada estilo arquitetural fornece um nível de abstração para a interações entre os componentes, definindo os padrões de interação sem adentrar nos detalhes da arquitetura. Embora existam muitos estilos arquiteturais, eles não conseguem satisfazer os requisitos de integração em alguns contextos específicos, especialmente quando envolve

a integração de muitos sistemas legados. A escolha de um estilo arquitetural errado pode afetar a arquitetura e, por consequência, as aplicações de diferentes formas, seja na confiabilidade destas, assim como no seu desempenho ou na velocidade de desenvolvimento (Zhao; Zhao, 2010).

Diante o desafio de responder a diferentes contextos, surge o conceito de restrição arquitetural. As restrições arquiteturais exercem o importante papel de definir as regras a serem seguidas ao aplicar um determinado estilo (di Nitto; Rosenblum, 1999). O seu principal objetivo é o auxiliar arquitetos e desenvolvedores de *software* no mapeamento das implementações da arquitetura ao seguir as suas prescrições e restrições do estilo, oferecendo assim um conjunto de princípios e guias para o *design* dos componentes e conectores que compõem a solução, bem como o relacionamento entre eles. É responsabilidade do estilo arquitetural definir os componentes e conectores que estarão presentes, bem como as restrições e como elas devem ser aplicadas (AKMEL et al., 2017).

O estilo arquitetural serve como um guia, ajudando a organizar cada modelo do domínio, subsistemas e as integrações entre os sistemas, encapsulando decisões importantes e enfatizando as restrições. Cada decisão arquitetural pode ser vista como uma aplicação do estilo, alguns exemplos de estilos arquiteturais são o REST (FIELDING; TAYLOR, 2000), *Event Sourcing* (FOWLER, 2005), microsserviços e monólitos (RICHARDSON, 2018).

A literatura apresenta também o conceito denominado como modelo arquitetural, de acordo com Wong et al. (2008) um modelo arquitetural é utilizado tipicamente para expressar os componentes, suas propriedades e principalmente a comunicação entre tais componentes utilizando conectores. A definição dos elementos que compõem a estrutura dos modelos arquiteturais (componentes e conectores) podem emergir de definições próprias ou a partir de estilos arquiteturais existentes, direcionando o modelo arquitetural para uma arquitetura específica (CALVARY; COUTAZ; NIGAY, 1997).

Esse trabalho tem o objetivo de propor um estilo arquitetural devido a possibilidade de impor restrições arquiteturais, não ser orientado para uma arquitetura específica e a possibilidade de agregar novos estilos, como descrito na Seção 4.3.

2.2 Arquitetura de Software

Pode ser definida como um conjunto de escolhas e decisões, é através da arquitetura que é definida a estrutura, as limitações e as características dos componentes e interfaces. De acordo com Bass, Clements e Kazman (2003) a arquitetura de *software* de um sistema é o conjunto de estruturas necessárias para raciocinar sobre o mesmo, que compreende os elementos de *software*, as relações entre eles e as propriedades de ambos.

Ainda segundo Bass, Clements e Kazman (2003) a arquitetura de *software* é composta por uma coleção de estruturas, as quais podem ser classificadas em três categorias a saber:

- **Módulos**: são estruturas estáticas responsáveis por implementar funções computacionais específicas, é o trabalho de base dos programadores. Um módulo pode ser decomposto como funcionalidade macro, como por exemplo, uma regra de negócio, ou ainda, no seu menor nível, pode ser uma classe em uma linguagem de programação orientada a objetos.

- **Componentes e conectores (C&C)**: são estruturas dinâmicas que focam na iteração entre os elementos em tempo de execução. Os componentes são unidades operacionais como por exemplo, servidores e, representam o comportamento das estruturas em tempo de execução.

- **Alocação**: descreve o mapeamento das estruturas de *software* para a organização do sistema, seu desenvolvimento e sua instalação.

As estruturas desempenham um papel importante na perspectiva da arquitetura devido ao poder analítico que fornece, estando diretamente associada aos atributos relevantes de qualidade. Os atributos de qualidade de *software* são uma propriedade mensurável ou testável de um sistema, são utilizados para indicar a satisfação em relação aos requisitos. A seguir são apresentados os principais atributos de qualidade abordados nesse estudo:

- **Escalabilidade**: Os dois tipos de escalabilidade são o horizontal e o vertical. A escalabilidade horizontal se refere a adição de recursos para as unidades lógicas,

como mais um servidor a um *cluster*. Já a escalabilidade vertical se refere a adição de recursos para as unidades físicas, como memória e processamento. O sucesso da escalabilidade significa dizer que os recursos adicionais resultam em uma melhoria mensurável, não exigindo esforço desnecessário ao adicionar novos recursos sem interromper as operações;

- **Adequação funcional**: Grau em que um produto ou sistema fornece funções que atendem às necessidades declaradas e implícitas quando usado sob as condições específicas;
- **Eficiência de desempenho**: Desempenho em relação a quantidade de recursos disponibilizados nas condições estabelecidas;
- **Confiabilidade**: Grau em que um sistema, produto ou componente executa as funções sob determinadas condições por um período de tempo;
- **Segurança**: Grau em que um sistema ou produto protege as informações e dados de pessoas ou outros sistemas de acordo com o nível de acesso especificado;

A arquitetura de *software* fornece uma abstração para gerenciar a complexidade do sistema e estabelecer um mecanismo de coordenação e comunicação entre os elementos que a compõem. Define as estruturas que atenderão aos requisitos técnicos e de negócios, buscando otimizar os atributos de qualidade, como por exemplo, desempenho e segurança. Tais definições impactam diretamente na qualidade, manutenção, desempenho e sucesso do produto final (JAISWAL, 2019).

A arquitetura é constituída por decisões, algumas grandes e outras pequenas, e a maioria precisa ser tomada no início do projeto, pois podem ter impacto profundo em outras fases do projeto que estão por vir. Projetar arquiteturas de *software* é uma tarefa crítica e altamente exigente. É uma tentativa de abstrair os pontos comuns inerentes ao projeto do sistema, como as atividades, conceitos, métodos, abordagens e resultados esperados (BASS; CLEMENTS; KAZMAN, 2003).

2.3 Estilo Arquitetural Monolítico

Em arquiteturas monolíticas, as abstrações de modularização dependem do compartilhamento de recursos da mesma máquina (memória, bancos de dados ou arquivos)

e, portanto, os componentes não são executáveis de forma independente (Bucchiarone et al., 2018). De acordo com o Microservices: Yesterday, Today, and Tomorrow (2017) um *software* monolítico é um aplicativo de *software* composto por módulos que não são independentes do aplicativo ao qual pertencem. Em uma arquitetura monolítica os serviços que compõem a aplicação são organizados de forma lógica no mesmo código fonte e unidade de instalação, sempre com um alto acoplamento entre os componentes.

A abordagem monolítica possui algumas vantagens como simplicidade de implantação e facilidade de desenvolvimento de acordo com Gos e Zabierowski (2020). Um exemplo clássico dessa implementação é o padrão MVC, que separa a aplicação em 3 camadas:

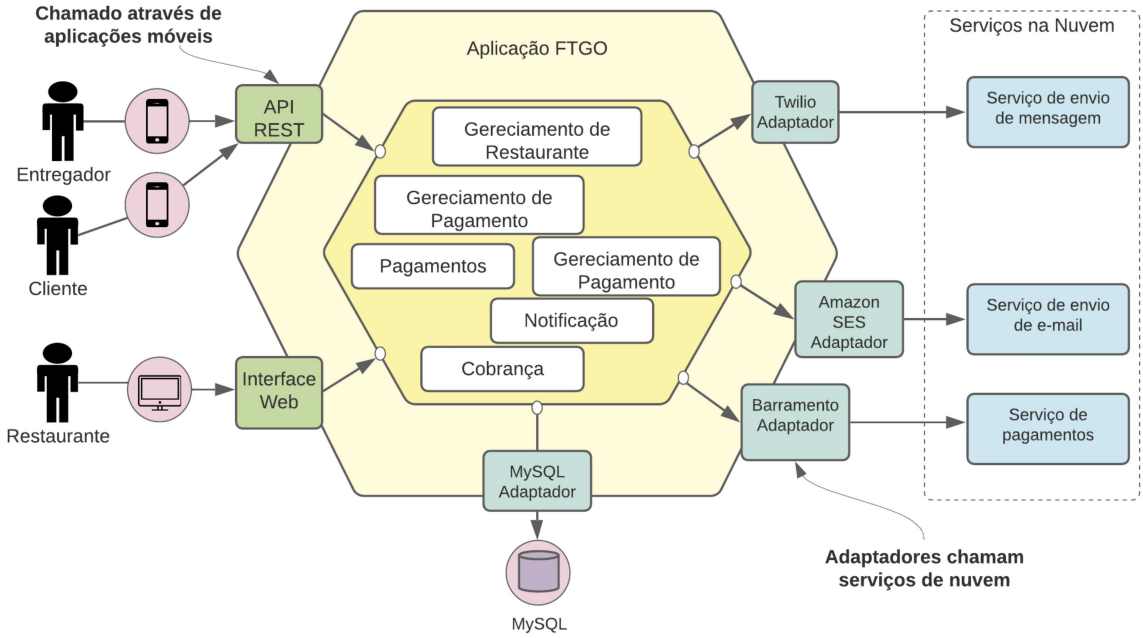
- ***View***: Tem a responsabilidade de renderizar a interface para o usuário, exibir os modelos, enviar e receber requisições de atualização do modelo;
- ***Controller***: Define o comportamento da aplicação, mapeia as ações que o usuário pode executar e seleciona as respostas que serão enviadas como respostas às ações realizadas pelo usuário;
- ***Model***: Encapsula o estado da aplicação, respondendo as perguntas sobre o estado, bem como expõe as funcionalidades da aplicação e notifica a *View* sobre as mudanças realizadas;

Um exemplo de implementação do estilo arquitetural monolítico pode ser visto na Figura 2. Todos os componentes são executados em uma única instância da aplicação e a mesma possui um banco dados centralizado.

No entanto, conforme apontado em diferentes estudos recentes como Gos e Zabierowski (2020), Microservices: Yesterday, Today, and Tomorrow (2017) e Bucchiarone et al. (2018), as arquiteturas monolíticas possuem diversas limitações. A seguir são apresentadas as principais limitações desse estilo.

- **Manutenibilidade**: Sistemas monolíticos grandes são difíceis de manter devido a sua complexidade. O rastreio, identificação e correção de *bugs* envolve uma longa investigação do código base. Resultando em uma maior complexidade para realizar as tarefas e também em mais tempo de depuração para realização de correções e testes;

Figura 2 – Exemplo de arquitetura monolítica



Fonte: Adaptado de Richardson (2018)

- **Gerenciamento de dependências**: A atualização ou inserção de novas bibliotecas podem impactar o sistema por completo, resultando estados situações inconsistentes. Gerando esforço para reteste da aplicação por completo ou em alguns casos, forçando a re-implementação de código para atender aos requerimentos da biblioteca inserida ou atualizada;

- **Dependência de tecnologia**: Sistemas monolíticos representam a dependência de uma tecnologia única, limitando a evolução da arquitetura ou da aplicação. Dessa forma, os desenvolvedores precisam lidar com as limitações sem terem liberdade para adicionar outras tecnologias ou *frameworks*.

- **Implementação de mudanças**: Qualquer alteração que seja realizada necessita de uma recompilação completa da aplicação, gerando a necessidade retestar toda a aplicação e refazer a implantação. Ao refazê-lo o, todas as instâncias da aplicação reiniciadas podendo causar períodos de indisponibilidade;

- **Acoplamento**: Mesmo que a aplicação seja construída de forma modular, o có-

digo base ainda é o mesmo, dessa forma, diferentes times tem dificuldade de compartilhar o mesmo ambiente para realizar as suas alterações;

- **Escalabilidade e tolerância à falhas**: Para escalar uma aplicação monolítica é necessário adicionar uma cópia da mesma. Por exemplo, se determinado módulo precisa ser escalado uma cópia completa da aplicação será realizada, causando uma alocação de recursos desnecessária. Ao escalar as aplicações monolíticas a pressão sobre os recursos compartilhados aumenta, podendo causar falhas em toda a aplicação se o banco de dados falhar, por exemplo.

2.4 Estilo Arquitetural de Microsserviços

O estilo arquitetural de microsserviços foi cunhado em 2005 por Rodgers (2005), durante uma apresentação na conferência *"Web Services Edge"*. Nesta época Rodgers (2005) referiu-se ao termo como "Micro-Web-Services". sendo as principais características deste apontadas como:

- Os componentes de *software* são micro *webservices* acessados por URI;
- URIs são publicados em um espaço de endereço URI interno;
- Os serviços são compostos usando *pipelines* do tipo Unix (utilizando baixo acoplamento);
- Os serviços podem chamar serviços (muitas linguagens de programação podem estar envolvidas);
- A complexidade dos serviços são abstraídos por trás de uma interface URI simples;
- Qualquer serviço, em qualquer granularidade, pode ser exposto;

Baseada na arquitetura Orientada a Serviços, mas ao invés de mensagens SOAP a arquitetura proposta por Rodgers (2005) foi projetada para usar o estilo REST.

Algumas definições do que é um “microsserviço” de acordo com Rodgers (2005):

- **Pequeno e com uma única responsabilidade**. Cada aplicação faz apenas uma coisa, enfatizando dessa forma a separação de funcionalidades que não se relacionam e agrupando as funcionalidades de um domínio em um microsserviço;

- **Não possui dependência de contêineres externos**. Seguindo o comportamento dos serviços Unix, a aplicação deve ser autossuficiente, possuindo internamente a capacidade de executar um servidor *web* ou ser um *jar* executável por exemplo;

- **Cada aplicativo é completamente separado**. Separado por domínio ou contextos, os microsserviços de um contexto devem ter a capacidade de executar múltiplas instâncias com suas próprias funcionalidades sem afetar uns aos outros;

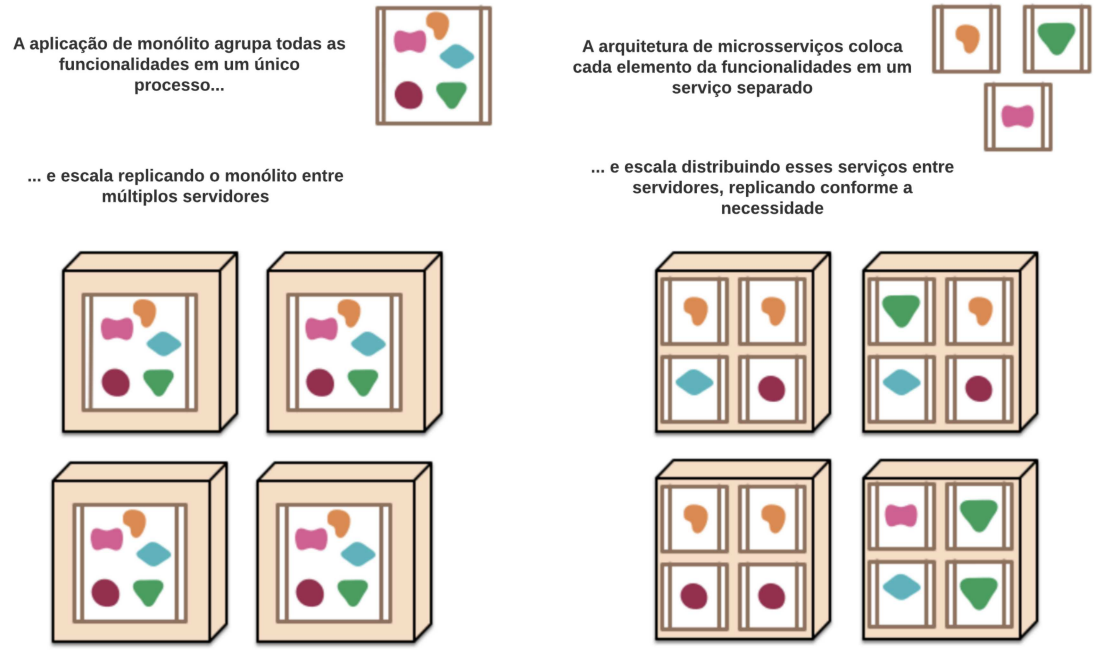
- **Contexto delimitado**. Domínios que fazem parte de diferentes contextos delimitados devem ser distintos, a duplicação de domínios nesse caso é aceitável.

- **Compartilhamento de código limitado**: O código compartilhado deve ser código de biblioteca ou infraestrutura, evitando ao máximo o acoplamento de código;

- **Provisionado automaticamente**: Um balanceamento de carga para cada microsserviço deve ser utilizado, permitindo assim que o mesmo possa ser escalonado automaticamente.

Usando o estilo arquitetural REST de Fielding e Taylor (2000) como base, os microsserviços são geralmente expostos ao mundo como uma API sob o protocolo HTTP. Para comparar o estilo de arquitetura de microsserviço com estilo de arquitetura monolítico, é necessário ter em mente que o estilo de arquitetura de microsserviço agregará essencialmente as características dos sistemas distribuídos. A arquitetura monolítica é um estilo que centraliza todos os componentes da aplicação em um único pacote, a aplicação tratará das requisições HTTP, executará a lógica de negócio e, se for necessário, consumirá outros *webservices*, assim como enviará ou consumirá mensagens de *message brokers*. O estilo de arquitetura monolítica é simples de desenvolver, implantar e escalar. Toda a lógica é executada em uma única unidade de processo, a transação ACID é fácil, por exemplo. Uma comparação entre os estilos arquiteturais é apresentada na Figura 3.

Figura 3 – Comparando o estilo monolítico com o estilo de microsserviços



Fonte: Adaptado de Fowler (2014)

Existem várias vantagens no uso de arquiteturas de microsserviços ao compará-lo com o estilo monolítico, Al-Debagy e Martinek (2018) elencou os 5 principais benefícios do uso de microsserviços:

- **Heterogeneidade tecnológica**: Os times tem a liberdade de escolher a tecnologia mais adequada para cada implementação para atingir os objetivos e desempenho desejados;

- **Resiliência**: Tendo em mente que aplicações de microsserviços podem falhar por diversos motivos devido a indisponibilidade de recursos, os microsserviços são desenvolvidos com essa premissa. Dessa forma, cada aplicação pode tratar separadamente as falhas diferentemente da arquitetura monolítica, que seria impactada totalmente nesse cenário;

- **Escalabilidade**: É possível escalar microsserviços individualmente, ao contrário de um sistema monolítico, onde só é possível escalar a aplicação por completo;

- **Fácil implantação**: Dado que os microsserviços executam de formam independente, a disponibilização de novas versões é mais rápida e não afeta outras partes do sistema;

- **Alinhamento organizacional**: Dividindo as bases de código é possível criar times de desenvolvimento independentes que refletem melhor a estrutura organizacional.

Estudos recentes apontam muitas vantagens de uso do estilo arquitetural de microsserviços (Al-Debagy; Martinek, 2018; Bucchiarone et al., 2018; DRAGONI et al., 2017). No entanto, o estilo de arquitetura de microsserviço possui algumas desvantagens. É comum para arquiteturas distribuídas a dificuldade e complexidade de implantação, o teste é difícil em comparação com a arquitetura monolítica e, em diversos cenários, é preciso lidar com transações distribuídas. O controle das transações distribuídas é complexo exigindo um esforço para manter a consistência dos dados (Santos; Rito Silva, 2020).

O último tópico para comparar arquiteturas de microsserviços e arquiteturas monolíticas são as transações, o objetivo das transações é manter os dados consistentes. O gerenciamento de transações é um dos tópicos mais discutidos em microsserviços, enquanto em uma aplicação monolítica o controle de transações é mais simples devido às transações estarem em uma única unidade, nos microsserviços cada uma tem sua própria base de dados (FOWLER, 2014).

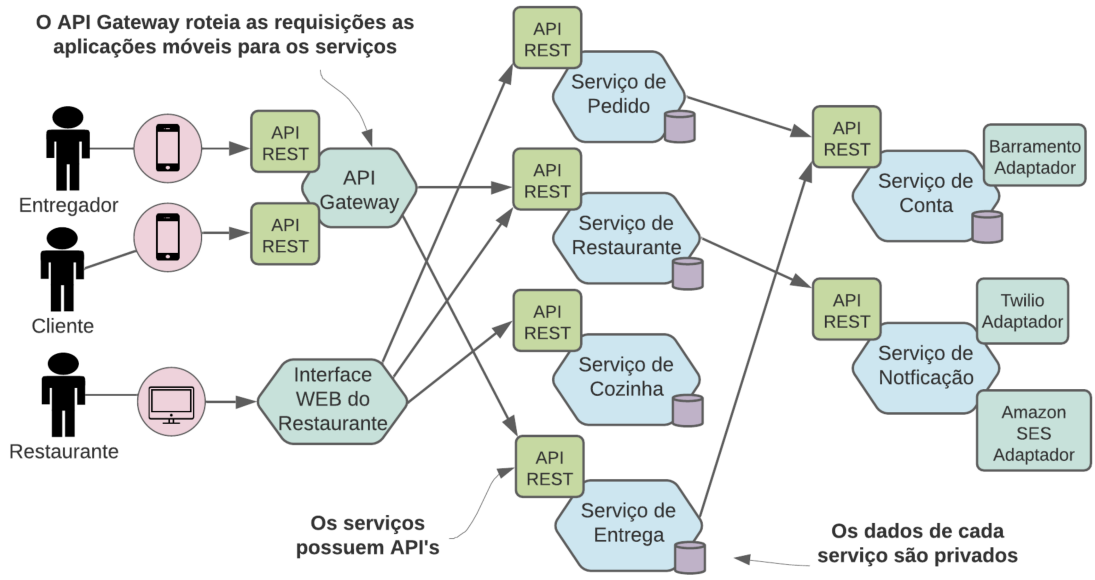
Richardson (2018) explica sobre duas estratégias comuns para manter dados consistentes em arquiteturas de microsserviços, (1) *commit* de duas fases (2PC) e (2) padrão SAGA. O *commit* de duas fases garante que todos os participantes de uma transação confirmem ou desfaçam as operações, é necessário que os participantes da transação como bancos de dados, *brokers* de mensagens e outros suportem o protocolo de *commit* de duas fases, este não é o caso de bancos de dados modernos como MongoDB[1] ou Cassandra[2] e *brokers* como RabbitMQ[3] e Apache Kafka[4]. O padrão SAGA aplica uma sequência de comandos que são coordenados, o coordenador pede um serviço para aplicar uma transação local, geralmente o serviço usa o ACID localmente para garantir

[1] https://www.mongodb.com
[2] https://cassandra.apache.org
[3] https://www.rabbitmq.com
[4] https://kafka.apache.org

os dados e retornar o resultado para o coordenador, então o coordenador pode decidir se pede para outro serviço para continuar a transação ou pode decidir pedir por uma sequência de comandos de compensação para reverter os dados.

Um exemplo de uma aplicação de microsserviços pode ser visto na Figura 4.

Figura 4 – Exemplo de arquitetura de microsserviços



Fonte: Adaptado de Richardson (2018)

2.5 Estilo Arquitetural *Event Sourcing*

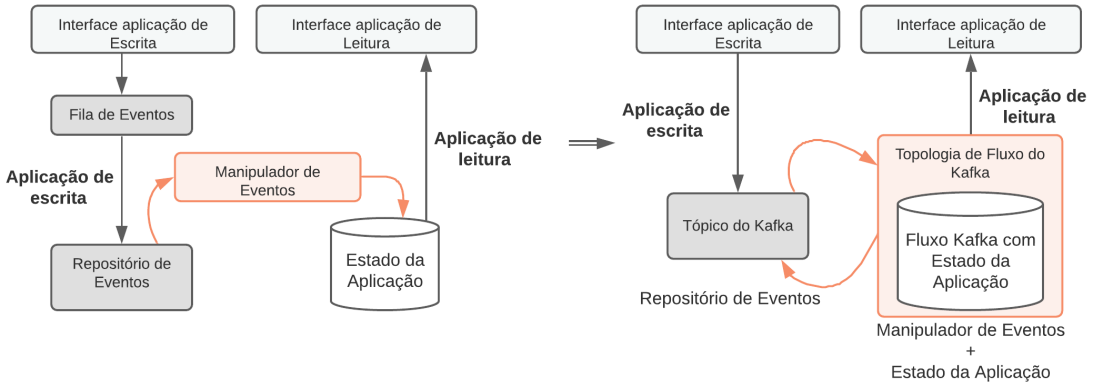
É um estilo definido por Fowler (2005) que descreve que, todas as alterações de estado devem ser capturadas e armazenadas como uma sequência de eventos. Ao contrário da abordagem tradicional em que as aplicações armazenam o último estado dos dados, o *Event Sourcing* persiste a cada mudança. Este padrão tem algumas vantagens:

- (1) permite a reconstrução do sistema reproduzindo os eventos;
- (2) permite a consulta temporal, é possível verificar o estado dos dados em um determinado momento;
- (3) provê um mecanismo de auditoria com todas as alterações realizadas;

Outro ponto é que novos sistemas podem ser adicionados ao fluxo de comunicação lendo os eventos e populando seus modelos, a integração de sistemas usando o padrão de eventos é mais fácil. Um exemplo de aplicação do *Event Sourcing* pode ser visto na Figura 5.

Para armazenar os dados e manter todo o histórico, é necessário que o fluxo de dados seja persistido, uma vez que o estilo *Event Sourcing* define que somente inserções com as informações do estado podem ser realizadas, conforme visto na Figura 5. É necessário armazenar um evento para modificar o estado do modelo de domínio. O *Event Sourcing* trata todos os eventos registrados sequencialmente que ocorrem no modelo de domínio como dados de entidades de primeira classe, esses dados são chamados de "evento de domínio". É uma coleção de eventos do início até o presente.

Figura 5 – Exemplo de arquitetura *Event Sourcing*



Fonte: Adaptado de https://www.confluent.io/blog/event-sourcing-cqrs-stream-processing-apache-kafka-whats-connection

Os eventos de domínio são registrados no *"Event Store"* e a restauração de todos os eventos armazenados no *"Event Store"* restaura o estado atual dos eventos de domínio. Eventos individuais incluídos em eventos de domínio não são modificados ou excluídos porque devem ser restaurados usando eventos de domínio (HAN; CHOI, 2020).

Alguns exemplos de uso do *Event Sourcing* nas aplicações financeiras para mapear eventos de domínio seriam: 1) domínio de cliente, 2) domínio de crédito e 3) domínio de login. Dessa forma a cada mudança de estado dessas informações um novo evento será criado e armazenado no *"Event Store"* para armazenar o histórico das mudanças.

Em uma arquitetura onde a escalabilidade é necessária, *Event Sourcing* pode ser combinado com uma arquitetura orientada a eventos (FOWLER, 2005), devido à natu-

reza desse estilo, onde os sistemas se comunicam por meio de mensagens de eventos, os sistemas podem cooperar em paralelo de forma fracamente acoplada, fornecendo uma escalabilidade horizontal e resiliência a falhas. De acordo com Evans (2003), um dos principais desafios na depuração de código em produção é a rastreabilidade e reprodutibilidade. O *Event Sourcing* enfrenta esses desafios permitindo confiar não apenas nos *logs* do aplicativo, mas armazenando todos os estados de transição e todos os eventos recebidos durante o processamento, ajudando a equipe a reproduzir os eventos para depurar e reproduzir exatamente as etapas da aplicação.

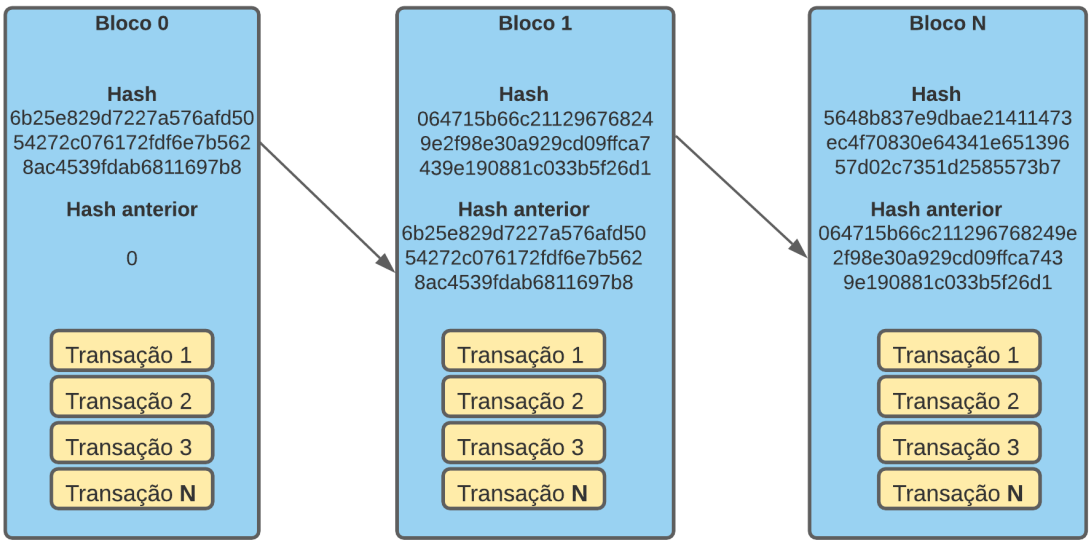
2.6 Blockchain

A tecnologia Blockchain foi aplicada em grande escala quando o Bitcoin foi introduzido por Nakamoto (2008). Caracteriza-se por ser um livro-razão distribuído que fornece apenas a possibilidade de adicionar novas informações. As transações são verificadas e aprovadas por seus usuários (computadores) distribuídos, criando uma rede de computadores que se comunicam entre si, após isso, as transações são registradas em um bloco de transações. Cada bloco é identificado por um *hash* criptográfico e armazenado em ordem cronológica, os blocos são ligados entre si formando uma cadeia de blocos conforme a Figura 6. Cada computador na rede possui uma cópia completa e atualizada do livro-razão salva localmente, não requerendo uma autoridade centralizadora para o controle de acesso às informações do livro-razão.

É possível classificar dois tipos de implementação de Blockchain: pública ou privada. Na Blockchain pública, qualquer computador pode se conectar a rede Blockchain. Para solucionar a questão de privacidade das informações, surgiu a Blockchain privada, onde o acesso a rede é restrito. Os elementos chaves da Blockchain são: a imutabilidade das informações registradas, descentralização da rede, processamento distribuído e segurança reforçada.

Outra característica importante do Blockchain é a introdução dos *Smart Contract*. Eles representam um novo conjunto de ferramentas e possibilidades. Essencialmente, os *Smart Contract* são pequenos programas de *software* que podem ser implantados na rede do Blockchain, podendo ser executados de várias maneiras, por exemplo: (1) após a transação ter ocorrido no Blockchain; (2) para validar os dados de entrada; (3) escrever os dados no bloco ou (4) executar uma ação antes da transação iniciar.

Figura 6 – Estrutura da Blockchain



Fonte: Elaborado pelo autor.

Vários benefícios são fornecidos pelo Blockchain:

- **Transparência**: Blockchain é transparente porque a implementação elimina a possibilidade de manipulação de dados;
- **Redução de custos**: As transações são ponto a ponto (P2P), desta forma não é necessário nenhum intermediário para completar a transação;
- **Segurança**: Apenas membros autorizados podem acessar e observar as transações, são configurações específicas que precisam ser feitas para entrar na rede;
- **Integridade**: A natureza descentralizada da Blockchain garante que os dados não podem ser modificados e controlados por um único membro da rede;

3 TRABALHOS RELACIONADOS

Este capítulo tem o objetivo de apresentar uma visão geral dos estudos já realizados, tendo como foco dois objetivos: (1) obter um panorama do estado-da-arte sobre estilos arquiteturais de *software* orientados a eventos; e (2) fornecer uma visão geral sobre os estudos realizados com Blockchain focado em aplicações financeiras (LUZ; FARIAS, 2020). Obtendo dessa forma um maior conhecimento dos estilos arquiteturais estudados, bem como da pesquisa e introdução de Blockchain por parte dos pesquisadores e instituições financeiras envolvidas.

Para atingir os objetivos definidos, esse capítulo foi organizado em duas seções principais: a Seção 3.1 apresenta uma revisão da literatura relacionada aos estilos arquiteturais que possuem como base o uso de eventos e a Seção 3.2 apresenta um mapeamento sistemático da literatura tendo como foco o uso de Blockchain nas arquiteturas e aplicações presentes na área financeira.

3.1 Estilos Arquiteturais Orientados a Eventos

Estilos arquiteturais não são fundamentalmente um conceito novo, tampouco os estilos arquiteturais que se utilizam de eventos para comunicação interna. Conforme exposto previamente, o uso de eventos e, principalmente, de arquiteturas orientadas a eventos no contexto das aplicações financeiras, tem ganhado evidência recentemente, apoiado principalmente pela combinação com microsserviços. Sendo o estilo arquitetural orientado a eventos algo relativamente recente, foi identificado que existem oportunidades de pesquisa a fim de preencher as lacunas relacionadas à temática.

3.1.1 Metodologia para Seleção dos Trabalhos

Diante do exposto, uma pesquisa envolvendo as principais bibliotecas digitais de publicação de pesquisa na área de computação foi realizada, com o objetivo de encontrar os trabalhos relacionados especificamente ao estilo arquitetural que faça uso de eventos. Foram consultadas as principais bibliotecas digitais, conforme enumerado na Tabela 1.

Para realizar as buscas nas bibliotecas mencionadas, foi utilizada uma combinação de palavras chaves na *string* de busca, resultando na seguinte expressão:

Tabela 1 – Bibliotecas digitais utilizadas para identificar os trabalhos relacionados

Biblioteca	**Endereço**
ACM Digital Library	http://dl.acm.org
Google Scholar	http://scholar.google.com
IEEE Xplore	http://ieeexplore.ieee.org
Science Direct	https://www.sciencedirect.com
Scopus	https://www.scopus.com
Semantic Scholar	https://www.semanticscholar.org
Springer Link	http://www.springerlink.com/

Fonte: Elaborado pelo autor.

("Software Architecture" ***OR*** *"Architecture Style")* ***AND*** *("Event")*

Após a obtenção dos resultados. o próximo passo foi a aplicação dos filtros para encontrar os estudos que mais se aproximavam com os objetivos deste trabalho. O primeiro filtro foi a busca por estudos escritos no idioma inglês, seguido pela leitura dos resumos dos trabalhos para verificar se o mesmo estava dentro do tema buscado. O segundo passo foi a leitura completa do estudo para a correta obtenção do entendimento da proposta. O terceiro foi a seleção dos estudos que de fato se encaixavam no tema procurado.

Após a aplicação dos critérios de busca, assim como os filtros, foram selecionados **8 artigos** para realizar a análise comparativa dos estudos. A Subseção 3.1.2 apresenta uma breve descrição com a análise de cada trabalho selecionado, enquanto a Subseção 3.1.3 discorrerá sobre os critérios de comparação e as oportunidades identificadas.

3.1.2 Análise dos Trabalhos Relacionados

A Formal Approach to Software Architecture (ALLEN, 1997): Dissertou sobre uma nova abordagem para descrever novos estilos arquiteturais, através de uma linguagem que descreve de forma abstrata os comportamentos da arquitetura, bem como dos componentes e conectores. A abordagem também permite descrever as configurações e os estilos arquiteturais. O estudo foca em uma abordagem mais generalista, não fazendo menção a nenhuma ferramenta de persistência das informações ou outra abordagem que trate a consistência e integridade das informações.

A formally specified framework for elaborating event-based architectural sty-

les correct by design (LOULOU et al., 2006): Propõe uma nova forma para projetar estilos de arquitetura baseado em eventos. Utilizando classes pré-definidas em conjunto com anotações, o estudo busca uma forma de identificar alguns estilos arquiteturais específicos que podem ser atendidos, tendo em vista diferentes estilos e habilitando a criação de novos estilos de arquitetura híbridos. Todavia, o armazenamento dos eventos gerados, bem como a manutenção da integridade das informações não foi endereçada neste estudo.

Compositional specification of event-based software architectural styles (Loulou et al., 2006): Investigou diferentes estilos arquiteturais baseados em eventos para, a partir disso, compor novos estilos arquiteturais ou padrões de arquitetura mais complexos. Os estudos focaram em três estilos de arquitetura: centralizada, completamente distribuída e parcialmente distribuída, para ao final fornecer um guia de como criar estilos de arquitetura genéricos para arquiteturas baseadas em eventos. O propósito do estudo visa atender a uma ampla gama de domínios de negócio. As informações contidas nos eventos de comunicação não foram.

Toward a real-time event flow architecture style (Schwanke, 2001): Apresenta os requisitos que são comuns em arquiteturas baseadas em eventos para sistemas que precisam processar dados em tempo real. Uma revisão de diferentes estilos arquiteturais já existentes é apresentada e, ao final, o autor apresenta de forma resumida três estilos de arquitetura que são catalogados para resolver os desafios de sistemas em tempo real. Com foco específico em um tipo de aplicação, o propósito atende a diferentes domínios, tendo os eventos como foco, não há nenhuma diretriz de como registrar e manter as informações que são processadas a partir dos eventos.

A Message-based Software Architecture Style for Distributed Application (Guozhen Tan et al., 2005): Propõe um novo estilo arquitetural para arquiteturas baseadas em eventos, focando em um componente de suporte para fornecer roteamento das mensagens trafegadas. O armazenamento de forma segura das mensagens que passam pelo roteador ou mesmo nos sistemas não são o foco do estudo. Não são tratados temas como consistência e integridades das informações, sendo um componente genérico que visa atender a diversos domínios.

Event Actors Based Approach for Supporting Analysis and Verification of Event-Driven Architectures (Tran; Zdun, 2013): Investiga e apresenta um nova forma de desenvolvimento de sistemas baseados em eventos, utilizando o modelo baseado em

atores como base para essa proposta. Discorre também sobre uma abordagem formal para especificar esse tipo de arquitetura e sistemas baseados em eventos. Porém, não apresenta nenhuma forma específica de armazenamento dos eventos gerados pelos sistemas, também não especifica como é realizada a consistência das informações utilizadas em arquiteturas de eventos.

A component-and-message-based architectural style for GUI software (Taylor et al., 1996): Apresenta um estilo de arquitetura baseado em eventos para componentes de interface do usuário, permitindo que os componentes possam trocar mensagens de forma assíncrona. A abordagem tem como base aplicações de interface do usuário no estilo cliente-servidor, não focando nos eventos como informações que devem ser armazenadas, sendo de um propósito genérico que visa atender a diferentes áreas.

Architectural styles for runtime software adaptation (Taylor; Medvidovic; Oreizy, 2009): Faz uma revisão de diferentes estilos arquiteturais para verificar as capacidades de fornecer suporte a adaptação da aplicação em tempo de execução. Ao final, é apresentada uma discussão com os prós e contras de cada estilo arquitetural e conclui que a demanda por novos sistemas auto-adaptáveis aumentará e que será necessário investir em mais pesquisas nessa área. O suporte para consistência à integridade das informações não foi mencionado, o estilo arquitetural visa atender a diferentes áreas de negócio sendo uma abordagem generalista.

3.1.3 Análise Comparativa e Oportunidades

Para realizar o processo de comparação dos estudos e identificar similaridade e diferenças, foram definidos cinco Critérios de Comparação (CC). O objetivo da comparação é identificar as oportunidades de pesquisa utilizando critérios objetivos, tais critérios são descritos a seguir:

- **Consistência e integridades dos dados (CC01):** estudos que analisaram ou utilizaram o estilo arquitetural para endereçar questões relacionadas à integridade e consistência dos dados em arquiteturas baseadas em eventos;

- **Blockchain (CC02):** estudos que pesquisaram ou utilizaram algum estilo de arquitetura que utilize dados conectados visando manter a integridade das informações armazenadas;

- **Área específica (CC03):** visa identificar se os trabalhos das pesquisas realizadas são genéricos ou se são aplicadas a uma área específica de negócio.
- **Propósito (CC04):** busca identificar nos estudos se os estilos arquiteturais são de múltiplos propósitos ou de propósito específico;
- **Avaliação empírica (CC05):** esse critério busca verificar se os estudos propostos realizaram alguma avaliação empírica incluindo: estudo de caso, experimento controlado, etc.

A Tabela 2 apresenta a comparação dos estudos selecionados, onde será possível comparar os estudos selecionados com o trabalho proposto nessa pesquisa.

Tabela 2 – Análise comparativa dos trabalhos relacionados selecionados

Trabalho Relacionado	Critério de Comparação				
	CC01	CC02	CC03	CC04	CC05
Trabalho Proposto	●	●	●	●	●
Allen (ALLEN, 1997)	○	○	○	●	●
Loulou *et al.* (LOULOU et al., 2006)	○	○	○	●	●
Loulou *et al.* (Loulou et al., 2006)	○	○	○	●	◐
Schwanke (Schwanke, 2001)	○	○	○	●	◐
Tan *et al.* (Guozhen Tan et al., 2005)	○	○	○	●	◐
Tran and Zdun (Tran; Zdun, 2013)	○	○	○	●	◐
Taylor *et al.* (Taylor et al., 1996)	○	○	○	●	◐
Taylor *et al.* (Taylor; Medvidovic; Oreizy, 2009)	○	○	○	●	○

● Atende Completamente ◐ Atende Parcialmente ○ Não Atende

Fonte: Elaborado pelo autor.

Através da análise comparativa dos estudos selecionados e do estudo proposto, é possível identificar as lacunas e oportunidades de pesquisa que podem ser exploradas, sendo estas descritas a seguir:

1. Nenhum trabalho focou em solucionar o problema da consistência e integridade dos dados quando utilizadas arquiteturas baseadas em eventos;
2. Nenhum trabalho relacionado explorou algum estilo arquitetural que envolva Blockchain;

3. Não foi encontrado nenhum trabalho que estabelecesse um estilo arquitetural para uso da Blockchain.

Dessa forma, considerando-se os critérios de comparação aplicados, apenas o trabalho proposto nessa pesquisa atende a todos os critérios de comparação. Foi identificado uma potencial oportunidade de pesquisa que envolva um estilo arquitetural que contemple os cinco critérios de comparação definidos previamente. Sendo assim, essa oportunidade de pesquisa será explorada nas seções a seguir.

3.2 Mapeamento Sistemático do Uso de Blockchain na Área Financeira

Esta Seção explora a questão de pesquisa QP1, o propósito de um mapeamento sistemático é realizar uma rigorosa revisão da literatura utilizando um protocolo de revisão estabelecido e bem definido, evitando o tendenciamento. O processo de SMS utilizado segue o modelo estabelecido por Petersen, Vakkalanka e Kuzniarz (2015a), que estabelece 3 principais etapas: (1) planejamento, (2) condução e (3) documentação dos resultados. Optou-se pela realização de um mapeamento sistemático da literatura (SMS) relacionado aos estudos de Blockchain na área financeira, pelo fato desta ser uma tecnologia relativamente recente e que começa a ganhar mais atenção nas pesquisas realizadas, demandando por sua vez a obtenção do estado-da-arte acerca da temática, visando a sua maior compreensão.

Devido ao recente crescimento de pesquisas realizadas acerca do Blockchain e ao interesse em particular nesse estudo pelo seu uso na área financeira, o SMS mostra-se dessa forma como uma importante ferramenta, que visa categorizar e resumir os estudos com base nas questões de pesquisas definidas no SMS.

A Subseção 3.2.1 apresenta o planejamento para a realização do SMS, na sequência, a Subseção 3.2.2 discorre sobre a execução do mapeamento. A Subseção 3.2.3 apresenta os resultados obtidos pelo mapeamento, adicionalmente à Subseção 3.2.4 discute as oportunidades de pesquisa encontradas.

3.2.1 Planejamento de Pesquisa

Seguindo o definido pela metodologia em Petersen, Vakkalanka e Kuzniarz (2015a) a primeira etapa para a realização do SMS é o planejamento do estudo que será con-

duzido. Essa etapa importante para garantir a padronização da pesquisa bem como a possibilidade de repetição por outros estudos, nas próximas subseções será apresentado em detalhes as etapas implementadas para a correta condução do mapeamento sistemático.

3.2.1.1 Questões de Pesquisa

Tendo como base o interesse particular sobre o uso de Blockchain na área financeira, as questões de pesquisa foram utilizadas para orientar a condução do mapeamento. Para atingir este objetivo, seis questões de pesquisa (QPs) foram definidas para explorar diferentes aspectos. A Tabela 3 enumera todas as questões de pesquisa usadas neste estudo, assim como descreve seus objetivos e variáveis relacionadas.

Tabela 3 – Questões de Pesquisa (QP), suas descrições e variáveis relacionadas

Questões de Pesquisa	**Descrição**	**Variável**
QP1: Quais plataformas foram utilizadas?	Criar uma lista de plataformas que foram estudadas	Plataformas
QP2: Qual foi a motivação para aplicar a Blockchain?	Esclarecer o que motivou as empresas a considerar a adoção de Blockchain	Propósitos
QP3: Qual foi algoritmo de consenso aplicado?	Identificar o algoritmo de consenso mais utilizados	Algoritmos
QP4: Quais foram as áreas de atuação de cada estudo?	Descobrir a área de cada contribuição dos estudos	Áreas de atuação
QP5: Quais foram os métodos de pesquisa utilizados?	Investigar e entender quais métodos de pesquisa foram empregados pelos pesquisadores	Métodos de pesquisa
QP6: Onde as pesquisas foram publicadas?	Descobrir em quais veículos os estudos foram publicados	Veículos de publicação

Fonte: Elaborado pelo autor.

3.2.1.2 Estratégia de Busca

Esta define os termos de pesquisa que correspondem ao objetivo da pesquisa. Esses termos foram usados para identificar e referenciar os estudos nas bibliotecas digitais. Para selecionar os estudos foi aplicado um processo de busca automática em todas as bases de dados utilizando os termos de busca previamente selecionados. Após a identificação dos estudos, aplicou-se os critérios de inclusão e exclusão dos estudos como referência. Foram aplicadas as seguintes etapas: (1) seleção das bases de dados relacionadas a esta pesquisa; (2) definição da *string* de busca utilizando os principais termos e sinônimos e (3) definição dos critérios de inclusão e exclusão.

Para a seleção dos estudos, foram definidas as bases de dados eletrônicas onde foram publicados os estudos para aplicação do filtro definido. Foram selecionadas seis bases de dados, apresentadas na Tabela 4. As bases de dados selecionadas são relacionadas, ou permitem aplicar filtro em informática ou áreas correlatas. Também foi baseado na cobertura de seu mecanismo de busca, a fim de encontrar periódicos, conferências, artigos mais relevantes para o contexto da pesquisa, etc.

Tabela 4 – Bibliotecas digitais utilizadas no SMS

Biblioteca	**Endereço**
ACM Digital Library	http://dl.acm.org
IEEE Xplore	http://ieeexplore.ieee.org
Science Direct	https://www.sciencedirect.com
Scopus	https://www.scopus.com
Semantic Scholar	https://www.semanticscholar.org
Springer Link	http://www.springerlink.com

Fonte: Elaborado pelo autor.

A *string* de busca utilizada para encontrar os estudos nas bases de dados foi a combinação dos termos primários e seus sinônimos. Essa combinação foi aplicada em cada mecanismo de busca de cada banco de dados. Os resultados foram usados para criar uma lista de estudos que poderiam contribuir para esta revisão sistemática. A *string* de pesquisa e seus sinônimos são apresentados em Tabela 5.

Foi criada uma combinação de termos principais e sinônimos usando os operadores lógicos “AND” e “OR”, resultando na seguinte sentença:

Tabela 5 – Termos primários e seus sinônimos

Termo primário	**Sinônimos**
Architecture	Design
Blockchain	Ledger
Bank	Financial, Fintech, Inter-Bank

Fonte: Elaborado pelo autor.

(“Architecture” **OR** “Design”) AND (“Blockchain” **OR** “Ledger”) AND (“BANK” **OR** “Fintech” **OR** “Financial” **OR** “Inter-Bank”)

3.2.1.3 Critérios de Inclusão e Exclusão

Os estudos encontrados pela *string* de pesquisa aplicada nas bibliotecas digitais resultam em uma coleção de estudos relacionados. Dessa forma, foram definidos os Critérios de Inclusão (CI) e os Critérios de Exclusão (CE) para esta coleção. Os critérios de inclusão foram utilizados para realizar um filtro a fim de encontrar os estudos mais relevantes a serem utilizados no mapeamento sistemático. Os critérios de exclusão foram criados para identificar quais estudos devem ser retirados da mesma.

Critério de inclusão aplicado aos estudos:

- **CI1:** Relacionado com a *string* de busca e as questões de pesquisa;
- **CI2:** Escrito no idioma inglês;
- **CI3:** Publicados até 2020;
- **CI4:** Disponível de forma completa nas bibliotecas digitais;

Após aplicado os critérios de inclusão, foram aplicados os seguintes critérios de exclusão:

- **CE1:** Corresponde com a *string* de busca mas está fora do contexto da pesquisa;
- **CE2:** Não escrito no idioma inglês;
- **CE3:** O título não possui nenhum termo que corresponde à *string* de busca ou o significado do título é totalmente o contrário das questões abordadas nas questões de pesquisa;

- **CE4:** O resumo não abordou nenhum aspecto das questões de pesquisa;
- **CE5:** Apareceu em duplicidade, ou seja, já havia sido selecionado previamente;
- **CE6:** Não está de acordo com as motivações da pesquisa;

3.2.2 Execução do Mapeamento

Após a apresentação dos passos preparatórios, iniciar-se-á de fato a execução do mapeamento. Nessa serão selecionados os estudos iniciais e, com isso, será iniciado o processo de filtragem dos mesmos. Para selecionar os estudos mais representativos, foi desenvolvido um processo de filtragem. Tal processo consiste em realizar 8 etapas para de filtragem dos estudos, sendo estas explicadas a seguir.

- **Etapa 1: Pesquisa Inicial**. Pesquisa e coleta dos resultados dos estudos da consulta aplicada nos motores de busca usando a *string* de pesquisa definida na Subseção 3.2.1.2. Nesta etapa foram encontrados **1884 estudos** nas bases de dados definidas na Tabela 4.
- **Etapa 2: Remoção de impurezas (CE1 e CE2)**. Aplicou-se os critérios de exclusão CE1 e CE2 para excluir os estudos que não se enquadram no contexto deste mapeamento sistemático.
- **Etapa 3: Filtrar por Título e Resumo (CE3 e CE4)**. Aplicou-use o CE3 e o CE4 para remover os estudos que retornaram como resultado da *string* de pesquisa. Nesta etapa, foram retirados os estudos que não se enquadrassem nos critérios CE1 e CE2 e não estivessem relacionados às questões de pesquisa.
- **Etapa 4: Combinação dos estudos**. Os estudos filtrados foram reunidos na mesma coleção para prosseguir para a próxima etapa.
- **Etapa 5: Remoção de Duplicatas (CE5)**. Considerando que um estudo pode ser encontrado em mais de uma base de dados, nesta etapa foi aplicado o CE5 para retirar todos os estudos duplicados.

- **Etapa 6: Adição por Heurísticas**. Para aumentar o número de estudos relevantes, os autores decidiram adicionar alguns estudos aplicando o *backward snowballing* (PETERSEN; VAKKALANKA; KUZNIARZ, 2015b). Esses estudos foram incluídos de acordo com os critérios de inclusão e exclusão.

- **Etapa 7: Filtragem por Texto Completo**. Todos os estudos foram lidos na íntegra para aplicação do CE6, excluindo os estudos que não estavam relacionados à questão de pesquisa ou à engenharia de *software*.

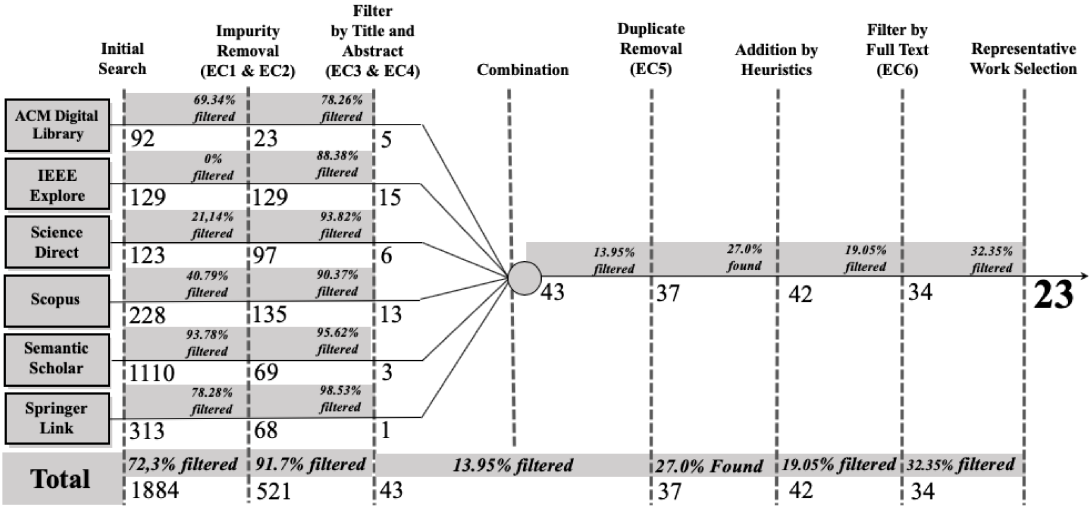
- **Etapa 8: Seleção dos estudos**. Seleção de todos os estudos filtrados como estudos primários para responder às QPs.

A Figura 7 mostra o resultado de cada etapa do processo de filtragem. Na busca inicial foram selecionados **1884 estudos**, na segunda etapa, após a aplicação dos critérios de remoção de impurezas, foram filtrados **521 estudos**, o que resultou em um total de 1363 estudos removidos. Seguindo o processo de filtragem, todos os títulos e resumos dos estudos foram revisados para verificar se condiziam com esta pesquisa, restando **43 estudos**. Na terceira e quartas etapas, os estudos foram combinados para remover as duplicatas, pois o mesmo estudo pode retornar de bases de dados diferentes, garantindo que apenas um estudo fosse utilizado. Depois de remover as duplicatas, mais 5 estudos foram adicionados por heurística, esses estudos não foram retornados dos bancos de dados na primeira pesquisa, e foram adicionados aplicando *backward snowballing*. No filtro de texto completo, todo o conteúdo estudado foi revisado, **34 estudos** foram filtrados nesta etapa. Na última etapa, foram selecionados os estudos mais representativos que se enquadraram nesta pesquisa, como resultado, foram encontrados **23 estudos primários**.

3.2.3 Resultado do Mapeamento

Esta seção apresenta os resultados coletados em relação às questões de pesquisa formuladas, cada QP é explicada detalhadamente abaixo.

Figura 7 – Processo de filtragem dos estudos primários



Fonte: Elaborado pelo autor.

3.2.3.1 QP1: Quais plataformas foram utilizadas?

A Tabela 6 mostra os resultados. O objetivo desta QP foi descobrir quais são as plataformas mais utilizadas em maior escala atualmente. Considerando que existe um grande número de tecnologias e algumas delas estão em fase de adoção inicial, ou já foram testadas pela comunidade e não apontam para este caminho, descartando o uso de tal tecnologia.

Tabela 6 – QP1: Quais plataformas foram utilizadas?

Plataformas	**Estudos**	**Percentual**	**Estudos Primários**
Ethereum	10	43.48%	E01, E02, E03, E04, E05, E06, E07, E08, E09, E10
Hyperledger Fabric	6	26.09%	E11, E12, E13, E14, E15, E16
Not Informed	3	13.04%	E17, E18, E19
BeihangChain	1	4.35%	E20
Hydrachain	1	4.35%	E20
Proprietary	1	4.35%	E21
QTUM	1	4.35%	E22
Tendermint	1	4.35%	E23

Fonte: Elaborado pelo autor.

A plataforma de Blockchain mais usada foi a Ethereum (ETHEREUM.ORG, 2019). A Ethereum é uma plataforma de Blockchain *open-source* criada por Vitaly Dmitriyevich, baseada no Bitcoin criado por Satoshi Nakamoto (NAKAMOTO, 2008), mas com modificações. Ao usar uma rede pública, a Ethereum pode executar milhões de transações por dia, suportando a execução de *smart contracts*.

A segunda plataforma de Blockchain mais usada é o Hyperledger Fabric (HYPERLEDGER.ORG, 2019). Esta é uma plataforma de Blockchain de código-aberto descrita de acordo com o site como *"uma estrutura de livro-razão distribuída com permissão de nível empresarial para o desenvolvimento de soluções e aplicativos. Seu design modular e versátil atende a uma ampla gama de casos de uso da indústria"*. Criado pela Linux Foundation, em 2016, com a combinação de 2 bases de código, uma dessas criada pela IBM e a outra pela Intel. No momento, o Hyperledger Fabric é um dos muitos projetos sob o guarda-chuva Hyperledger.

Finalmente, as outras plataformas de Blockchain mencionadas por outros estudos não têm toda a fama das primeiros concorrentes. O QTUM (QTUM, 2019) é uma plataforma de Blockchain *open-source*, baseada na Blockchain Ethereum, prometendo um desempenho significativo comparado às outras plataformas. O QTUM também é uma *criptomoeda*. O Hydrachain (HYDRACHAIN, 2019) é outro Blockchain baseado em Ethereum e BeihangChain, ele usa votação bizantina e a coleta de dados é realizada concomitantemente para aumentar a velocidade do processo, portanto, possui um bloco único para o processo de criação (TSAI et al., 2016). O Tendermint (TENDERMINT.COM, 2019) é outra plataforma de Blockchain de código-aberto, ao contrário de outras plataformas, o Tendermint não é baseado em Ethereum e tem suas próprias implementações de plataforma de Blockchain. Também é um algoritmo de consenso. Um dos estudos primários relatou que usou uma implementação de Blockchain proprietária.

3.2.3.2 QP2: Qual foi a motivação para aplicar a Blockchain?

O objetivo desta QP foi identificar a motivação por trás das decisões de testar e aplicar o Blockchain, conforme os resultados apresentados na Tabela 7, além de verificar se os recursos fornecidos pelas plataformas Blockchain atendem aos requisitos das aplicações financeiras em geral. Neste QP foi possível identificar as motivações dos estudos para aplicar as funcionalidades da Blockchain para aumentar a segurança das

informações armazenadas.

Tabela 7 – QP2: Qual foi a motivação para aplicar a Blockchain?

Propósito	**Estudos**	**Percentual**	**Estudos Primários**
Aumento de segurança	14	60.87%	E01, E04, E06, E08, E09, E10, E11, E13, E15, E16, E18, E19, E22, E23
Aumento de transparência	8	34.78%	E01, E03, E07, E12, E14, E15, E16, E19
Solução descentralizada	5	21.74%	E04, E15, E16, E19, E21
Integridade das informações	5	21.74%	E03, E04, E06, E17, E21
Prevenção a fraudes	4	17.39%	E01, E17, E20, E22
Compartilhamento seguro de informações	4	17.39%	E10, E12, E13, E14
Garantia de imutabilidade	2	8.70%	E12, E23
Privacidade das informações	2	8.70%	E07, E14
Evitar transações duplicadas	1	4.35%	E23

Fonte: Elaborado pelo autor.

A preocupação com a segurança (14, 60,87%) é uma motivação para o uso do Blockchain, tendo em visto que é uma das áreas que mais tem recebido atenção. No entanto, é claro ao observar esta QP que mais possibilidades de uso da Blockchain podem ser exploradas, como por exemplo a imutabilidade, sendo esta explorada em dois (8.70%) estudos apenas.

3.2.3.3 QP3: Qual foi algoritmo de consenso aplicado?

O objetivo desta QP foi saber qual algoritmo de consenso foi aplicado, sendo os resultados apresentados na Tabela 8. Tendo em vista que o consenso é um dos aspectos mais importantes da Blockchain, este QP mostra que os estudos são notavelmente frágeis nesse aspecto. A maioria dos estudos não apresenta o consenso aplicado, não sendo possível afirmar se eles tiveram a intenção de não informar ou se foi apenas por falta de conhecimento. Estes estudos utilizaram a implementação padrão da plataforma escolhida.

Por exemplo, o algoritmo de consenso Kafka utilizado no Hyperledger Fabric, é

um algoritmo baseado em uma votação, que fornece tolerância à falhas de travamento e bom desempenho, e que não é um tolerante à falhas do tipo Bizantina, o que impede o sistema de chegar a um acordo no caso de nós maliciosos ou defeituosos (HYPERLEDGER.ORG, 2017). A implementação do consenso Kafka usa o Apache Kafka (APACHE, 2019), uma plataforma de *streaming* distribuída.

Tolerância de falha Bizantina (PBFT)[1] é um algoritmo de consenso genérico, que permite que os sistemas distribuídos continuem funcionando mesmo que um nó na rede Blockchain esteja com defeito ou atue como um nó malicioso. Esses problemas são comuns em sistemas distribuídos, entretanto, com esta implementação, os nós honestos chegam a um consenso e a rede não é afetada por um nó malicioso ou com defeito.

Tabela 8 – QP3: Qual foi algoritmo de consenso aplicado?

Algoritmos	**Estudos**	**Percentual**	**Estudos Primários**
Não informado	15	65.22%	E01, E02, E03, E04, E05, E06, E07, E08, E09, E10, E14, E17, E18, E19, E20
Kafka	2	8.70%	E13, E15
PBFT	2	8.70%	E16, E21
Solo	2	8.70%	E11, E12
Proof of Stake	1	4.35%	E22
Tendermint	1	4.35%	E23

Fonte: Elaborado pelo autor.

Proof of Stake (PoS)[2] representa uma classe de algoritmos de consenso em que os validadores votam no próximo bloco, e o peso da votação depende do tamanho de sua aposta. É considerado uma melhoria em relação ao algoritmo Proof of Work (PoW)[3] por causa do menor consumo de energia, riscos de centralização reduzidos, segurança contra diferentes tipos de ataques de 51% e mais (ETHEREUM.ORG, 2019). Pode ser aplicado em plataformas de Blockchain Ethereum.

Solo é um algoritmo de consenso simples para o Hyperledger Fabric, ele assim chamado porque executa uma única instância do serviço de pedidos. É útil para o desenvolvimento, mas não é recomendado para ambiente de produção, visto que será a

[1] `https://sawtooth.hyperledger.org/docs/pbft/releases/latest/architecture.html`

[2] `https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/`

[3] `https://ethereum.org/en/developers/docs/consensus-mechanisms/pow/`

única instância do serviço de pedidos na rede, caracterizando assim um ponto único de falha (SPOF).

Tendermint (TENDERMINT.COM, 2019) é uma replicação de máquina de estado tolerante a falhas bizantinas (BFT). Alega-se que o Tenedermint tolera até 1/3 de máquinas falhando arbitrariamente e pode replicar máquinas de estado determinístico escritas em qualquer linguagem de programação. O consenso fornecido pelo Tendermint pode ser aplicado para trabalhar em Ethereum, Hyperledger Fabric e outras plataformas de Blockchain.

3.2.3.4 QP4: Quais foram as áreas de atuação de cada estudo?

O objetivo deste QP foi descobrir quais áreas foram usadas para aplicar o Blockchain. Os resultados apresentados na Tabela 9 sugerem que na área financeira existe muitos setores e oportunidades. Foi possível identificar que a maioria das pesquisas foi aplicada em bancos ou *fintechs*.

Tabela 9 – QP4: Quais foram as áreas de atuação de cada estudo?

Áreas de Atuação	**Estudos**	**Percentual**	**Estudos Primários**
Bancos ou *Fintechs*	12	52.17%	E02, E05, E09, E10, E12, E13, E16, E17, E19, E20, E21, E23
Concessão de empréstimos	4	17.39%	E06, E07, E14, E15
Investimento	3	13.04%	E03, E04
Sistema de pagamento	2	8.70%	E22
Seguro	1	4.35%	E11
Pagamento de impostos	1	4.35%	E01

Fonte: Elaborado pelo autor.

O item Bancos ou *Fintechs* introduziram a Blockchain no caminho crítico transacional, ou seja, as transações básicas do sistema financeiro como depósitos, retiradas e transferências estava de alguma forma envolvidas com a Blockchain. Embora a maioria dos estudos tenha sido aplicada em Bancos ou *Fintechs* (12, 52,17%), o resultado apresenta uma diversidade de estudos de Blockchain em outras subáreas.

Enquanto, "Concessão de empréstimos"e "Investimento"representam 17,39% e 13,04% dos estudos, respectivamente, somados esses 2 itens representam 30,43% do total. Isso

demonstra que mais oportunidades de uso da Blockchain têm sido exploradas, e que o espectro de atuação pode ser ainda maior.

Essa diversidade mostra o potencial da Blockchain, em conjunto com os *smart contracts* ações automatizadas podem ser executadas acelerando a execução dos processos e aumentando a transparência dos contratos executados. Um *smart contract* é um código executável que atua dentro da Blockchain para executar e fazer cumprir os termos de um acordo entre partes que não se confiam. Pode ser pensado como um sistema que libera ativos digitais para todas ou algumas das partes envolvidas, uma vez que as regras predefinidas foram atendidas (BUTERIN, 2019).

3.2.3.5 QP5: Quais foram os métodos de pesquisa utilizados?

A Tabela 10 mostra os resultados desta QP. O objetivo da mesma foi descobrir quais os métodos de pesquisa foram aplicados para cada estudo. Foi possível identificar que a maioria dos estudos (65,22%) caracterizavam-se pelo desenvolvimento de um protótipo.

Tabela 10 – QP5: Quais foram os métodos de pesquisa utilizados?

Métodos de pesquisa	**Estudos**	**Percentual**	**Estudos Primários**
Protótipo	15	65.22%	E01, E02, E03, E04, E06, E07, E08, E09, E11, E12, E13, E14, E15, E19, E21
Framework	4	17.39%	E10, E16, E18, E23
Estudo de caso	3	13.04%	E17, E20, E22
Design	1	4.35%	E05

Fonte: Elaborado pelo autor.

Os protótipos representam a maioria dos métodos aplicados, os mesmos foram desenvolvidos em um contexto controlado para solucionar ou provar a aplicabilidade das plataformas de Blockchain e a solução projetada. O que demonstra o quão complexo para arquitetos e desenvolvedores é introduzir essa nova tecnologia nas arquiteturas existentes, visto que a maioria dos estudos ainda busca testar e provar suas hipóteses dado que ainda não existe um padrão ou estilo arquitetural que possa ajudar nessa tarefa.

Foi observado que os "*Frameworks*"(17,39%) possuem poucos estudos, ao todo quatro, assim como o item "*Design*"com 4,35% e somente um estudo. O que sugere que

estudos buscando uma solução genérica e mais ampla são necessários, auxiliando o time de desenvolvimento na concepção e produção de novas soluções. Tais componentes possuem uma enorme importância dado que que podem ser reutilizados em outras situações, enquanto o protótipo fornece um exemplo de uso das plataformas de Blockchain para uma determinada situação. O "Estudo de Caso" apresenta o estado atual do Blockchain (13,04%), os casos de uso relatam os resultados do uso da Blockchain na prática e reforçam que é uma tecnologia que tende a ser utilizada com maior frequência.

3.2.3.6 QP6: Onde as pesquisas foram publicadas?

O objetivo desta QP foi quantificar onde os estudos primários foram publicados. A Tabela 11 mostra os resultados obtidos.

Tabela 11 – QP6: Onde as pesquisas foram publicadas?

Veículos de Publicação	**Estudos**	**Percentual**	**Estudos Primários**
Conferência	16	69.57%	E02, E03, E04, E06, E07, E09, E10, E11, E12, E13, E17, E18, E19, E21, E22, E23
Revista	7	30.43%	E01, E05, E08, E14, E15, E16, E20

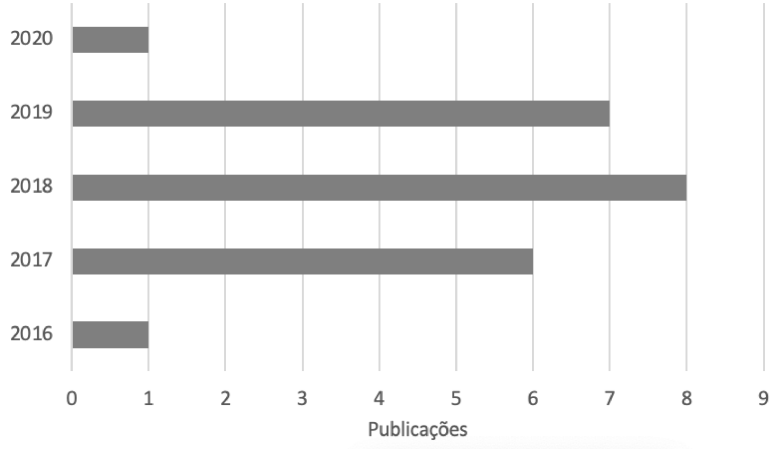
Fonte: Elaborado pelo autor.

Foi possível identificar que a maioria dos estudos foi publicada em conferências (69,57%), o que indica um estágio de incipiência dos estudos identificados. Ademais, poucos destes foram publicados em revistas (30,43%), apesar representar uma quantidade menor, tais estudos são importantes para compartilhar o conhecimento em veículos específicos consolidando tanto os estudos realizados como os resultados obtidos. Mediante a Figura 8, é possível observar-se um recente crescimento no interesse pelo tema de Blockchain na área financeira até a data do desenvolvimento deste trabalho.

3.2.4 Oportunidades de Pesquisa

Esta seção apresenta discussões e as oportunidades de pesquisa que foram identificados por meio da análise dos estudos primários selecionados. Além das lacunas

Figura 8 – Quantidade de publicações por ano



Fonte: Elaborado pelo autor.

apontadas em 1.1 algumas oportunidades de pesquisa foram encontradas a partir do mapeamento sistemático sendo apresentadas abaixo, tais oportunidades de pesquisa serão exploradas ao longo dessa dissertação.

(1) Como utilizar Blockchain em uma aplicação orientada a eventos. A maior parte dos estudos foram conduzidos utilizando aplicações que predominantemente seguem o modelo de comunicação síncrono. Tornando dessa forma as aplicações mais suscetíveis ao acoplamento e aumentando a dependência entre os componentes que compõem a arquitetura.

Atualmente, a utilização de arquiteturas que adotam o estilo arquitetural orientado a eventos já é uma realidade, sendo empregado no desenvolvimento de novos *softwares*. Dessa forma as pesquisas futuras devem responder as seguintes questões: (1) Como integrar aplicações orientada a eventos com a implementação de Blockchain? e (2) Como tirar melhor proveito dos eventos gerados pelas aplicações que utilizando o estilo arquitetural orientado a eventos?

(2) Métodos de pesquisa. O método de pesquisa aplicado na maior parte dos estudos foi o protótipo. Neste caso, os estudos visam desenvolver um projeto e uma implementação para resolver o problema em seus contextos. Porém, cada pesquisa apontou uma solução para o caso específico. Nesta situação o ambiente foi controlado e o problema faz parte de algo maior.

Falta de estudos que diversifiquem os métodos de pesquisa aplicados, sendo identi-

ficados apenas dois estudos de caso e dois estudos acerca de *frameworks*. Esse número é inexpressivo para a área financeira, quando comparado aos muitos protótipos desenvolvidos. É necessário observar as aplicações atuais utilizadas no contexto financeiro, como a aplicação de *core banking*, a prevenção à lavagem de dinheiro, os sistemas de pagamentos, entre outros, e analisar como esses sistemas interagirão com o Blockchain, um novo paradigma comparando com o estado atual.

Para pesquisas futuras, é necessário responder às seguintes questões: (1) Qual é o método de pesquisa ideal para mostrar potencial do Blockchain? e (2) Qual estilo de *design* deve ser aplicado para permitir a transição para plataformas de Blockchain?

(3) Estilo arquitetural. Foi possível identificar a necessidade de mais estudos que cubram um problema mais genérico e abordem a solução de projeto de forma mais moderna. Para aplicar a Blockchain em larga escala é necessário antes estabelecer um estilo de arquitetura e desenvolvimento de *software* adequado, tanto em uma solução de Blockchain quanto em aplicações que pretendam interagir com a Blockchain. Para expandir a adoção do Blockchain, é importante definir o *design* entre este e o mundo externo. A Engenharia de *Software* Orientada a Blockchain (BOSE) (Porru et al., 2017) lista algumas práticas que precisam ser abordadas, como teste de *smart contracts*, teste de transação de Blockchain e os critérios para selecionar a implementação de Blockchain adequada.

Por natureza, as aplicações financeiras são críticas e, em geral, funcionam 24 horas, 7 dias por semana, envolvendo diversos sistemas complexos. Nos sistemas financeiros existe muito sistemas legados, requisitos regulatórios, integração entre sistemas, requisitos de segurança, entre outros. Este cenário pode levar os aplicativos de Blockchain a vários *corner cases* ou *edge cases*. Para evitar situações inesperadas e evitar um grande esforço para implementar e testar novas soluções usando Blockchain, é importante desenvolver um conjunto de estilos de arquitetura e desenvolvimento para lidar com esses desafios.

O projeto da arquitetura de *software* para um *software* financeiro típico lida com concorrência, alta disponibilidade e tolerância a falhas, não é por acaso que o Blockchain também lida com esses requisitos. Mas agora, o estilo de arquitetura mudou com a introdução do Blockchain. Algumas etapas para construir arquiteturas de *software* descentralizadas híbridas são apresentadas em Wessling et al. (2019).

A pesquisa futura deve responder às seguintes questões: (1) Como escolher o imple-

mentar um estilo de arquitetura para usar a Blockchain em uma aplicação financeira?; (2) Quais são os projetos de arquitetura de que os *softwares* financeiros precisam para se comunicar com os componentes da Blockchain? e; (3) Quais são os requisitos que influenciam no estilo de arquitetura para aplicar Blockchain em aplicações financeiras?

Sumário para a QP 1: Foi realizado um comparativo dos trabalhos relacionados com o presente estudo. Na sequência foi realizado um mapeamento sistemático do uso de Blockchain em arquiteturas de *software* na área financeira. Ao final foi possível identificar as lacunas no uso de estilos arquiteturais que fazem uso de eventos bem como a aplicação desses estilos junto com a Blockchain na área financeira.

4 ESTILO ARQUITETURAL PROPOSTO

Este Capítulo apresenta o EventChain, o qual trata-se de um estilo arquitetural baseado em cadeia de eventos para o suporte ao desenvolvimento de aplicações na área financeira. Tal Capítulo busca responder a questão de pesquisa QP2.

Sendo organizado da seguinte forma: a Seção 4.1 apresenta uma visão geral onde são apontados os principais requisitos que o estilo arquitetural irá contemplar, bem como o foco principal de atuação. A Seção 4.2 traz de forma detalhada, o papel e funcionamento do Resolvedor de Eventos e da Cadeia de Eventos. A Seção 4.3 introduz as restrições arquiteturais que caracterizam o estilo arquitetural e a Seção 4.4 apresenta os aspectos de implementação do estilo arquitetural.

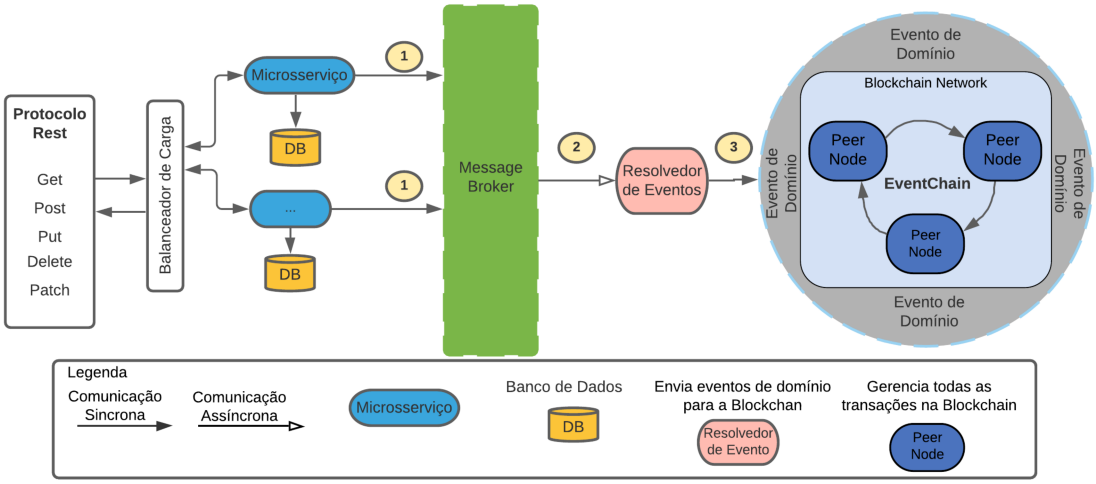
4.1 Visão Geral do Estilo Arquitetural Proposto

O EventChain trata-se de um estilo arquitetural que transforma eventos de baixa ordem gerados pelas aplicações em eventos de mais alta ordem, armazenando os eventos de mais alta ordem transformados e com mais valor de negócio de forma imutável na Blockchain. A Figura 9 apresenta uma visão geral do estilo arquitetural proposto, destacando a sua estrutura, seus principais conceitos e transições entres tais conceitos. Para gerar os eventos de mais alta ordem e armazená-los de forma imutável, o estilo proposto define três etapas:

- **Etapa 1: Eventos de Domínio.** Esta etapa tem o propósito de ser a primeira camada que recebe as requisições. Utilizando o protocolo REST (FIELDING; TAYLOR, 2000) como porta de entrada das requisições, as aplicações distribuídas no formato de microsserviços (RODGERS, 2005) processam as requisições guardando as informações no banco de dados e gerando eventos. Os eventos são enviados ao *broker* de mensageria à medida que cada interação ocorre, podendo ser uma interação do usuário ou de um microsserviço com outro.

- **Etapa 2: Resolvedor de Eventos.** Esta etapa tem o propósito de analisar e inspecionar os eventos de mais baixa ordem gerados pelas aplicações, em busca de resolver os eventos que serão agregados em um evento de mais alta ordem. Focando nos eventos de mais baixa ordem os resolvedores de eventos são micros-

Figura 9 – Visão geral do estilo arquitetural proposto



Fonte: Elaborado pelo autor.

serviços especializados que observam tais eventos (ou a composição de múltiplos eventos), visando mapeá-los para um evento com mais valor agregado, o evento de mais alta ordem. Após realizar o mapeamento, os eventos de mais alta ordem (mais próximos do negócio) são inseridos no Blockchain, dando origem à uma cadeia de eventos (ou fatos imutáveis de domínio ou negócio). O Resolvedor de Eventos implementa uma estratégia para descoberta automática de eventos de baixa ordem, para que ao se conectar no *Message Broker*, consiga processar as mensagens sem necessidade de alterar as aplicações envolvidas na Etapa 1.

- **Etapa 3: Cadeia de Eventos.** O propósito dessa etapa é persistir de forma imutável as informações contidas nos eventos de mais alta ordem. Formando dessa forma uma sequência de informações ligadas entre si. Nessa etapa, os eventos de domínio são recebidos e armazenados na Blockchain. Após a confirmação de que a informação foi persistida, novos eventos são gerados dentro da Blockchain para que componentes especializados que ficam ouvindo tais eventos possam realizar alguma ação, dando sequência ao fluxo e caracterizando a cadeia de eventos. A cadeia de eventos torna os fatos do domínio de negócio imutáveis ao serem registrados no Blockchain, ao mesmo tempo assegura aspectos relacionados a questões de segurança — fator crítico na área financeira.

As etapas 2 e 3 serão apresentadas de forma detalhada na próxima seção.

4.2 Resolvedor de Eventos e Cadeia de Eventos

Os **resolvedores de eventos** são microsserviços especializados que observam os eventos gerados a partir dos microsserviços e os mapeiam para eventos mais alta ordem. Isto é, dado um conjunto de eventos de mais baixa ordem o resolvedor mapeia para um evento de mais alta ordem. Eventos de baixa ordem são eventos que possuem somente a informação que possuem, se limitando a trafegar as informações entre os componentes da arquitetura. Enquanto, os eventos de mais alta ordem são eventos que possuem mais valor de negócio, são gerados a partir da combinação de outras informações para gerar uma terceira informação e que será persistida na Blockchain.

Os eventos gerados pelas aplicações podem ser divididos em duas grandes categorias: (1) Eventos de domínio ou de mais alta ordem e (2) Eventos de aplicação ou baixa ordem. Os eventos de domínio são definidos por Eric Evans (EVANS, 2003) como algo que aconteceu e que os especialistas de domínio deveriam se preocupar. De acordo com o (VERNON, 2013), os eventos de mais alta ordem são utilizados para capturar a ocorrência de algo que aconteceu com o domínio. Dessa forma, quando um evento desse tipo é criado normalmente ele é transmitido para sistemas externos e as ocorrências do domínio devem ser seguidas através de contextos limitados. Por outro lado, os eventos de aplicação ou de baixa ordem são eventos que tem como objetivo comunicar outro sistema ou aplicação dentro do seu contexto delimitado ou não para que realizem alguma ação, tal tipo de evento não impacta o domínio de forma que pode ser gerado livremente. Normalmente os eventos de baixa ordem são utilizados por uma aplicação com o objetivo de se comunicar com outra de forma assíncrona, permitindo enfileirar mensagens para posterior processamento, uma sequência de eventos de aplicação pode resultar na geração de um evento de domínio modificando o mesmo de forma definitiva.

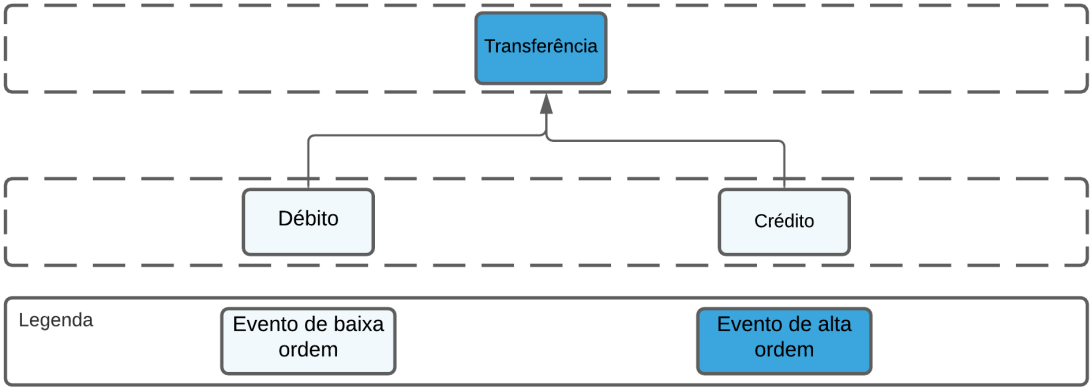
A Figura 10 apresenta um exemplo de fluxo de geração de eventos bem como a divisão entre cada tipo de evento. O evento de mais alta ordem *transferência*, é gerado a partir da resolução de dois eventos de baixa ordem, *Débito* e *Crédito*. Esse componente implementa uma estratégia para descoberta automática dos eventos baseado em alguma condição, para que ao se conectar no *Message Broker*, consiga processar tais eventos sem necessidade de alterar as aplicações envolvidas na Etapa 1 da Figura 9.

Ao mapear um evento de mais alta ordem, como demonstrado na Figura 10, é possível extrair informações que não estavam no contexto de nenhuma das aplicações. O processo de resolução pode ser representando pelo Algoritmo 1, que detalha a lógica para resolver uma combinação dos eventos de baixa ordem em um evento de mais alta ordem. Dessa forma quando dos eventos se combinam dado algum critério especificado, o resolvedor recebe a lista dos eventos (linha 1), identifica quais os eventos se combinem (linha 3 a 7) e chama o *smart contratc* para gerar o evento de mais alta ordem (linha 9).

Algoritmo 1: Resolvedor de eventos. Exemplo evento de transferência.

```
1  function resolverEventos(eventosDeCredito, eventosDeDebito)
2      // Para cada evento de crédito
3      foreach evtCred in eventosDeCredito do
4          // Compara com o evento de débito
5          foreach evtDebt in eventosDeDebito do
6              // Identifica o evento correspondente
7              if evtCred.recibo == evtDebt.recibo then
8                  // Gera um evento de mais alta ordem
9                  return evtTransferencia
10         end
11     end
12 end
```

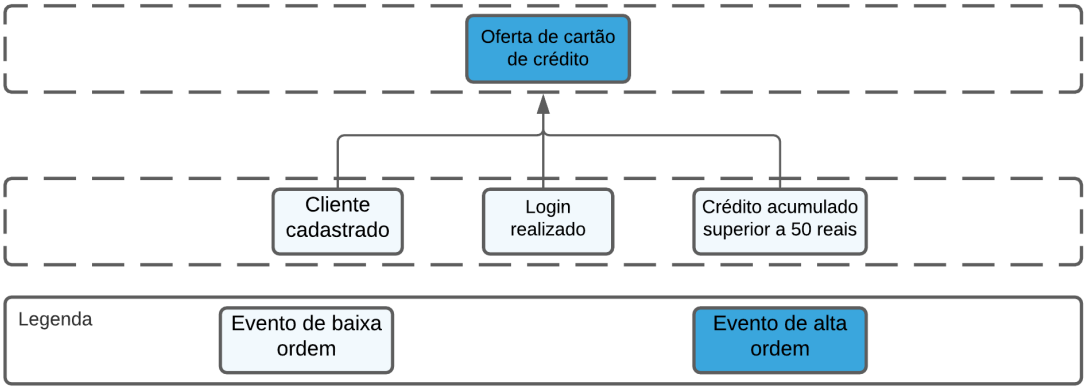
Figura 10 – Fluxo de eventos



Fonte: Elaborado pelo autor.

Conforme mencionando na Seção 2.5, o uso de eventos de domínio é tipicamente utilizado em arquiteturas orientadas a eventos. A Figura 11 apresenta a estratégia de implementação dessa funcionalidade utilizando o estilo arquitetural proposto nesse estudo, ao interpolar três eventos de baixa para gerar um evento um de alta ordem. Gerando um evento que proporciona fazer uma oferta ao cliente de acordo com uma sequência de eventos.

Figura 11 – Fluxo de eventos complexos



Fonte: Elaborado pelo autor.

A **cadeia de eventos** é o elemento central do estilo arquitetural proposto, é onde todos os eventos resolvidos pelo resolvedor de eventos são armazenados de forma imutável. Dessa forma os especialistas de domínio podem observar os eventos e tomar ações, a cadeia de eventos se concretiza através de *Smart Contracts* que são disponibilizados para acesso externo através de API's. Ao receber uma requisição um *Smart Contract* tem a responsabilidade de validar a estrutura da informação, o conteúdo e as demais parâmetros recebidos. Após a gravação das informações na *Blockchain* a cadeia de eventos continua seu fluxo, um evento é gerado após a confirmação da gravação dos dados internamente pela própria *Blockchain*, ouvintes de eventos de confirmação escutam por esses eventos podendo tomar ações e a partir desse momento gravar novas informações gerando novos eventos, seguindo sucessivamente esse fluxo.

Aos persistir os eventos de mais alta ordem como ilustrado na Figura 10, uma sequência de eventos de *transferência* serão armazenadas de forma conectada. Permitindo assim que outras ações possam ser realizadas, seja por pessoas ou por sistemas de forma automatizada. Podendo ser útil no contexto de aplicações financeiras para

monitoramento de sistemas, monitoramento de comportamento do usuário ou trilhas de auditoria, prevenção a fraudes ou prevenção à lavagem de dinheiro.

A Figura 12 apresenta um exemplo de evento de mais alta ordem registrada na Blockchain. Eventos de mais alta ordem oferecem uma nova perspectiva de visão das informações, permite novas possibilidades de desenvolvimento para os arquitetos e desenvolvedores ao habilitar o desenvolvimento de aplicações assíncronas e seguras ao persistir as informações de forma imutável. O Algoritmo 2 apresenta o processo para registrar um evento de mais alta ordem na Blockchain. Ao receber o evento de transferência (linha 1), o algoritmo solicita uma instância do *smart contract*, que busca uma instância da classe que manipula o estado da Blockchain (linha 5) e por fim realiza o adição da informação na Blockchain (linha 7).

A implementação do estilo arquitetural proposto pode ser visualizada no repositório de código do GITHUB em **https://github.com/Eventchain-Example/exemplo-base**. O exemplo contém o código fonte do protótipo, das classes que compõem a arquitetura da solução, diagramas e toda a documentação produzida. Através do arquivo README.md é possível navegar entre os projetos e a documentação.

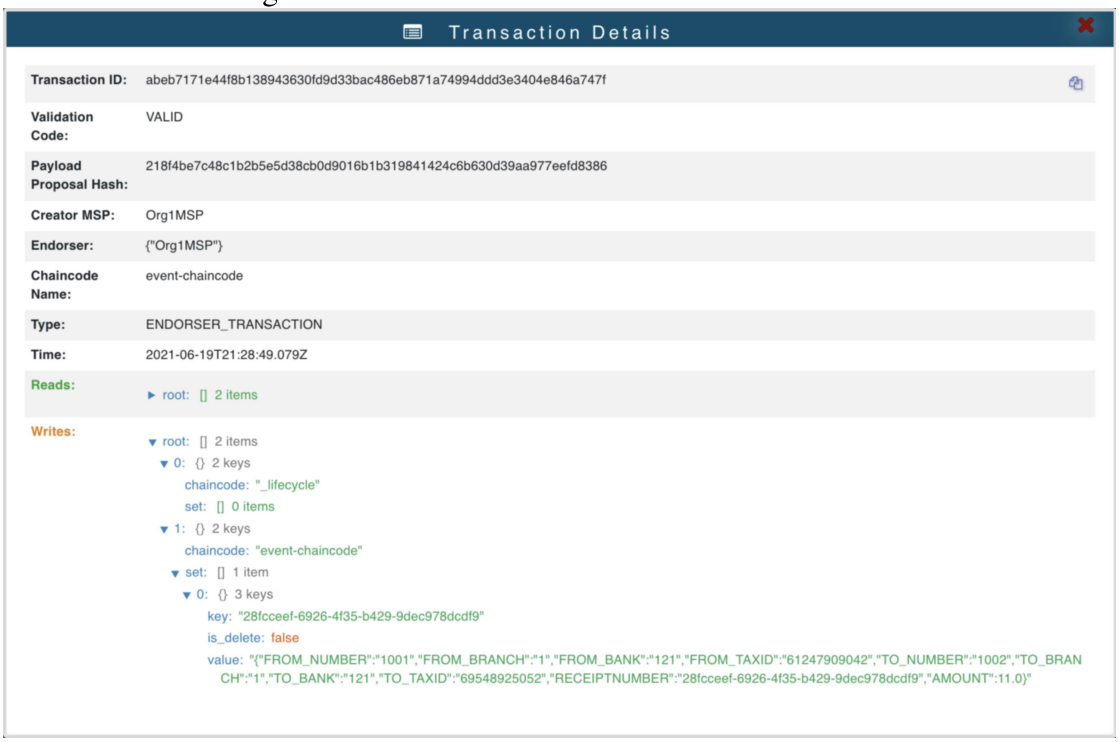
Algoritmo 2: Cadeia de eventos

```
1  procedure saveToEventChain(evtTransferencia)
2      // Localiza o smartContract
3      smartContract ← getTransferSmartContract()
4      // Busca o bloco correspondente na Blockchain
5      transferLedger ← smartContract.getLedger()
6      // Adiciona a informação ao bloco
7      transferLedger.append(evtTransferencia)
8  end
9  function getTransferSmartContract()
10     accountTransferChaincode ← network
11     return accountTransferChaincode
12 end
```

4.3 Restrições Arquiteturais

Esta seção descreve as restrições arquiteturais definidas no estilo arquitetural *EventChain*. Essas restrições buscam criar normas que irão definir como a arquitetura de uma

Figura 12 – Evento de transferência na Blockchain



Fonte: Elaborado pelo autor.

aplicação deverá ser estruturada, auxiliam arquitetos e desenvolvedores na tomada de decisões. As restrições podem também ser vistas como uma forma de definir características comuns para arquiteturas (di Nitto; Rosenblum, 1999). As restrições são:

- **Uso de token JWT:** O estilo arquitetural faz uso de comunicação HTTP e para garantir a segurança dos serviços expostos o mecanismo de autenticação escolhido é o de token JWT[1]. Os arquitetos e os desenvolvedores rotineiramente precisam definir e implementar mecanismos de segurança. De acordo com Ahmed e Mahmood (2019) o uso de token JWT contribui para diminuir o risco de ataques de predição, ataques de imagens pré-definidas e o risco de atacantes assumirem perfis indevidos em aplicações WEB. Essa restrição além de acrescentar confiabilidade e segurança na troca de mensagens, tem como objetivo utilizar um mecanismo de autenticação performático sendo, de acordo com Albertengo et al. (2020), um mecanismo de autenticação para *web services* REST bastante performático.

[1] `https://tools.ietf.org/html/rfc7519`

- **Comunicação assíncrona e *message broker*:** A segunda restrição adicionada ao estilo arquitetural é o uso de eventos através do *Message Broker*. *Message brokers* formam uma uma parte essencial e insubstituível de microsserviços, aplicações baseadas em nuvem ou aplicações de processamento tem tempo real sendo utilizada para comunicação entre aplicações (HEGDE; NAGARAJA, 2020). No entanto, no estilo proposto, o uso de *message broker* agrega mais um propósito: observar e interpretar eventos. Ao observar os eventos gerados pelas aplicações, o *Resolver de Eventos*, como mostra a Figura 9 na Etapa 2, extrai e transforma os eventos de mais baixa ordem em outro de mais alta ordem. Essa abordagem permite implementar o *Resolver de Eventos* diminuindo o acoplamento entre os componentes e sem interferir no fluxo das aplicações, Etapa 1 da Figura 9, oferecendo suporte para aumentar o desempenho das aplicações de forma independente e com baixo acoplamento.

  O uso de comunicação assíncrona também é uma restrição que está presente no estilo arquitetural *Event Driven*, onde os componentes executam unidades de processamento de forma independente e desacoplada para processar um conjunto de eventos de forma reativa (LAIGNER et al., 2020).

- **Eventos de domínio:** A terceira restrição implica em somente observar eventos de domínio. Somente esse tipo de evento deve ser enviado pelo *Resolvedor de Eventos* para a *Cadeia de Eventos*. Em Han e Choi (2020) os eventos são utilizados para a forma de uso clássica de *Event Sourcing* e CQRS[2], armazenando os eventos de domínio em bancos de dados para posterior processamento de *dashboards* e recuperação da informação. O *EventChain* se caracteriza nesse aspecto principalmente ao tratar os eventos de domínio de 2 formas diferentes: 1) se preocupa em analisar e transformar os eventos em outros de mais alta ordem, ou seja, de mais valor agregado e 2) se preocupa em armazenar os dados dos eventos transformados na Blockchain, uma estrutura que garante a imutabilidade e a auditoria das informações.

- **Resolvedor de eventos:** Esse componente foi adicionado ao estilo para permitir que os eventos de domínio de mais baixa alta ordem possam ser observados e transformados em eventos de mais alta ordem, é a etapa 2 da Figura 9. Esse é

[2]Do Inglês: *Command Query Responsibility Segregation*

um dos conceitos introduzidos pelo estilo arquitetural proposto, ao observar os eventos como um agente inteligente as demais aplicações ficam desobrigadas de gerar eventos de domínio para uma finalidade específica.

Nenhum estado será mantido pela componente, ou seja, nenhum banco de dados ou outro repositório manterá informações dos eventos. As agregações devem ser realizadas em tempo real em memória. Possibilitando a execução de múltiplas instâncias do *Resolvedor de eventos* de forma independente.

- **Blockchain:** Outro conceito introduzido pela estilo arquitetural proposto é a *Cadeia de eventos*, dentro de cadeia somente poderão ser armazenados eventos confirmados que não serão modificados. O estilo arquitetural se baseia no uso intensivo de Blockchain e *Smart Contracts*.

  Essa restrição garante que o desenvolvedor não precisa se preocupar com nenhum código adicional para garantir que a informação gerada a partir do *Resolvedor de eventos* se mantenha íntegra e consistente.

- **Microsserviços e componentes distribuídos:** Para garantir a alta disponibilidade e a tolerância a falhas é necessário que os componentes tenham a capacidade de executar múltiplas instâncias ao mesmo. De acordo com Valdivia et al. (2020) o uso de microsserviços permite o desenvolvimento de sistemas distribuídos.

  Os componentes envolvidos na etapa 2 e 3 da Figura 9 devem ser implementados utilizando os conceitos de microsserviços permitindo que possam ser executados múltiplas instâncias dos serviços de forma distribuída em máquinas distintas a fim de garantir que em casa de falha em uma instância as demais continuem funcionando. Os demais componentes como *message brokers* e implementações de Blockchain devem ter a mesma capacidade de executar os processos de forma distribuída.

A *EventChain* faz uso dos estilos arquiteturais já estabelecidos para compor o seu estilo, focando na área financeira para criar um estilo mais específico que busca atender os requisitos comuns desse contexto. Para comparação entre os estilos arquiteturais, foram então estabelecidos os seguintes requisitos comuns (RC) da área financeira: (RC1) segurança, (RC2) performance, (RC3) alta disponibilidade e tolerância a falhas e (RC4)

Tabela 12 – Análise comparativa dos estilos arquiteturais

Estilo Arquitetural	Seg	Perf	AD e TF	Cons e ID
EventChain	●	●	●	●
Event Sourcing	◐	●	●	○
Microsserviço	◐	●	●	○
Model-View-Controller	○	○	○	○
REST	◐	●	○	○

Legenda: ● Suportado ◐ Parcialmente Suportado ○ Não Suportado
Seg: Segurança
Perf: *Performance*
AD e TF: Alta Disponibilidade e Tolerância a Falhas
Cons e ID: Consistência e Integridade dos Dados

consistência e integridade dos dados. A Tabela 12 resume o que cada estilo arquitetural atende de forma nativa.

4.4 Aspectos de Implementação

Um protótipo foi desenvolvido para demonstrar a viabilidade de implementação do estilo arquitetural proposto. Para isso, as seguintes tecnologias foram utilizadas:

- **Spring boot + Java**: A linguagem de programação escolhida para a implementação dos microsserviços foi a linguagem de programação Java [3], por ser uma das linguagens de programação mais utilizadas atualmente[4] em o conjunto com o *framework* Spring Boot [5]. A escolha de uso do Spring Boot teve como a motivação o fato de ser um *framework* de código-aberto focado em microsserviços, muito utilizado na indústria de *software* por ser simples e robusto ao mesmo tempo. Que oferece um ecossistema completo para desenvolver microsserviços com suporte e integração para diversos bancos de dados SQL e NoSQL, *brokers* de mensageria, bancos de dados de cache além da integração com outros *frameworks*, possui um ótimo suporte para a implementação de serviços REST. Esta abordagem permite que os desenvolvedores escolham uma das muitas possibilidades suportadas pelo

[3] `https://www.oracle.com/java/`
[4] `https://www.tiobe.com/tiobe-index/`
[5] `https://spring.io/projects/spring-boot`

Spring Boot para desenvolver microsserviços, usando o ecossistema maduro do Java, uma linguagem de código-aberto e gratuita e em constante evolução.

Dessa forma a contribuição da combinação do Java com o Spring Framework para o estilo arquitetural proposto se concentra no seu maduro ecossistema, que permite a construção de microsserviços com alto desempenho e grande suporte para integração. O protótipo faz amplo uso dessas integrações para validações de segurança bem como para estabelecer a conexão com o Apache Kafka e dessa forma enviar os eventos de mais baixa ordem de forma assíncrona.

- **Apache Kafka**: Os servidores de mensagens são componentes de *software* amplamente usados na indústria. São utilizados para permitir a comunicação entre sistemas em modo assíncrono, desacoplando os sistemas e aplicações. Desta forma, é possível integrar sistemas usando diferentes linguagens ou plataformas. Como componente de servidor de mensagens, o Apache Kafka[6] cumpre com os requisitos do *design* da arquitetura de *software* proposta, é uma plataforma de alto desempenho e baixa latência com tolerância a falhas Essas características tornam o Apache Kafka ideal em projetos que precisam processar um grande número de mensagens em tempo real, por exemplo, é usado como base para implementar um dos protocolos de consensos internamente no Hyperledger Fabric.

  Como contribuição principal o Apache Kafka permitiu que a implementação dos resolvedores de eventos pudessem se conectar a uma plataforma com alto desempenho para processar os eventos de mais baixa ordem. Além de fornecer um componente que permitisse fazer agregações dos eventos em tempo real, fator extremamente importante para habilitar a resolução dos eventos com o mínimo de atraso.

- **Hyperledger Fabric**: O Hyperledger Fabric[7] é uma plataforma Blockchain de código-aberto. Descrita de acordo com o site como *"uma estrutura de livro razão distribuída com permissões de nível corporativo para o desenvolvimento de soluções e aplicativos. Seu design modular e versátil satisfaz uma ampla gama de casos de uso da indústria"*, em tradução livre. Criado pela Linux Foundation em 2016, com a combinação de duas bases de código, uma dessas bases de código foi

[6] `https://kafka.apache.org/`
[7] `https://www.hyperledger.org/use/fabric`

criada pela IBM e a outra criada pela Intel. No momento, o Hyperledger Fabric é um dos muitos projetos sob o guarda-chuva Hyperledger.

A escolha do Hyperledger Fabric se deve ao fato de ser uma plataforma com boa documentação e com casos de uso que comprovam a eficiência da plataforma, tanto no quesito de versatilidade como no quesito de performance. Além disso, a plataforma já possui um SDK para aplicações Java, o que permite integrar os microsserviços desenvolvidos de forma fácil e reduz a curva de aprendizado, além de permitir que os *Smart Contracts* possam ser escritos em diferentes linguagens. Por fim, uma das principais contribuições do uso do Hyperledger Fabric é a capacidade de escalabilidade e o suporte a execução de múltiplos nós da rede de forma independente. Fornecendo dessa forma um poderoso suporte para tolerância a falhas, o que permite que uma parte da rede possa continuar operando mesmo que ocorra uma falha em alguns de seus servidores.

O uso do Hyperledger Fabric teve um papel fundamental na implementação da cadeia de eventos. Os resolvedores de eventos utilizaram a rede para instanciar os *Smart Contracts* e tornar os mesmos imutáveis, formando assim a cadeia de eventos.

A implementação do estilo de arquitetura de *software* proposto nesse estudo é baseada em projetos de código-aberto amplamente utilizados na indústria de *software* e também em instituições financeiras. Busca-se, dessa forma, reaproveitar o conhecimento adquirido pelos desenvolvedores agilizando o desenvolvimento de novas soluções.

Embora o estilo arquitetural proposto possa ser implementado em outras áreas, esse estudo se limitou a atacar os problemas típicos da área financeira. A principal motivação para isso é alta complexidade dos requisitos envolvidos nas aplicações financeiras, os processos regulatórios e a crescente demanda por softwares nessa área.

Sumário para a QP 2: Foi proposto e implementado o estilo arquitetural EventChain. O qual faz uso de uma arquitetura orientada a eventos para aplicações financeiras. Tal estilo arquitetural introduz novos conceitos: 1) o resolvedor de evento que inspeciona diferentes eventos de menor ordem para transformar para um evento de mais alta ordem e, 2) a cadeia de eventos, que faz uso intensivo da Blockchain para armazenamento imutável, garantido integridade e a consistência das informações geradas pelo resolvedor de eventos.

5 AVALIAÇÃO

Este capítulo apresenta a avaliação do estilo arquitetural proposto, explorando a questão de pesquisa QP3. Para tal, foi desenvolvido um estudo de caso e aplicado um questionário de aceitação de tecnológica. A Seção 5.1 apresenta um estudo de caso de uma aplicação que faz uso do estilo arquitetural proposto no Capítulo 4, tendo como base os aspectos de implementação expostos na Seção 4.4. A Seção 5.2 discute os resultados obtidos através da aplicação do questionário de aceitação tecnológica sobre o estilo arquitetural proposto utilizando como base o estudo de caso.

5.1 Estudo de Caso

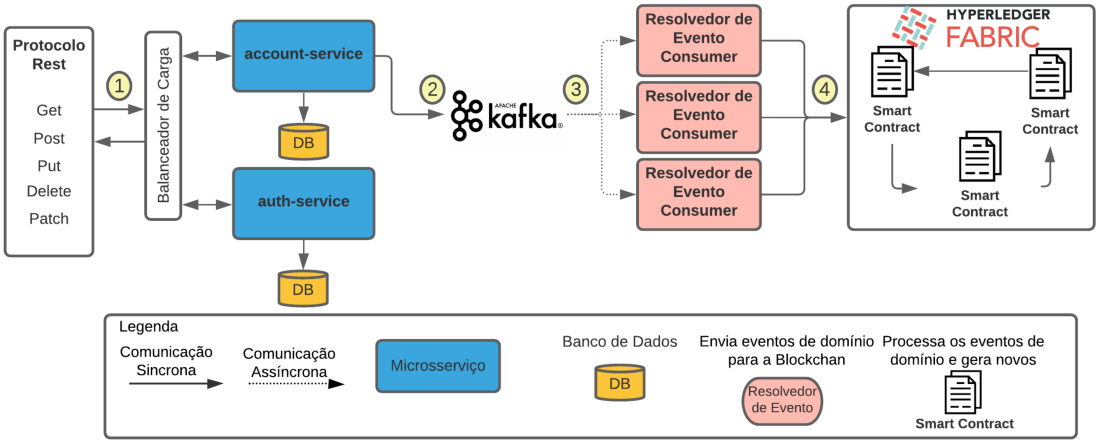
Sendo o estilo arquitetural proposto um estilo novo que se diferencia dos demais já pesquisados pela academia, a primeira parte da avaliação consiste em um estudo de caso. De acordo com Han e Park (2017), essa metodologia é eficiente quando é necessário verificar, expandir teorias conhecidas ou desafiar uma teoria específica. Permitindo que a pesquisa se foque na estratégia de explicar as fronteiras entre o fenômeno e contexto, e não em uma verificação teórica (YIN, 2008), dessa forma o estudo de caso busca demonstrar a viabilidade de implementação do estilo arquitetural proposto.

Para explorar o contexto financeiro algumas funcionalidades dessa área foram implementadas. O estudo de caso possui as seguintes funcionalidades: (1) cadastro de conta; (2) login e autenticação de usuários; (3) autorização de acesso; (4) transação de débito; (5) transação de crédito; (6) geração de eventos transacionais; (7) resolvedor de eventos e (9) registro dos eventos na Blockchain utilizando *smart contracts*.

A Figura 13 ilustra a instanciação de uma arquitetura que aplica o estilo arquitetural proposto, o diagrama que pode ser visualizado 14. O componente *auth-service* instancia os controles para auntenticação e geração de token JWT, os componentes *account-service e loan-service* geram os eventos de mais baixa ordem que são enviados para o kafka. E por fim, os dois últimos componentes que foram implementados, o *event-resolver* que representa uma instância do resolvedor de eventos responsável por ler e identificar os eventos que são correspondentes e, o componente *event-chaincode* que representa uma instância da cadeia de eventos responsável por fazer a interface com o Blockchain para as transações. Tal estilo está sendo respeitado mediante a aplicação

das restrições arquiteturais, que observando os números em destaque na Figura 13 se manifestam da seguinte forma:

Figura 13 – Estudo de caso



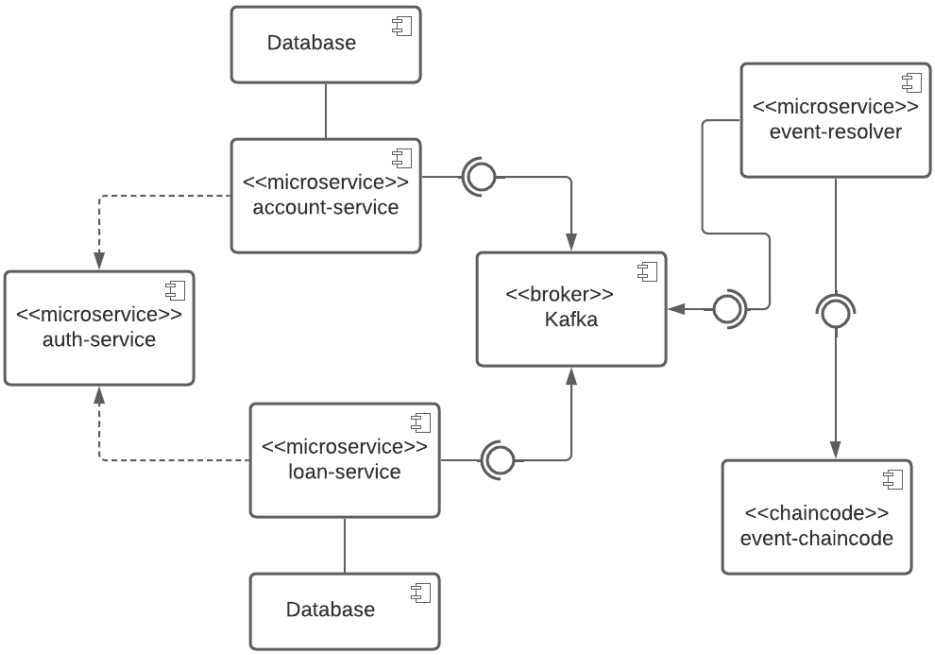
Fonte: Elaborado pelo autor.

- **Segurança**: A comunicação HTTP entre os serviços utiliza *token* JWT assinado. A implementação dessa restrição arquitetural faz uso do módulo de segurança *spring-boot-starter-security* provido nativamente pelo *framework* do *Spring Boot* em conjunto com a biblioteca *jsonwebtoken* para geração e validação do Token JWT. A Figura 15 apresenta o trecho de código que implementa a geração do *token*, enquanto que na Figura 16 é possível visualizar o trecho de código executa a validação do *token*.

  Ao realizar o processo de autenticação do usuário se as credenciais, do usuário forem válidas o, *token* JWT é gerado. Todos os demais serviços recebem no cabeçalho das requisições um parâmetro que deve conter o conteúdo do *token* gerado. Se o conteúdo for válido, a requisição segue o fluxo estabelecido, caso contrário o acesso ao recurso será negado.

- **Eventos de domínio e Comunicação assíncrona**: Os eventos de domínio são enviados para o *broker* à medida que novas transações ocorrem. A implementação de *broker* escolhida foi o Kafka, que acrescenta os seguintes benefícios: (1) capacidade de manter as mensagens já recebidas para futuro reprocessamento; e (2)

Figura 14 – Diagrama de componentes



Fonte: Elaborado pelo autor.

possibilidade de agregar eventos. A implementação do envio dos eventos utilizou o módulo *spring-kafka* que fornece implementações utilitárias para comunicação com o Kafka, conforme ilustrado pela Figura 17.

- **Resolvedor de eventos**: É o componente capaz de identificar os eventos de baixa ordem e transformar em eventos de mais alta ordem. A implementação do resolvedor de eventos utiliza o KSQLDB[1] para agregar as informações previamente

[1] `https://ksqldb.io/`

Figura 15 – Geração de *token* JWT

```
private String doGenerateToken(Map<String, Object> claims, String subject) {

return Jwts.builder().setClaims(claims).
    setSubject(subject).setIssuedAt(new Date(System.currentTimeMillis()))
    .setExpiration(new Date(System.currentTimeMillis() + JWT_TOKEN_VALIDITY * 1000))
    .signWith(SignatureAlgorithm.HS512, secret).compact();
}
```

Fonte: Elaborado pelo autor.

Figura 16 – Validação do *token* JWT

```
public Boolean validateToken(String token, UserDetails userDetails) {
    final String username = getUsernameFromToken(token);
    return (username.equals(userDetails.getUsername()) && !isTokenExpired(token));
}

public String getUsernameFromToken(String token) {
    return getClaimFromToken(token, Claims::getSubject);
}

public Date getExpirationDateFromToken(String token) {
return getClaimFromToken(token, Claims::getExpiration);
}

public <T> T getClaimFromToken(String token, Function<Claims, T> claimsResolver) {
    final Claims claims = getAllClaimsFromToken(token);
    return claimsResolver.apply(claims);
}

private Claims getAllClaimsFromToken(String token) {
    return Jwts.parser().setSigningKey(secret).parseClaimsJws(token).getBody();
}

private Boolean isTokenExpired(String token) {
    final Date expiration = getExpirationDateFromToken(token);
    return expiration.before(new Date());
}
```

Fonte: Elaborado pelo autor.

Figura 17 – Envio de eventos

```
public class PaymentProducer {

    private final KafkaTemplate<String, String> kafkaTemplate;
    private final String topicName;

    public PaymentProducer(final KafkaTemplate<String, String> kafkaTemplate,
                           @Value("${kafka.topic.payment}") final String topic) {
        this.kafkaTemplate = kafkaTemplate;
        this.topicName = topic;
    }

    public void sendMessage(final PaymentEvent message) {
        ListenableFuture<SendResult<String, String>> future = kafkaTemplate.send(topicName, message.toString());
        future.addCallback(new PaymentProducerCallback());
    }
}
```

Fonte: Elaborado pelo autor.

enviadas para o *broker*, identificando o relacionamento entre os eventos de mais baixa ordem e gerando eventos de mais alta ordem. Tais eventos são enviados para um novo e específico tópico no Kafka que possui a finalidade de receber esse tipo de evento. Para isso dentro do KSQLDB é necessário realizar a seguinte sequência para configurar a ferramenta: 1) criar um ou mais *streams* dos tópicos de origem e 2) criar uma tabela que irá realizar a agregação da informação com base nos critérios estabelecidos.

Para a implementação da Figura 10 no estudo de caso foi utilizada a estratégia de criar três *streams* e uma tabela da seguinte forma:

- ***Stream* geral:** A primeira *stream* representa todos os eventos transacionais de contas que são enviadas para um tópico do Kafka.
- ***Stream* de crédito:** A segunda *stream* filtra os eventos transacionais de cré-

dito da *stream* geral.

- ***Stream* de débito:** A terceira *stream* filtra os eventos transacionais de débito da *stream* geral.
- **Tabela de agregação:** Uma tabela que representa a agregação dos eventos correspondentes com base em um critério estabelecido.

A Figura 18 apresenta o código utilizado para criar a *stream* geral que recebe os eventos enviados ao tópico *accountTransaction*, as Figuras 19 e 20 apresentam o código utilizado para filtrar os eventos de crédito e débito respectivamente. A Figura 21 apresenta o código utilizado para agregar os eventos correspondentes, no estudo de caso comparando os números de recibo iguais e enviando os eventos correspondentes para o tópico *transfers*. O arquivo com os códigos executados está disponível no repositório de código do GITHUB com o nome scripts_ksql.txt.

Figura 18 – Stream de eventos transacionais

```
create stream accountTransaction_st
(
    ID VARCHAR key, number VARCHAR, BRANCH VARCHAR, bank VARCHAR, balance double, taxId VARCHAR,
    eventType VARCHAR, receiptNumber VARCHAR, amount double
)
WITH (KAFKA_TOPIC='accountTransaction', VALUE_FORMAT='JSON');
```

Fonte: Elaborado pelo autor.

Figura 19 – Stream filtrando eventos de crédito

```
create stream credit_st as
    SELECT
        id,
        number,
        branch,
        bank,
        taxId,
        eventType,
        receiptNumber,
        amount
    FROM
        accountTransaction_st
    WHERE
        eventType = 'CREDIT'
emit changes;
```

Fonte: Elaborado pelo autor.

Após isso, microsserviços especializados em consumir os eventos enviados ao tópico de transferência utilizam o módulo *spring-kafka* para se conectar ao *broker*, identificando o tipo de evento e o seu *smart contract* correspondente. Após

Figura 20 – Stream filtrando eventos de débito

```
create stream credit_st as
    SELECT
        id,
        number,
        branch,
        bank,
        taxId,
        eventType,
        receiptNumber,
        amount
    FROM
        accountTransaction_st
    WHERE
        eventType = 'DEBIT'
emit changes;
```

Fonte: Elaborado pelo autor.

Figura 21 – Tabela de agregação dos eventos

```
create table transfers as
select
    LATEST_BY_OFFSET(fromAccount.number) AS from_number,
    LATEST_BY_OFFSET(fromAccount.branch) AS from_branch,
    LATEST_BY_OFFSET(fromAccount.bank) AS from_bank,
    LATEST_BY_OFFSET(fromAccount.taxId) AS from_taxId,

    LATEST_BY_OFFSET(toAccount.number) AS to_number,
    LATEST_BY_OFFSET(toAccount.branch) AS to_branch,
    LATEST_BY_OFFSET(toAccount.bank) AS to_bank,
    LATEST_BY_OFFSET(toAccount.taxId) AS to_taxId,

    LATEST_BY_OFFSET(fromAccount.receiptNumber) AS receiptNumber,
    LATEST_BY_OFFSET(toAccount.amount) AS amount,
    fromAccount.receiptNumber
from debit_st as fromAccount
inner join credit_st as toAccount
WITHIN 10 MINUTES
on fromAccount.receiptNumber = toAccount.receiptNumber
group by fromAccount.receiptNumber
emit changes;
```

Fonte: Elaborado pelo autor.

identificar o *smart contract* uma conexão com a rede Blockchain é estabelecida utilizando o SDK para envio do evento.

Novos resolvedores de eventos podem ser desenvolvidos e plugados na arquitetura a qualquer momento sem impactar os demais ou as aplicações geradoras de eventos de ordem mais baixa. Os resolvedores podem agregar informações de um ou mais eventos de mais baixa ordem para um evento de mais alta ordem. A Figura 22 demonstra o código fonte que foi utilizado para desenvolvedor o gerador de eventos que se encaminha os eventos para*broker* de mensagens, enquanto a Figura 23 demostra a implementação para o consume de mensagens produzidas.

- **Cadeia de eventos**: Os *smart contracts* são o ponto inicial para formar e manter

Figura 22 – Produtor de evento

```java
public class PaymentProducer {

    private final KafkaTemplate<String, String> kafkaTemplate;
    private final String topicName;

    public PaymentProducer(final KafkaTemplate<String, String> kafkaTemplate,
                           @Value("${kafka.topic.payment}") final String topic) {
        this.kafkaTemplate = kafkaTemplate;
        this.topicName = topic;
    }

    public void sendMessage(final PaymentEvent message) {
        ListenableFuture<SendResult<String, String>> future = kafkaTemplate.send(topicName, message.toString());
        future.addCallback(new PaymentProducerCallback());
    }
}
```

Fonte: Elaborado pelo autor.

Figura 23 – Resolvedor de evento

```java
@Component
@Slf4j
public class EventResolverConsumer {

    @Autowired
    private Gateway gateway;
    private Network network;
    @Value("${event-resolver.blockchain.network}")
    private String networkName;
    private ObjectMapper objectMapper = new ObjectMapper();
    private Contract contract;

    @PostConstruct
    public void setup() {
        network = gateway.getNetwork(networkName);
    }

    @KafkaListener(topics = "${event-resolver.topic.transfers}", groupId = "${spring.kafka.consumer.group-id}",
            containerFactory = "kafkaListenerContainerFactory")
    public void listen(String event, Acknowledgment ack, @Header(KafkaHeaders.RECEIVED_PARTITION_ID) String partition,
            @Header(KafkaHeaders.OFFSET) String offset, @Header(KafkaHeaders.RECEIVED_TOPIC) String topicName) {
        try {
            log.info("Partition {}, Offset {}, {} event received {}", partition, offset, topicName, event);
            switch (topicName) {
                case "TRANSFERS":
                    sendTransferToBlockchain(event);
                    break;
                default:
                    throw new RuntimeException("Topic not configured");
            }
            ack.acknowledge();
        } catch (Exception e) {
            log.error("Error to read transfer event", e);
        }
    }

    public void sendTransferToBlockchain(String transferJSON) {
        try {
            if (contract == null) {
                contract = network.getContract("event-chaincode");
            }
            contract.submitTransaction("CreateTransfer", transferJSON);
        } catch (ContractException | InterruptedException | TimeoutException e) {
            log.error("Was not possible to send event to blockchain", e);
        }
    }
}
```

Fonte: Elaborado pelo autor.

a cadeia de eventos. Quando um evento é enviado para a rede um *smart contract* é sempre associado, sendo responsabilidade do mesmo aplicar as validações que possam ser necessárias, registrar na Blockchain a informação e sinalizar o status final da transação indicando sucesso ou falha.

Eventos de mais alta ordem são armazenadas de forma independente, ou seja, para cada tipo de evento um bloco específico é reservado. A medida que novos eventos são recebidos os mesmos são agregados formando dessa a cadeia de eventos. Para a implementação do estudo de caso foi implementado um *smart contract* customizado com a lógica de persistência dos eventos de transferência, a Figura 24 apresenta o código fonte utilizado na implementação.

Figura 24 – Smart contract desenvolvido

```
'use strict';

const { Contract } = require('fabric-contract-api');

class AccountTransfer extends Contract {
    async CreateTransfer(ctx, event) {
        const eventTransfer = JSON.parse(event);
        ctx.stub.putState(eventTransfer.RECEIPTNUMBER, event);
        return event;
    }

    async GetAll(ctx) {
        const allResults = [];
        const iterator = await ctx.stub.getStateByRange('', '');
        let result = await iterator.next();
        while (!result.done) {
            const strValue = Buffer.from(result.value.value.toString()).toString('utf8');
            let record;
            try {
                record = JSON.parse(strValue);
            } catch (err) {
                console.log(err);
                record = strValue;
            }
            allResults.push({ Key: result.value.key, Record: record });
            result = await iterator.next();
        }
        return JSON.stringify(allResults);
    }
}
module.exports = AccountTransfer;
```

Fonte: Elaborado pelo autor.

- **Consistência e integridade**: Se manifestam através da combinação dos elementos no estilo a arquitetural, ao plugar os Resolvedores de Eventos de forma assíncrona e escutar os eventos de domínio, garantimos que somente eventos já confirmados pelas demais aplicações serão consumidos. Para isso as aplicações que

enviam os eventos de baixa ordem para o Kafka utilizam transação[2], ou seja, ao final da transação na Etapa 2 da Figura 13 se não houver um comando de *commit* confirmando o fim, nenhum evento será enviado para o *broker*. O mesmo ocorre para o Resolvedor de Eventos: ao transformar os eventos de mais alta ordem, ao final da Etapa 3 da Figura 13, se não houver um comando de *commit*, o evento é retornado para o Kafka que fará uma nova tentativa de entrega.

A integridade das informações se manifesta de duas formas: (1) com o uso do *token* JWT como mencionado no item 1 dessa listagem e (2) com a aplicação da cadeia de eventos, tornando dessa forma os eventos imutáveis.

- **6) Alta disponibilidade e tolerância a falhas**: Dado que os sistemas da área financeira processam milhares de requisições, a opção de utilizar componentes que possam escalar foi um fator determinante.

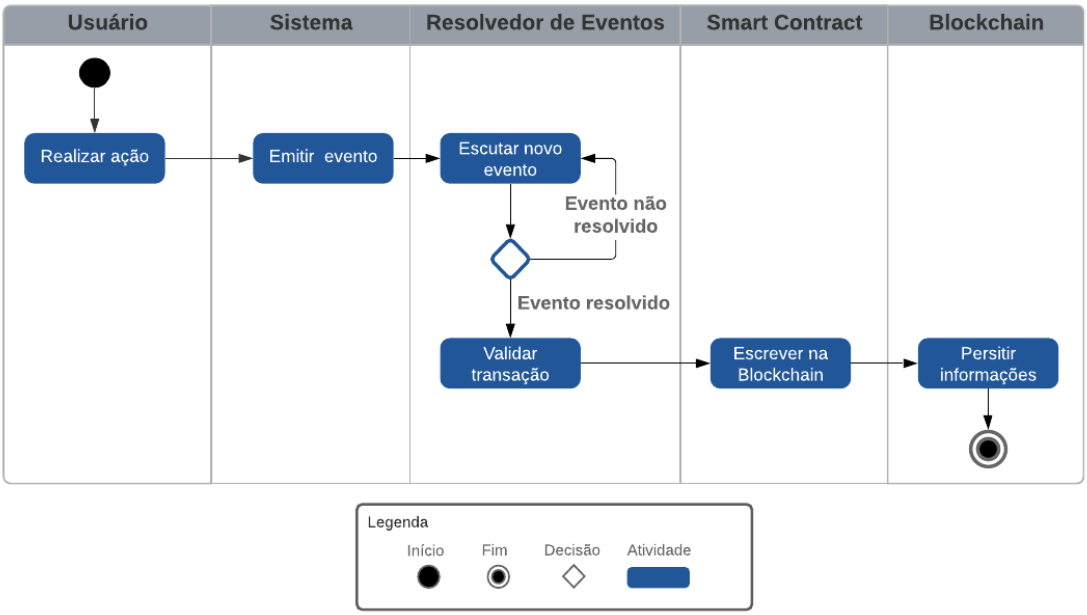
  A Etapa 1 da Figura 13 consiste no uso de um balanceador de carga com o objetivo de minimizar o impacto em caso de falha em uma das instâncias, onde múltiplas instâncias dos mesmos microsserviços recebem e processam as requisições distribuídas pelo balanceador. Para isso os microsserviços foram implementados de forma que nenhum estado seja mantido em memória, permitindo que novas instâncias possam ser adicionadas ou removidas do balanceador de carga sem afetar o processamento. Na Etapa 2 da Figura 13 o *broker* de mensagens Kafka possui a característica de ser uma plataforma distribuída, permitindo que em caso de falha as aplicações possam se conectar automaticamente a outras instâncias do *cluster*.

  A adição de um *broker* de mensagens distribuído permite também que múltiplas instâncias dos resolvedores de eventos (Etapa 3 da Figura 13) possam processar os eventos de mais alta ordem paralelamente, minimizando os impactos em caso de falha e, se ocorrer uma falha, permite que possam se reconectar no *broker* e continuar o processamento a partir do ponto onde pararam. Na cadeia de eventos (Etapa 4 da Figura 13) a disponibilidade e tolerância a falhas se manifesta através do uso de múltiplos *peers* executando as mesmas instâncias de *Smart Contracts*, utilizando um dos principais benefícios da *Blockchain* que é a capacidade de processamento distribuído.

[2] `https://docs.spring.io/spring-kafka/reference/html/#transactions`

A Figura 25 apresenta o diagrama de atividades contendo as atividades macro implementas para o estudo de caso da Figura 13.

Figura 25 – Diagrama de atividades



Fonte: Elaborado pelo autor.

5.1.1 Atributos de qualidade

A seguir será apresentado os atributos de qualidade que o estilo arquitetural proposto demonstrou após ser submetido a diferentes cenários de teste de carga. Os testes foram realizados em única máquina, utilizando o seguinte *hardware*: processador Intel Core i7 Quad-Core com 2,9 GHz, memória de 16 GB com 2133 MHz, disco rígido *SSD* com 500 GB e sistema operacional MacOs Big Sur. Toda a infraestrutura necessária para execução dos microsserviços foi executada utilizando contêineres Docker[3], foram utilizados contêineres para o banco de dados Postgres[4], Kafka, KSQLDB e Hyperledger Fabric.

Os testes de carga foram divididos em duas partes. A primeira parte consistiu na

[3]https://www.docker.com/
[4]https://www.postgresql.org/

realização de testes de carga utilizando a ferramenta JMeter[5], onde foram estabelecidos três cenários de testes utilizando como base a métrica de quantidade de transações por segundo (TPS), tendo a duração de cinco minutos, sendo: 1) dez TPS, 2) vinte TPS e, 3) trinta TPS no microsserviço *account-service* responsável por fazer as transações financeiras e enviar eventos para o *broker* de mensagens. A segunda parte focou no microsserviço *event-resolver* responsável por resolver os eventos e enviá-los para a Blockchain, testando dessa forma a capacidade de resolução e envio para a Blockchain.

A primeira parte dos testes utilizou uma única instância do microsserviço *account-service*, a Figura 26 apresenta os gráficos envolvendo os recursos de processador e memória que foram extraídos da ferramenta de monitoramento de aplicações JVisualVM[6]. Foi possível visualizar que o consumo de processamento foi satisfatório considerando as circunstâncias de compartilhamento de recursos, se mantendo estável próximo de 35% com trinta TPS. Os recursos de memória foram muito bem utilizados, mesmo sob a carga mais alta não consumiu mais que 200 MB.

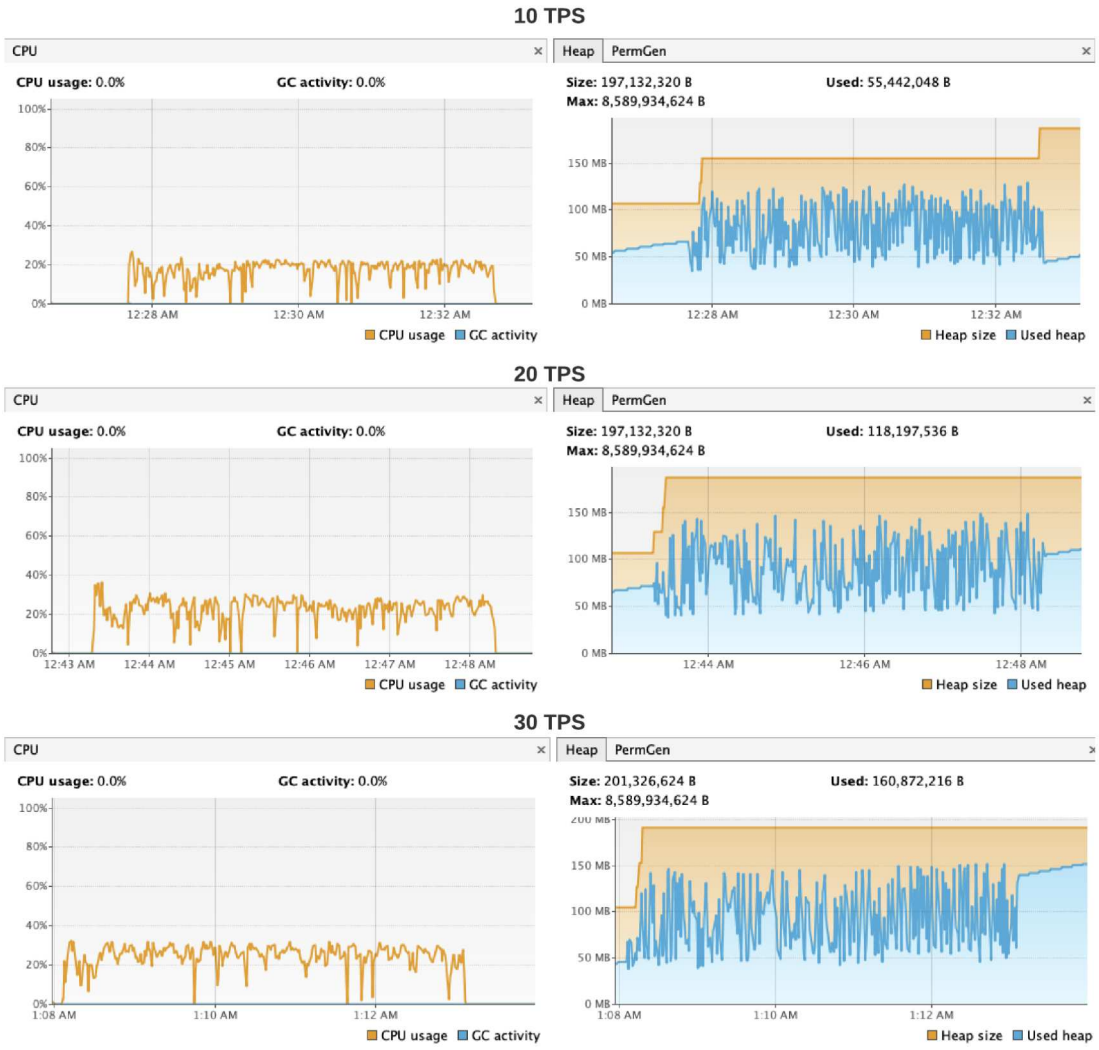
Como resultado, foi possível atingir o pico máximo de 350 TPS no microsserviço *account-service*, oscilando entre 300 e 350 TPS durante o período de teste. A Figura 27 apresenta o comparativo de TPS durante o período de testes dos três cenários.

A segunda parte do teste de performance focou no teste do microsserviço *event-resolver*, dessa forma além de testar a capacidade de resolver os eventos também foram testados a capacidade do *smart contract* e da Blockchain. Todos os eventos gerados na primeira parte foram enviados para o Kafka, no entanto, para isolar os testes o microsserviço *event-resolver* na foi iniciado. Dessa forma foi possível realizar os testes utilizando a grande quantidade de mensagens que microsserviço precisaria processar.

Diferentemente da primeira parte, o teste de performance no microsserviço *event-resolver* não foi separado em cenários, dado a característica de consumir mensagens de forma assíncrona. Foram monitorados os recursos do microsserviço da mesma forma que na primeira parte utilizando o JVisualVM, porém, o tempo do teste teve um acréscimo de cinco de minutos para melhor visualização dos resultados através dos gráficos, totalizando dez minutos. O monitoramento da Blockchain foi realizado utilizando a fer-

[5]https://jmeter.apache.org/
[6]https://visualvm.github.io/

Figura 26 – Teste de carga - Consumo de recursos *account-service*



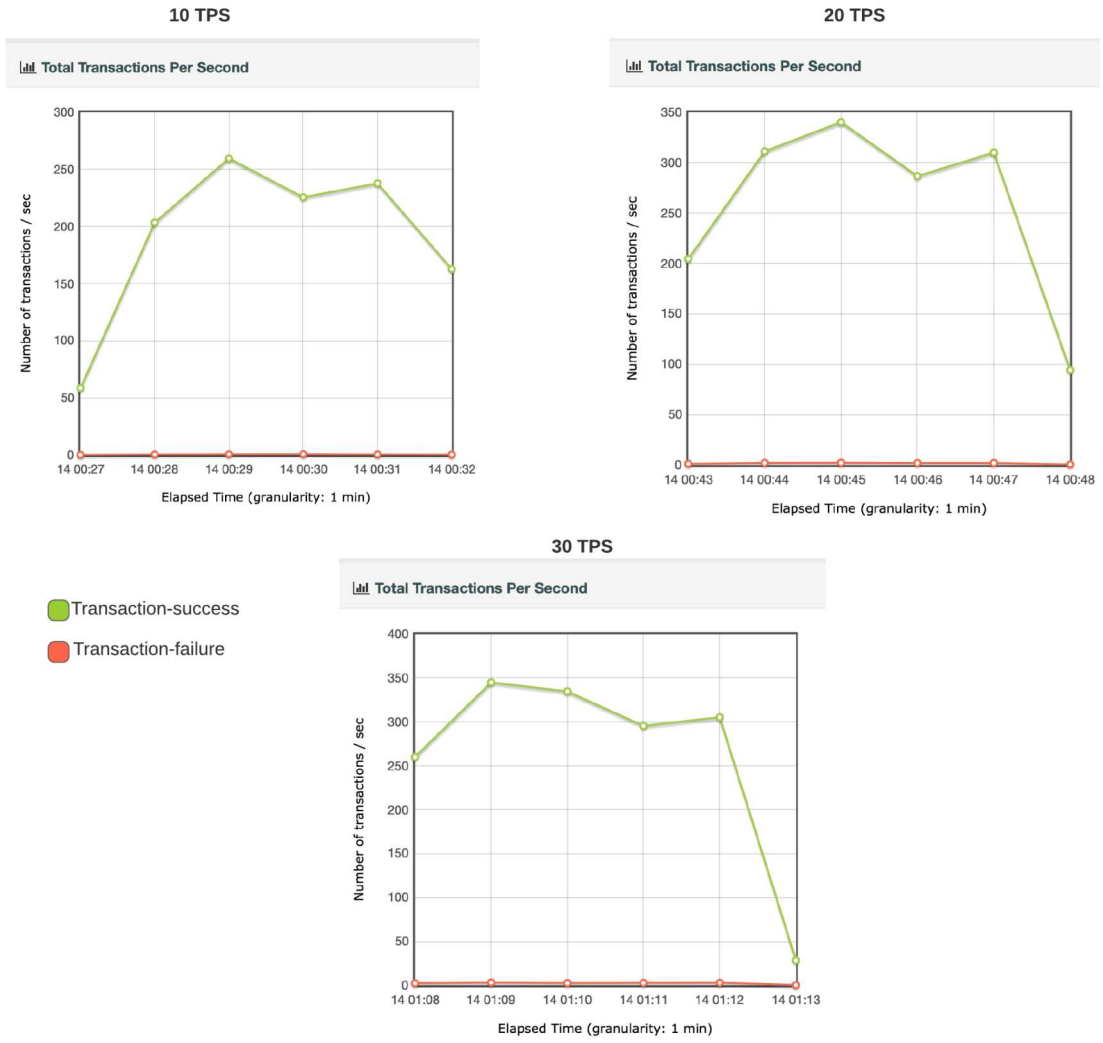
Fonte: Elaborado pelo autor.

ramenta Hyperledger Explorer[7], tal ferramenta é indicada para explorar a Blockchain e realizar o monitoramento da mesma.

O monitoramento do microsserviço *event-resolver* demonstrou um ótimo resultado em termos de consumo de recursos de *hardware*, a Figura 28 apresenta os gráficos dos recursos utilizados. Se destaca o baixo uso de processamento ficando na média de 5% de uso do processador durante o teste, a utilização de memória também apresentou um

[7]https://www.hyperledger.org/use/explorer

Figura 27 – Teste de carga - TPS *account-service*

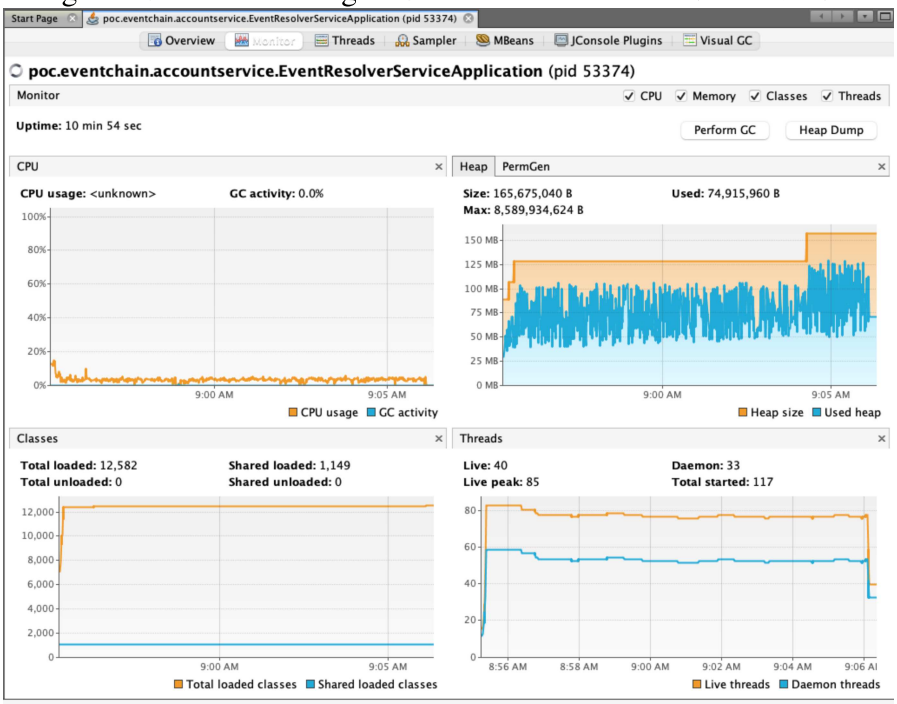


Fonte: Elaborado pelo autor.

bom resultado não utilizando mais do que 150 MB.

A Figura 29 apresenta a configuração inicial antes do início dos testes, que já possuía seiscentas transações registradas e duzentos e três blocos. Após os dez minutos de teste foi possível observar uma curva crescente de performance atingindo o pico de mil e quinze TPS no término do teste, como pode ser visto na Figura 30. Após o teste de performance a Blockchain registrou onze mil duzentas e oitenta seis transações

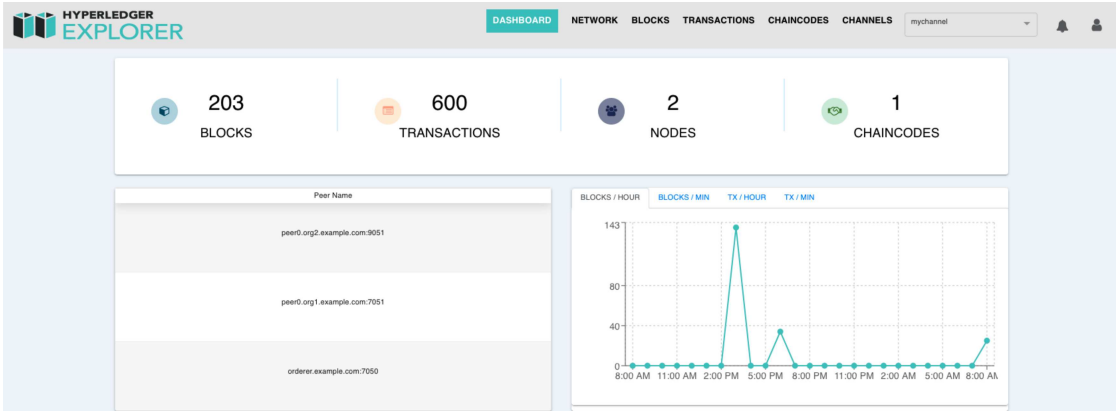
Figura 28 – Teste de carga - Consumo de recursos *event-resolver*



Fonte: Elaborado pelo autor.

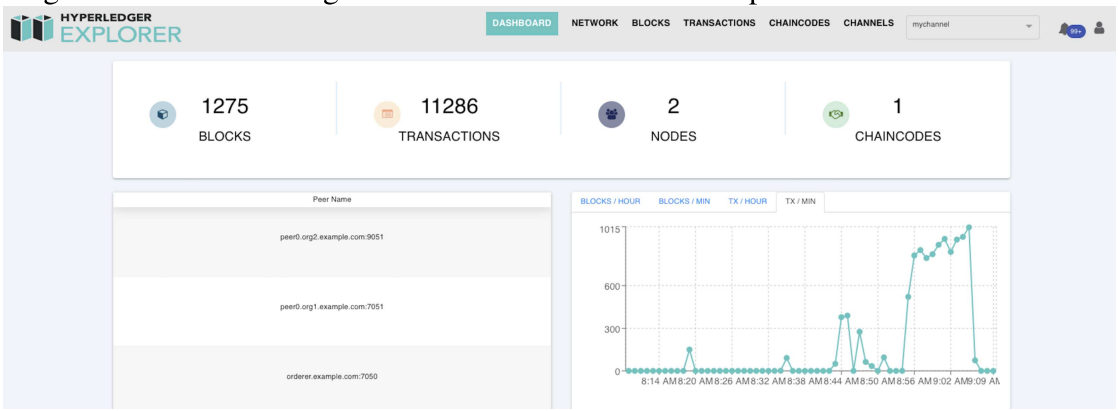
e mil duzentos e setenta e cinco blocos. Foram criadas dez mil seiscentas e oitenta seis transações em dez minutos de teste, obtendo uma média de mil e sessenta e oito transações por minuto.

Figura 29 – Teste de carga - Estado inicial da Blockchain



Fonte: Elaborado pelo autor.

Figura 30 – Teste de carga - Resultados finais do teste de performance na Blockchain



Fonte: Elaborado pelo autor.

- **Escalabilidade**: Como demostrado nos testes de performance o estilo arquitetural apresentou bons resultados em termos de escalabilidade, permitindo que os componentes possa ser escalados de forma individualizada. Pode-se escalar os componentes tanto de forma horizontal como vertical. Em comparação com o estilo arquitetural de monólito o EventChain possui um suporte mais flexível em termos de escalabilidade;

- **Adequação funcional**: O estilo arquitetural proposto demonstrou um grau de adequação alinhado com as exigências e necessidades das aplicações financeira, sob o aspecto de volume, tempo de resposta e capacidade de processamento;

- **Eficiência de desempenho**: Dado o cenário apresentado ficou evidente as capacidades do estilo arquitetural proposto em termos de desempenho, o que atende as exigências de performance das aplicações financeiras;

- **Confiabilidade**: Os testes demonstraram que o estilo arquitetural proposto se mostrou confiável durante a execução dos testes, não apresentando oscilações, falhas ou erros que desabonem o grau de confiança nos componentes implementados.

- **Segurança**: Um dos atributos de maior destaque do EventChain é sem dúvida a segurança. Tornando as informações seguras contra acessos e manipulação, o estilo arquitetural proposto apresenta um alto grau de consistência e integridade das

informações armazenadas. Nenhum outro estilo arquitetural apresenta tamanho grau de segurança quanto o EventChain.

5.2 Modelo de Aceitação Tecnológica

Essa etapa de avaliação consiste busca investigar a aceitação do EventChain por profissionais da indústria através da aplicação do Modelo de Aceitação Tecnológica (TAM)[8]. Essa Seção explora a questão de pesquisa Q3, buscando validar e analisar a aceitação do estilo arquitetural proposto entre os participantes que responderam ao questionário.

O questionário TAM foi escolhido como ferramenta de avaliação dado que a literatura (FARIAS et al., 2019; CHAGAS et al., 2019) tem demonstrado a sua eficácia na avaliação de contextos que são similares ao apresentado nesse estudo. Sendo escolhido por essas razões: 1) identificar a percepção de facilidade de uso, 2) identificar a percepção de usabilidade, 3) identificar a atitude em relação ao uso e, 3) identificar comportamento com relação a intenção de uso.

Para isso, o questionário TAM (MARANGUNIć; GRANIć, 2014) criado apresenta questões sobre a usabilidade e aceitação da técnica apresentada nesse estudo. Foram criadas nove questões, que foram respondidas utilizando a escala *Likert*[9] da seguinte forma: Concordo Totalmente, Concordo Parcialmente, Neutro, Discordo Parcialmente e Discordo Totalmente. As questões formuladas (Q) tratam vários tópicos, incluindo a percepção de facilidade de uso (Q1-3), percepção de usabilidade (Q4-6), atitude em relação ao uso (Q7 e Q8) e o comportamento com relação a intenção de uso (Q9).

5.2.1 Processo de avaliação

A Figura 31 apresenta o processo experimental, o qual é formado por um conjunto de atividades que foram agrupadas em três fases, que serão descritas a seguir:

Fase 1: Desenvolvimento do protótipo. Possui uma única atividade, focando no desenvolvimento de um protótipo. O protótipo teve o objetivo de implementar um sistema financeiro aplicando os requisitos mínimos utilizando o estilo arquitetural proposto

[8]Do inglês: *Technology Acceptance Model*
[9]Do inglês: *Likert Scale*

para demonstrar a viabilidade técnica. O protótipo implementa um sistema tipicamente utilizado em softwares da área financeira para simular a aplicação do estilo arquitetural proposto.

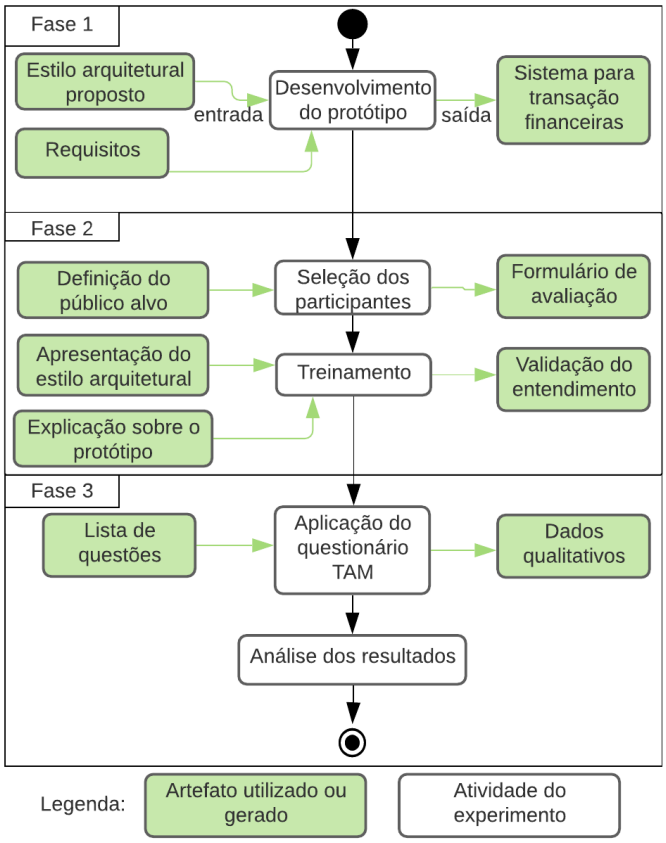
Fase 2: Seleção e treinamento. Possui duas atividades e teve como tema central os participantes da pesquisa. Sendo a primeira delas a seleção dos participantes para responder ao questionário de aceitação tecnológica. Para isso foi estabelecido alguns critérios a fim de garantir que os participantes tivessem o perfil técnico compatível, sendo eles: 1) atuação na área de desenvolvimento de *software*; 2) conhecimento prático e teórico na engenharia de *software* e 3) com formação completa ou estudantes de cursos de graduação e pós-graduação. A segunda atividade consistiu na elaboração de uma apresentação e de um vídeo para demonstrar para os participantes o estilo arquitetural proposto, o problema, a motivação e a sua aplicação na construção do protótipo. Além da apresentação, também foi disponibilizado o código-fonte do protótipo para que os participantes pudessem analisar a sua implementação, bem como a arquitetura aplicando o estilo arquitetural EventChain.

Fase 3: Aplicação do questionário e análise dos resultados. É a última fase e possui duas atividades, focando no questionário de aceitação tecnológica. A primeira atividade consiste na aplicação do questionário. Os participantes responderam questões relacionadas ao estilo arquitetural proposto, bem como questões sobre o perfil individual. Após a coleta dos dados os resultados foram mensurados e analisados, a análise completa será apresentada na seção a seguir.

5.2.2 Análise dos resultados

Esta seção apresenta e discute os resultados coletados através da aplicação do questionário de aceitação tecnológica. Para isso, os resultados foram divididos em duas partes. A primeira apresenta os dados relativos ao perfil dos participantes da pesquisa, enquanto a segunda apresenta os dados relativos ao questionário de aceitação tecnológica, Questões de um (Q1) a nove (Q9).

Figura 31 – Processo experimental



Fonte: Elaborado pelo autor.

5.2.2.1 Perfil dos participantes

As perguntas sobre o perfil dos participantes buscam entender a área de atuação, experiência, idade e nível educacional, a Tabela 13 descreve as respostas dos participantes e a Figura 32 apresenta os gráficos com o perfil dos participantes. Os resultados foram coletados entre os dias 20 de junho e 09 de novembro de 2021. No total, 34 participantes responderam o questionário.

A seleção dos participantes foi realizada de acordo com os seguintes critérios de inclusão:

- **Critério 1:** Estar atuando na área financeira.

- **Critério 2:** Ter experiência com desenvolvimento de *software*.
- **Critério 3:** Possuir no mínimo o ensino médio completo.
- **Critério 4:** Ter respondido todas as perguntas do questionário TAM.

Os participantes tinham entre 20 e 40 anos, sendo que a maior parte entre 30 e 39 anos (73,5%). Com relação ao nível educacional, foi possível identificar o alto grau de escolaridade, mais de 97,1% das pessoas que responderam o questionário tinham ensino superior em andamento ou concluído. Dessas pessoas, pode-se destacar que quatorze tinham pós-graduação (41,2%), quatorze com ensino superior completo (41,2%) e três com mestrado (8,8%).

Com relação a atuação profissional dos participantes, é possível identificar que os participantes possuem bastante experiência. A maior parte possui mais de oito anos de experiência na indústria (76,5%) e apenas um participante tinha menos de dois anos de experiência (2,9%). Desses, a maioria atua como desenvolvedores ou arquitetos de *software* (76,5%), sendo 20,6% como desenvolvedor e 55,9% como arquiteto.

5.2.2.2 Questionário de aceitação tecnológica

A Tabela 14 apresenta as questões utilizadas no questionário TAM. Por meio do questionário TAM buscou-se avaliar a facilidade de uso percebida, utilidade percebida, atitude em relação ao uso e a intenção comportamental de uso, no que diz respeito ao estilo de arquitetura proposto. A seguir será apresentada uma discussão sobre os resultados obtidos do questionário.

Facilidade de uso percebida: Composta por três questões (Q1, Q2 e Q3), busca entender entre os participantes o quanto o estilo arquitetural proposto é fácil de ser aplicado. Nenhum dos participantes discordou totalmente que o estilo arquitetural é fácil de utilizar, pelo contrário, a maioria (67,7%) concorda que o estilo arquitetural é fácil de ser aplicado, enquanto 17,6% discordam parcialmente. A mesma percepção também pode ser apontada em relação a facilidade de aprendizagem (73,5%). Se destaca em relação a facilidade de uso a percepção de flexibilidade provida, onde uma grande maioria (91,2%) concorda que o estilo arquitetural proposto é flexível. A Figura 33 apresenta os gráficos com as respostas de cada pergunta.

Tabela 13 – Perfil dos participantes

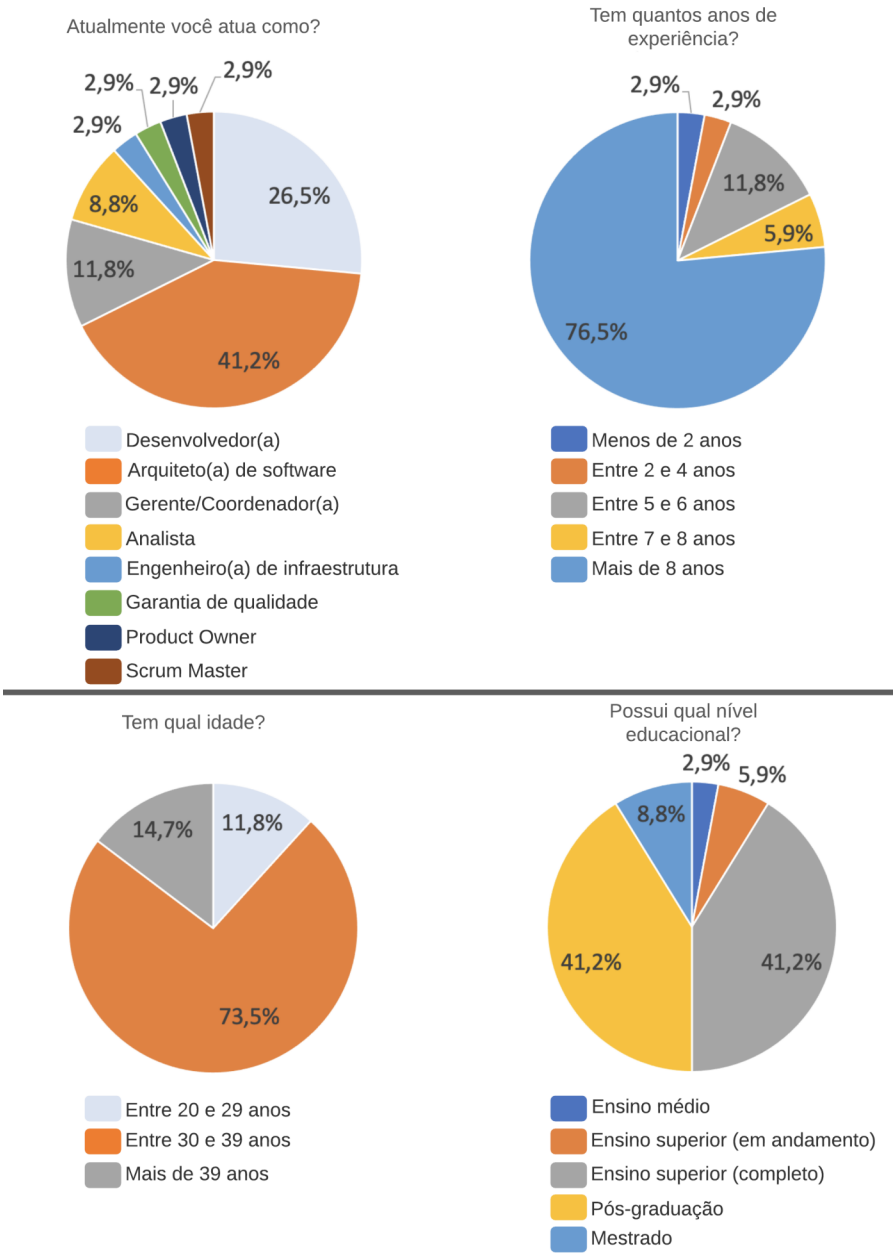
Perfil (n=34)	**Resposta**	**Total**	**Percentual**
Atuação			
	Desenvolvedor(a)	9	26,5%
	Arquiteto(a) de *software*	14	41,2%
	Gerente/coordenador(a)	4	11,8%
	Analista	3	8,9%
	Engenheiro(a) de infraestrutura	1	2,9%
	Garantia de qualidade	1	2,9%
	Product owner	1	2,9%
	Scrum master	1	2,9%
Experiência			
	> 8 anos	26	76,5%
	7–8 anos	2	5,9%
	5–6 anos	4	11,8%
	2–4 anos	1	2,9%
	< 2 anos	1	2,9%
Idade			
	30–39 anos	25	73,5%
	> 39 anos	5	14,7%
	20–29 anos	4	11,8%
Nível educacional			
	Pós-graduação	14	41,2%
	Ensino superior completo	14	41,2%
	Mestrado	3	8,9%
	Ensino superior em andamento	2	5,9%
	Ensino médio	1	2,9%

Fonte: Elaborado pelo autor.

É possível observar que a percepção em relação a facilidade de uso do estilo arquitetural proposto encontra concordância entre os participantes. Sendo o ponto alto a percepção de flexibilidade, justamente um dos objetivos propostos pelo estilo arquitetural. Sendo umas das principais características que o estilo arquitetural precisa para se adequar às diferentes aplicações.

Usabilidade percebida: Tem como objetivo entender a usabilidade do estilo arquitetural proposto. Para isso foram criadas três perguntas (Q4, Q5 e Q6). Com relação a usabilidade, nenhum dos participantes discordou totalmente, pelo contrário, houve uma

Figura 32 – Perfil dos participantes



Fonte: Elaborado pelo autor.

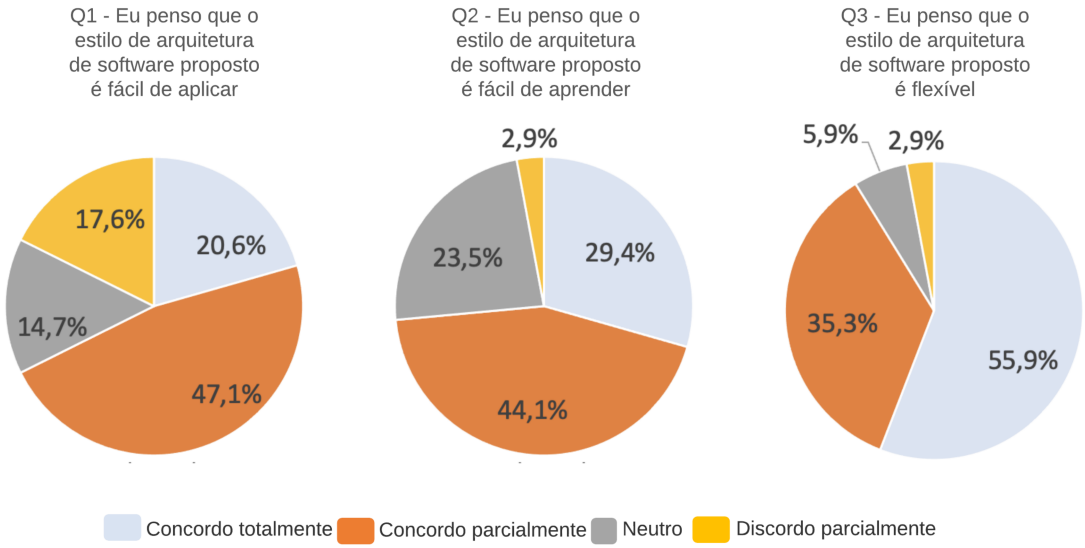
clara percepção de usabilidade. Muitos do participantes (82,4%) concordam que o estilo arquitetural torna mais seguro o armazenamento dos dados gerados através de *Event Sourcing*. E uma percentagem ainda maior (91,2%) concorda que o estilo arquitetural

Tabela 14 – Questionário TAM

Questão (Q)	**CT**	**CP**	**N**	**DP**	**DT**
Facilidade de uso percebida					
(Q1) Eu penso que o estilo é fácil de aplicar	7	16	5	6	0
(Q2) Eu penso que o estilo é fácil de aprender	10	15	8	1	0
(Q3) Eu penso que o estilo é flexível	19	12	2	1	0
Usabilidade percebida					
(Q4) O estilo torna mais seguro o armazenamento dos dados gerados através de *Event Sourcing*?	14	14	6	0	0
(Q5) O estilo faz bom uso de Blockchain?	20	11	2	1	0
(Q6) O estilo é uma implementação viável?	22	11	1	0	0
Atitude em relação ao uso					
(Q7) O estilo encoraja o uso de *Event Sourcing*?	15	14	5	0	0
(Q8) Estou confiante sobre o estilo proposto	15	11	8	0	0
Intenção comportamental de uso					
(Q9) Pretendo testar o estilo proposto	16	8	6	4	0
Legenda:					
CT: Concordo totalmente **CP**: Concordo parcialmente **N**: Neutro					
DP: Discordo parcialmente **DT**: Discordo totalmente					

Fonte: Elaborado pelo autor.

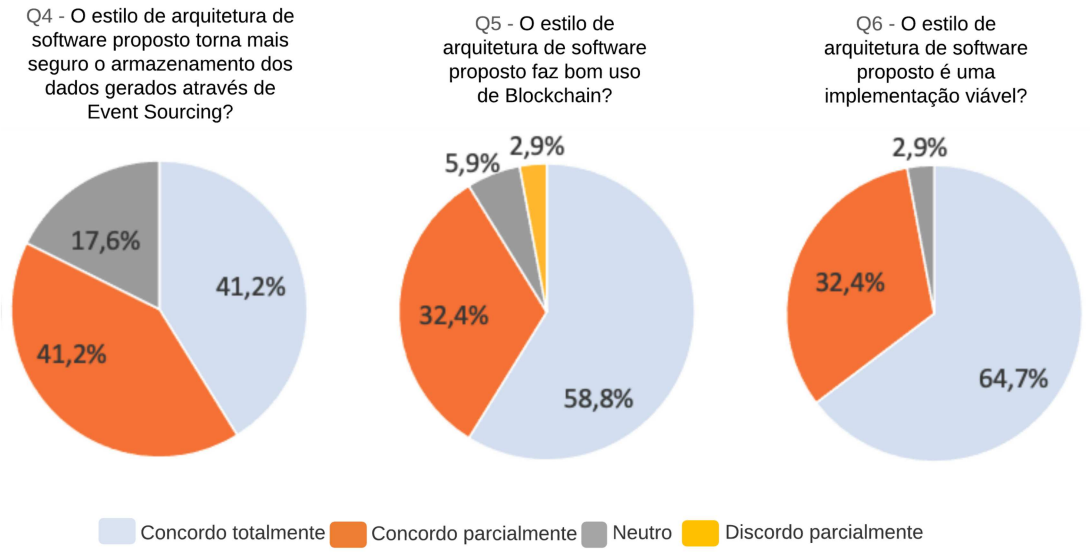
Figura 33 – Gráficos das respostas sobre facilidade de uso percebida



Fonte: Elaborado pelo autor.

proposto faz bom uso de Blockchain. E a grande maioria dos participantes (97,1%) concorda que o estilo arquitetural proposto é uma implementação viável. A Figura 34 apresenta os gráficos com as respostas de cada pergunta.

Figura 34 – Gráficos das respostas sobre usabilidade percebida



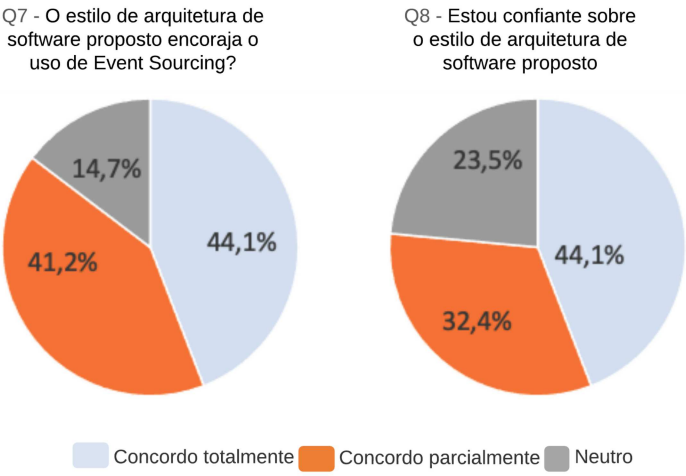
Fonte: Elaborado pelo autor.

De forma geral a grande maioria concorda que o estilo arquitetural possui uma proposta de viabilidade possível. A abordagem da EventChain é a de justamente fornecer um estilo arquitetural que possa responder as exigências das aplicações da área financeira atual e futuro.

Atitude em relação ao uso: Tem como objetivo verificar junto aos participantes a intenção de uso do estilo arquitetural proposto, composta por duas questões (Q7 e Q8). De acordo com 85,3% dos participantes o estilo arquitetural encoraja o uso de *Event Sourcing* nas aplicações. Grande parte destes (76,5%) concordam em estar confiantes no uso do estilo arquitetural, vale ressaltar que não houve nenhum participante que discordou com relação a atitude ao uso. O resumo das respostas pode ser visualizado na Figura 35.

Intenção comportamental de uso: Esse tópico tem uma única questão (Q9) e um objetivo específico de verificar entre os participantes a intenção de testar o estilo arquitetural proposto. A grande maioria (70,6%) concordam que têm a intenção testar estilo

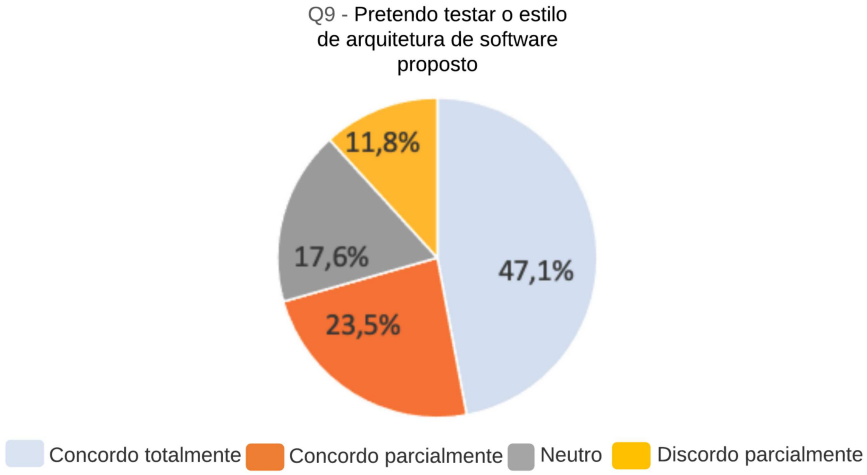
Figura 35 – Gráficos das respostas sobre atitude em relação ao uso



Fonte: Elaborado pelo autor.

arquitetural, sendo que apenas 11,8% discordam parcialmente. Não houve respostas que discordassem totalmente da intenção de testar o estilo arquitetural, os resultados estão na Figura 36.

Figura 36 – Gráficos das respostas sobre intenção comportamental de uso



Fonte: Elaborado pelo autor.

Dentre os participantes que responderam ao questionário de aceitação tecnológica

as áreas de atuação com mais representantes foram a arquitetura de *software* (14) e desenvolvimento (9), a Tabela 15 apresenta uma comparação direta da concordância entre esses dois grupos.

Tabela 15 – Comparação de concordância entre arquitetos(as) e desenvolvedores(as)

Questão (Q)	**ARQ**	**DEV**
Facilidade de uso percebida		
(Q1) Eu penso que o estilo é fácil de aplicar	71,4%	55,5%
(Q2) Eu penso que o estilo é fácil de aprender	57,1%	77,7%
(Q3) Eu penso que o estilo é flexível	100%	66,6%
Usabilidade percebida		
(Q4) O estilo torna mais seguro o armazenamento dos dados gerados através de *Event Sourcing*?	71,4%	77,7%
(Q5) O estilo faz bom uso de Blockchain?	92,8%	88,8%
(Q6) O estilo é uma implementação viável?	100%	100%
Atitude em relação ao uso		
(Q7) O estilo encoraja o uso de *Event Sourcing*?	85,7%	77,7%
(Q8) Estou confiante sobre o estilo proposto	85,7%	44,4%
Intenção comportamental de uso		
(Q9) Pretendo testar o estilo proposto	92,8%	44,4%
Legenda:		
ARQ:Arquitetos(as) **DEV**: Desenvolvedores(as)		

Fonte: Elaborado pelo autor.

Também foi um realizada a comparação entre arquitetos(as) e desenvolvedores(as) utilizando o coeficiente de concordância *Kappa* de Cohen (COHEN, 1960). O coeficiente *Kappa* é uma estatística que busca medir o grau confiabilidade ou reprodutibilidade intraobservadores ou intraobservador para variáveis categóricas nominais.

A utilização do coeficiente *Kappa* de Cohen nesse estudo busca verificar o grau de concordância entre arquiteto(as) e desenvolvedores(as) com elevado nível de experiência (mais de 8 anos) com o desenvolvimento de *software*, foram selecionados quatro arquitetos(as) e quatro desenvolvedores(as) que responderam ao questionário TAM e que tinham diferentes respostas que pudessem ser comparadas. Para isso, observando os participantes que responderam ao questionário TAM foram selecionados os observadores: arquiteto(a) 3 (ARQ3), arquiteto(a) 4 (ARQ4), arquiteto(a) 6 (ARQ6), arquiteto(a) 8 (ARQ8), desenvolvedor(a) 1 (DEV1), desenvolvedor(a) 2 (DEV2), desenvolvedor(a) 5 (DEV5) e desenvolvedor(a) 7 (DEV7).

Foram comparados três grupos que foram organizados da seguinte forma: comparação entre arquitetos(as) e desenvolvedores(as) experientes, comparação entre arquitetos(as) experientes e comparação entre desenvolvedores(as) experientes. A comparação de cada grupo pode ser visualizado abaixo na Figura 37.

Figura 37 – Comparação entre arquitetos(as) e desenvolvedores(as) utilizando *Kappa*

Comparação entre Arquitetos e Desenvolvedores Experientes				
	ARQ 03	ARQ 04	ARQ 06	ARQ 08
DEV 01	0,2941	-0,125	-0,125	-0,125
DEV 03	0,2653	-0,25	-0,25	0,25
DEV 05	0,1176	0	0	0,4
DEV 07	0,1176	0,0625	0,0625	0,4375
Comparação entre Arquitetos Experientes				
	ARQ 03	ARQ 04	ARQ 06	ARQ 08
ARQ 03		0,2653	0,2653	0,2653
ARQ 04	0,2653		1	0,55
ARQ 06	0,2653	1		0,55
ARQ 08	0,2653	0,55	0,55	
Comparação entre Desenvolvedores Experientes				
	DEV 01	DEV 03	DEV 05	DEV 07
DEV 01		0,4375	0,4706	-0,3125
DEV 03	0,4375		0,5714	0,0625
DEV 05	0,4706	0,5714		0,1176
DEV 07	-0,3125	0,0625	0,1176	

Fonte: Elaborado pelo autor.

Através da aplicação do coeficiente de *Kappa* utilizando os três grupos da Figura 37 pode realizar quatro análises possíveis:

- **Análise 1**: Arquitetos(as) e desenvolvedores(as) experientes tiveram um nível de concordância baixo.
- **Análise 2**: Desenvolvedores(as) experientes tiveram um nível de concordância melhor, quando comparado à Análise 1.
- **Análise 3**: Arquitetos(as) experientes tiveram um nível de concordância melhor, quando comparado às Análises 1 e 2.

- **Análise 4**: O perfil dos profissionais favoreceu, até certo ponto, ao nível de concordância em relação ao percepção de aceitação do EventChain..

Os resultados coletados através da aplicação do questionário TAM demonstraram uma boa aceitação entre os profissionais da indústria que responderam as questões. A maioria concordou que o estilo arquitetural proposto possui bastante potencial para uso, inclusive entre a maioria dos participantes houve concordância com relação a pretensão de testar o estilo arquitetural.

Sumário para a QP 3: Foi apresentado e desenvolvido um novo estilo arquitetural, que foi testado sob diferentes casos de testes de performance atendendo de forma consistente as necessidades das aplicações financeiras. Um estudo de caso foi desenvolvido utilizando o estilo arquitetural proposto nesse estudo. Na sequência foi aplicado um questionário de aceitação tecnológica com profissionais da indústria do desenvolvimento de software que atuam na área financeira. Os resultados demonstram uma boa aceitação do estilo arquitetural por parte dos profissionais bem como a intenção de utilização e teste por parte dos mesmos.

6 CONCLUSÃO

Este trabalho propôs um estilo arquitetural para aplicações na área financeira, investigou cinco itens da problemática e três explorou três questões de pesquisa. A utilização de diferentes estilos arquiteturais, bem como a utilização de Blockchain em empresas da área financeira, tendo em vista a necessidade da crescente demanda, a complexidade e o esforço exigido no desenvolvimento de novos *softwares*. Dado o cenário apresentado, esta pesquisa apresentou o EventChain, um novo estilo arquitetural para aplicações da área financeira, que consiste na observação de eventos de baixa ordem gerados pelas aplicações para transformá-los em eventos com mais valor agregado – eventos de mais alta ordem – os quais são armazenados de forma imutável utilizando a Blockchain para isso. O EventChain tem o objetivo de ser um estilo arquitetural que estende outros estilos a fim de simplificar o desenvolvimento de novas funcionalidades, utilizando para isso uma abordagem assíncrona que não interfere nas aplicações e fornece insumos para os especialistas de domínio contando com segurança de que as informações não serão modificadas.

A pesquisa foi organizada em quatro capítulos, excluindo a introdução e a conclusão. O Capítulo 2 apresentou os principais conceitos sobre estilos arquiteturais, trazendo diferentes estilos utilizados, tais como: estilo monolítico, estilo de microsserviços e estilo orientado a eventos, arquiteturas de *software* e Blockchain. No Capítulo 3 foi apresentado o estado-da-arte no que tange a estilos arquiteturais orientados a eventos, realizando uma análise comparativa entre os estudos de acordo com os critérios estabelecidos nesse estudo. Também foi realizado um mapeamento sistemático do uso de Blockchain na área financeira, dado o ineditismo de tal tecnologia.

No Capítulo 4 foi apresentada uma visão geral do estilo arquitetural proposto, assim como os principais conceitos desenvolvidos especificamente para o estilo arquitetural, bem como suas restrições e seu aspecto de implementação. Já no Capítulo 5 foi realizada a avaliação do estilo arquitetural e para isso foram realizadas duas atividades. A primeira consistiu na construção de um protótipo para demonstrar a viabilidade e uso do estilo. Na segunda atividade foi realizada a aplicação do questionário de aceitação tecnológica para entender a aceitação do estilo arquitetural proposto em um dado conjunto de participantes.

Finalmente, este trabalho apresentou o desenvolvimento do EventChain, um novo

estilo arquitetural que possui grande potencial de uso e aceitação dado as suas características. Ser focado em uma área específica, dessa forma atendendo aos anseios da mesma e tornando mais fácil o desenvolvimento de novas aplicações para a área financeira. Aplicando novas tecnologias que tornam muito mais seguro o armazenamento das informações, tornado-as seguras e íntegras por padrão. Permitindo que o desenvolvimento de novos *softwares* possam se concentrar no desenvolvimento de novos requisitos de negócio e endereçando a implementação dos demais requisitos não-funcionais a cargo da aplicação do estilo arquitetural proposto.

6.1 Contribuições

O presente estudo contribuiu cientificamente sob diversos aspectos, a seguir será destacado as três principais contribuições.

Mapeamento sistemático do uso de Blockchain na área financeira. Este estudo realizou um mapeamento sistemático do uso de Blockchain na área financeira. Dessa forma foi possível identificar o quanto as instituições envolvidas na área financeira estão interessadas no uso da tecnologia. Diversas abordagens de uso da tecnologia Blockchain estão em estudos ou mesmo já foram aplicadas de forma corporativa, no entanto, ficou claro que a tecnologia sempre foi utilizada para resolver um problema específico e não como uma plataforma que pode ser aplicada em conjunto com outras tecnologias. O estudo apresentou o uso da tecnologia como base para armazenar informações com mais valor agregado, além de inserir a tecnologia em um estilo arquitetural de forma que proporcione um uso mais amplo para responder aos desafios da área financeira, um contexto de negócio complexo e com alta regulação.

Análise crítica do estado-da-arte. Esta pesquisa realizou uma análise crítica do estado-da-arte de estilos arquiteturais. A partir da análise realizada foi possível identificar as lacunas que os trabalhos não endereçavam em suas análises. Por exemplo, nenhum estilo arquitetural era orientado a resolver os desafios de uma área específica, a tecnologia Blockchain não foi empregada em nenhum dos trabalhos encontrados e nenhuma dos trabalhos utilizou a abordagem de observar os eventos para gerar novos eventos com mais valor agregado. Tal análise se mostrou extremamente importante para constatar as oportunidades de pesquisa no desenvolvimento de novos estilos arquiteturais que utilizassem tecnologias modernas e resolvessem problemas de forma mais

específica.

O estilo arquitetural proposto. O estilo arquitetural proposto atufou diversas lacunas deixadas por diferentes estilos arquiteturais que atuam de forma generalista focando em diferentes áreas de negócio. Nessa pesquisa, para atender o estilo arquitetural, foram criados os componentes *Resolvedor de eventos* e *Cadeia de eventos*, que são componentes especializados desenvolvidos especificamente para observar eventos de baixa ordem por outras aplicações e interpretar esses eventos para enviá-los para a Blockchain. Através da aplicação do estilo arquitetural (Figura 13) em um estudo de caso foi possível observar comparando com o cenário atual (Figura 1) que a arquitetura implementada seguindo, as restrições impostas, possui as seguintes vantagens: (1) fornece um mecanismo que simplifica o processamento dos eventos gerados pelas demais aplicações; (2) garante que as informações contidas nos eventos estarão íntegras e (3) que estarão consistentes devido a cadeia de eventos.

Com o estilo arquitetural apresentado foi verificado que uma nova abordagem utilizando Blockchain pode tornar as aplicações mais robustas e seguras ao mesmo tempo que torna o desenvolvimento de novas funcionalidades mais simples. Dessa forma os desenvolvedores não precisam se preocupar com o complexo ambiente de geração de eventos e garantia de atomicidade das transações que as aplicações desenvolvidas para o mundo financeiro exigem, além de manter os dados íntegros. Os resolvedores de eventos, em conjunto com a cadeia de eventos, fornecem um poderoso mecanismo para novas arquiteturas ou mesmo para se plugar em arquiteturas existentes.

6.2 Limitações e Trabalhos Futuros

Como discutido no decorrer deste estudo, a pesquisa desenvolveu e implementou um estilo arquitetural para aplicações da área financeira. Todavia, novas pesquisas para evoluir o estilo arquitetural são necessárias dado o cenário desafiador e complexo que os desenvolvedores enfrentam no dia-à-dia para implementar novas funcionalidades. Abaixo serão apresentadas as limitações deste estudo bem como as oportunidades de trabalhos futuros que se apresentam.

Área de negócio específica. Este estudo se limitou a definir um estilo de arquitetura de *software* para a área financeira, onde se faz necessário atender requisitos específicos como desempenho, disponibilidade, segurança e integridade das informações. A apli-

cação do estilo arquitetural proposto com outros estilos de arquitetura ou até mesmo outros requisitos não foram estudados. No Capítulo 5 foi apresentada a avaliação de um protótipo utilizando tecnologias modernas e utilizadas amplamente, outras tecnologias não foram validadas quanto a viabilidade de implementação.

Adicionar funcionalidades ao resolvedor de eventos. No Capítulo 4 o estilo arquitetural EventChain propõe o uso de resolvedores de eventos para transformar os eventos de baixa ordem em eventos de alta ordem. Existe a possibilidade de utilizar os eventos de alta ordem em um resolvedor de eventos de tipo especial, permitindo que os eventos de mais alta ordem retroalimentem sistemas ou gerem outros eventos de uma ordem ainda maior. Essa oportunidade de pesquisa expandiria ainda mais a utilização do estilo arquitetural em conjunto com estilo arquitetural orientado a eventos.

Revisar as restrições arquiteturais. Os estilos arquiteturais propõem uma série de restrições arquiteturais que caracterizam cada estilo e diferenciam-se entre si pelas suas particularidades. Com o EventChain não é diferente, uma série de restrições foram apresentadas no Capítulo 4 e aplicadas na prática no Capítulo 5. Uma oportunidade de pesquisa seria a revisão das restrições arquiteturais do estilo arquitetural proposto, validando diferentes cenário na busca de encontrar novas restrições ou aprimorar as existentes.

Realizar experimentos de comparação. No Capítulo 5 foi apresentado a avaliação de um protótipo utilizando uma das funcionalidades básicas da área financeira, porém, outras funcionalidades não foram exploradas. Novas comparações com outros estilos arquiteturais poderão ser realizadas para enriquecer esse estudo e as possibilidades que dele advém.

Por fim, é esperado que as questões e apontamentos levantados em torno dos estilos arquiteturais encorajem novos pesquisadores a utilizar novas arquiteturas de *software* aplicando o estilo arquitetural proposto, bem como a avaliação do estilo arquitetural em conjunto com outras tecnologias atuais.

REFERÊNCIAS

à Nijeholt, H.; OUDEJANS, J.; ERKIN, Z. DecReg: a framework for preventing double-financing using blockchain technology. In: ACM WORKSHOP ON BLOCKCHAIN, CRYPTOCURRENCIES AND CONTRACTS, 2017, New York, NY, USA. **Proceedings...** ACM, 2017. p. 29–34. (BCC '17).

AHMED, S.; MAHMOOD, Q. An authentication based scheme for applications using json web token. In: INTERNATIONAL MULTITOPIC CONFERENCE (INMIC), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p. 1–6.

AKMEL, F. et al. A comparative analysis on software architecture styles. **International Journal in Foundations of Computer Science & Technology**, [S.l.], v. 7, p. 11–22, 11 2017.

Al-Debagy, O.; Martinek, P. A comparative review of microservices and monolithic architectures. In: IEEE 18TH INTERNATIONAL SYMPOSIUM ON COMPUTATIONAL INTELLIGENCE AND INFORMATICS (CINTI), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p. 000149–000154.

ALBERTENGO, G. et al. On the performance of web services, google cloud messaging and firebase cloud messaging. **Digital Communications and Networks**, [S.l.], v. 6, n. 1, p. 31–37, 2020.

ALLEN, R. J. **A formal approach to software architecture**. [S.l.]: CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 1997.

APACHE. **Apache Kafka, a distributed streaming platform**. 2019.

BASS, L.; CLEMENTS, P.; KAZMAN, R. **Software Architecture in Practice, Second Edition**. [S.l.]: Addison-Wesley Professional, 2003.

BELLO, G.; PEREZ, A. J. Adapting Financial Technology Standards to Blockchain Platforms. In: ACM SOUTHEAST CONFERENCE, 2019., 2019, New York, New York, USA. **Proceedings...** ACM Press, 2019. p. 109–116. (ACM SE '19).

BHATTACHARYA, R.; WHITE, M.; BELOFF, N. A blockchain based peer-to-peer framework for exchanging leftover foreign currency. In: COMPUTING CONFERENCE, 2017., 2017. **Anais...** IEEE, 2017. p. 1431–1435.

BOSCO, F.; CROCE, V.; RAVEDUTO, G. **Blockchain Technology for Financial Services Facilitation in RES Investments**. [S.l.]: IEEE, 2018. 1–5 p.

Bucchiarone, A. et al. From monolithic to microservices: an experience report from the banking domain. **IEEE Software**, [S.l.], v. 35, n. 3, p. 50–55, 2018.

BUTERIN, V. **A next-generation smart contract and decentralized application platform.** 2019.

CALVARY, G.; COUTAZ, J.; NIGAY, L. From single-user architectural design to pac*: a generic software architecture model for cscw. In: ACM SIGCHI CONFERENCE ON HUMAN FACTORS IN COMPUTING SYSTEMS, 1997, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 1997. p. 242–249. (CHI '97).

CHAGAS, M. W. et al. Hermes: a natural language interface model for software transformation. In: XV BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEMS, 2019. **Proceedings...** [S.l.: s.n.], 2019. p. 1–8.

CHANG, S. E.; HE, S. Y. Exploring Blockchain Technology for Capital Markets: a case of angel fund. In: IEEE INTERNATIONAL CONFERENCE ON INTERNET OF THINGS (ITHINGS) AND IEEE GREEN COMPUTING AND COMMUNICATIONS (GREENCOM) AND IEEE CYBER, PHYSICAL AND SOCIAL COMPUTING (CPSCOM) AND IEEE SMART DATA (SMARTDATA), 2018., 2018. **Anais...** IEEE, 2018. p. 1941–1948.

CHEN, B.; TAN, Z.; FANG, W. Blockchain-Based Implementation for Financial Product Management. In: INTERNATIONAL TELECOMMUNICATION NETWORKS AND APPLICATIONS CONFERENCE (ITNAC), 2018., 2018. **Anais...** IEEE, 2018. p. 1–3.

CLEMENTS, P. et al. **Documenting software architectures**: views and beyond. [S.l.]: Addison-Wesley Professional, 2002.

COHEN, J. A Coefficient of Agreement for Nominal Scales. **Educational and Psychological Measurement**, [S.l.], v. 20, n. 1, p. 37, 1960.

DANEZIS, G.; MEIKLEJOHN, S. Centrally banked cryptocurrencies. **CoRR**, [S.l.], v. abs/1505.06895, 2015.

di Nitto, E.; Rosenblum, D. Exploiting adls to specify architectural styles induced by middleware infrastructures. In: INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING (IEEE CAT. NO.99CB37002), 1999., 1999. **Proceedings...** [S.l.: s.n.], 1999. p. 13–22.

MAZZARA, M.; MEYER, B. (Ed.). Microservices: yesterday, today, and tomorrow. In: _____. **Present and ulterior software engineering**. Cham: Springer International Publishing, 2017. p. 195–216.

EGELUND-MÜLLER, B. et al. Automated Execution of Financial Contracts on Blockchains. **Business and Information Systems Engineering**, [S.l.], v. 59, n. 6, p. 457–467, 2017.

ETHEREUM.ORG. **Ethereum**. 2019.

EVANS, E. **Domain-driven design**. [S.l.: s.n.], 2003.

FARIAS, K. et al. Uml2merge: a uml extension for model merging. **IET Software**, [S.l.], v. 13, n. 6, p. 575–586, 2019.

FIELDING, R. T.; TAYLOR, R. N. **Architectural styles and the design of network-based software architectures**. 2000. Tese (Doutorado em Ciência da Computação) — University of California, Irvine, 2000. AAI9980887.

FOWLER, M. **Event Sourcing**. Disponível em: <https://martinfowler.com/eaaDev/EventSourcing.html>. Acesso em: 5 maio 2020.

FOWLER, M. **Microservices**. Disponível em: <https://martinfowler.com/articles/microservices.html>. Acesso em: 21 fevereiro 2021.

GIL, D. G.; DÍAZ-HEREDERO, R. A. A microservices experience in the banking industry. In: EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE: COMPANION PROCEEDINGS, 12., 2018, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2018. (ECSA '18).

Gos, K.; Zabierowski, W. The comparison of microservice and monolithic architecture. In: IEEE XVITH INTERNATIONAL CONFERENCE ON THE PERSPECTIVE TECHNOLOGIES AND METHODS IN MEMS DESIGN (MEMSTECH), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p. 150–153.

Guozhen Tan et al. A message-based software architecture style for distributed application. In: SIXTH INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING APPLICATIONS AND TECHNOLOGIES (PDCAT'05), 2005. **Anais...** [S.l.: s.n.], 2005. p. 583–585.

HAN, J.; PARK, C.-m. Case study on adoption of new technology for innovation. **Asia Pacific Journal of Innovation and Entrepreneurship**, [S.l.], v. 11, n. 2, p. 144–158, Jan 2017.

HAN, S.; CHOI, J.-i. V2x-based event acquisition and reproduction architecture with event-sourcing. In: INTERNATIONAL CONFERENCE ON COMPUTING AND DATA ENGINEERING, 2020., 2020, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2020. p. 164–167. (ICCDE 2020).

HEGDE, R. G.; NAGARAJA, G. Low latency message brokers. **Int. Res. J. Eng. Technol.**, [S.l.], v. 7, p. 5, 2020.

HOFMEISTER, C.; NORD, R.; SONI, D. **Applied software architecture**. [S.l.]: Addison-Wesley, Reading, MA, USA, 1999.

HU, Y. et al. A Delay-Tolerant Payment Scheme Based on the Ethereum Blockchain. **IEEE Access**, [S.l.], v. 7, p. 33159–33172, 2019.

HYDRACHAIN. **Hydrachain**. 2019.

HYPERLEDGER.ORG. **Hyperledger Architecture**. [S.l.]: Hyperledger.org, 2017.

HYPERLEDGER.ORG. **Hyperledger FABRIC**. 2019.

HYVÄRINEN, H.; RISIUS, M.; FRIIS, G. A Blockchain-Based Approach Towards Overcoming Financial Fraud in Public Sector Services. **Business and Information Systems Engineering**, [S.l.], v. 59, n. 6, p. 441–456, 2017.

JAISWAL, M. Software architecture and software design. **International Research Journal of Engineering and Technology (IRJET) e-ISSN**, [S.l.], p. 2395–0056, 2019.

KABRA, N. et al. MudraChain: blockchain-based framework for automated cheque clearance in financial institutions. **Future Generation Computer Systems**, [S.l.], v. 102, p. 574–587, jan 2020.

Kamogawa, T.; Okada, H. Enterprise architecture and information systems: in japanese banking industry. In: INTERNATIONAL SYMPOSIUM ON APPLICATIONS AND THE INTERNET, 2008., 2008. **Anais...** [S.l.: s.n.], 2008. p. 433–436.

KIM, J. S.; GARLAN, D. **Analyzing architectural styles**. [S.l.]: Carnegie Mellon University, 2010.

KIS, M.; SINGH, B. A Cybersecurity Case for the Adoption of Blockchain in the Financial Industry. In: IEEE INTERNATIONAL CONFERENCE ON INTERNET OF THINGS (ITHINGS) AND IEEE GREEN COMPUTING AND COMMUNICATIONS (GREENCOM) AND IEEE CYBER, PHYSICAL AND SOCIAL COMPUTING (CPSCOM) AND IEEE SMART DATA (SMARTDATA), 2018., 2018. **Anais...** IEEE, 2018. p. 1491–1498.

LAIGNER, R. et al. From a monolithic big data system to a microservices event-driven architecture. In: EUROMICRO CONFERENCE ON SOFTWARE ENGINEERING AND ADVANCED APPLICATIONS (SEAA), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p. 213–220.

LOULOU, I. et al. A formally specified framework for elaborating event-based architectural styles correct by design. In: MAGHREBIAN CONFERENCE ON SOFTWARE ENGINEERING AND ARTIFICIAL INTELLIGENCE (MCSEAI'06), 2006. **Proceedings...** [S.l.: s.n.], 2006. p. 132–138.

Loulou, I. et al. Compositional specification of event-based software architectural styles. In: IEEE INTERNATIONAL CONFERENCE ON COMPUTER SYSTEMS AND APPLICATIONS, 2006., 2006. **Anais...** [S.l.: s.n.], 2006. p. 337–344.

LUZ, M. A. d.; FARIAS, K. The use of blockchain in financial area: a systematic mapping study. In: XVI BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEMS, 2020, New York, NY, USA. **Anais...** Association for Computing Machinery, 2020. (SBSI'20).

MARANGUNIć, N.; GRANIć, A. Technology acceptance model: a literature review from 1986 to 2013. **Universal Access in the Information Society**, [S.l.], v. 14, n. 1, p. 81–95, 2014.

MAYHEW, K.; CHEN, W. Blockchain - Can It Solve the Security Issues and Fraud Expenses for Credit Card Commerce? In: IEEE 5TH INTL CONFERENCE ON BIG DATA SECURITY ON CLOUD (BIGDATASECURITY), IEEE INTL CONFERENCE ON HIGH PERFORMANCE AND SMART COMPUTING, (HPSC) AND IEEE INTL CONFERENCE ON INTELLIGENT DATA AND SECURITY (IDS), 2019., 2019. **Anais...** IEEE, 2019. p. 37–41.

MCCALLIG, J.; ROBB, A.; ROHDE, F. Establishing the representational faithfulness of financial accounting information using multiparty security, network analysis and a blockchain. **International Journal of Accounting Information Systems**, [S.l.], 2019.

Moises Arantes, G. et al. Improving the Process of Lending, Monitoring and Evaluating Through Blockchain Technologies: an application of blockchain in the brazilian development bank (bndes). In: IEEE INTERNATIONAL CONFERENCE ON INTERNET OF THINGS (ITHINGS) AND IEEE GREEN COMPUTING AND COMMUNICATIONS (GREENCOM) AND IEEE CYBER, PHYSICAL AND SOCIAL COMPUTING (CPSCOM) AND IEEE SMART DATA (SMARTDATA), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p. 1181–1188.

NAKAMOTO, S. **Bitcoin: a peer-to-peer electronic cash system**. 2008.

NORTA, A.; LEIDING, B.; LANE, A. Lowering Financial Inclusion Barriers with a Blockchain-Based Capital Transfer System. In: IEEE INFOCOM 2019 - IEEE CONFERENCE ON COMPUTER COMMUNICATIONS WORKSHOPS (INFOCOM WKSHPS), 2019. **Anais...** IEEE, 2019. p. 319–324.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: an update. **Inf. Softw. Technol.**, [S.l.], v. 64, p. 1–18, 2015.

PETERSEN, K.; VAKKALANKA, S.; KUZNIARZ, L. Guidelines for conducting systematic mapping studies in software engineering: an update. **Information and Software Technology**, [S.l.], v. 64, p. 1 – 18, 2015.

PO, C. et al. **Decentralizing the stock exchange using blockchain an ethereum-based implementation of the bucharest stock exchange**. 2018. 459–466 p.

Porru, S. et al. Blockchain-oriented software engineering: challenges and new directions. In: IEEE/ACM 39TH INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING COMPANION (ICSE-C), 2017., 2017. **Anais...** [S.l.: s.n.], 2017. p. 169–171.

QTUM. **QTUM: defining the blockchain economy**. 2019.

RAIKWAR, M. et al. **A Blockchain Framework for Insurance Processes**. [S.l.]: IEEE, 2018. 1–4 p. v. 2018-Janua.

RICHARDSON, C. **Microservices patterns**. [S.l.: s.n.], 2018.

RODGERS, P. **Service-Oriented Development on NetKernel- Patterns, Processes & Products to Reduce System Complexity**. Disponível em: https://web.archive.org/web/20180520124343/ http://www.cloudcomputingexpo.com/node/80883. Acesso em: 1 maio 2020.

Santos, N.; Rito Silva, A. A complexity metric for microservices architecture migration. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE (ICSA), 2020., 2020. **Anais...** [S.l.: s.n.], 2020. p. 169–178.

Schwanke, R. W. Toward a real-time event flow architecture style. In: EIGHTH ANNUAL IEEE INTERNATIONAL CONFERENCE AND WORKSHOP ON THE ENGINEERING OF COMPUTER-BASED SYSTEMS-ECBS 2001, 2001. **Proceedings...** [S.l.: s.n.], 2001. p. 94–102.

SHAW, M.; GARLAN, D. **Software architecture — perspectives on an emerging discipline**. Upper Saddle River, NJ, USA: [s.n.], 1996.

SOETHOUT, T. Exploiting models for scalable and high throughput distributed software. In: COMPANION OF THE 2019 ACM SIGPLAN INTERNATIONAL CONFERENCE ON SYSTEMS, PROGRAMMING, LANGUAGES, AND

APPLICATIONS: SOFTWARE FOR HUMANITY, 2019, New York, NY, USA. **Proceedings...** Association for Computing Machinery, 2019. p. 35–37. (SPLASH Companion 2019).

SUN, H. et al. Multi-Blockchain Model for Central Bank Digital Currency. In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING, APPLICATIONS AND TECHNOLOGIES (PDCAT), 2017., 2017. **Anais...** IEEE, 2017. p. 360–367.

TAPSCOTT, A.; TAPSCOTT, D. **How Blockchain Is Changing Finance**. 2017.

Taylor, R. N. et al. A component- and message-based architectural style for gui software. **IEEE Transactions on Software Engineering**, [S.l.], v. 22, n. 6, p. 390–406, 1996.

Taylor, R. N.; Medvidovic, N.; Oreizy, P. Architectural styles for runtime software adaptation. In: JOINT WORKING IEEE/IFIP CONFERENCE ON SOFTWARE ARCHITECTURE EUROPEAN CONFERENCE ON SOFTWARE ARCHITECTURE, 2009., 2009. **Anais...** [S.l.: s.n.], 2009. p. 171–180.

TENDERMINT.COM. **Tendermint, tendermint core. byzantine-fault tolerant state machine replication. or blockchain, for short.** 2019.

Tran, H.; Zdun, U. Event actors based approach for supporting analysis and verification of event-driven architectures. In: IEEE INTERNATIONAL ENTERPRISE DISTRIBUTED OBJECT COMPUTING CONFERENCE, 2013., 2013. **Anais...** [S.l.: s.n.], 2013. p. 217–226.

TSAI, W.-T. et al. **A System View of Financial Blockchains**. 2016. 450–457 p.

VALDIVIA, J. A. et al. Patterns related to microservice architecture: a multivocal literature review. **Programming and Computer Software**, [S.l.], v. 46, n. 8, p. 594–608, Dec 2020.

VERNON, V. **Implementing domain-driven design**. [S.l.: s.n.], 2013.

WANG, H.; GUO, C.; CHENG, S. LoC — A new financial loan management system based on smart contracts. **Future Generation Computer Systems**, [S.l.], v. 100, p. 648–655, 2019.

WANG, X. et al. **Inter-Bank Payment System on Enterprise Blockchain Platform**. 2018. 614–621 p. v. 2018-July.

Wessling, F. et al. Towards blockchain tactics: building hybrid decentralized software architectures. In: IEEE INTERNATIONAL CONFERENCE ON SOFTWARE ARCHITECTURE COMPANION (ICSA-C), 2019., 2019. **Anais...** [S.l.: s.n.], 2019. p. 234–237.

WONG, S. et al. A scalable approach to multi-style architectural modeling and verification. In: IEEE INTERNATIONAL CONFERENCE ON ENGINEERING OF COMPLEX COMPUTER SYSTEMS (ICECCS 2008), 13., 2008. **Anais...** [S.l.: s.n.], 2008. p. 25–34.

WU, T.; LIANG, X. Exploration and practice of inter-bank application based on blockchain. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND EDUCATION (ICCSE), 2017., 2017. **Anais...** IEEE, 2017. p. 219–224.

YIN, R. K. **Case study research**: design and methods (applied social research methods). Fourth Edition.. ed. [S.l.]: Sage Publications, 2008.

ZENG, X. et al. A consortium blockchain paradigm on hyperledger-based peer-to-peer lending system. **China Communications**, [S.l.], v. 16, n. 8, p. 38–50, aug 2019.

Zhang, S.; Wang, Y.; Zhang, G. The service architecture of commercial bank. In: INTERNATIONAL CONFERENCE ON NEW TRENDS IN INFORMATION SCIENCE AND SERVICE SCIENCE, 4., 2010. **Anais...** [S.l.: s.n.], 2010. p. 180–184.

Zhao, C.; Zhao, L. The research about software integration oriented heterogeneous architecture style. In: THE 2ND INTERNATIONAL CONFERENCE ON SOFTWARE ENGINEERING AND DATA MINING, 2010. **Anais...** [S.l.: s.n.], 2010. p. 311–315.

APÊNDICE A – LISTA DE ESTUDOS PRIMÁRIOS

Abaixo é apresentado a listagem com os 23 estudos primários selecionando no SMS realizado.

E01 HYVÄRINEN, H.; RISIUS, M.; FRIIS, G. A Blockchain-Based Approach Towards Overcoming Financial Fraud in Public Sector Services. **Business and Information Systems Engineering**, [S.l.], v. 59, n. 6, p. 441–456, 2017

E02 EGELUND-MÜLLER, B. et al. Automated Execution of Financial Contracts on Blockchains. **Business and Information Systems Engineering**, [S.l.], v. 59, n. 6, p. 457–467, 2017

E03 BOSCO, F.; CROCE, V.; RAVEDUTO, G. **Blockchain Technology for Financial Services Facilitation in RES Investments**. [S.l.]: IEEE, 2018. 1–5 p

E04 PO, C. et al. **Decentralizing the stock exchange using blockchain an ethereum-based implementation of the bucharest stock exchange**. 2018. 459–466 p

E05 MCCALLIG, J.; ROBB, A.; ROHDE, F. Establishing the representational faithfulness of financial accounting information using multiparty security, network analysis and a blockchain. **International Journal of Accounting Information Systems**, [S.l.], 2019

E06 Moises Arantes, G. et al. Improving the Process of Lending, Monitoring and Evaluating Through Blockchain Technologies: an application of blockchain in the brazilian development bank (bndes). In: IEEE INTERNATIONAL CONFERENCE ON INTERNET OF THINGS (ITHINGS) AND IEEE GREEN COMPUTING AND COMMUNICATIONS (GREENCOM) AND IEEE CYBER, PHYSICAL AND SOCIAL COMPUTING (CPSCOM) AND IEEE SMART DATA (SMARTDATA), 2018., 2018. **Anais...** [S.l.: s.n.], 2018. p. 1181–1188

E07 NORTA, A.; LEIDING, B.; LANE, A. Lowering Financial Inclusion Barriers with a Blockchain-Based Capital Transfer System. In: IEEE INFOCOM 2019 - IEEE CONFERENCE ON COMPUTER COMMUNICATIONS WORKSHOPS (INFOCOM WKSHPS), 2019. **Anais...** IEEE, 2019. p. 319–324

E08 HU, Y. et al. A Delay-Tolerant Payment Scheme Based on the Ethereum Blockchain. **IEEE Access**, [S.l.], v. 7, p. 33159–33172, 2019

E09 CHANG, S. E.; HE, S. Y. Exploring Blockchain Technology for Capital Markets: a case of angel fund. In: IEEE INTERNATIONAL CONFERENCE ON INTERNET OF THINGS (ITHINGS) AND IEEE GREEN COMPUTING AND COMMUNICATIONS (GREENCOM) AND IEEE CYBER, PHYSICAL AND SOCIAL COMPUTING (CPSCOM) AND IEEE SMART DATA (SMARTDATA), 2018., 2018. **Anais...** IEEE, 2018. p. 1941–1948

E10 BHATTACHARYA, R.; WHITE, M.; BELOFF, N. A blockchain based peer-to-peer framework for exchanging leftover foreign currency. In: COMPUTING CONFERENCE, 2017., 2017. **Anais...** IEEE, 2017. p. 1431–1435

E11 RAIKWAR, M. et al. **A Blockchain Framework for Insurance Processes**. [S.l.]: IEEE, 2018. 1–4 p. v. 2018-Janua

E12 CHEN, B.; TAN, Z.; FANG, W. Blockchain-Based Implementation for Financial Product Management. In: INTERNATIONAL TELECOMMUNICATION NETWORKS AND APPLICATIONS CONFERENCE (ITNAC), 2018., 2018. **Anais...** IEEE, 2018. p. 1–3

E13 WANG, X. et al. **Inter-Bank Payment System on Enterprise Blockchain Platform**. 2018. 614–621 p. v. 2018-July

E14 WANG, H.; GUO, C.; CHENG, S. LoC — A new financial loan management system based on smart contracts. **Future Generation Computer Systems**, [S.l.], v. 100, p. 648–655, 2019

E15 ZENG, X. et al. A consortium blockchain paradigm on hyperledger-based peer-to-peer lending system. **China Communications**, [S.l.], v. 16, n. 8, p. 38–50, aug 2019

E16 KABRA, N. et al. MudraChain: blockchain-based framework for automated cheque clearance in financial institutions. **Future Generation Computer Systems**, [S.l.], v. 102, p. 574–587, jan 2020

E17 KIS, M.; SINGH, B. A Cybersecurity Case for the Adoption of Blockchain in the Financial Industry. In: IEEE INTERNATIONAL CONFERENCE ON INTERNET OF THINGS (ITHINGS) AND IEEE GREEN COMPUTING AND COMMUNICATIONS (GREENCOM) AND IEEE CYBER, PHYSICAL AND SOCIAL COMPUTING (CPSCOM) AND IEEE SMART DATA (SMARTDATA), 2018., 2018. **Anais...** IEEE, 2018. p. 1491–1498

E18 MAYHEW, K.; CHEN, W. Blockchain - Can It Solve the Security Issues and Fraud Expenses for Credit Card Commerce? In: IEEE 5TH INTL CONFERENCE ON BIG DATA SECURITY ON CLOUD (BIGDATASECURITY), IEEE INTL CONFERENCE ON HIGH PERFORMANCE AND SMART COMPUTING, (HPSC) AND IEEE INTL CONFERENCE ON INTELLIGENT DATA AND SECURITY (IDS), 2019., 2019. **Anais...** IEEE, 2019. p. 37–41

E19 WU, T.; LIANG, X. Exploration and practice of inter-bank application based on blockchain. In: INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND EDUCATION (ICCSE), 2017., 2017. **Anais...** IEEE, 2017. p. 219–224

E20 TSAI, W.-T. et al. **A System View of Financial Blockchains**. 2016. 450–457 p

E21 SUN, H. et al. Multi-Blockchain Model for Central Bank Digital Currency. In: INTERNATIONAL CONFERENCE ON PARALLEL AND DISTRIBUTED COMPUTING, APPLICATIONS AND TECHNOLOGIES (PDCAT), 2017., 2017. **Anais...** IEEE, 2017. p. 360–367

E22 BELLO, G.; PEREZ, A. J. Adapting Financial Technology Standards to Blockchain Platforms. In: ACM SOUTHEAST CONFERENCE, 2019., 2019, New York, New York, USA. **Proceedings...** ACM Press, 2019. p. 109–116. (ACM SE '19)

E23 à Nijeholt, H.; OUDEJANS, J.; ERKIN, Z. DecReg: a framework for preventing double-financing using blockchain technology. In: ACM WORKSHOP ON BLOCKCHAIN, CRYPTOCURRENCIES AND CONTRACTS, 2017, New York, NY, USA. **Proceedings...** ACM, 2017. p. 29–34. (BCC '17)