INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO
MESTRADO EM ENGENHARIA INFORMÁTICA




# Harmonização de desempenho, tolerância a falhas e escalabilidade em arquiteturas de microsserviços orientadas a eventos

**PEDRO HENRIQUE LINHAS E MARQUES**
Junho de 2023

POLITÉCNICO
DO PORTO

ISEP INSTITUTO SUPERIOR DE ENGENHARIA DO PORTO

# Harmonização de desempenho, tolerância a falhas e escalabilidade em arquiteturas de microsserviços orientadas a eventos

**Pedro Henrique Linhas e Marques**

**Dissertação para obtenção do Grau de Mestre em**
**Engenharia Informática**
**Especialização em Engenharia de Software**

**Orientadora: Isabel Azevedo**

Porto, Junho 2023

# Dedicatória

Este documento é dedicado a todas as pessoas que me apoiaram e acreditaram sempre em mim.

# Declaração de Integridade

Declaro ter conduzido este trabalho académico com integridade.

Não plagiei ou apliquei qualquer forma de uso indevido de informações ou falsificação de resultados ao longo do processo que levou à sua elaboração. Portanto, o trabalho apresentado neste documento é original e de minha autoria, não tendo sido utilizado anteriormente para nenhum outro fim.

Declaro ainda que tenho pleno conhecimento do Código de Conduta Ética do P.Porto.

ISEP, Junho 2023

Pedro Marques

Pedro Henrique Linhas e Marques

# Resumo

Uma arquitetura de microsserviços é composta por diferentes serviços independentes, que possuem as suas próprias camadas conseguindo comunicar entre si. Os mecanismos de comunicação síncrona utilizados neste tipo de arquitetura conduzem ao aumento da latência e ao risco da existência de *timeouts*. Sabendo-se da possível ocorrência de falhas mecanismos de controlo sobre o estado do sistema são recomendados. Complementarmente, a escalabilidade deve de estar presente consoante as necessidades. Este tipo de arquitetura combinada com abordagens orientadas a eventos pode-se revelar uma alternativa razoável quando se possui requisitos de desempenho, tolerância a falhas e escalabilidade.

Uma visão geral do estado de arte das arquiteturas de microsserviços orientadas a eventos é realizada, com recurso a uma revisão sistemática da literatura. Aqui são exploradas as preocupações, desafios, topologias, tecnologias e métricas a considerar aquando da exploração da temática das arquiteturas de microsserviços orientadas a eventos. Após a recolha de informação, uma análise de valor sobre o valor deste projeto de tese é construída, com recurso ao processo de inovação *New Concept Development* (*NCD*). Alcançando-se a ideia mais relevante de explorar, apoiada pela aplicação do método *Analytic Hierarchy Process* (*AHP*), ou seja, o desenvolvimento de dois protótipos de microsserviços orientados a eventos arquitecturalmente distintos. O primeiro aplicando a topologia mediadora e o segundo aplicando o modelo ator.

Escolhida a ideia mais relevante de se explorar é realizada a análise e a conceção da solução. Abordando-se o problema de negócio de estacionamento de veículos, bem como são construídos os processos de engenharia de requisitos e de análise funcional com recurso ao método *Quality Function Deployment* (*QFD*). Depois evidencia-se as alterações arquiteturais a realizar às aplicações típicas da empresa, para que estas contemplem as abordagens referentes à aplicação da topologia mediadora e do modelo ator. Posteriormente, a implementação dos protótipos correspondentes é realizada, começando-se pela definição das dependências principais de cada protótipo, depois os detalhes de implementação mais relevantes dos mesmos são expressos. O processo de implementação é concluído com a realização de testes, assegurando a qualidade dos protótipos alcançados.

Ulteriormente, a avaliação dos protótipos desenvolvidos é suportada segundo certas métricas, alcançadas com base na abordagem *Goal, Questions, Metrics* (*GQM*), permitindo avaliar o desempenho, a escalabilidade, a disponibilidade e a monitorabilidade de cada protótipo. Os resultados atingidos permitiram comparar os diferentes protótipos, sendo o protótipo baseado na topologia mediadora aquele que melhor harmoniza desempenho, disponibilidade, monitorabilidade e escalabilidade. Por fim, as conclusões do trabalho de mestrado são expressas, onde se apresentam os resultados atingidos, as contribuições, ameaças e pontos de trabalho futuro.

**Palavras-chave**: Arquitetura de software, Microsserviços, Orientados a eventos

# Abstract

A microservices architecture is composed of different independent services, which have their own layers and can communicate with each other. The synchronous communication mechanisms used in this type of architecture lead to increased latency and the risk of timeouts. Knowing the possible occurrence of failures, control mechanisms over the state of the system are recommended. In addition, scalability must be present depending on the needs. This type of architecture combined with event-driven approaches can prove to be a reasonable alternative when performance, fault tolerance and scalability requirements are met.

An overview of the state of the art of event-driven microservices architectures is performed, using a systematic literature review. Here, the concerns, challenges, topologies, technologies and metrics to consider when exploring the topic of event-driven microservices architectures are explored. After collecting information, a value analysis on the value of this thesis project is built, using the New Concept Development (NCD) innovation process. Reaching the most relevant idea to explore, supported by the application of the Analytic Hierarchy Process (AHP) method, that is, the development of two different prototypes using event-driven microservices. The first applying the mediator topology and the second applying the actor model.

Having chosen the most relevant idea to be explored, the analysis and design of the solution is carried out. Addressing the business problem of parking vehicles, requirements engineering and functional analysis processes are built using the Quality Function Deployment (QFD) method. Afterwards, the architectural changes to be made to the company's typical applications are highlighted, so that they include the approaches related to the application of the mediator topology and the actor model. Subsequently, the implementation of the corresponding prototypes is carried out, starting with the definition of the main dependencies of each prototype, then the most relevant implementation details of each prototype are expressed. The implementation process concludes with testing, ensuring the quality of the prototypes achieved.

Next, the evaluation of the developed prototypes is supported according to certain metrics, achieved based on the Goal, Questions, Metrics (GQM) approach, allowing to evaluate the performance, scalability, availability and monitorability of each prototype. The achieved results allowed comparing the different prototypes, being the protype based on mediator topology the one that best harmonizes performance, availability, monitorability and scalability. Finally, the conclusions of the master's work are expressed, where the results achieved, contributions, threats and points for future work are presented.

**Keywords**: Software Architecture, Microservices, Event-driven

# Agradecimentos

Quero agradecer a todos os envolvidos que, direta ou indiretamente, participaram na minha formação em Engenharia Informática e que acompanharam as minhas conquistas até esta etapa.

Primeiramente quero agradecer ao David Mota pelo seu trabalho de supervisor na empresa Devscope. O David mostrou-se sempre disponível para realizar o acompanhamento do projeto de tese, revelando-se bastante curioso e atento durante o desenvolvimento do mesmo.

De seguida pretendo expressar a minha gratidão à professora Isabel Azevedo pelo seu intenso trabalho de mentoria ao longo das diferentes fases de desenvolvimento do projeto de tese. A professora apresentou-se bastante presente e sempre disponível para o esclarecimento de dúvidas.

Por fim quero agradecer especialmente aos meus pais e aos meus amigos. Forneceram-me sempre o apoio necessário, ajudando-me a tomar as melhores decisões de uma forma assertiva e racional, ao longo do desenvolvimento da minha dissertação de mestrado.

# Índice

# Lista de Figuras

# Lista de Tabelas

# Lista de Excertos de Código

# Acrónimos e Símbolos

## Lista de Acrónimos

| | |
|---|---|
| **ACID** | *Atomicity, Consistency, Isolation, Durability* |
| **ACM** | *Association of Computing Machinery* |
| **ADR** | *Action Design Research* |
| **AHP** | *Analytic Hierarchy Process* |
| **AMQP** | *Advanced Message Queuing Protocol* |
| **API** | *Application Programming Interface* |
| **AWS** | *Amazon Web Services* |
| **C4** | *Context, Containers, Components, Code* |
| **CPU** | *Central Processing Unit* |
| **CQRS** | *Command and Query Responsibility Segregation* |
| **CRUD** | *Create, Read, Update and Delete* |
| **CSV** | *Comma-separated values* |
| **DSL** | *Domain Specific Language* |
| **DTO** | *Data Transfer Object* |
| **FCFS** | *First Come First Serve* |
| **FFE** | *Fuzzy Front End* |
| **FIFO** | *First In First Out* |
| **GQM** | *Goal, Questions, Metrics* |
| **HTTP** | *Hypertext Transfer Protocol* |
| **IEC** | *International Electrotechnical Commission* |
| **IEEE** | *Institute of electrical and electronics engineers* |
| **IOT** | *Internet of Things* |
| **ISO** | *International Organization for Standardization* |

| | |
|---|---|
| **IT** | *Information Technology* |
| **JMS** | *Java Messaging Service* |
| **NCD** | *New Concept Development* |
| **NPD** | *New Product Development* |
| **OTP** | *Open Telecom Platform* |
| **QFD** | *Quality Function Deployment* |
| **RAM** | *Random Access Memory* |
| **REST** | *Representational State Transfer* |
| **SLA** | *Service Level Agreement* |
| **UDP** | *User Datagram Protocol* |
| **URL** | *Uniform Resource Locators* |

## Lista de Símbolos

| | |
|---|---|
| **λ** | Lambda |

# 1 Introdução

Este capítulo contextualiza o trabalho de mestrado, apresenta os seus objetivos e o problema subjacente. Também a metodologia de investigação é sintetizada, por fim a descrição da estrutura do documento é exposta.

## 1.1 Contexto

A quantidade total de dados criados, capturados, copiados e consumidos globalmente pelos sistemas de software tem vindo a aumentar expressivamente de ano para ano. No ano de 2019 estima-se que o volume de dados/informação tenha atingido o valor de 41 *zettabytes*, enquanto no ano de 2025 prevê-se o valor de 181 *zettabytes* referente ao volume de dados/informação em circulação (P. Taylor, 2022).

Considerando as previsões de aumento significativo anual do volume de informação, espera-se que os sistemas sejam desenvolvidos de uma forma rápida e económica, apresentando-se também como altamente disponíveis, escaláveis e resilientes (Ambre, 2020). Posto isto, nos últimos anos a procura por sistemas distribuídos tem aumentado e cada vez mais empresas adotam uma arquitetura baseada em microsserviços. Empresas como a Netflix, Uber e Spotify adotam este tipo de arquitetura para conseguirem construir e manter sistemas complexos com sucesso (Bos, 2020). A combinação desta arquitetura com abordagens orientadas a eventos pode ser uma alternativa razoável quando se possui requisitos de desempenho, de tolerância a falhas e de escalabilidade (Ambre, 2020).

A empresa Devscope, que atua em diferentes áreas de negócio, tem evidenciado dificuldade no balanceamento dos requisitos de software mencionados no parágrafo anterior. Pelo que pretende explorar abordagens alternativas de arquiteturas de microsserviços orientadas a eventos, face à abordagem adotada nas aplicações típicas da empresa, com o intuito de perceber e avaliar opções que permitam entregar valor aos seus clientes.

## 1.2 Problema

As aplicações modernas com uma arquitetura baseada em microsserviços visam um desacoplamento e devem incluir vários microsserviços independentes (Cerny, Donahoo, & Trnka, 2017).

A comunicação síncrona conduz ao aumento da latência e ao risco da existência de *timeouts*, sobretudo se existir uma elevada carga de pedidos simultâneos num sistema (Oliveira Rocha, 2022). Falhas de hardware ou de software podem ocorrer em qualquer tipo de sistema, não devendo afetar o correto funcionamento do mesmo (Oliveira Rocha, 2022), pelo que mecanismos de redundância e de controlo sobre o estado do mesmo podem minimizar esta questão. A existência de períodos de elevada realização de pedidos, bem como o contrário, é algo a ter em conta para se conseguir responder à elevada procura e também combater o desperdício de recursos, pelo que os serviços devem de ser escalonados conforme as necessidades (Bellemare, 2020).

A existência de assincronismo, tolerância a falhas e escalabilidade das várias partes de sistemas com uma arquitetura de microsserviços orientada a eventos formam um conjunto de características que, em conjunto, dificultam a obtenção de alguns atributos de qualidade (Ambre, 2020). Assim alcança-se uma classe de problemas comuns a diferentes áreas de negócio que partilham um conjunto de características (Parulkar, 2020)(Aley, 2022). Pelo que mais do que se resolver um problema, pretende-se resolver vários.

Surge então a necessidade de se avaliar as possibilidades para as aplicações que possuem uma arquitetura orientada a eventos e microsserviços, de modo a identificar as que melhor se adaptam à classe de problemas definida. Conclui-se que é relevante a perceção de padrões e tecnologias que permitem considerar o assincronismo e a tolerância a falhas em pedidos realizados entre serviços. Complementarmente, a identificação da topologia numa arquitetura orientada a eventos mais apropriada e as tecnologias relevantes quando desejadas aplicações com alto desempenho e escaláveis é algo a explorar.

## 1.3 Objetivo

O trabalho tem por objetivo explorar arquiteturas de microsserviços orientadas a eventos que permitam harmonizar desempenho, tolerância a falhas e escalabilidade como características num sistema. Para a sua concretização, foram definidas as seguintes atividades/tarefas:

- Sintetizar as preocupações, topologias, tecnologias e métricas relevantes na aplicação de arquiteturas de microsserviços orientadas a eventos;
- Realizar o processo de engenharia de requisitos referente ao negócio de estacionamento de veículos;
- Conceber um protótipo inicial que represente as aplicações típicas da empresa;
- Conceber diferentes protótipos orientados a eventos, evidenciando as diferenças arquiteturais existentes relativamente ao protótipo inicial;

- Implementar os protótipos orientados a eventos concebidos;
- Avaliar os diferentes protótipos implementados segundo as métricas exploradas previamente;

## 1.4 Metodologia de investigação

O método *Action Design Research (ADR)* foi adotado como metodologia de investigação, pois permite a geração de conhecimento através da construção e da avaliação de um conjunto de artefactos num ambiente organizacional (Sein, Henfridsson, Purao, Rossi, & Lindgren, 2011).

Primeiramente a fase de formulação do problema é endereçada sendo composta pelas seguintes ações:

- Realização de revisão sistemática da literatura com o objetivo de se construir o estado de arte, com base nas seguintes questões de investigação (presente no capítulo 2):
    - Quais são as preocupações a ter em conta aquando da construção de arquiteturas de microsserviços orientadas a eventos? (presente na secção 2.2.1);
    - Quais são as tecnologias mais apropriadas na construção de arquiteturas de microsserviços orientadas a eventos? (presente na secção 2.2.2);
    - Que soluções reais segundo uma arquitetura de microsserviços orientada a eventos permitem equilibrar desempenho, tolerância a falhas e escalabilidade? (presente na secção 2.2.3);
    - Quais são as métricas para avaliação de arquiteturas de microsserviços orientadas a eventos? (presente na secção 2.2.4);
- Realização de tomadas de decisão, com recurso a métodos formais, que influenciam a conceção e a implementação da solução mais relevante de exploração (presente no capítulo 3);

De seguida a fase de construção, intervenção e de avaliação é alcançada através das seguintes ações:

- Análise de negócio e requisitos para uma aplicação de estacionamento de veículos, negócio este proposto pela empresa para se endereçar a temática das arquiteturas de microsserviços orientadas a eventos (presente nas secções 4.2, 4.3 e 4.4);
- Conceção do protótipo inicial que representa as aplicações típicas da empresa, que vai permitir a exploração e familiarização das tecnologias usadas nos projetos da mesma; (presente na secção 4.6);
- Conceção dos diferentes protótipos, idealizados na fase de formulação do problema, evidenciando as diferenças existentes entre estes e o protótipo inicial (presente na secção 4.6);
- Implementação e respetiva documentação dos diferentes protótipos; (presente no capítulo 5);

- Realização de experiências e avaliação dos diferentes protótipos segundo diversas métricas. (presente no capítulo 6).

A fase de reflexão e aprendizagem é endereçada paralelamente às duas fases anteriores, sendo possível aqui ajustar as ações das fases anteriores para se conseguir refletir e analisar sobre o trabalho produzido. Finaliza-se com a fase de formalização da aprendizagem onde se sintetizam os resultados atingidos, as contribuições, ameaças à validade da solução e possível trabalho futuro (presente no capítulo 7).

## 1.5 Estrutura do documento

Este documento é composto pelos seguintes capítulos:

- Introdução: aborda o contexto do problema, seguido do problema inerente a este projeto de dissertação, depois os objetivos são delineados e a metodologia de trabalho é exposta. Por fim a estrutura do documento é realçada.
- Estado de Arte: realiza uma revisão sistemática da literatura, foram formuladas quatro questões de investigação que permitiram a angariação e a sintetização de conhecimento sobre a temática das arquiteturas de microsserviços orientadas a eventos.
- Análise de Valor: expõe o valor da solução a entregar, enriquecido com a definição do processo de inovação, onde é escolhida a ideia mais relevante de exploração e respetivas tecnologias de implementação;
- Análise e Conceção: documenta o processo de análise do problema de negócio, proporcionando a compreensão do mesmo. A adoção de determinados padrões é justificada. A conceção dos diferentes protótipos é exposta nesta secção complementarmente.
- Implementação da solução: sumariza os detalhes principais relatados durante o processo de desenvolvimento dos protótipos. Primeiro, a configuração de cada protótipo é exposta, depois os detalhes de implementação dos padrões de software adotados são descritos. Seguem-se os testes unitários, de integração, de sistema e funcionais efetuados.
- Experimentação e Avaliação: define o processo de avaliação que permite classificar os protótipos implementados segundo uma escala, com base nos valores das métricas referentes aos atributos de qualidade que cada protótipo apresenta. Complementarmente, a configuração das ferramentas que permitem a obtenção dos valores das métricas é exposta. Depois, os protótipos também são comparados entre si consoante os valores que possuem por atributo de qualidade. Os testes de hipóteses e de *stress* efetuados aos protótipos são abordados neste capítulo adicionalmente.
- Conclusão: resume o trabalho de mestrado realizado. Começa-se com o nível de concretização dos objetivos e as contribuições referentes ao projeto desenvolvido, depois as ameaças à validade da solução são expressas e termina-se com a enumeração de pontos de trabalho futuro que podem minimizar o impacto das ameaças detetadas.

- Anexo: capítulo adicional que contém a perceção de valor por parte do cliente e a proposta de valor alcançada, revelando-se o conjunto de produtos e serviços que despoletam valor para o cliente Devscope.

# 2 Estado de Arte

Este capítulo descreve o estado atual da temática das arquiteturas de microsserviços orientadas a eventos. Para tal, foi efetuada uma revisão de literatura sistemática com recurso a quatro questões de investigação. Adicionalmente, os padrões de software identificados no processo de revisão de literatura foram introduzidos detalhadamente, assim no momento de conceção da solução torna-se possível ponderar e escolher os padrões mais relevantes de exploração.

## 2.1 Construção de arquiteturas de microsserviços orientadas a eventos

Foi aplicado o método de revisão de literatura sistemática para se obter conhecimento sobre preocupações, tecnologias e métricas a considerar aquando da construção de uma arquitetura de microsserviços orientada a eventos.

A figura 1 sintetiza o processo de realização de uma revisão sistemática de literatura. O processo começa com a definição das questões de investigação, seguem-se os passos de triagem de artigos, de obtenção de palavras-chave consoante os resumos e por fim ocorre a extração de dados e respetivo mapeamento (K. Petersen, Feldt, Mujtaba, & Mattsson, 2008).

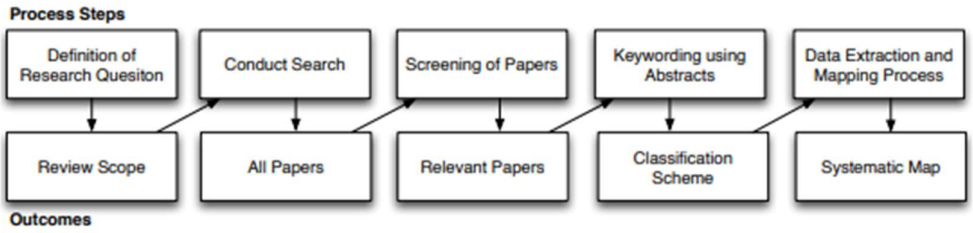



Figura 1 - Passos para realização de uma revisão sistemática da literatura – imagem de (K. Petersen et al., 2008)

### 2.1.1 Questões de investigação

Primeiramente foram formuladas as questões de investigação, que pretendem revelar as intenções que o investigador possui para a condução da revisão sistemática. A tabela 1 contém as questões de investigação e respetiva justificação de adoção.

Tabela 1 - Questões de investigação

| Id | Questão | Justificação |
|---|---|---|
| RQ1 | Quais são as preocupações a ter em conta aquando da construção de arquiteturas de microsserviços orientadas a eventos? | Para se conseguir perceber quais os aspetos a considerar que permitem a construção de uma arquitetura de microsserviços orientada a eventos robusta. |
| RQ2 | Quais são as tecnologias mais apropriadas na construção de arquiteturas de microsserviços orientadas a eventos? | Para se conseguir perceber quais as tecnologias que permitem a construção de uma arquitetura de microsserviços orientada a eventos robusta. |
| RQ3 | Que soluções reais segundo uma arquitetura de microsserviços orientada a eventos permitem equilibrar desempenho, tolerância a falhas e escalabilidade? | Para se conseguir perceber como se deve construir uma solução orientada a eventos, com determinadas características. |
| RQ4 | Quais são as métricas para avaliação de arquiteturas de microsserviços orientadas a eventos? | Para se conseguir avaliar e comparar diferentes soluções orientadas a eventos, de uma forma objetiva. |

### 2.1.2 Critério de investigação adotado

*Institute of electrical and electronics engineers (IEEE), Association of computing machinery* (*ACM*) e *Google Scholar* foram as fontes de informação adotadas. Adicionalmente um conjunto de blogs técnicos, presentes na tabela 2, foram também incluídos como fontes de informação plausíveis para esta revisão de literatura sistemática.

Tabela 2 - Fontes de informação adicionais para a revisão de literatura sistemática

| Blog | URL |
|---|---|
| Martin Fowler | https://martinfowler.com/ |
| O'Reilly | https://www.oreilly.com/ |

A pesquisa foi efetuada através da junção de termos relevantes, resultando nas seguintes consultas de pesquisa:

- Event driven AND microservices AND (patterns OR principles);
- Event driven AND microservices AND (technologies OR tools OR frameworks);
- Event driven AND microservices AND (performance OR fault tolerance OR scalability);
- Event driven AND microservices AND (metrics OR evaluation).

### 2.1.3 Critérios de inclusão e exclusão

Foram aplicados critérios de inclusão e de exclusão para se conseguir identificar quais os estudos relevantes para se obter resposta às questões de investigação formuladas (K. Petersen et al., 2008), processo este representado na figura 2.

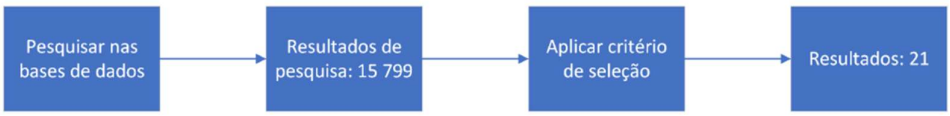



Figura 2 - Processo de seleção de artigos

A tabela 3 expõe os critérios definidos a aplicar no processo de seleção de artigos.

Tabela 3 - Critérios de inclusão e exclusão

| **Critério** | **Justificação** |
|---|---|
| Inclusão | Artigos que mencionem padrões, princípios, ferramentas e tecnologias a implementar numa arquitetura de microsserviços orientada a eventos. |
| | Artigos que reflitam casos práticos de aplicação de uma arquitetura de microsserviços orientada a eventos. |
| | Artigos que apresentem avaliações de desempenho, de tolerância a falhas e de escalabilidade em arquiteturas de microsserviços orientadas a eventos. |
| | Artigos que mencionem métricas de avaliação em arquiteturas de microsserviços orientadas a eventos. |
| | Livros, artigos científicos e relatórios técnicos. |
| Exclusão | Publicações comerciais. |
| | Artigos que não estão escritos em Inglês. |
| | Artigos que não são do âmbito da engenharia de software. |
| | Artigos que não referenciam de uma forma explicita os conceitos de *event driven* e de *microservices*. |
| | Artigos que não demonstram evidências da perspetiva do autor. |

Após aplicação dos critérios expostos foram obtidos os seguintes artigos, presentes na tabela 4.

Tabela 4 - Lista de artigos obtida

| ID | Título | Referência |
|---|---|---|
| 1 | Building Event Driven Microservices | (Bellemare, 2020) |
| 2 | Practical Event-Driven Microservices Architecture | (Oliveira Rocha, 2022) |
| 3 | What do you mean by “Event-Driven”? | (Fowler, 2017) |
| 4 | Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture | (Rudrabhatla, 2018) |
| 5 | Using Event Sourcing and CQRS to build a High-Performance Point Trading system | (Zhong, Li, & Wang, 2019) |
| 6 | Literature Review: A Comparative study of real time streaming technologies and Apache Kafka | (Vyas, Tyagi, Jain, & Sahu, 2021) |
| 7 | AMQP and beyond | (Prajapati, 2021) |
| 8 | A Fair Comparison of Message Queuing Systems | (Fu, Zhang, & Yu, 2021) |
| 9 | Operational stream processing: Towards Scalable and Consistent Event-Driven Applications | (Katsifodimos & Fragkoulis, 2019) |
| 10 | Using microservices and event driven architecture for big data stream processing | (Zhelev & Rozeva, 2019) |
| 11 | Apache Camel based implementation of an Industrial Middleware solution | (Gosewehr, Wermann, Borsych, & Colombo, 2018) |
| 12 | Comparing interservice communications of microservices of e-commerce industry | (Gordesli & Varol, 2022) |
| 13 | Intelligent Sleep monitoring system based on microservices and event driven architecture | (Surantha, Utomo, Lionel, Gozali, & Isa, 2022) |
| 14 | Performance Analysis of Message Queue in the Different Actor System Implementation | (Al-Twajre, 2019) |
| 15 | The actor model based distributed fault tolerant control system | (Lv et al., 2020) |
| 16 | Understanding and addressing quality attributes of microservices architecture: A Systematic literature review | (Li et al., 2021) |
| 17 | Investigating performance metrics for scaling microservices in cloudIoT-Environments | (Gotin, Lösch, Heinrich, & Reussner, 2018) |
| 18 | Chapter 2. Event-Driven Architecture | (Richards, 2019) |
| 19 | Designing microservices systems using patterns: An Empirical Study of quality trade-offs | (Vale et al., 2022) |
| 20 | Quality Assurance for microservices architectures | (Schirgi & Brenner, 2021) |
| 21 | Building Microservices - Designing Fine-Grained Systems | (Newman, 2021) |

### 2.1.4 Categorização

Primeiramente são lidos os resumos com o intuito de se encontrar palavras-chave e conceitos que reflitam a contribuição do artigo, conseguindo assim identificar o contexto da pesquisa (K. Petersen et al., 2008). Assim as palavras-chave de cada artigo são combinadas, resultando um esquema de nível elevado sobre o contexto da pesquisa (K. Petersen et al., 2008). Caso o resumo de um artigo seja insuficiente para se alcançar palavras-chave relevantes, a leitura das secções de introdução e conclusão é aconselhada (K. Petersen et al., 2008). Após a seleção das palavras-chave relevantes, estas são usadas para a formação das categorias usadas na pesquisa. A figura 3 sumariza o processo de construção do esquema de classificação.

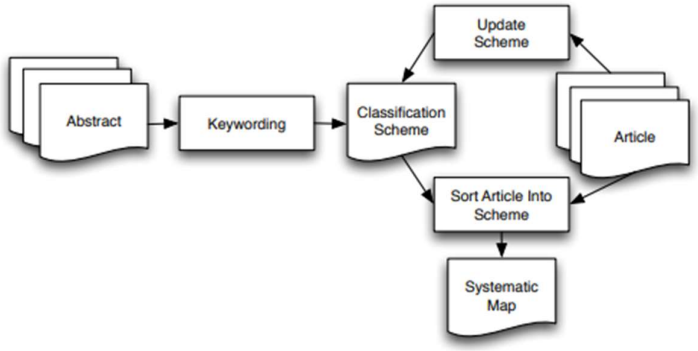



Figura 3 - Construção do esquema de classificação – imagem de (K. Petersen et al., 2008)

A tabela 5 expõe as categorias de investigação alcançadas.

Tabela 5 - Categorias de investigação

| **Categoria de Investigação** | **Descrição** |
|---|---|
| Desenvolvimento de uma arquitetura de microsserviços orientada a eventos | Investigar padrões, princípios e topologias para se implementar uma arquitetura de microsserviços orientada a eventos. |
| Ferramentas usadas na construção de arquiteturas de microsserviços orientadas a eventos | Investigar *frameworks* e *brokers* de mensagens para se implementar uma arquitetura de microsserviços orientada a eventos. |
| Soluções implementadas segundo uma arquitetura de microsserviços orientada a eventos | Investigar casos práticos, segundo uma arquitetura orientada a eventos, que revelem preocupações relacionadas com o desempenho, tolerância a falhas e escalabilidade. |
| Métricas para se avaliar uma arquitetura de microsserviços orientada a eventos | Investigar as métricas de qualidade a considerar aquando do desenvolvimento e da avaliação deste tipo de arquitetura. |

### 2.1.5 Extração de dados e mapeamento

Após a criação das categorias de investigação, foram criados para cada categoria subtópicos. As tabelas em baixo representadas, tabelas 6 até 9, apresentam os artigos distribuídos pelos respetivos subtópicos de cada categoria.

Tabela 6 - Desenvolvimento de uma arquitetura de microsserviços orientada a eventos

| **Descrição** | **Id do Documento** |
|---|---|
| Desafios | 1, 18, 4, 21 |
| Comunicação entre microsserviços | 1, 2, 10, 12, 21 |
| Padrões | 1, 2, 3, 4, 5, 21 |
| Topologias | 1, 18 |

Tabela 7 - Ferramentas usadas na construção de arquitetura de microsserviços orientada a eventos

| **Descrição** | **Id do Documento** |
|---|---|
| Ferramentas de topologia *broker* | 6, 7, 8 |
| Análise de ferramentas de topologia *broker* | 8 |
| Ferramentas de topologia mediadora | 11, 18 |
| *Frameworks* para programação orientada a atores | 9, 15 |
| Análise de *frameworks* para programação orientada a atores | 15 |

Tabela 8 - Soluções implementadas segundo uma arquitetura de microsserviços orientada a eventos

| **Descrição** | **Id do Documento** |
|---|---|
| Soluções | 5, 11, 13, 15 |

Tabela 9 - Métricas para se avaliar uma arquitetura de microsserviços orientada a eventos

| **Descrição** | **Id do Documento** |
|---|---|
| Atributos de qualidade | 16, 20 |
| Avaliação de métricas | 2, 16, 17, 19 |

## 2.2 Análise de dados

Esta secção tem como objetivo dar resposta às questões de investigação identificadas na secção 2.1.1, torna-se possível obter respostas pois os dados relevantes já foram recolhidos e caracterizados em categorias previamente.

### 2.2.1 RQ1 - Quais são as preocupações a ter em conta aquando da construção de arquiteturas de microsserviços orientadas a eventos?

#### 2.2.1.1 Padrões

Foram analisados artigos e livros referentes a padrões, mas realça-se a relevância do estudo efetuado por Martin Fowler, no seu *blog*, onde sintetiza os conceitos inerentes a cada padrão de software a aplicar em arquiteturas orientadas a eventos.

**Event Notification**

*Event Notification* é um padrão de software usado quando um determinado sistema necessita de notificar outros, que existiu uma alteração no seu domínio, para tal são enviadas mensagens de eventos (Fowler, 2017). Martin Fowler acrescenta que a parte principal deste padrão é o facto de o sistema produtor de mensagens não esperar por uma resposta, e caso espere deve receber essa resposta de uma forma indireta. Através da utilização deste padrão cada evento criado não deve conter informação expressiva, por norma contém somente os identificadores de evento e de produtor, para que este possa ser questionado sobre a informação de negócio alterada (Fowler, 2017).

Conclui-se que o consumidor é notificado sobre a ocorrência de algo, mas depois necessita de realizar um pedido ao produtor para se conseguir manter atualizado, pelo que existe um elevado fluxo de comunicações, favorecendo a consistência face à disponibilidade.

**Event Carried State Transfer**

Este padrão de software, como o nome sugere, anuncia que os eventos transferidos possuem estado, pelo que os consumidores não necessitam de recorrer aos produtores para obtenção de informação adicional (Fowler, 2017). Para tal, os consumidores são obrigados a construir uma réplica privada de estado referente aos eventos que querem acompanhar. Conclui-se que cada consumidor apresenta uma cache local que vai sendo atualizada segundo o estado dos eventos recebidos, levando a que exista uma maior disponibilidade, resultado numa diminuição da latência do sistema.

**Command Query Responsibility Segregation**

A aplicação deste padrão permite a separação de responsabilidades, através da divisão dos sistemas em duas partes, a parte de comandos e a parte de consultas (Zhong et al., 2019). Martin Fowler, no seu *blog*, complementa que a ideia deste padrão é possuir estruturas de

dados separadas para leitura e escrita de informação, e realça que a comunicação entre sistemas não é alcançada por este padrão. Na figura 4 observa-se a independência existente entre as partes de comando e de consulta, realça-se que a comunicação pode ser assegurada através do uso de eventos.

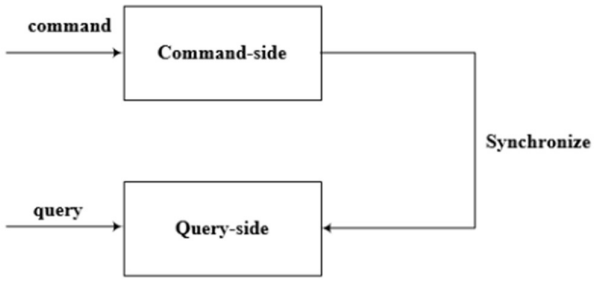



Figura 4 - Padrão CQRS – imagem de (Zhong et al., 2019)

O autor (Oliveira Rocha, 2022) sublinha que possuir uma clara separação entre comandos e consultas, permite a utilização da parte de consultas de uma forma mais despreocupada, visto que esta parte não tem a capacidade de mudar estado. Contrariamente, a parte de comandos pode requerer cuidados adicionais devido à sua capacidade de alteração de estado.

Sistemas que adotam este padrão revelam-se mais simples de conceber e de implementar, comparativamente com outros (Zhong et al., 2019), e o processo de localização de excertos de código que alteram ou que devolvem estado fica facilitado (Oliveira Rocha, 2022).

**Event Sourcing**

A adoção deste padrão de software sugere uma alternativa ao armazenamento de dados. O autor (Oliveira Rocha, 2022) afirma que uma entidade deve possuir o histórico de mudanças efetuadas ao longo do tempo, em vez de possuir somente a alteração mais recente. Logo, uma entidade possui um fluxo de eventos não alteráveis e torna-se a principal fonte de verdade, sendo possível reconstruir os dados de um sistema através do reprocessamento desta fonte (Fowler, 2017). Martin Fowler conclui que é inquestionável a pertinência de se saber o estado atual de algo, mas por vezes torna-se útil saber-se como é que se alcançou um determinado estado.

Vulgarmente, este padrão costuma ser conjugado com o padrão *CQRS*, onde em vez de se armazenar o estado atual de uma entidade como um único registo, projeta-se o estado da entidade observando o histórico de eventos relacionados com essa entidade (Newman, 2021).

**Send/Receive**

Este padrão é caracterizado por oferecer uma comunicação ponto a ponto entre dois serviços para satisfazer um propósito concreto, tipicamente um pedido que pretende efetuar uma determinada ação no segundo serviço (Oliveira Rocha, 2022). Comandos, do inglês *commands*, baseiam-se tipicamente no padrão *Send/Receive*, visto que possuem um propósito específico e

apresentam uma relação próxima relativamente ao domínio que pretendem alterar (Oliveira Rocha, 2022).

**Publish/Subscribe**

Este padrão é usado quando um serviço publica alterações efetuadas ao seu domínio, e os serviços interessados em acompanhar essas alterações subscrevem o canal de publicação. Assim torna-se possível acompanhar e processar as alterações efetuadas noutros serviços (Oliveira Rocha, 2022). Realça-se que o serviço publicador simplesmente garante que a mensagem é publicada no *broker* (Oliveira Rocha, 2022).

**Request/Response assíncrono**

Este padrão é útil de se aplicar quando um serviço necessita de obter informação sobre se as alterações que este despoletou já foram processadas pelos outros serviços (Oliveira Rocha, 2022). Para tal, quando o serviço que irá processar o evento despoletado, pelo serviço original, efetuar esse processamento este deve de publicar um novo evento sinalizando tal ocorrência (Oliveira Rocha, 2022). Os dois eventos, despoletados anteriormente, podem ser correlacionados através de um identificador único, sendo possível assim ao serviço original perceber se as alterações que despoletou já foram processadas pelos outros serviços (Oliveira Rocha, 2022). Conclui-se que caso um serviço seja publicador e recetor de eventos, este por norma estará a aplicar o padrão *Request/Response*.

#### 2.2.1.2 Topologias

O autor (Bellemare, 2020) salienta que o conceito topologia está presente quando se discute a temática de arquiteturas de microsserviços orientadas a eventos. Acrescenta que através desta discussão é percetível qual a relação entre microsserviços, fluxos de eventos e *Application Programming Interface (API)*.

**Topologia Broker**

Na figura 5 sintetiza-se os dois principais componentes arquiteturais deste tipo de topologia, o *broker* e o processador de eventos. O componente *broker* possui os canais de eventos, que podem ser filas de mensagens, tópicos de mensagens ou uma combinação de ambos, a serem usados no fluxo de negócio delineado (Richards, 2019). Nesta topologia cada processador de eventos tem a responsabilidade de publicar os eventos e também de realizar o processamento de um evento recebido, através do uso do componente *broker* (Richards, 2019).

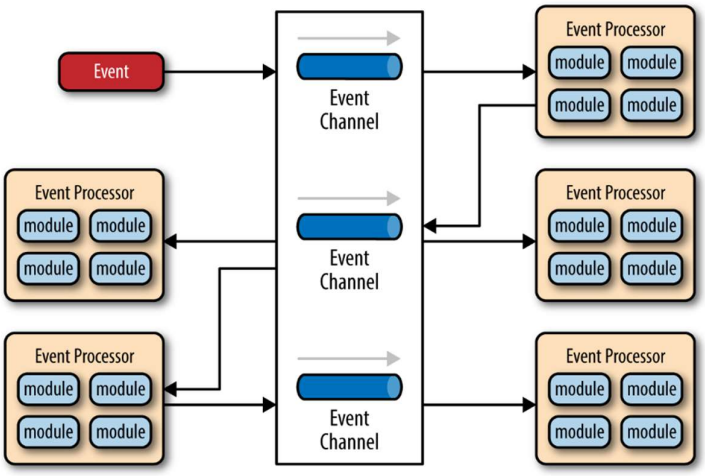



Figura 5 - Arquitetura orientada a eventos através de uma topologia Broker – imagem de (Richards, 2019)

O autor (Richards, 2019) realiza uma analogia referente à topologia, dizendo que esta se compara a uma corrida de estafetas onde cada corredor transporta um bastão durante uma distância, visto que cada corredor só possui o bastão durante um espaço temporal bem definido, cada processador de eventos, também, após transferir o evento não está mais envolvido no processamento desse mesmo evento.

Conclui-se que esta topologia se torna útil quando existe um fluxo simples de processamento de eventos, não existindo a necessidade de existir um orquestrador que realize a gestão destes (Richards, 2019).

**Topologia Mediator**

Na figura 6 os quatro principais componentes arquiteturais deste tipo de topologia, a fila de eventos, o mediador de eventos, o canal de eventos e o processador de eventos estão representados. O autor (Richards, 2019) salienta a distinção entre o evento inicial, que é recebido pelo mediador de eventos, e o evento a processar que vai ser gerado pelo mediador de eventos e recebido pelos processadores de eventos. O mediador de eventos é responsável pela orquestração do evento inicial recebido, consoante o evento recebido este componente envia eventos específicos, para os canais de eventos, a processar por cada processador de eventos (Richards, 2019). Os processadores de eventos possuem a lógica de negócio necessária para processar os eventos recebidos. O autor (Richards, 2019) anuncia que esta topologia não especifica a implementação do componente fila de eventos, podendo ser por exemplo implementado com recurso a uma fila de mensagens. Relativamente ao componente canal de eventos, este pode ser implementado com recurso a uma fila de mensagens ou a tópicos de mensagens.

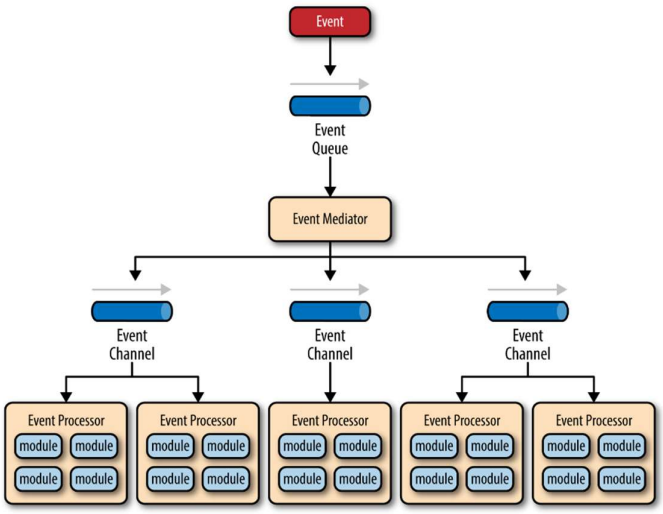



Figura 6 - Arquitetura orientada a eventos através de uma topologia Mediator – imagem de (Richards, 2019)

Conclui-se que esta topologia é relevante de se aplicar, quando se necessita de orquestrar múltiplos passos referentes a um evento.

#### 2.2.1.3 Comunicação entre microsserviços

A comunicação entre microsserviços pode ser efetuada de uma maneira síncrona ou assíncrona.

**Microsserviços síncronos**

Microsserviços síncronos estão altamente ligados a uma abordagem do tipo pedido-resposta, sendo a comunicação entre serviços assegurada por *API's*, que tencionam endereçar os requisitos de negócio (Bellemare, 2020). Para tal, podem fazer uso do padrão *Representational State Transfer (REST)* juntamente com o protocolo *Hypertext Transfer Protocol (HTTP)*, onde as palavras-chave *GET*, *POST*, *PUT* e *DELETE* deste protocolo podem ser usadas para implementação de uma comunicação *REST* (Gordesli & Varol, 2022). Roy Fielding introduziu o estilo arquitetural *REST* na sua dissertação e descreveu como este padrão pode ser usado no desenvolvimento de uma arquitetura para a web (Fielding, 2000).

O autor (Bellemare, 2020), conclui que este tipo de comunicação usada em microsserviços rege-se por determinadas características:

- Acoplamento ponto a ponto:

Cada microsserviço possui as suas próprias dependências e por norma é dependente de outros serviços, que possuem dependências, para conseguir endereçar as suas próprias tarefas de negócio. Isto pode levar a que o número de conexões existentes seja bastante expressivo, resultando na dificuldade de efetuar mudanças futuras (Bellemare, 2020).

- Escalamento dependente:

A escalabilidade de um determinado microsserviço está condicionada pela capacidade de escalabilidade de todos os serviços dependentes, mas também se encontra limitada pela quantidade de microsserviços a percorrer para se alcançar o resultado pretendido inicialmente (Bellemare, 2020).

**Microsserviços orientados a eventos**

A comunicação existente nestes sistemas é diferente daquela explorada no ponto anterior. Aqui os microsserviços não comunicam diretamente entre si, não existindo o problema da espera ativa (Gordesli & Varol, 2022), para conseguirem comunicar fazem uso de plataformas assíncronas.

O autor (Bellemare, 2020) categoriza as plataformas assíncronas em *brokers* de mensagens ou *brokers* de eventos.

A adoção de *brokers* de mensagens permite a comunicação entre sistemas, ao longo da rede, através do mecanismo de *Publish/Subscribe*, por exemplo, em filas de mensagens (Bellemare, 2020). Os produtores são responsáveis pela colocação de mensagens nas filas, para que os consumidores as consumam e realizem o processamento adequado (Bellemare, 2020). Salienta-se que as mensagens são apagadas da fila, após o seu consumo pelos consumidores.

A adoção de *brokers* de eventos permite a construção de um registo de factos ordenamos, mantido durante um tempo que pode ser configurável. O autor (Bellemare, 2020) evidencia que este tipo de *broker* realiza a gestão individual de acesso aos registos através de índices, e conclui que qualquer consumidor consegue aceder aos respetivos eventos.

O autor (Bellemare, 2020) defende que a adoção de *broker* de mensagem ainda é relevante na aplicação de uma arquitetura de microsserviços orientada a eventos. Mas, afirma que este tipo de *broker* não é suficiente para satisfazer todas as exigências deste tipo de arquitetura.

#### 2.2.1.4 Desafios

Aqui exploram-se os desafios resultantes da existência de uma comunicação assíncrona, bem como os desafios inerentes a transações distribuídas.

**Comunicação**

Os autores (Gotin et al., 2018) endereçam os desafios existentes na comunicação entre serviços, através do uso de filas de mensagens, para tal definem certos conceitos que ajudam no entendimento da temática.

Existem três estados que uma fila pode assumir, relacionados com o tamanho desta (Gotin et al., 2018):

- *Steady*: quando o número de mensagens recebidas é igual ao número de mensagens consumidas;
- *Filling*: quando o número de mensagens recebidas é superior ao número de mensagens consumidas;
- *Draining*: quando o número de mensagens recebidas é inferior ao número de mensagens consumidas;

As filas apresentam uma política de *First-come First-serve (FCFS)*, sendo que é introduzido um atraso na camada de aplicação causado pelo tempo de espera de cada mensagem na fila (Gotin et al., 2018). Uma fila fica congestionada quando o tempo necessário para a mensagem passar pela fila ultrapassar o tempo máximo estipulado. Uma fila encontra-se inundada quando o número de mensagens na fila excede o tamanho máximo da fila.

Surge então o desafio de evitar ou recuperar o alcance de filas inundadas ou congestionadas sobretudo em sistemas *cloud* (Gotin et al., 2018). Por norma os microsserviços são de ordem mais consumidora ou produtora. No caso de apresentarem uma componente produtora mais forte, possuem uma taxa de produção superior à de consumo, podendo levar a que fila fique no estado de *filling*. Assim sendo, uma fila eventualmente pode ficar inundada ou congestionada, pelo que a camada de aplicação irá sofrer de degradação de desempenho e de rejeição de mensagens, prejudicando a confiabilidade do sistema (Gotin et al., 2018). Os microsserviços que apresentam uma taxa de consumo superior à de produção não degradam o estado da fila de mensagens. Conclui-se que é fundamental a avaliação do impacto de cada recurso na *cloud*, visto que a implementação deste aumenta os custos operacionais totais, e também é pertinente a recolha de informação sobre o estado de cada fila (Gotin et al., 2018).

**Saga**

As aplicações baseadas em microsserviços possuem as suas tabelas espalhadas por múltiplas bases de dados, pelo que as transações *Atomicity, Consistency, Isolation, Durability (ACID)* não podem ser usadas (Rudrabhatla, 2018). Surge então o conceito de saga, onde cada microsserviço muda o estado da base de dados através de uma transação distribuída, podendo gerar um evento que despolete o microsserviço seguinte (Rudrabhatla, 2018). Caso aconteça alguma falha são efetuados cenários de compensação, que percorrem os microsserviços envolvidos na direção contrária à estabelecida na saga, com o intuito de se recuperar da falha (Rudrabhatla, 2018).

Para se responder a este desafio podem ser usadas duas técnicas distintas, *Event Choreography* ou *Event Orchestration*. *Event Choreography* permite que um microsserviço despolete eventos referentes a outros microsserviços sem existir a necessidade de um coordenador central (Rudrabhatla, 2018). O autor (Bellemare, 2020) associa esta técnica a uma dança, onde cada dançarino efetua o seu papel de uma forma independente sem ser avisado sobre o que deve fazer na sua atuação. *Event Orchestration* adota um coordenador central responsável por

despoletar os eventos relevantes da saga. O autor (Bellemare, 2020) associa esta técnica a uma orquestra musical, onde existe um maestro que comanda os músicos durante a atuação.

### 2.2.2 RQ2- Quais são as tecnologias mais apropriadas na construção de arquiteturas de microsserviços orientadas a eventos?

Realizou-se a divisão das tecnologias por topologia, e adicionalmente concluiu-se que as *frameworks* de programação orientadas a atores também são relevantes no contexto das arquiteturas orientadas a eventos. Pelo que, uma secção referente a estas *frameworks* também foi adicionada. No fim da análise das tecnologias por topologia e das *frameworks* de programação orientada a atores são realizadas as comparações entre estas.

A investigação inerente às tecnologias mais apropriadas teve em consideração alguns critérios de exclusão. Primeiramente a tecnologia tem de ser *open-source* e tem de ser possível de se integrar com uma linguagem de programação conhecida pelo investigador.

#### 2.2.2.1 Ferramentas de topologia broker

**RabbitMQ**

RabbitMQ é um *broker* de mensagens, que faz uso do protocolo *Advanced Message Queuing Protocol (AMQP)*, construído sobre a plataforma *Erlang Open Telecom (OTP)*, permitindo o envio massivo de mensagens entre sistemas (Prajapati, 2021).

Na figura 7 apresentam-se os principais componentes da arquitetura do RabbitMQ. Esta arquitetura permite a existência de elevada disponibilidade devido à aplicação de espelhamento de filas. Cada fila espelhada possui exatamente um mestre, denominado de *master node*, e pode conter múltiplos espelhos, sendo que cada *broker* contém toda a informação de uma determinada fila (Fu et al., 2021). Os autores (Fu et al., 2021) salientam que as operações efetuadas sobre uma fila primeiramente são aplicadas ao nó mestre da fila, sendo posteriormente propagadas para os respetivos espelhos. O componente *Exchange*, que se encontra vinculado a uma fila, é um agente reencaminhador de mensagens para a respetiva fila.

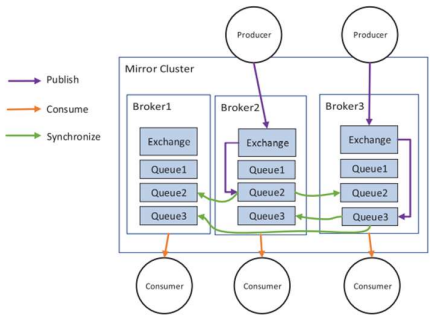



Figura 7 - Arquitetura RabbitMQ – imagem de (Fu et al., 2021)

**Apache Kafka**

Apache Kafka é uma plataforma de processamento de *streams*, permitindo guardar a informação em forma de *stream*. O estudo comparativo entre tecnologias de *streaming* realizado pelos autores (Vyas et al., 2021), segundo diferentes parâmetros de qualidade, concluiu que o Kafka possui uma taxa de transferência superior e é facilmente escalável de acordo com os requisitos de negócio a satisfazer.

A figura 8 representa os principais componentes da arquitetura da tecnologia Kafka. Esta arquitetura é do tipo *peer-to-peer*, onde todos os *brokers* mantêm o mesmo estado (Fu et al., 2021). Os produtores publicam as mensagens no *broker*, modo *push*, e os consumidores de um grupo consomem as partições de um determinado tópico que subscrevem, modo *pull* (Fu et al., 2021). Uma partição é uma fila de mensagens imutáveis ordenadas, e cada mensagem presente aqui possui um número de sequência único por norma intitulado de *offset*. Caso um consumidor necessite de consumir outra vez uma mensagem passada, este pode fazê-lo através do ajuste do *offset*. Os consumidores e os *brokers* são geridos pelo Zookeeper para se alcançar o mecanismo de *load balancing* (Fu et al., 2021).

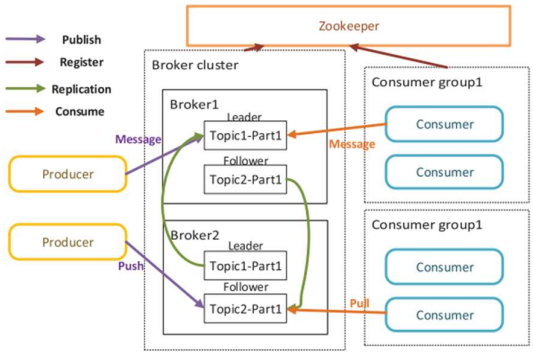



Figura 8 - Arquitetura Kafka – imagem de (Fu et al., 2021)

**ActiveMQ**

ActiveMQ é um *broker* de mensagens que integra a infraestrutura *Java Messaging Service (JMS).*

A figura 9 reflete a arquitetura inerente ao ActiveMQ. O Zookeeper é responsável pela gestão dos *brokers* registados, caso o *broker* mestre esteja indisponível é da responsabilidade do Zookeeper eleger um novo mestre dentro dos escravos disponíveis (Fu et al., 2021). Os produtores e os consumidores comunicam com o *broker* mestre do *cluster*, pois só o mestre consegue providenciar serviço (Fu et al., 2021). Os *brokers* escravos sincronizam a sua informação com o *broker* mestre. ActiveMQ suporta os modos *publish-subscribe* através do conceito tópico, e *point-to-point* através do conceito fila (Fu et al., 2021).

No modo *publish-subcribe* qualquer mensagem possui uma categoria, intitulada de tópico, sendo que um subscritor tem acesso a todas as mensagens do tópico que subscreve, e cada

tópico pode ser subscrito por agentes diferentes (Fu et al., 2021). No modo *point-to-point* as mensagens são enviadas para uma fila específica e somente um consumidor pode obter essa mensagem (Fu et al., 2021).

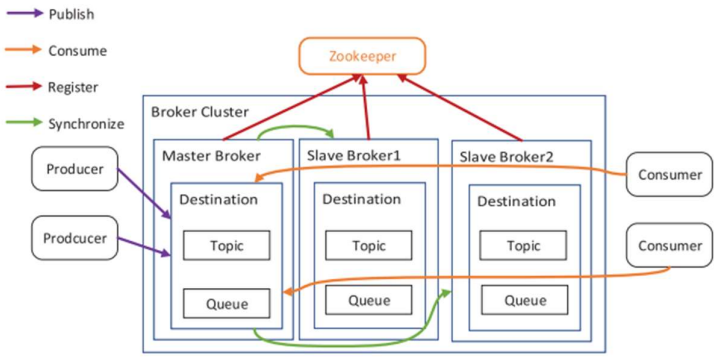



Figura 9 - Arquitetura ActiveMQ – imagem de (Fu et al., 2021)

#### 2.2.2.2 Análise de ferramentas de topologia broker

A tabela 10 sumariza as principais características das 3 tecnologias do tipo *broker*, sendo possível assim realizar uma comparação entre as funcionalidades disponibilizadas por cada tecnologia. A tabela foi construída com recurso ao estudo realizado por (Fu et al., 2021), e atualizada consoante as páginas de Github das tecnologias, onde se identificaram as categorias popularidade, funcionalidades de produção, funcionalidades de qualidade de serviço e funcionalidades adicionais.

Tabela 10 - Comparação entre diferentes tecnologias broker

| **Categoria** | **Tópico** | **Kafka** | **RabbitMQ** | **ActiveMQ** |
|---|---|---|---|---|
| Popularidade | Estrelas | 23.7k | 10.2k | 2.1k |
| | Forks | 12.1k | 3.8k | 1.4k |
| | Issues | 1,031k | 240 | 42 |
| Funcionalidade de produção | Linguagem de desenvolvimento | Scala | Erlang | Java |
| | Protocolo Comum | TCP | AMQP | Multiple |
| | Gestor de cluster | Zookeeper | Erlang | Zookeeper |
| | Arquitetura | P2P | Master-slave | Master-slave |
| | Modelo de fila | Pub-sub | P2P | Pub-sub e P2P |
| | Modo de | Pull | Push/Pull | Pull |
| | Persistência | Disco | Memória/Disco | Memória/Disco/BD |
| | Usabilidade | Mediana | Elevada | Mediana |
| | Compatibilidade | Excelente | Boa | Fraca |

| | | | | |
|---|---|---|---|---|
| Funcionalidades de qualidade de serviço | Entrega garantida | ALL | At-least/most-once | At-least-once |
| | Ordem garantida | Ordem da partição | Não possui | Ordem da fila |
| | Confiabilidade | Elevada | Elevada | Elevada |
| | Escalabilidade | Boa | Fraca | Boa |
| | Transações | Possui | Possui | Possui |
| Funcionalidades adicionais | Batching | Sim | Sim | Sim |
| | Fila de espera | Não | Sim | Não |
| | Fila prioritária | Não | Sim | Sim |

#### 2.2.2.3 Ferramentas de topologia Mediator

O componente principal de uma arquitetura de topologia *mediator*, é o mediador de eventos. A maneira mais usual de implementação de um mediador de eventos é através de *integration hubs*, como o Apache Camel ou o MuleESB (Richards, 2019).

**Apache Camel**

Apache Camel é uma *framework* de integração *open-source*, baseada nos padrões de integração empresariais, que permite a integração de sistemas consumidores e produtores (Camel, 2009). A *framework* permite a definição de regras de mediação e de roteamento através do uso de *DSLs* (Camel, 2021b). Apache Camel caracteriza-se por ser do tipo *standalone*, podendo ser também embebido como uma biblioteca em Spring Boot, Quarkus e servidores de aplicações (Camel, 2009). Possui cerca de uma centena de componentes que são usados para acesso a bases de dados, filas de mensagens e *APIs*. Salienta-se o componente ActiveMQ, que permite o envio e a receção de mensagens, que estende o componente Camel JMS (Camel, 2021a). Por fim, suporta cerca de 50 formatos de dados distintos incluindo os formatos padrão das indústrias financeira, saúde e de comunicações (Camel, 2009).

**Mule ESB**

Mule ESB é uma *framework* de integração que permite a conexão entre sistemas, de uma forma intuitiva e facilitada, permitindo a troca de mensagens entre estes (Mulesoft, 2015). A *framework* não integra a implementação de *JMS*, mas providencia *APIs* e mecanismos para integração de qualquer *JMS* compatível, como por exemplo o ActiveMQ e o MuleMQ (García-Jiménez, Martínez-Carreras, & Gómez-Skarmeta, 2010). A *framework* permite a orquestração de eventos em tempo real ou em *batch* e possui conectividade universal (Mulesoft, 2015). Importa destacar a capacidade de mediação de serviços, que a *framework* possui, que permite separar a lógica de negócio da lógica de mensagens existindo a possibilidade de criação de fluxos concretos orientados ao negócio (Mulesoft, 2015).

#### 2.2.2.4 Análise de ferramentas de topologia mediator

A tabela 11 sumariza as principais características das duas tecnologias do tipo *mediator*, sendo possível assim realizar uma comparação entre as funcionalidades disponibilizadas por cada tecnologia. A tabela foi construída após análise do *post* presente em (T. Taylor, 2020), após análise das páginas de documentação das tecnologias e consoante as páginas de Github das tecnologias. Assim identificaram-se as categorias popularidade, funcionalidades de produção e outras funcionalidades.

Tabela 11 - Comparação entre diferentes tecnologias mediator

| **Categoria** | **Tópico** | **Apache Camel** | **Mule ESB** |
|---|---|---|---|
| Popularidade | Estrelas | 4.6k | 339 |
| | Forks | 4.6k | 653 |
| | Issues | 8.8k | 11.9k |
| Funcionalidades de produção | Linguagem de programação | Java | Java |
| | Criação de routes | Uso de DSLs | Uso de ficheiros de configuração XML. |
| | Criação de fluxos de integração | Uso de DSLs | Uso de ficheiros de configuração XML. |
| | Documentação | Forte | Fraco |
| | Suporte | Exclusivamente suporte da comunidade | MuleSoft oferece suporte dedicado |
| | Implantação em cloud | Possível | Possível |
| | Java Messaging Service | Via Componentes | Via APIs |
| | Conectividade | Via componentes | Via conectores |
| | Criação de conectividade | Via Maven Archetypes | Via Maven Archetypes |
| | User Interface | Não visual | Visual |
| Outras funcionalidades | Batching | Sim | Sim |

#### 2.2.2.5 Frameworks para programação orientada a atores

A programação orientada a atores permite a construção de sistemas concorrentes, distribuídos, reativos, resilientes e tolerantes a falhas (Lv et al., 2020). Consequentemente a decisão de

exploração de uma forma mais detalhada sobre este estilo de programação surgiu, para se complementar a temática deste projeto de tese, em detrimento da programação orientada a agentes. A programação orientada a agentes é mais adequada para sistemas complexos com um número avultado de componentes em interação, enquanto a programação orientada a atores por norma é aplicada em sistemas altamente responsivos e com necessidades de escalonamento. No parágrafo seguinte é realizada uma breve descrição do modelo agente somente para se introduzir o tema.

No estudo (Ingham, 1999) realizado por James Ingham reporta-se que o conceito agente é algo ambíguo, existindo diferentes pontos de vista sobre a verdadeira essência de um agente. Um agente, segundo James, é uma entidade que tem a capacidade de controlar a sua própria ação independentemente de outras entidades, com exceção da possível comunicação dependente que pode existir entre entidades (Ingham, 1999). Um agente pode ser reativo e proativo, apenas reativo ou apenas proativo. Ou seja, um agente pode ser: 1. Um sistema puramente orientado a eventos, reativo; 2. Um sistema orientado a objetivos, proativo; ou 3. Uma combinação de ambas as abordagens 1 e 2 (Ingham, 1999).

O modelo ator apresenta-se como um modelo conceptual, presente na figura 10, cada ator é uma unidade fundamental de computação que pode realizar as seguintes tarefas: 1. Criar outro ator; 2. Enviar uma mensagem; 3. Anunciar como vai lidar com a próxima mensagem.

Os atores são caracterizados por não partilharem memória e por possuírem a sua própria caixa de correio. As mensagens neste modelo são imutáveis e são acessíveis de se enviarem via rede. Cada ator pode estar hospedado localmente ou remotamente e possui um endereço, através deste endereço torna-se possível a comunicação entre atores (Tiwari, 2019). Os atores processam uma mensagem de cada vez, utilizando um mecanismo de comunicação assíncrona (Tiwari, 2019). A caixa de correio de cada ator utiliza o mecanismo *First In First Out (FIFO)*.

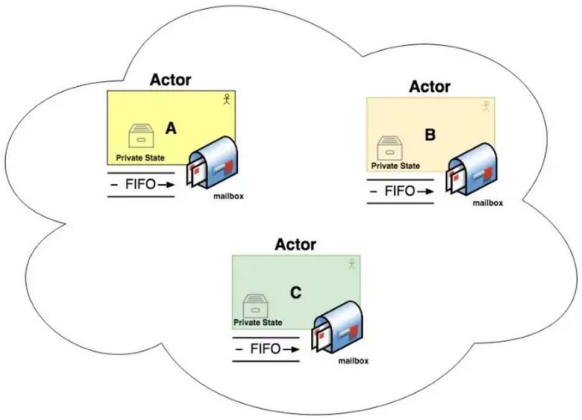



Figura 10 - Modelo Conceptual Ator – imagem de (Tiwari, 2019)

Os autores (Katsifodimos & Fragkoulis, 2019), concluem que o Akka e o Microsoft Orleans são das *frameworks* orientadas a atores mais populares, permitindo a aplicação de padrões úteis,

*CQRS* e *event sourcing* por exemplo, na construção de arquiteturas de microsserviços orientadas a eventos.

**Akka.NET**

Akka.NET é uma *framework* que permite a construção de aplicações tolerantes a falhas orientadas a eventos, permitindo aos desenvolvedores de C# utilizarem as funcionalidades originais do Akka desenvolvido inicialmente em Java/Scala (Akka, 2017).

Os atores estão distribuídos segundo uma árvore de supervisão. É da responsabilidade do nó supervisor, nó pai, supervisionar os seus nós, nós filhos. Caso um nó filho falhe, o nó pai recebe uma notificação sendo possível este decidir o que fazer com o nó filho. Sempre que um nó é reiniciado as suas mensagens não são perdidas, ficando armazenadas na respetiva caixa de correio.

**Microsoft Orleans**

A *framework* Orleans permite a construção de sistemas distribuídos de uma forma robusta e escalável, simplificando a construção destes através do fornecimento de padrões e de *APIs* (Microsoft, 2022).

Nesta *framework* uma *grain* é a implementação de um ator, sendo composta por identidade, comportamento e estado como consta na figura 11. Uma *grain* pode-se encontrar ativada, a executar ou desativada. Uma *grain* pode ser volátil ou persistida em qualquer sistema de armazenamento (Microsoft, 2022).


Figura 11 - Composição de uma grain – imagem de (Microsoft, 2022)

*Silo* é outro conceito relevante nesta *framework*, considera-se um anfitrião de *grains*, sendo responsável pela ativação ou desativação das *grains*. A utilização de agrupamentos de *silos* permite obter um sistema escalável e tolerante a falhas (Microsoft, 2022). Na figura 12 é percetível a relação entre os conceitos falados anteriormente.

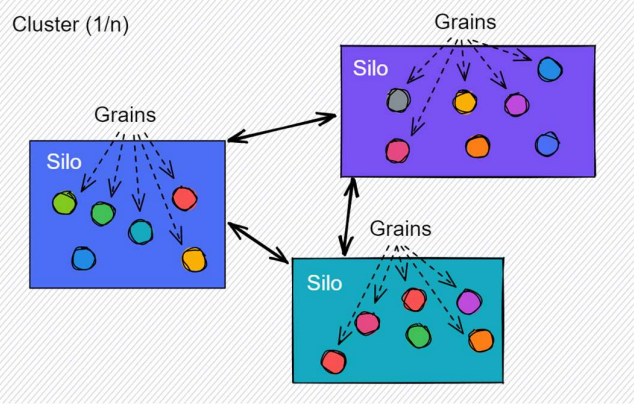



Figura 12 - Relação entre conceitos de relevo adotados nesta framework – imagem de (Microsoft, 2022)

#### 2.2.2.6 Análise de frameworks de programação orientada a atores

A tabela 12 sumariza as principais características das duas *frameworks* de programação orientada a atores, sendo possível assim realizar uma comparação entre as funcionalidades disponibilizadas por cada *framework*. A tabela foi construída com recurso ao estudo efetuado por (Kuhn, 2015), onde se identificaram as categorias popularidade, funcionalidades de produção e funcionalidades de qualidade de serviço.

Tabela 12 - Comparação entre diferentes frameworks de programação orientada a atores

| **Categoria** | **Tópico** | **Orleans** | **Akka.NET** |
|---|---|---|---|
| Popularidade | Estrelas | 9k | 4.2k |
| | Forks | 2k | 1k |
| | Issues | 384 | 335 |
| Funcionalidades de produção | Linguagem de programação | C# com .NET | C# com .NET |
| | Ciclo de vida | Grains Activations possuem do início ao fim. | Atores possuem. |
| | Criação automática | Grains criadas automaticamente quando necessário. | Atores são criados explicitamente pelos seus pais. |
| | Implantação | Silos. | ClusterSharding. |
| | Atores virtuais | Tipo nominal + GUID. | ActorRef. |
| | Comunicação entre atores | Via envio de mensagens | Via envio de mensagens |

| | | | |
|---|---|---|---|
| | Instanciação de atores | Uma Grain pode possuir uma ou mais ativações no mesmo período. | Um ActorRef referencia somente um ator criado previamente. |
| | Transparência da localização | Uma Grain não possui conhecimento sobre a sua localização física. | ActorRef é a abstração central de localização. |
| | Referências ao ator | A referência a uma Grain não possui informação de localização. | ActorRef contém toda a informação necessária para se enviar uma mensagem a um ator. |
| | Persistência | Baseado em Snapshots. | Baseado em Event Sourcing. |
| | Multi-Tasking | Processa uma mensagem de cada vez sem interrupção. | Processa uma mensagem de cada vez sem interrupção. |
| | Serialização | Possui geradores de código que emitem código de serialização. | Utiliza bibliotecas externas. |
| Funcionalidades de qualidade de serviço | Garantia de entrega de mensagens | At-most-once | At-most-once |
| | Load Balancing | Silos | ClusterSharding |
| | Distributed Directory | Utiliza uma tabela hash distribuída. | Utiliza uma abordagem sharding. |

### 2.2.3 RQ3 – Que soluções reais segundo uma arquitetura de microsserviços orientada a eventos permitem equilibrar desempenho, tolerância a falhas e escalabilidade?

Os autores (Zhong et al., 2019) expõem um caso prático de uso do modelo conceptual ator juntamente com os padrões *event sourcing* e *CQRS* para a construção de um sistema de transação de pontos. Os autores adotaram esta abordagem pois os sistemas tradicionais de transação de pontos apresentam problemas de competição de recursos e *deadlocks* da base de dados, levando a que os requisitos de desempenho e de escalabilidade não sejam devidamente

alcançados (Zhong et al., 2019). Para tal os conceitos principais de negócio como por exemplo Utilizador e Ordem, foram implementados com recurso a atores Akka, pois cada ator consiste num conjunto de atributos e comportamentos privados, sendo possível a adequação do comportamento do mesmo consoante a diversidade das mensagens recebidas (Zhong et al., 2019). A transferência de pontos entre utilizadores é realizada através da criação de um evento que contém identificador, utilizador que envia os pontos, utilizador que recebe os pontos e os respetivos pontos. Cada utilizador consegue a obtenção dos pontos disponíveis através do varrimento de todos os eventos que possuem o seu identificador, obtendo o balanço de pontos até ao momento. Concluem que através desta abordagem já não existe a necessidade de trancar a base de dados, sendo que as operações de transferência de pontos podem ser efetuadas concorrentemente e com tolerância a falhas (Zhong et al., 2019).

Os autores (Surantha et al., 2022) efetuaram um estudo sobre a viabilidade de implementação de um sistema de monitorização de sono com recurso a sensores, assente na plataforma *Internet Of Things* (*IOT*), aplicando uma arquitetura de microsserviços orientada a eventos. A figura 13 evidencia a arquitetura alcançada. O componente *Sensor Gateway* é responsável por receber os dados provenientes dos sensores, usa um mecanismo de comunicação assíncrona para que o componente *Sensor data Persister* consiga registar a informação na base de dados (Surantha et al., 2022). Realça-se a separação de responsabilidades entre o consumo e o registo de dados, que permitiu a maximização do desempenho e da disponibilidade do sistema proposto. A realização de testes experimentais permitiu concluir que a arquitetura proposta permitiu aumentar a taxa de transferência em 34,76%, diminuir o tempo de resposta em 55,85% e reduzir o consumo de memória em 37,26% por cada instância replicada quando comparada com uma solução de microsserviços não orientada a eventos (Surantha et al., 2022).

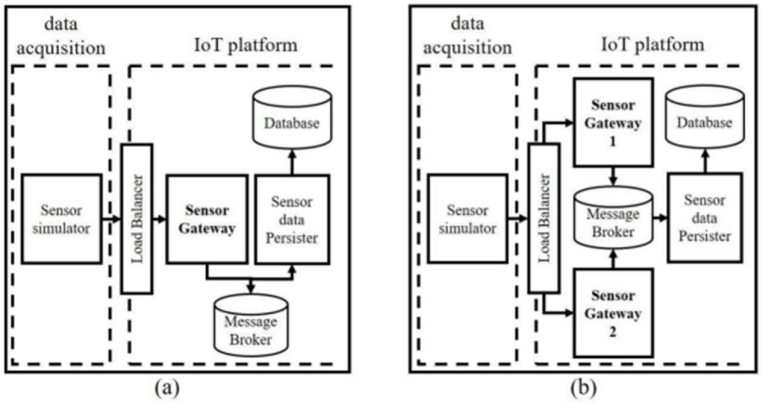



Figura 13 - Arquitetura orientada a eventos de microsserviços alcançada – imagem de (Surantha et al., 2022)

Os autores (Gosewehr et al., 2018) expõem um cenário de implementação de um *middleware* industrial com recurso a Apache Camel. O *middleware,* para satisfazer os requisitos de negócio expostos, deve conseguir agregar dados, através da recolha e armazenamento de dados de diferentes fontes, e processá-los através do roteamento dos dados agregados para os múltiplos

serviços de destino (Gosewehr et al., 2018). Concluem que a adoção desta tecnologia é útil em casos onde existem microsserviços, e onde existe a necessidade de mediar/gerir microsserviços distintos em termos de protocolos e *APIs* que fazem uso (Gosewehr et al., 2018).

### 2.2.4 RQ4 – Quais são as métricas para avaliação de arquiteturas de microsserviços orientadas a eventos?

Nesta secção abordam-se os atributos de qualidade a considerar numa arquitetura de microsserviços orientada a eventos e as técnicas e métricas que permitem endereçar os atributos de qualidade descobertos.

#### 2.2.4.1 Atributos de qualidade

Um atributo de qualidade é considerado um requisito não funcional, podendo ser usado como algo mensurável que permite avaliar se um determinado sistema satisfaz os interesses das partes interessadas (Schirgi & Brenner, 2021).

Os autores (Li et al., 2021) realizaram uma revisão sistemática da literatura com o objetivo de identificar e sintetizar estudos que relatem os atributos de qualidade usados em arquiteturas de microsserviços. Foram identificados 72 estudos principais, e consoante os dados extraídos destes foi realizada uma visão geral dos 6 atributos de qualidade mais relevantes a aplicar numa arquitetura de microsserviços (Li et al., 2021). De seguida, identificaram 19 táticas que permitem endereçar de uma forma objetiva os atributos de qualidade. Os autores (Li et al., 2021) consideram escalabilidade e desempenho como os atributos de qualidade mais críticos a ter em conta, face aos outros 4 identificados: disponibilidade, monitorabilidade, segurança e testabilidade, na construção de sistemas de microsserviços. Na figura 14 estão representadas as táticas a aplicar em cada atributo de qualidade, para se conseguir construir um sistema que apresente qualidade.

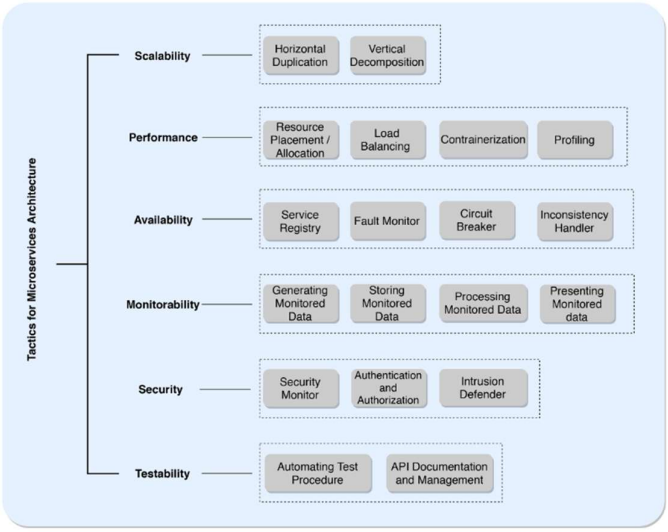



Figura 14 -Táticas a aplicar em atributos de qualidade – imagem de (Li et al., 2021)

#### 2.2.4.2 Avaliação de métricas

As compensações arquiteturais, *architectural trade-offs*, por norma são expressas como atributos de qualidade que são afetados de alguma maneira através da aplicação de determinada estratégia, como por exemplo o uso de uma nova infraestrutura de implantação ou a aplicação de um novo padrão (Vale et al., 2022). Pelo que, a obtenção de um conhecimento objetivo sobre a qualidade de software deve ir mais além face à perceção subjetiva do desenvolvedor, devendo ser identificadas métricas para se medirem os atributos de qualidade (Vale et al., 2022). O estudo conduzido pelos autores (Vale et al., 2022) revelou que os intervenientes possuem dificuldade na identificação de métricas de qualidade, por outro lado conseguem identificar técnicas e táticas com facilidade que permitem endereçar os atributos de qualidade.

**Escalabilidade**

A escalabilidade permite avaliar a capacidade de um sistema conseguir acrescentar recursos para responder a uma quantidade variável de pedidos (Li et al., 2021).

Duplicação horizontal anuncia o aumento ou a diminuição das instâncias de microsserviços consoante o número de pedidos a responder por estes (Li et al., 2021). Decomposição vertical pretende a separação de um sistema em diferentes serviços e bases de dados para se alcançar um sistema independentemente escalável (Li et al., 2021). Surge então a métrica **Escalonamento horizontal/vertical**, que pretende que os microsserviços funcionem corretamente apesar do seu tamanho sofrer alterações regularmente, horizontalmente ou verticalmente. O estudo, conduzido pelos autores (Vale et al., 2022), revela que os participantes fazem uso da técnica de escalonamento automático, utilizando a infraestrutura Amazon Web Services (*AWS*) ou Kubernetes, que endereça a métrica abordada na frase anterior.

O **número de dependências síncronas entre serviços** é uma métrica a considerar, caso este valor seja expressivo significa que o sistema possui fragilidades do ponto de vista da escalabilidade (Oliveira Rocha, 2022). A existência de **baixo acoplamento** é uma métrica relevante, pois permite que cada serviço faça uso somente do necessário dos outros serviços (Oliveira Rocha, 2022).

**Desempenho**

O desempenho permite avaliar a capacidade de um sistema conseguir alcançar os requisitos temporais supostos aquando da resposta a um evento (Li et al., 2021).

A **utilização média do *Central Processing Unit*** (CPU) e a **utilização média de *Random Access Memory*** (RAM) de cada microsserviço são métricas a considerar pois permitem a avaliação do consumo de recursos (Gotin et al., 2018)(Vale et al., 2022). O **crescimento da fila de mensagens** é uma métrica que permite o cálculo da diferença entre as taxas de chegada e de partida de mensagens de uma determinada fila (Gotin et al., 2018). O **atraso de uma fila de mensagens** é

outra métrica relevante que permite a obtenção do tempo de espera de uma mensagem antes de ser efetivamente processada (Gotin et al., 2018). O **tempo de resposta** é uma métrica que permite obter o tempo decorrido desde o momento do pedido até à obtenção de resposta por parte de um microsserviço.

O estudo, conduzido pelos autores (Vale et al., 2022), revela que os participantes fazem uso de *dashboards* para endereçarem as métricas de desempenho, permitindo assim a consulta dos indicadores de uma forma sintetizada mantendo o nível de abstração elevado.

**Disponibilidade**

A disponibilidade permite avaliar a capacidade de um sistema conseguir recuperar de falhas tendo em conta que o período de indisponibilidade, provocado pela falha, não deve ser superior ao estipulado previamente (Li et al., 2021).

A **deteção de falhas** é uma métrica relevante para o atributo de qualidade em questão, as falhas idealmente devem de ser detetadas antes de o sistema proceder a ações que permitam recuperar o seu estado (Li et al., 2021). A utilização de um monitor de falhas, do inglês *fault monitor*, é uma técnica de monitoramento contínuo usada para descobrir falhas nos microsserviços que é vulgarmente usada para identificar a ocorrência de falhas (Li et al., 2021).

O estudo, conduzido pelos autores (Vale et al., 2022), anuncia que os participantes refletem sobre a disponibilidade de um sistema aquando da análise do nível de serviço esperado pelo cliente (*SLA*). A **medição do tempo de disponibilidade** e a **medição do número de pedidos concluídos com sucesso** de cada microsserviço são métricas a considerar para se avaliar de uma forma objetiva a disponibilidade de algo. Conclui-se que os participantes fazem uso de estatísticas em tempo de execução para sustentarem as métricas de **latência**, que permite a obtenção do tempo necessário de transferência dos dados de um ponto para outro na rede, e de **tráfego**, que permite a perceção do uso geral da rede (Vale et al., 2022).

**Monitorabilidade**

A monitorabilidade permite avaliar a capacidade de um sistema suportar operações de monitorização enquanto o sistema se encontra a ser executado (Li et al., 2021).

No estudo, efetuado pelos autores (Vale et al., 2022), os participantes possuem uma opinião subjetiva sobre as métricas a aplicar para endereçar a monitorabilidade como atributo de qualidade. Identificou-se a técnica de geração e armazenamento de dados monitorizados, onde devem ser gerados e guardados: 1. ficheiros de análise das métricas dos serviços recolhidos em tempo de execução; 2. ficheiros de log que possuem informação de pedidos recebidos e efetuados pelos microsserviços e respetivas comunicações entre partes interessadas (Li et al., 2021). Os dados devem ser guardados em sistemas centralizados ou descentralizados (Li et al., 2021). A técnica de apresentação de dados monitorizados também foi identificada, aqui importa realçar que após a recolha de dados existe o processamento dos mesmos, sendo que os dados devem ser apresentados de uma forma agregada ou não agregada tendo em conta a

informação que cada parte interessada espera observar. Os participantes do estudo (Vale et al., 2022) utilizam ferramentas de monitorização externas, como o Grafana, Prometheus e Google Analytics, de acordo com as suas necessidades de monitorização.

**Segurança**

A segurança permite avaliar a capacidade de um sistema proteger os dados de acesso não autorizado, permitindo o acesso a utilizadores e sistemas autorizados (Li et al., 2021).

Semelhante ao que se registou no atributo de qualidade monitorabilidade os participantes não conseguiram identificar objetivamente as métricas de segurança (Vale et al., 2022). Autenticação e autorização é uma técnica de relevo utilizada do ponto de vista da segurança. A autenticação é um processo que confirma a identidade de um determinado utilizador, enquanto a autorização é a definição do mapeamento das funcionalidades que certo utilizador é autorizado a efetuar (Li et al., 2021). A utilização de monitor de segurança, do inglês *security monitor*, é uma técnica que permite a observação de comportamentos anómalos e de ataques em diferentes níveis de uma arquitetura orientada a microsserviços (Li et al., 2021). Os participantes endereçam as preocupações de segurança através da realização de auditorias internas e externas, com o intuito de se certificarem que o sistema construído possui qualidade e pode continuar a ser utilizado pelas partes interessadas (Vale et al., 2022).

**Testabilidade**

A testabilidade permite avaliar a capacidade de um sistema demonstrar as suas falhas através de testes (Li et al., 2021).

O **número de testes automáticos** e a **percentagem de código coberto** foram identificadas como métricas quando se tem em conta a testabilidade em soluções de microsserviços (Vale et al., 2022). Para se endereçar estas métricas os desenvolvedores fazem uso de testes funcionais, unitários e de integração (Vale et al., 2022). A documentação e gestão de *API* é uma técnica, vulgarmente usada, pois permite a criação de uma página de documentação e gestão necessária para automatização de testes e para a realização de certos cenários de testes particulares (Li et al., 2021).

## 2.3 Padrões

Nesta secção pretende-se explorar com mais detalhe os padrões identificados na secção 2.2.1 durante o processo de revisão de literatura realizado. Assim, torna-se possível no capítulo de Análise e Conceção selecionar de uma forma informada os padrões de software a utilizar em detrimento de outros.

### 2.3.1 Event Collaboration

#### 2.3.1.1 Problema

Em qualquer arquitetura de software está presente a colaboração entre componentes, sejam estes objetos de pequena dimensão num único espaço de endereço ou de dimensões superiores como aplicações que comunicam sobre a infraestrutura da internet, sendo por norma o estilo de colaboração presente orientado por solicitações (Fowler, 2006). Ou seja, o componente que precisa de informações, que outro componente possui, realiza a solicitação, e caso o componente solicitado necessite de informação proveniente de outro componente mais uma solicitação é enviada (Fowler, 2006).

Mantendo a premissa de que cada componente segue o princípio da responsabilidade única, isto é, não possui informação sobre o fluxo arquitetural completo sendo responsável somente por parte do sistema, torna-se necessário explorar outros tipos de colaboração entre as partes interessadas caso se pretenda construir serviços refinados utilizando eventos (Stopford, 2018).

Surge então o padrão *event collaboration* que sugere um estilo de colaboração diferenciado face ao anunciado no primeiro parágrafo desta secção. Isto é, em vez dos componentes realizarem solicitações/pedidos quando necessitam de informações adicionais, estes lançam eventos quando existe a mudança de algo (Fowler, 2006). Existindo complementarmente outros componentes que recebem esses eventos despoletados e reagem apropriadamente (Fowler, 2006). O problema conduzido por este padrão leva ao pensamento critico sobre como deve de ser assegurada a interação entre as partes interessadas, tendo sido identificadas duas soluções preconizadas que pretendem responder às intenções despoletados por este padrão.

#### 2.3.1.2 Solução preconizada de Event Notification

A solução preconizada de *event notification* utiliza eventos para notificar outros microsserviços que uma mudança em algo ocorreu. O evento de notificação contém somente uma referência do estado que foi modificado, apresentando-se como um identificador único, permitindo aos microsserviços consumidores ponderar se a mudança de estado despoletada é relevante de ser acompanhada ou não (Bos, 2021b).

Um canal de comunicação, por exemplo um *broker* de mensagens ou de eventos, e uma estratégia de comunicação assíncrona, por exemplo *publish/subscribe* ou *send/receive*, são mecanismos que o sistema a implementar o padrão de *event notification* necessita de contemplar. A solução referente à aplicação do padrão *event notification* para um sistema que utilize um corretor de eventos e a estratégia de comunicação assíncrona *publish/subscribe* segue o seguinte fluxo de informação (Bos, 2021b):

- Os microsserviços que necessitam de acompanhar os eventos despoletados subscrevem os eventos que desejam do *broker* de eventos;
- Um microsserviço produtor ao aplicar a sua lógica de negócio manipula certos dados, produzindo e armazenando um evento de um tipo específico no corretor de eventos;
- Qualquer microsserviço que tenha subscrito o tipo de evento despoletado pelo microsserviço produtor consegue então consumir o evento de notificação despoletado.

Este evento possui somente o identificador único e a fonte que indica de onde os detalhes do evento podem ser recuperados;

- Os microsserviços após consumirem o evento de notificação despoletado, estão aptos a solicitarem os detalhes de mudança de estado efetuados pelo microsserviço produtor. A solicitação pode ser efetuada de uma maneira síncrona, com recurso a um *endpoint HTTP* do microsserviço produtor, ou de uma maneira assíncrona, com recurso a uma fila de mensagens a processar pelo microsserviço produtor.

#### 2.3.1.3 Solução preconizada de Event Carried State Transfer

A solução preconizada de *event carried state transfer* utiliza eventos que possuem estado, diferenciando-se da solução de *event notification* que utiliza eventos que contêm somente um identificador para se obter estado do microsserviço produtor. Os microsserviços consumidores já não necessitam de solicitar o microsserviço produtor para obterem estado, devido ao facto de os eventos possuírem estado. Os microsserviços consumidores constroem uma réplica privada de estado, através do armazenamento do estado dos eventos que consomem.

O fluxo de informação exposto na solução de *event notification* também se aplica ao padrão de *event carried state transfer*, diferenciando-se em certos aspetos (Bos, 2021a):

- O evento consumido pelo microsserviço consumidor, despoletado pelo microsserviço produtor, possui estado eliminando a necessidade do microsserviço consumidor solicitar ao microsserviço produtor os detalhes da mudança de estado ocorrida;
- Os microsserviços consumidores atualizam a sua réplica privada de estado consoante a informação do evento consumido, realizando o processamento do evento tendo em conta as necessidades do microsserviço consumidor em questão.

### 2.3.2 CQRS

#### 2.3.2.1 Problema

Nas arquiteturas tradicionais é utilizado o mesmo modelo de dados para consultar e atualizar informação numa base de dados, apresentando-se como uma abordagem simples, funcionado bem para operações *CRUD (Create, Read, Update and Delete)* básicas (Arambarri, Dennis, Dahan, Sherer, & Hallihan, 2023). Contudo, a aplicação desta abordagem pode tornar-se desafiadora em cenários de aplicações mais complexos. No lado de leitura deve ser possível a realização de consultas diversificadas, retornando *DTOs (Data Transfer Objects)* com formatos distintos, podendo tornar o mapeamento dos objetos em algo não trivial (Arambarri, Dennis, et al., 2023). No lado de escrita o modelo de dados pode apresentar validações complexas associadas a lógica de negócio, resultando num modelo complexo que possui demasiadas responsabilidades (Arambarri, Dennis, et al., 2023).

A gestão da segurança e das permissões de cada entidade, em arquiteturas tradicionais, pode revelar-se complexa pois cada entidade encontra-se disponível para operações de leitura e escrita.

#### 2.3.2.2 Solução preconizada

Através da aplicação do padrão *CQRS* pretende-se separar as leituras e as escritas em modelos de dados diferentes, através da utilização de comandos para atualizar dados e consultas para leitura de dados. Para tal, os comandos devem ser baseados em tarefas em vez de serem centrados nos dados, e as consultas, não modificando a base de dados, devem de devolver um *DTO* que não encapsule qualquer conhecimento de domínio (Arambarri, Dennis, et al., 2023). Salienta-se que os comandos podem ser processados de uma forma assíncrona, com recurso a filas de mensagens, em contrapartida de serem processados sincronamente (Arambarri, Dennis, et al., 2023).

Os modelos de dados de leitura e de escrita podem ser isolados, apesar de não ser condição obrigatória, podendo inclusive estar presentes em bases de dados de tipos diferentes (Arambarri, Dennis, et al., 2023). Por exemplo, a base de dados de escrita pode ser do tipo relacional, enquanto a base de dados de leitura pode ser do tipo não relacional, otimizando-se estas consoante as necessidades de carga. Com isto, surge a necessidade de sincronização dos dados nas diferentes bases de dados, possível de se alcançar através do uso de eventos sempre que algo é despoletado, com recurso ao padrão de software *event sourcing*.

A segregação proposta por este padrão permite a divisão da parte de comandos e de consultas em componentes diferenciados. Por exemplo, dois serviços diferentes podem ser criados, um possui a capacidade de endereçar as mudanças de estado, enquanto o outro endereça as consultas devolvendo informação sem a modificar (Oliveira Rocha, 2022).

A aplicação da solução preconizada por este padrão inclui certos aspetos que permitem endereçar os problemas que este se propõe a enfrentar (Arambarri, Dennis, et al., 2023):

- Escalonamento independente, por possuir as partes de comandos e de consultas bem delineadas, permitindo que as cargas de trabalho de leitura e de escrita sejam dimensionadas isoladamente;
- Modelos de dados otimizados, ou seja, cada parte do sistema está otimizado ou para leituras ou para consultas;
- Segurança, do ponto de vista que as entidades estão segregadas consoante a escrita ou a leitura de informação;
- Separação das responsabilidades criando modelos de dados mais flexíveis à mudança, apresentando a lógica de negócio complexa no modelo de escrita, simplificando o modelo de leitura.

### 2.3.3 Event Sourcing

#### 2.3.3.1 Problema

Tipicamente a leitura de dados de um repositório, seguida da modificação destes e respetiva atualização do estado atual dos mesmos, utilizando transações que bloqueiam os dados, é o caso de uso/processo típico presente nas arquiteturas tradicionais (Arambarri, Hallihan, Ganji, Leskis, & Sherer, 2023).

A abordagem descrita no primeiro parágrafo desta secção, assente nas operações *CRUD*, realiza as operações de atualização diretamente na base de dados, podendo reduzir o desempenho e capacidade de resposta devido à sobrecarga de processamento necessário para se efetivar as operações (Arambarri, Hallihan, et al., 2023). Adicionalmente, conflitos de atualização podem existir, devido ao facto de as operações de atualização serem executadas num único registo de uma determinada base de dados (Arambarri, Hallihan, et al., 2023). A abordagem não regista o histórico das operações realizadas ao longo do tempo, ficando somente disponível para consulta o estado atual de uma entidade (Arambarri, Hallihan, et al., 2023).

#### 2.3.3.2 Solução preconizada

O padrão *event sourcing* sugere uma abordagem para realizar as operações sobre os dados controladas com recurso a uma sequência de eventos, eventos estes, registados num repositório que permite somente a operação de acréscimo de dados (Arambarri, Hallihan, et al., 2023). Assim, o sistema que faz uso deste padrão envia uma série de eventos que descrevem, de uma forma inequívoca, cada ação ocorrida nos dados para o repositório de eventos (Arambarri, Hallihan, et al., 2023). O repositório de eventos, que persiste os eventos despoletados, apresenta-se como a fonte de verdade referente ao estado atual dos dados. Usualmente, o repositório de eventos publica os eventos permitindo que os consumidores destes possam ser notificados e os manipulem consoante as suas necessidades (Arambarri, Hallihan, et al., 2023). Permitindo assim, por exemplo, aos serviços consumidores efetuarem tarefas necessárias para a conclusão da operação despoletada pelo evento, pois a parte de geração de eventos é independente das partes que assinam/acompanham os eventos.

Através da aplicação deste padrão torna-se possível aos sistemas efetuarem a leitura do histórico de eventos, podendo reconstruir o estado atual de uma entidade através do consumo de todos os eventos relacionados com essa.

A aplicação da solução preconizada por este padrão inclui certos aspetos que permitem endereçar os problemas que este se propõe a enfrentar (Arambarri, Hallihan, et al., 2023):

- Eventos são imutáveis e são armazenados utilizando uma operação de acréscimo;
- Eventos são objetos simples que descrevem ações que ocorreram;
- O problema das atualizações simultâneas de dados não se coloca;
- O repositório de eventos permite a realização de auditorias, permitindo monitorizar as ações executadas no repositório;
- Existe a desassociação das operações/tarefas a executar dos eventos, favorecendo a flexibilidade e extensibilidade da solução;

### 2.3.4 Publish/Subscribe

#### 2.3.4.1 Problema

Em arquiteturas de sistemas distribuídos e de microsserviços os diferentes componentes, destes sistemas, necessitam de fornecer informações a outros componentes do sistema. A utilização de mensagens assíncronas permite separar os serviços produtores dos serviços

consumidores, combatendo-se o problema da espera ativa. Contudo, a utilização de uma fila de mensagens exclusiva por cada serviço consumidor não se torna prática quando se possui um número avultado de serviços consumidores (Dennis, Dahan, & Sherer, 2022). Por outro lado, pode existir a necessidade de certos serviços consumidores estarem interessados somente num subconjunto de informação (Dennis et al., 2022).

#### 2.3.4.2 Solução preconizada

A solução preconizada por este padrão sugere a utilização de um sistema de mensagens assíncronas que possua os seguintes artefactos (Dennis et al., 2022):

- Um canal de mensagens de entrada a utilizar pelo publicador de mensagens. O publicador constrói as mensagens, com recurso a um formato de mensagem padrão, e envia-as através do canal de entrada.
- Um canal de mensagens de saída a ser utilizado pelos consumidores, ou seja, os subscritores;
- Um mecanismo que permita copiar as mensagens do canal de entrada para os canais de saída, com o intuito de os subscritores interessados na mensagem poderem processá-la devidamente. Este mecanismo por norma é implementado com recurso a um *broker* de mensagens ou eventos, como por exemplo RabbitMQ, ActiveMQ ou Apache Kafka.

A aplicação da solução preconizada por este padrão inclui certos aspetos que permitem endereçar os problemas que este se propõe a solucionar (Dennis et al., 2022):

- Divide os sistemas que precisam de comunicar, permitindo a sua gestão de forma independente;
- Aumenta a escalabilidade, melhorando a capacidade de resposta do publicador de mensagens;
- Possibilidade de processamento diferido ou programado, ou seja, os serviços subscritores podem configurar quando pretendem efetuar o processamento das mensagens;
- Possibilidade de monitorização dos canais de comunicação, sendo possível inspecionar as mensagens que circulam nestes;
- Possibilidade de criação de tópicos de mensagens, onde cada tópico está associado a um canal de saída, sendo possível aos consumidores subscreverem os tópicos relevantes;
- Possibilidade de realização de filtragem de conteúdo, onde as mensagens são inspecionadas e distribuídas tendo em conta o seu conteúdo. Posteriormente, torna-se possível a cada subscritor configurar o conteúdo que está interessado em acompanhar.

### 2.3.5 Request/Response assíncrono

#### 2.3.5.1 Problema

Normalmente as aplicações cliente dependem de outros serviços, por exemplo *APIs*, para executarem lógica de negócio (T. Petersen, Arambarri, Sperker, & Urnun, 2022). Usualmente, essas invocações ocorrem com recurso ao protocolo *HTTP* aplicando o padrão *REST*. Existem cenários onde o processamento efetuado, pelos serviços invocados, pode ser complexo e penoso em termos temporais. Pelo que, não se torna plausível a espera pelo fim do processamento dos serviços invocados para se responder à solicitação, revelando-se um problema quando se utiliza exclusivamente mecanismos de comunicação *request/response* síncronos (T. Petersen et al., 2022).

#### 2.3.5.2 Solução preconizada

Uma solução preconizada para o problema exposto é a utilização de sondagem *HTTP*, que se torna útil quando a utilização de conexões de execução prolongada não se torna viável. A solução alcançada sugere a aplicação ordenada dos seguintes passos (T. Petersen et al., 2022):

- A aplicação cliente realiza uma chamada síncrona a um serviço, despoletando uma tarefa/ação no serviço invocado;
- O serviço invocado responde de uma forma síncrona, o mais brevemente possível, retornando o código de estado *HTTP* 202 sinalizando que o pedido foi recebido para processamento posterior;
- A resposta possui uma referência para um ponto de estado que a aplicação cliente pode sondar para verificar o estado da operação no serviço invocado;
- O serviço invocador delega o processamento da tarefa num outro componente, semelhante a uma fila de mensagens;
- Caso a operação ainda esteja em processamento, o ponto de estado devolve um recurso indicando que a tarefa ainda se encontra em processamento. Caso a operação termine o processamento, o ponto de estado devolve um recurso indicando que a tarefa foi concluída.

### 2.3.6 Send/Receive

#### 2.3.6.1 Problema

Em arquiteturas de sistemas distribuídos e de microsserviços os diferentes componentes, destes sistemas, necessitam de fornecer informações a outros componentes do sistema. A utilização de mecanismos síncronos de comunicação, por exemplo recorrendo do protocolo *HTTP*, permite separar os serviços produtores dos serviços consumidores. Contudo, as implicações da comunicação síncrona, como por exemplo o problema da espera ativa e da existência de *timeouts*, estão presentes. Surgindo o padrão *send/receive* que pretende introduzir uma alternativa ao problema da comunicação síncrona, quando não se necessita de uma resposta imediata do microsserviço a contactar.

#### 2.3.6.2 Solução preconizada

O padrão *send/receive* permite a troca de mensagens entre diferentes componentes ou sistemas. O padrão providencia um mecanismo de envio de mensagens de um produtor para um recetor de mensagens, permitindo a comunicação assíncrona entre as partes interessadas resultando num desacoplamento destas. O cenário típico de aplicação do padrão é composto pelas seguintes fases (Warren, Wenzel, Dahan, & Pine, 2022):

- O serviço produtor constrói a mensagem a ser transmitida, que contém os dados necessários;
- O serviço produtor envia a mensagem para um canal de comunicação, onde diversos serviços recetores possuem acesso de leitura às mensagens presentes;
- A mensagem é entregue a um dos serviços recetores que acompanha as mensagens do canal de comunicação;
- O serviço recetor processa a mensagem recebida, através da realização de ações de acordo com o conteúdo da mensagem.

## 2.4 Sumário

Através da questão de investigação RQ1 foi possível identificar padrões e diferentes topologias presentes em arquiteturas de microsserviços orientadas a eventos. Complementarmente, na mesma questão, foram identificados desafios e diferentes abordagens de comunicação entre partes interessadas. Na RQ2 exploraram-se tecnologias e *frameworks* que permitem a aplicação dos padrões e/ou topologias descobertos em RQ1. Adicionalmente, essas mesmas tecnologias e *frameworks* foram comparadas entre si, sendo possível de se perceber as diferenças entre as funcionalidades disponibilizadas por cada uma destas. Depois, em RQ3 foram descobertos casos práticos de aplicação de arquiteturas de microsserviços orientadas a eventos. Por fim, em RQ4 foram recolhidas métricas e abordagens para avaliação das arquiteturas de microsserviços orientadas a eventos. Conclui-se que através deste capítulo foi possível adquirir conhecimento sobre como conceber e implementar soluções de microsserviços orientadas a eventos com os padrões de software, tecnologias e *frameworks* mais adequadas para cada necessidade, salientando-se também as métricas e abordagens que permitem avaliar de uma forma objetiva e rigorosa as soluções implementadas.

# 3 Análise de Valor

O objetivo da realização de uma análise de valor é identificar as oportunidades que podem despoletar valor a um determinado produto ou ideia, salientando a redução dos custos sem comprometer a qualidade do produto ou ideia final.

Neste capítulo aplica-se o processo de inovação mediante os cinco elementos do modelo *NCD*. Através da adoção deste modelo seleciona-se a ideia mais relevante de exploração neste projeto de tese, com recurso ao método *AHP*. Depois, no último elemento do modelo as tecnologias, que permitem executar a ideia alcançada na prática, são escolhidas.

## 3.1 Processo de Inovação

O processo de inovação pode ser dividido em três partes: *Fuzzy Front End* (*FFE*), *New Product Development* (*NPD*) e *Commercialization*, como se observa na figura 15 (Belliveau, Griffin, & Somermeyer, 2002).

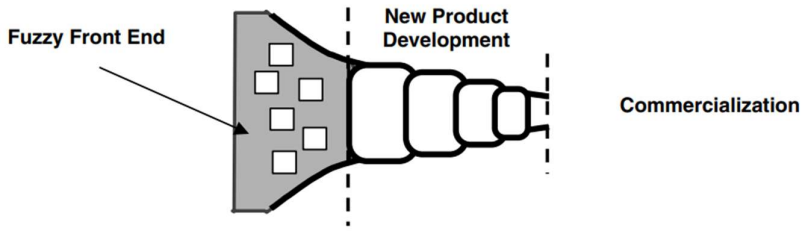



Figura 15 - Partes do processo de inovação – imagem de (Belliveau et al., 2002)

A primeira parte, *FFE*, é considerada como uma das secções do processo de inovação mais suscetíveis de melhoramento do processo geral de inovação (Belliveau et al., 2002). Contudo, existe escassez na definição dos princípios que permitem a avaliação geral do processo de

inovação, provocada pela falta de terminologia comum adotada pelas empresas, na definição dos elementos-chave aquando do processo de *FFE* (Belliveau et al., 2002). Para endereçar este problema, provocado pela falta de *standards* na fase de *FFE*, o modelo *NCD* foi criado que pretende fornecer informações e conceitos comuns a aplicar na parte de *FFE* (Belliveau et al., 2002).

A figura 16 apresenta as três partes fundamentais deste modelo (Belliveau et al., 2002):

- Motor: representa a cultura e estratégia de negócio da empresa, permitindo comandar os cinco elementos pertencentes ao interior do modelo;
- Interior do modelo: representa as cinco atividades de *FFE*: identificação da oportunidade, análise da oportunidade, geração de ideia, seleção de ideia e definição da conceção;
- Fatores de influência: representa os fatores externos à organização que influenciam o processo de inovação até à comercialização.

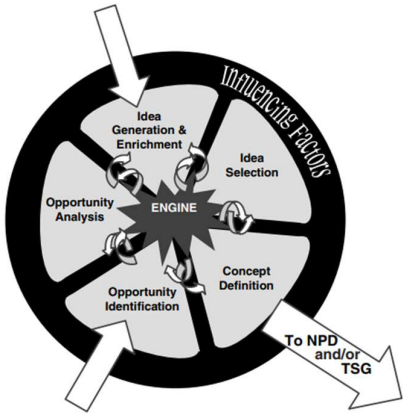



Figura 16 - Modelo NCD – imagem de (Belliveau et al., 2002)

As secções seguintes pretendem detalhar os cinco elementos interiores do modelo *NCD*, através da aplicação do modelo ao projeto de dissertação.

### 3.1.1 Identificação de oportunidade

Nesta fase pretende-se que a empresa clarifique as oportunidades relevantes num certo contexto, para tal torna-se útil a clarificação do mercado e área tecnológica que a empresa pretende alcançar (Belliveau et al., 2002).

No capítulo de Estado de Arte foi realizada uma revisão sistemática de literatura relativa ao tema desta dissertação. Após a extração de dados e respetivo mapeamento dos estudos encontrados torna-se possível a identificação de padrões. Foi identificado o padrão referente aos temas que utilizam arquiteturas de microsserviços orientadas a eventos, concluindo-se que os temas de *IOT*, de *Cloud* e de *Distributed Systems* são aqueles que apresentam maior número de ocorrências na revisão de literatura efetuada, como se observa na figura 17.

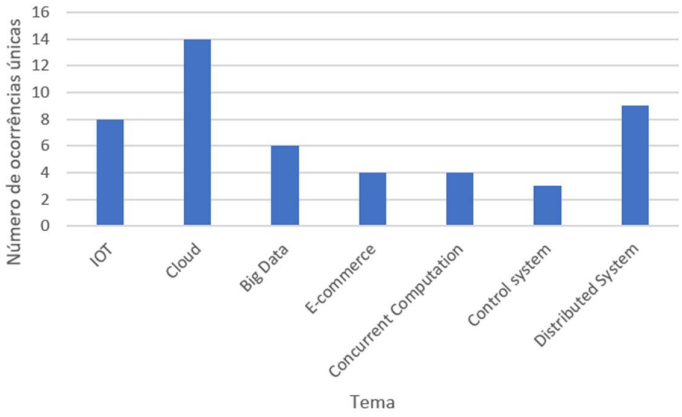



Figura 17 - Número de ocorrências únicas por tema

Justifica-se a identificação variada de temas, pois as empresas que atuam nesses setores de atividade procuram sistemas que possuam características como a tolerância a falhas, a escalabilidade, o elevado desempenho e a existência de monitorabilidade. Conclui-se que os temas de *IOT*, *Cloud* e *Distributed Systems* estão altamente conectados à temática das arquiteturas de microsserviços orientadas a eventos.

### 3.1.2 Análise da Oportunidade

Após realizada a identificação da oportunidade é necessário um processo de análise da mesma, para se confirmar se o desenvolvimento da oportunidade é vantajoso ou não (Belliveau et al., 2002).

Na secção de Análise de Dados do capítulo de Estado de Arte, presente em 2.2, foram compilados os desafios, possíveis padrões e tecnologias a implementar e também a descrição de casos práticos aplicados sobre a temática das arquiteturas de microsserviços orientadas a eventos.

A figura 18 evidencia o interesse, a nível mundial, do termo *event driven microservices* desde Novembro de 2019 até Novembro de 2022. Conclui-se que no ano de 2022 existiu um aumento da popularidade do termo, atingindo o pico de popularidade em Fevereiro de 2022 e em Novembro de 2022 foi atingido o segundo maior valor de popularidade dos últimos 3 anos.


Figura 18 - Popularidade do termo event driven microservices – imagem de (Trends, 2023)

### 3.1.3 Geração de ideias

Esta etapa pretende endereçar o nascimento, o desenvolvimento e o amadurecimento de uma ideia (Belliveau et al., 2002). Uma ideia vai sofrendo alterações ao longo das diferentes etapas do modelo *NCD* (Belliveau et al., 2002).

As ideias a construir devem de responder às seguintes questões:

- Como é que a solução a implementar pode endereçar requisitos de desempenho e de escalabilidade, e ao mesmo tempo possuir tolerância a falhas?
- Quais as alterações a efetuar às aplicações típicas da empresa, para que estas continuem competitivas nas áreas de negócio em que se inserem?
- Qual a melhor solução que permite endereçar a elevada existência de pedidos, e quais são os componentes arquiteturais principais que permitem tal cenário?

As ideias plausíveis de exploração são:

- S1: Desenvolvimento de duas soluções – uma solução através de uma topologia mediadora e outra solução através de uma *framework* orientada a atores;
- S2: Desenvolvimento de duas soluções – ambas as soluções através de uma topologia mediadora, diferindo a tecnologia adotada em cada solução;
- S3: Desenvolvimento de duas soluções – ambas as soluções através da adoção de uma *framework* orientada a atores diferentes entre elas.

### 3.1.4 Seleção da ideia

Após o processo de geração de ideias, é fundamental selecionar a ideia mais relevante. Para isso foi aplicado o método *AHP*, que consoante os critérios de avaliação consegue determinar de uma forma matemática a relevância de cada ideia em análise.

Foram definidos os seguintes critérios de decisão:

- **Tempo**: tendo em conta o tempo estipulado para a realização do projeto;
- **Interesse Empresarial**: tendo em conta a opinião da empresa referente à exploração de soluções não usuais no seu leque de projetos.
- **Relevância**: tendo em conta a pertinência de cada solução responder às perguntas formuladas na parte de geração de ideias.

A figura 19 representa a árvore hierárquica de decisão alcançada, que é composta pelos critérios de decisão e pelas alternativas a considerar:

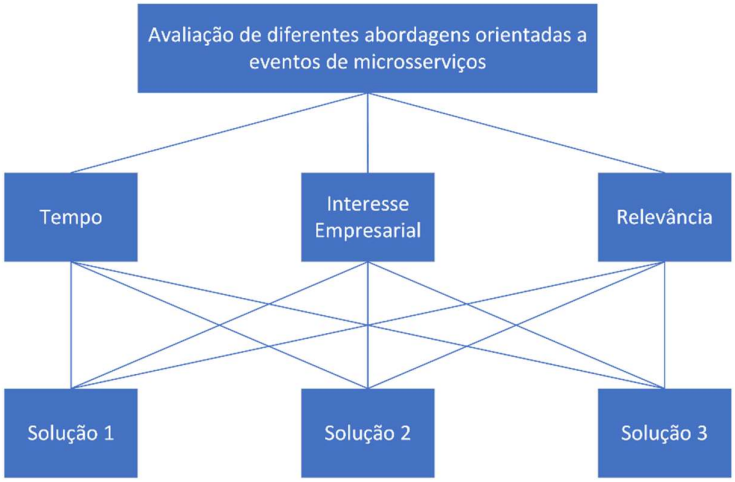



Figura 19 - Árvore hierárquica de decisão

Com o intuito de se atribuir níveis de importância nas comparações efetuadas ao longo da aplicação do método *AHP* a escala de Saaty foi usada, sendo os valores de nível de importância expressos na tabela 13.

Tabela 13 - Escala Fundamental - Níveis de Importância de comparações (Saaty, 1980)

| **Nível de Importância** | **Definição** | **Explicação** |
|---|---|---|
| 1 | Igual importância | As duas atividades contribuem igualmente para o objetivo. |
| 3 | Fraca importância | A experiência e o julgamento favorecem levemente uma atividade face á outra. |
| 5 | Forte importância | A experiência e o julgamento favorecem fortemente uma atividade em relação á outra. |
| 7 | Muito forte importância | Uma atividade é muito fortemente favorecida em relação à outra. |
| 9 | Importância absoluta | A evidência favorece uma atividade em relação à outra com o mais elevado grau de certeza. |
| 2,4,6,8 | Valores intermediários | Quando se necessita de um compromisso entre as duas atividades. |

A tabela 14 apresenta a matriz de comparação consoante os critérios definidos no passo anterior.

Tabela 14 - Matriz de comparação dos critérios

| | **Tempo** | **Interesse Empresarial** | **Relevância** |
|---|---|---|---|
| Tempo | 1 | 1/3 | 1/5 |
| Interesse Empresarial | 3 | 1 | 1/3 |
| Relevância | 5 | 3 | 1 |

De seguida, torna-se necessário normalizar a matriz de comparação de critérios alcançada, para tal divide-se o valor de cada entrada pelo valor total da soma da sua coluna. Depois é calculada a média aritmética dos valores de cada linha da matriz normalizada, dando origem ao vetor de prioridades. A tabela 15 apresenta os valores alcançados.

Tabela 15 - Matriz normalizada dos critérios

| | **Tempo** | **Interesse Empresarial** | **Relevância** | **Prioridade relativa** |
|---|---|---|---|---|
| Tempo | 0,1111 | 0,0769 | 0,1305 | 0,1062 |
| Interesse Empresarial | 0,3333 | 0,2308 | 0,2179 | 0,2607 |
| Relevância | 0,5555 | 0,6928 | 0,6536 | 0,6340 |

Depois entra a fase de avaliação da consistência das prioridades relativas, para se averiguar se os julgamentos efetuados são confiáveis. Para se conseguir calcular o valor de razão de consistência (RC) é necessário primeiro efetuar o cálculo de λmax. λmax é calculado através da multiplicação da matriz de comparação de critérios com o vetor de prioridades (1), sendo depois efetuada a média dos valores obtidos sobre o vetor de prioridades (2).

$$\begin{bmatrix} 1 & \frac{1}{3} & \frac{1}{5} \\ 3 & 1 & \frac{1}{3} \\ 5 & 3 & 1 \end{bmatrix} \text{X} \begin{bmatrix} 0.1062 \\ 0.2607 \\ 0.6340 \end{bmatrix} = \begin{bmatrix} 0.3199 \\ 0.7906 \\ 1.9471 \end{bmatrix} \text{(1)}$$

$$\lambda\text{max} = \text{Média} \left(\frac{0.3199}{0.1062}, \frac{0.7906}{0.2607}, \frac{1.9471}{0.6340}\right) = 3{,}0387 \text{ (2)}$$

Agora torna-se possivel calcular o índice de consistência da seguinte forma: $IC = \frac{\lambda \max - n}{n-1}$ , com n = 3, tendo sido obtido o valor de 0,01933.

Depois calcula-se o valor de RC, com recurso à seguinte expressão: $RC = \frac{\text{IC}}{IR}$, salienta-se o valor de IR = 0,58 devido à dimensão da matriz ser 3. Obteve-se o valor de 0,0332 para RC, pelo que se conclui que o valor das prioridades relativas é consistente por ser inferior a 0,1.

Na próxima fase são criadas as matrizes de comparação para cada critério, presentes nas tabelas 16 até 21, para se conseguir determinar a importância de cada alternativa. Após isso as matrizes são normalizadas, para se conseguir realizar o cálculo da prioridade relativa de cada solução por critério de avaliação.

Tabela 16 - Matriz de comparação para o critério tempo

| | **Solução 1** | **Solução 2** | **Solução 3** |
|---|---|---|---|
| Solução 1 | 1 | 3 | 3 |
| Solução 2 | 1/3 | 1 | 1/2 |
| Solução 3 | 1/3 | 2 | 1 |

Tabela 17 - Matriz normalizada para o critério tempo

| | **Solução 1** | **Solução 2** | **Solução 3** | **Prioridade relativa** |
|---|---|---|---|---|
| Solução 1 | 0,6024 | 0,500 | 0,667 | 0,5898 |
| Solução 2 | 0,2008 | 0,166 | 0,111 | 0,1593 |
| Solução 3 | 0,2008 | 0,333 | 0,222 | 0,2519 |

Tabela 18 - Matriz de comparação para o critério interesse empresarial

| | **Solução 1** | **Solução 2** | **Solução 3** |
|---|---|---|---|
| Solução 1 | 1 | 5 | 3 |
| Solução 2 | 1/5 | 1 | 1/3 |
| Solução 3 | 1/3 | 3 | 1 |

Tabela 19 - Matriz normalizada para o critério interesse empresarial

| | **Solução 1** | **Solução 2** | **Solução 3** | **Prioridade relativa** |
|---|---|---|---|---|
| Solução 1 | 0,6521 | 0,5555 | 0,6928 | 0,6335 |
| Solução 2 | 0,1307 | 0,1111 | 0,0770 | 0,1063 |
| Solução 3 | 0,2174 | 0,3333 | 0,2308 | 0,2605 |

Tabela 20 - Matriz de comparação para o critério relevância

| | **Solução 1** | **Solução 2** | **Solução 3** |
|---|---|---|---|
| Solução 1 | 1 | 3 | 3 |
| Solução 2 | 1/3 | 1 | 1/3 |
| Solução 3 | 1/3 | 3 | 1 |

Tabela 21 - Matriz normalizada para o critério relevância

| | **Solução 1** | **Solução 2** | **Solução 3** | **Prioridade relativa** |
|---|---|---|---|---|
| Solução 1 | 0,6002 | 0,4286 | 0,6924 | 0,5737 |
| Solução 2 | 0,2000 | 0,1429 | 0,0770 | 0,1400 |
| Solução 3 | 0,2000 | 0,4286 | 0,2308 | 0,2865 |

Com recurso às tabelas presentes nesta secção é possível o cálculo da prioridade real de cada alternativa, através da multiplicação da matriz onde cada coluna possui o vetor de prioridade de cada alternativa por critério, pelo vetor de prioridades relativas dos critérios:

$$\begin{bmatrix} 0.5898 & 0.6335 & 0.5737 \\ 0.1593 & 0.1063 & 0.1400 \\ 0.2519 & 0.2605 & 0.2865 \end{bmatrix} \text{X} \begin{bmatrix} 0.1062 \\ 0.2607 \\ 0.6340 \end{bmatrix} = \begin{bmatrix} 0.5915 \\ 0.1334 \\ 0.2763 \end{bmatrix}$$

Conclui-se que S1 é a solução prioritária, por apresentar um resultado geral superior (0,59).

### 3.1.5 Definição da conceção

No processo de seleção de ideia, concluiu-se que a alternativa mais vantajosa para este projeto é a de desenvolvimento de duas soluções, uma através da aplicação da topologia mediadora e outra através da aplicação de uma *framework* orientada a atores. Pelo que para se formular respostas às questões formuladas na fase de geração de ideias é necessário definir as tecnologias a usar, que irão conseguir evidenciar na prática o desenvolvimento das duas soluções, alcançadas no processo de seleção de ideias.

A tabela 11 presente na secção 2.2.2.4 – Análise de ferramentas de topologia *mediator,* do capítulo de Estado de Arte, sintetiza as diferenças entre tecnologias plausíveis de serem utilizadas para se implementar uma topologia do tipo mediadora. Com base na categoria popularidade, conclui-se que a tecnologia Apache Camel supera os valores da tecnologia Mule ESB. Relativamente à categoria de funcionalidades de produção, o Apache Camel possui uma documentação extensa face ao Mule ESB que possui escassez de documentação. Adicionalmente Apache Camel possui suporte exclusivamente através da comunidade

contrariamente ao Mule ESB onde a MuleSoft oferece suporte dedicado. Outro aspeto que se salienta é o facto de Apache Camel disponibilizar *JMS* através de componentes enquanto Mule ESB disponibiliza *APIs* para esta questão. Conclui-se que a tecnologia a adotar para implementação de topologia mediadora é o Apache Camel, devido às vantagens expressas nos tópicos anteriores, mas sobretudo pela diversidade de componentes que podem ser usados como mecanismos de comunicação assíncrona, o que é um ponto fundamental no tema desta dissertação.

A tabela 12 presente na secção 2.2.2.6 – Análise de *frameworks* de programação orientada a atores, do capítulo de Estado de Arte, sintetiza as diferenças entre *frameworks* plausíveis de serem utilizadas para se aplicar o modelo conceptual ator. Repara-se que as funcionalidades disponibilizadas pelas duas *frameworks* são idênticas, exceto que no Akka.NET os atores são criados exclusivamente pelos seus pais enquanto no Orleans os atores são criados de uma forma automática dependendo das necessidades. Outra diferença é que no Akka.NET as referências para atores possuem informação de localização, no Orleans tal não acontece. Conclui-se que a *framework* a adotar na implementação da solução baseada no modelo ator é o Akka.NET apesar de não ser tão popular face ao Orleans, mas permite um controlo mais restrito referente à utilização de atores, o que se pode revelar enriquecedor no momento de implementação desta solução.

## 3.2 Sumário

A análise de valor conduzida neste capítulo permitiu a seleção da ideia mais relevante, com recurso ao método *AHP*, de se explorar no processo de conceção e de implementação deste projeto de tese. A ideia resultante foi a de conceção e implementação de duas soluções, a primeira através da aplicação da topologia mediadora e a segunda com recurso ao modelo ator. Posto isto, as tecnologias de implementação foram escolhidas com base nas tabelas comparativas de tecnologias, construídas no capítulo de Estado de Arte. O protótipo segundo a topologia mediadora é construído com recurso a Apache Camel e o protótipo baseado no modelo ator é construído através da *framework* Akka.NET. Conclui-se que as duas escolhas alcançadas passaram por um processo de análise rigoroso, sustentando a exigência que o projeto de tese em questão necessita.

# 4 Análise e Conceção

Neste capítulo pretende-se evidenciar os processos de análise de requisitos e de conceção de arquiteturas de microsserviços orientadas a eventos. Primeiramente os códigos de ética e legais a cumprir são mencionados, justificando-se a necessidade de conceção e implementação de um protótipo inicial de raiz representativo das aplicações típicas da empresa. Depois é efetuado o processo de recolha de requisitos, referente ao negócio de estacionamento de veículos. Seguidamente o modelo de domínio da solução é exposto e a adoção de certos padrões de software em detrimento de outros é justificada. Posteriormente a arquitetura do protótipo inicial é exposta e as alterações a efetuar a este são documentadas, com recurso a diagramas arquiteturais de diferentes níveis de granularidade, alcançando-se assim a solução idealizada no processo de análise de valor. Isto é, a construção de um protótipo segundo uma topologia mediadora e a construção de outro segundo a aplicação de uma *framework* orientada a atores.

## 4.1 Código ético e legal

O código de ética e de conduta profissional deve de ser seguido pelos profissionais de informática. O código da *ACM* foi desenhado para orientar a conduta ética dos profissionais de informática, aqui incluem-se também estudantes e outras pessoas que utilizem a tecnologia de computação (ACM, 2018). Qualquer profissional de informática deve de respeitar a privacidade, visto que a tecnologia permite a recolha, a monitorização e a troca de informação pessoal por vezes sem a autorização das partes envolvidas (ACM, 2018). O princípio ético de honrar a confidencialidade deve de ser considerado na gestão de dados confidenciais, de estratégias de negócio, de informação financeira, entre outras (ACM, 2018).

Com o intuito de se salvaguardar as questões éticas e legais, foi tomada a decisão de conceção e implementação de um protótipo inicial, de raiz, que não é nenhuma aplicação existente na empresa, mas representa várias. Assim, apesar de existirem outras possibilidades, optou-se por

ter o protótipo inicial a simbolizar as aplicações típicas da empresa e apenas dados agregados de várias aplicações foram disponibilizados para este trabalho.

## 4.2 Engenharia de Requisitos

O negócio de estacionamento de veículos pretende efetuar o controlo de lugares de estacionamento destinados a cargas e descargas. Cada lugar de estacionamento possui um sensor, que tem a capacidade de envio de informação referente ao estacionamento. Isto é, se o lugar de estacionamento se encontra disponível ou não, bem como o instante em que esta informação foi obtida. Identificou-se que são cerca de 500 sensores que enviam mensagens de estado a qualquer instante, pelo que o sistema deve estar disponível para a receção destas mensagens e efetuar o devido processamento das mesmas. O sistema deve ser capaz de disponibilizar a informação referente aos sensores, para consulta por parte do administrador, bem como registar a ocorrência de violações de regras de negócio.

O padrão *ISO/IEC* (*International Organization of Standardization/Institute of Electrotechnical Commission*) 25010 expõe o modelo de qualidade de produto de software, composto por diferentes características de qualidade. As necessidades das partes interessadas estão representadas no modelo de qualidade, que por sua vez categoriza a qualidade do produto em características e subcaracterísticas. O padrão *ISO/IEC* 25010 foi adotado no processo de engenharia de requisitos, para se alcançar um produto de software que apresente qualidade.

A tabela 22 expõe os requisitos funcionais, tendo sido identificado exclusivamente o requisito de gestão de sensores de estacionamento. A tabela 23 expõe os requisitos não funcionais que irão permitir a construção de protótipos que considerem o desempenho, a escalabilidade, a disponibilidade, a monitorabilidade e a testabilidade como características de qualidade.

Tabela 22 - Requisitos funcionais

| **Requisito Funcional** | **Descrição** |
|---|---|
| Gestão de sensores de estacionamento | Gerir a informação sobre os sensores de estacionamento. |

Tabela 23 - Requisitos não funcionais

| **Requisito não funcional** | **Descrição** |
|---|---|
| Desempenho | O registo da informação de um sensor deve ser efetuado em menos de 100ms. |
| Desempenho | A listagem de informação dos sensores deve ser disponibilizada em menos de 100ms. |

| | |
|---|---|
| Escalabilidade | As comunicações existentes entre serviços devem ser asseguradas por mecanismos assíncronos. |
| Escalabilidade | O sistema deve ser capaz de dar resposta a pelo menos 500 solicitações simultâneas apresentando um tempo médio de resposta por pedido inferior a 2 segundos. |
| Disponibilidade | Mecanismos de tolerância a falhas devem de ser adotados para se maximizar a disponibilidade do sistema. |
| Disponibilidade | O sistema deve estar disponível 100% do tempo aquando do momento da realização de testes de carga. |
| Monitorabilidade | Registos de valores de métricas de desempenho, de disponibilidade e de escalabilidade devem de ser gerados e armazenados. |
| Monitorabilidade | Informação referente a violações de regras de negócio devem de ser registadas para verificação pelo administrador do sistema. |
| Testabilidade | Testes unitários, de integração, de sistema e de aceitação devem de ser realizados para suportar a qualidade da solução construída. |

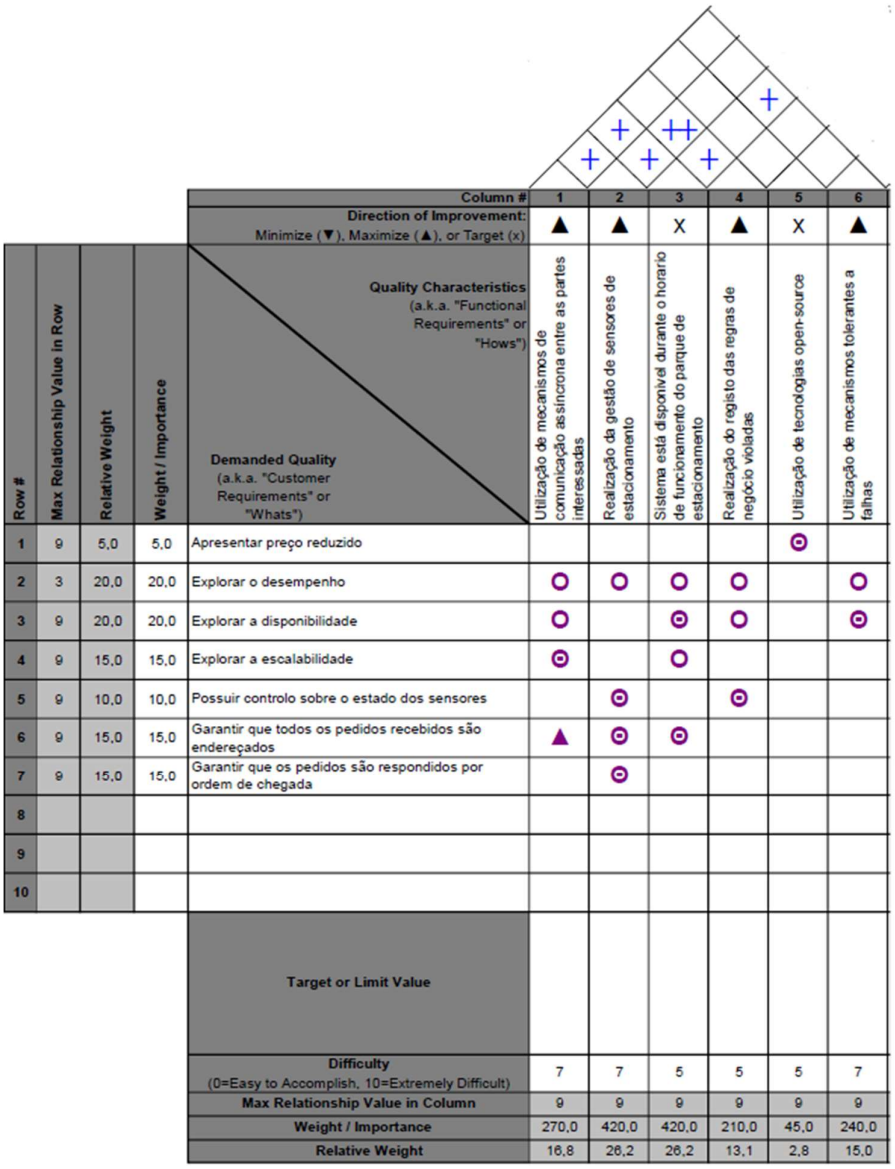
## 4.3 Análise funcional

*Quality Function Deployment* (*QFD*) é um processo e um conjunto de ferramentas usado para se desenhar novos produtos e serviços, e também permite a modificação destes com o intuito de endereçarem os requisitos especificados pelo cliente (Wolniak, 2017). Este método foi criado por um conjunto de engenheiros japoneses, com o objetivo de garantirem a implementação das necessidades reais do cliente aquando do desenvolvimento de produtos e serviços novos ou existentes (Wolniak, 2017).

No capítulo Anexo 1 o cliente deste projeto de mestrado foi identificado, sendo este a empresa Devscope que se encontra com problemas no balanceamento de requisitos de desempenho, disponibilidade e escalabilidade nos projetos que desenvolvem para os seus clientes.

Na figura 20 encontra-se a aplicação prática do modelo *QFD*. Através de uma reunião entre o supervisor deste projeto e o autor deste documento foi possível a discussão das necessidades reais do cliente, reforçando-se a conversão das expectativas do cliente em características de qualidade. Complementarmente, a importância de cada requisito, presente na coluna peso

relativo da figura 20, foi analisada e definida na mesma reunião, resultado numa hierarquização de relevância de requisitos para o cliente.



| Row # | Max Relationship Value in Row | Relative Weight | Weight / Importance | Demanded Quality (a.k.a. "Customer Requirements" or "Whats") | 1 Utilização de mecanismos de comunicação assíncrona entre as partes interessadas | 2 Realização da gestão de sensores de estacionamento | 3 Sistema está disponível durante o horário de funcionamento do parque de estacionamento | 4 Realização do registo das regras de negócio violadas | 5 Utilização de tecnologias open-source | 6 Utilização de mecanismos tolerantes a falhas |
|---|---|---|---|---|---|---|---|---|---|---|
| | | | | Column # | 1 | 2 | 3 | 4 | 5 | 6 |
| | | | | Direction of Improvement: Minimize (▼), Maximize (▲), or Target (x) | ▲ | ▲ | X | ▲ | X | ▲ |
| 1 | 9 | 5,0 | 5,0 | Apresentar preço reduzido | | | | | ⊙ | |
| 2 | 3 | 20,0 | 20,0 | Explorar o desempenho | O | O | O | O | | O |
| 3 | 9 | 20,0 | 20,0 | Explorar a disponibilidade | O | | ⊙ | O | | ⊙ |
| 4 | 9 | 15,0 | 15,0 | Explorar a escalabilidade | ⊙ | | O | | | |
| 5 | 9 | 10,0 | 10,0 | Possuir controlo sobre o estado dos sensores | | ⊙ | | ⊙ | | |
| 6 | 9 | 15,0 | 15,0 | Garantir que todos os pedidos recebidos são endereçados | ▲ | ⊙ | ⊙ | | | |
| 7 | 9 | 15,0 | 15,0 | Garantir que os pedidos são respondidos por ordem de chegada | | ⊙ | | | | |
| 8 | | | | | | | | | | |
| 9 | | | | | | | | | | |
| 10 | | | | | | | | | | |
| | | | | Target or Limit Value | | | | | | |
| | | | | Difficulty (0=Easy to Accomplish, 10=Extremely Difficult) | 7 | 7 | 5 | 5 | 5 | 7 |
| | | | | Max Relationship Value in Column | 9 | 9 | 9 | 9 | 9 | 9 |
| | | | | Weight / Importance | 270,0 | 420,0 | 420,0 | 210,0 | 45,0 | 240,0 |
| | | | | Relative Weight | 16,8 | 26,2 | 26,2 | 13,1 | 2,8 | 15,0 |

Figura 20 - Análise QFD

Nas linhas, da figura 20, foram recolhidos os requisitos do cliente destacando-se a exploração do desempenho e da disponibilidade, nas arquiteturas de microsserviços orientadas a eventos, que possuem um grau de importância superior face aos outros requisitos. Do ponto de vista do cliente, este pretende explorar o desempenho e a disponibilidade com o intuito de conseguir reduzir a latência, reduzir os tempos de resposta, alcançar a inexistência de *timeouts* e conseguir dar resposta a cenários de existência de um número de pedidos avultado nas aplicações que desenvolve. Sobre a exploração da escalabilidade, o cliente pretende perceber

como é que consegue mitigar o número de dependências síncronas, minimizando o desperdício de recursos informáticos quando pretender escalar os sistemas que desenvolve.

Nas colunas, da figura 20, foram expressas as características de qualidade que o produto a entregar deve contemplar. Salienta-se que o registo das informações disponibilizadas pelos sensores, a disponibilidade do sistema que permita endereçar todos os pedidos recebidos, a utilização de mecanismos assíncronos e a existência de mecanismos de tolerância a falhas são as características de qualidade mais relevantes.

## 4.4 Modelo de Domínio

O modelo de domínio é delineado, presente na figura 21, após o término dos processos de engenharia de requisitos e de *QFD*. Foi identificado o conceito principal lugar de estacionamento que possui os seguintes atributos:

- Identificador: permite identificar inequivocamente cada lugar de estacionamento;
- Endereço: contém informação sobre a localização do lugar de estacionamento;
- Estado: contém informação sobre se o lugar de estacionamento se encontra ocupado ou não;
- Data de recolha de informação: contém informação de dia/mês/ano sobre a última vez que o lugar de estacionamento foi monitorizado;
- Tempo de recolha de informação: contém informação de hora/minuto/segundo sobre a última vez que o lugar de estacionamento foi monitorizado;

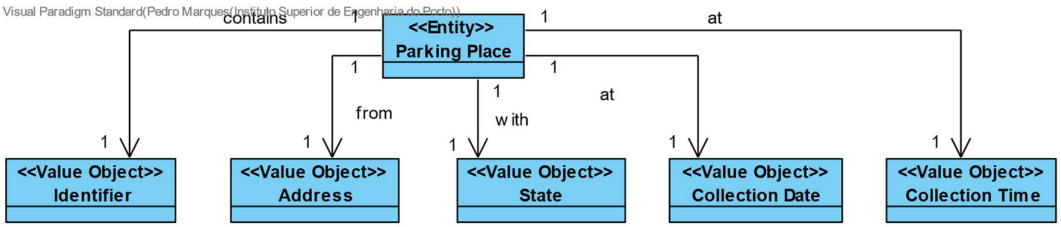



Figura 21 - Modelo de Domínio

## 4.5 Seleção de padrões de software

Finalizado o processo de recolha de requisitos e de construção do modelo de domínio da solução torna-se necessária a escolha dos padrões de software a adotar na conceção dos protótipos.

O estilo de colaboração a adotar, entre os microsserviços dos diferentes protótipos, é baseado no envio e receção de eventos para se conseguir endereçar a temática a explorar neste trabalho de mestrado. Os eventos das aplicações típicas da empresa possuem estado aplicando o padrão

*event carried state transfer*, introduzido na secção 2.3.1, de colaboração baseado em eventos. Na conceção dos protótipos baseados na topologia mediadora e no modelo ator foi tomada a decisão de aplicação do mesmo padrão de colaboração existente nas aplicações típicas da empresa. Uma alternativa seria a adoção do padrão de colaboração *event notification*, introduzido na secção 2.3.1, onde impera um elevado fluxo de comunicações, favorecendo a consistência face à disponibilidade. Optou-se pela adoção de *event carried state transfer* em detrimento de *event notification* para se garantir uma maior disponibilidade, resultando numa diminuição da latência do sistema. Obrigando-se os microsserviços consumidores a construírem uma réplica privada de estado que vai sendo atualizada consoante a informação de estado dos eventos recebidos.

Apesar do problema de domínio de estacionamento de veículos ser relativamente simples e as regras de negócio inerentes também o serem, foi tomada a decisão de aplicação dos padrões *CQRS*, introduzido na secção 2.3.2, e *event sourcing*, introduzido na secção 2.3.3, nos protótipos baseados na topologia mediadora e no modelo ator. As aplicações típicas da empresa são segregadas em partes de escrita e de leitura, mas não fazem uso do padrão de *event sourcing*. Optou-se pela adoção de *event sourcing* para se conseguir registar as alterações de estado dos eventos que ocorrem, construindo um histórico das alterações efetuadas ao longo do tempo, permitindo restaurar e reverter o estado de um sistema. Conjugou-se *event sourcing* com *CQRS* para se desacoplar o processo de geração de eventos do processo de consumo de eventos, existindo consequentemente a segregação das operações de escrita das operações de leitura suportada pela utilização de modelos de dados de escrita e de leitura independentes.

Do ponto de vista da comunicação as aplicações típicas da empresa utilizam o padrão de *publish/subscribe*, introduzido na secção 2.3.4, onde existe a criação de tópicos de mensagens, sendo que cada tópico está associado a um canal de saída permitindo aos consumidores subscreverem os tópicos relevantes. Na conceção do protótipo baseado na topologia mediadora optou-se pela utilização do padrão *send/receive*, introduzido na secção 2.3.6, em detrimento do padrão *publish/subscribe*. Ambos os padrões providenciam um mecanismo de envio de mensagens de um produtor para um recetor, utilizando comunicação assíncrona, contudo o padrão *send/receive* permite que a mensagem seja entregue somente a um dos serviços recetores que acompanha as mensagens no canal de comunicação, existindo concorrência pela obtenção da mensagem. No protótipo baseado no modelo ator optou-se pela exploração das funcionalidades assíncronas de comunicação que o modelo ator disponibiliza.

## 4.6 Modelo C4 e Vistas 4+1

O modelo C4 é composto por um conjunto de diagramas hierárquicos que possui o propósito de descrever a arquitetura de software em diferentes níveis de detalhe (Brown & Betts, 2018). Os diagramas encontram-se presentes em quatro diferentes níveis: contexto, *container*, componente e código. A vista arquitetural 4+1 permite organizar a composição de uma arquitetura de software através do uso de vistas diferentes: lógica, processo, implementação,

física e de casos de uso, sendo que cada vista possui preocupações distintas (Jayawardene, 2021).

Com o intuito de se descrever as arquiteturas de software idealizadas, para se resolver o problema deste projeto de dissertação, e sobretudo para se identificar as diferenças entre os diferentes protótipos as ferramentas descritas anteriormente foram adotadas.

### 4.6.1 Vista de casos de uso

Na figura 22, observa-se os casos de uso identificados após o processo de recolha de requisitos, efetuado na secção 4.2.

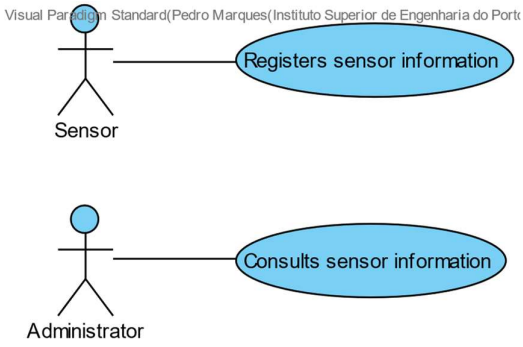



Figura 22 - Diagrama de casos de uso

### 4.6.2 Nível de Contexto

Primeiramente foi identificado o contexto do sistema, comum a todos os protótipos, tendo sido identificados dois atores: sensor e administrador na vista lógica presente na figura 23.

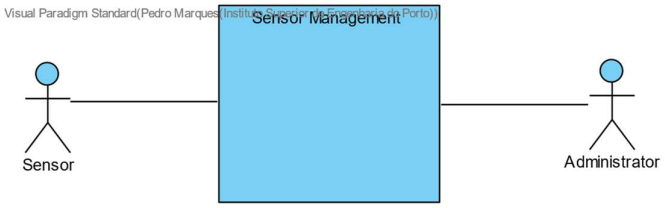



Figura 23 - Vista lógica no nível de contexto

As aplicações típicas da empresa, para o domínio explorado, utilizam RabbitMQ como sistema de fila de mensagens, MongoDB como sistema não relacional de persistência de dados e disponibilizam *end points* para obtenção de dados provenientes de sensores, como se observa na figura 24.

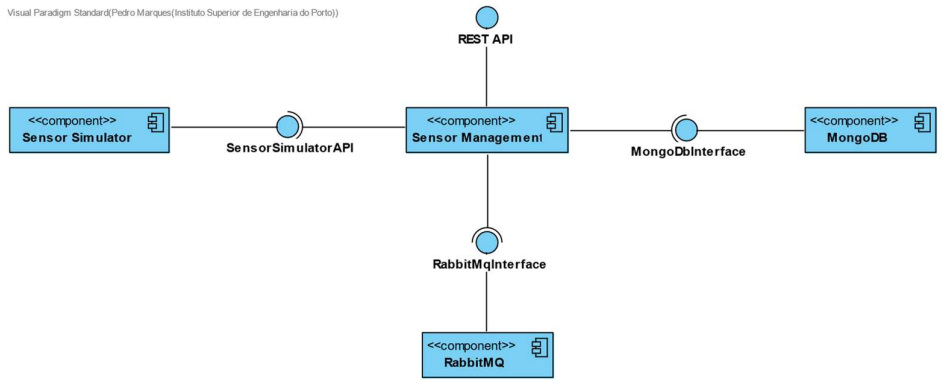



Figura 24 - Vista lógica de interação com outros sistemas no nível de contexto (Protótipo inicial)

Com o objetivo de se manter o estado dos eventos ao longo do tempo, e não só o último estado referente a algo, a *framework* Axon foi adotada que permite a aplicação facilitada do padrão de software *event sourcing*. Assim, o servidor Axon, de elevado desempenho, tem o propósito de armazenamento de eventos, existindo paralelamente o componente MongoDB para consultas de dados e registo dos estados mais recentes dos conceitos de negócio. Adicionalmente o sistema de fila de mensagens foi alterado para o ActiveMQ, que permite a implementação de filas duráveis por defeito (ActiveMQ, 2019a) e sobretudo porque possui integração com Apache Camel. Esta integração é assegurada através do uso do componente ActiveMQ disponibilizado pelo Apache Camel. A figura 25 é a nova representação gráfica de vista lógica de interação no nível de contexto para o protótipo mediador.

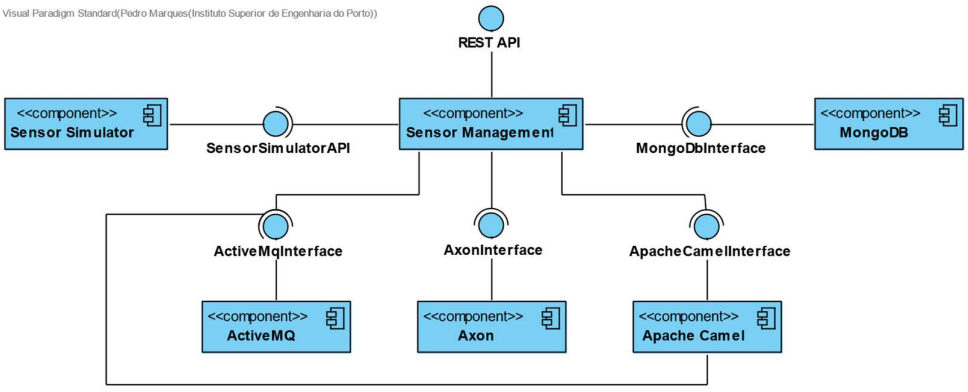



Figura 25 - Vista lógica de interação com outros sistemas no nível de contexto (Protótipo mediador)

Outra abordagem arquitetural surge da possibilidade de implementação do modelo conceptual ator, onde as funcionalidades provenientes do Akka podem ser utilizadas. A figura 26 evidencia a vista lógica de interação no nível de contexto para o protótipo baseado no modelo conceptual ator. Realça-se que o componente de MongoDB possui bases de dados diferentes para leituras e escritas.

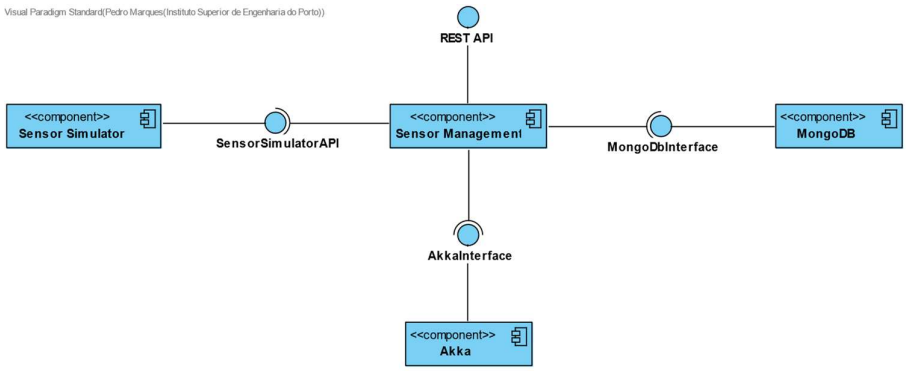



Figura 26 - Vista lógica de interação com outros sistemas no nível de contexto (Protótipo atores)

### 4.6.3 Nível de Container

Após a concretização do nível de contexto é possível explorar os protótipos através do nível de *container*. As aplicações típicas da empresa, para o domínio em questão, possuem dois microsserviços, serviço produtor e consumidor, sendo a comunicação assegurada entre eles através do *broker* de mensagens, pelo que não existe qualquer tipo de dependência direta entre microsserviços. A figura 27 salienta os aspetos referidos.

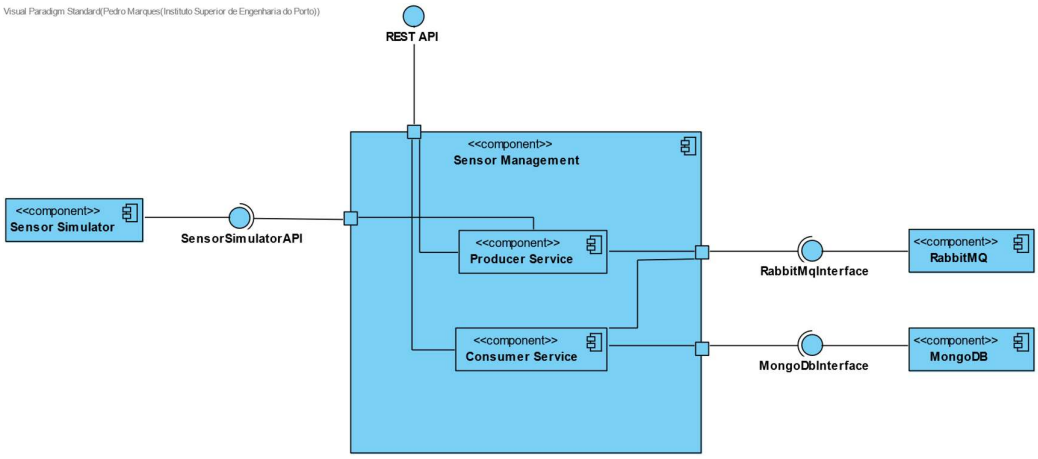



Figura 27 - Vista lógica no nível de container (Protótipo inicial)

A figura 28 evidencia as alterações descritas no nível de *container* referentes ao protótipo mediador. Onde o serviço produtor recorre ao servidor Axon para registo dos eventos recebidos. Complementarmente um novo microsserviço foi adicionado, serviço mediador, que permite ser a ponte entre os dois microsserviços iniciais, sendo possível neste novo serviço a definição de certas regras de negócio.

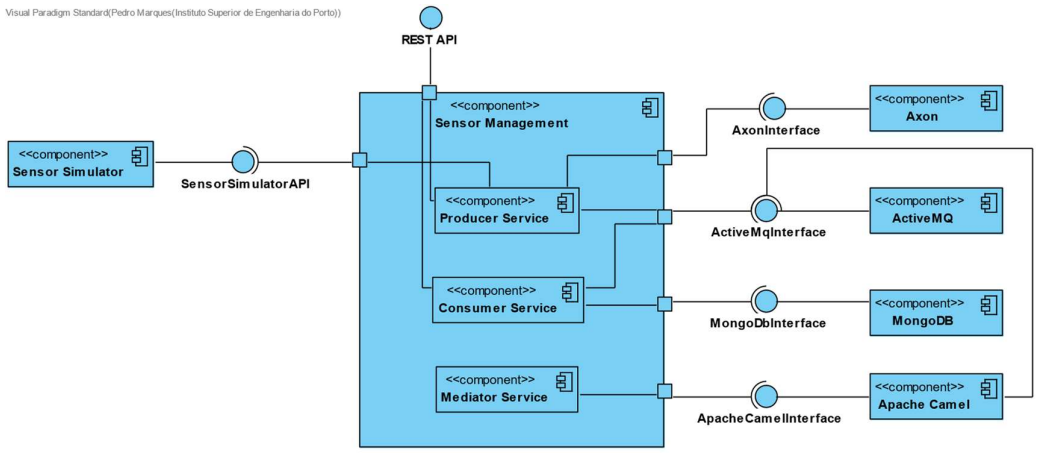



Figura 28 - Vista lógica no nível de container (Protótipo mediador)

Relativamente ao protótipo baseado no modelo ator o componente de gestão de sensores possui dois microsserviços, um serviço de registo e outro serviço de consulta, observável na figura 29. Salienta-se a capacidade do serviço de registo de utilização das capacidades orientadas a atores que a *framework* Akka.NET disponibiliza.

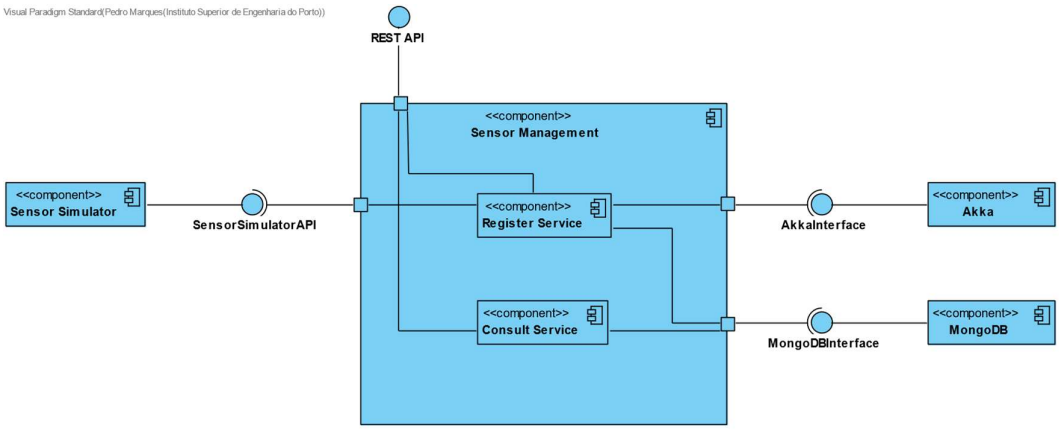



Figura 29 - Vista lógica no nível de container (Protótipo atores)

### 4.6.4 Nível de Componente

Neste nível pretende-se detalhar os componentes internos referentes aos componentes descobertos nos níveis anteriores.

Como os microsserviços produtor e consumidor possuem uma arquitetura diferenciada, optou-se pela representação gráfica de ambos os serviços. A figura 30 apresenta a arquitetura do microsserviço produtor, que se encontra dividida em quatro camadas principais:

- Camada de controlador: responsável pela gestão dos pedidos *post* recebidos pelo simulador de sensor, através do *end point* disponibilizado. Suporta a utilização de *DTO*;

- Camada de serviço: responsável pela preparação do conteúdo das mensagens, para posteriormente serem enviadas pela camada de mensagem;
- Camada de *mapper*: responsável por efetuar conversões entre objetos a serem usados por outras camadas aplicacionais;
- Camada de mensagem: responsável pelo envio das mensagens para o *broker* de mensagens.

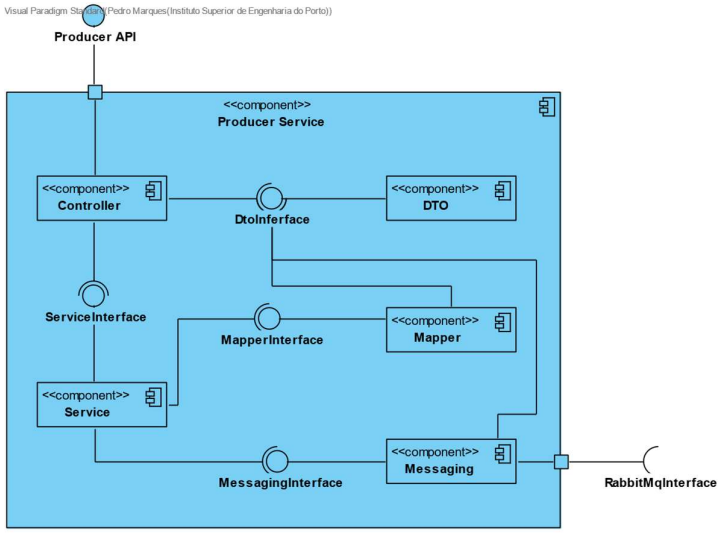



Figura 30 - Vista lógica do componente serviço produtor (protótipo inicial)

A figura 31 apresenta a arquitetura do microsserviço consumidor, que se encontra dividida em quatro camadas principais:

- Camada de controlador: responsável pela gestão dos pedidos *get* recebidos, através do *end point* disponibilizado;
- Camada de serviço: responsável pela validação dos objetos de domínio, utilizando a camada de repositório para obtenção e criação de dados de negócio;
- Camada de mensagem: responsável pela gestão das mensagens recebidas pelo *broker* de mensagens, e pelo envio destas para a camada de serviço;
- Camada de repositório: responsável pela gestão dos dados de negócio.

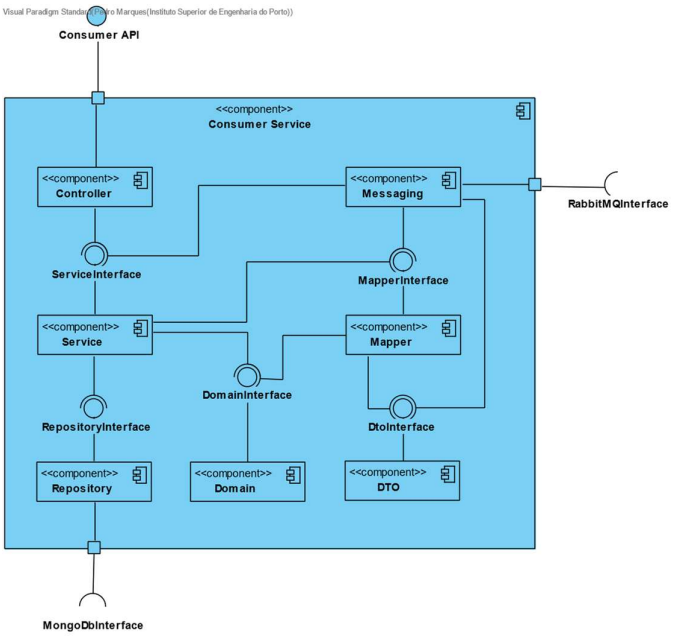



Figura 31 - Vista lógica do componente serviço consumidor (protótipo inicial)

Na figura 32 é percetível a vista de implementação referente ao serviço consumidor, assim torna-se mais explicita a relação entre os diferentes componentes arquiteturais.

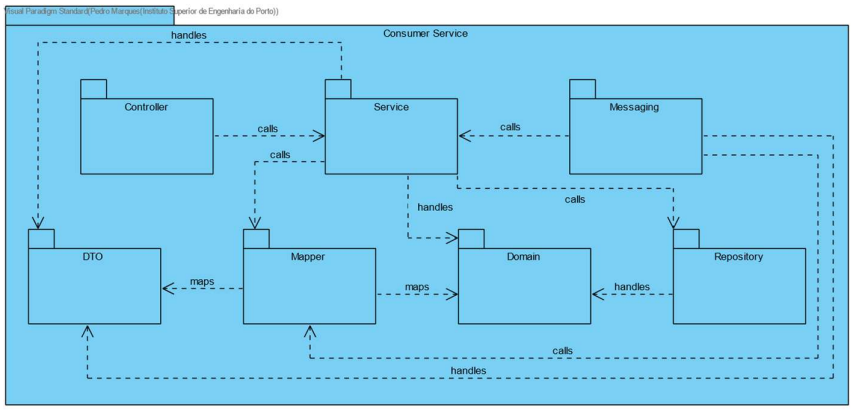



Figura 32 - Vista de implementação do componente serviço consumidor (protótipo inicial)

Para finalizar o desenho do protótipo que representa as aplicações típicas da empresa, referente ao nível de abstração presente, a vista de processo foi adotada para demonstrar o fluxo existente aquando da realização de um pedido, segundo a arquitetura idealizada. Nas figuras 33 e 34 é possível de se observar esta vista para o requisito de gestão de sensores, que se inicia com um pedido efetuado por um sensor que vai ser endereçado pelo serviço produtor, este após validar a informação envia esta informação para o *broker* de mensagens em forma de mensagem. Por fim o *broker* de mensagens envia a mensagem para o serviço consumidor, que aplica a lógica de negócio referente à informação recebida e armazena os objetos de domínio na base de dados.

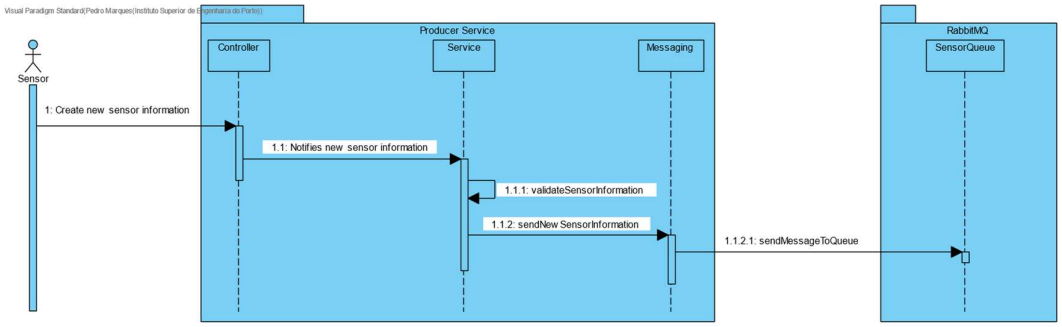



Figura 33 - Vista de processo referente ao requisito de gestão de sensores (parte 1) (protótipo inicial)

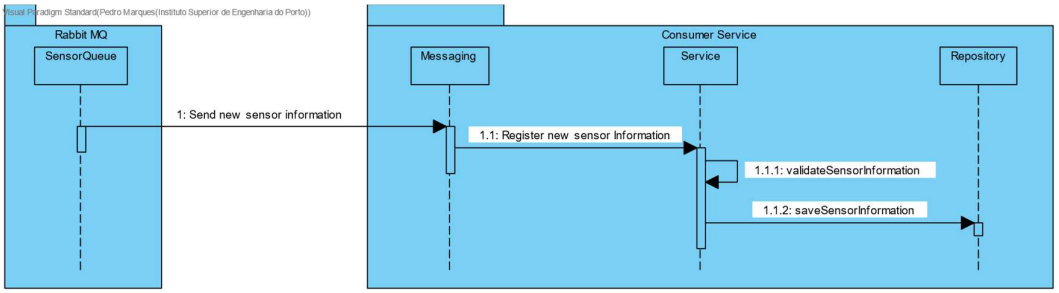



Figura 34 - Vista de processo referente ao requisito de gestão de sensores (parte 2) (protótipo inicial)

O protótipo mediador possui um mecanismo de *event sourcing*, como foi evidenciado nos níveis anteriores, pelo que o serviço produtor deste difere do serviço produtor do protótipo inicial. Na medida em que este novo protótipo possui a camada de repositório no serviço produtor, responsável pelo registo de todos os eventos capturados, observável na figura 35.

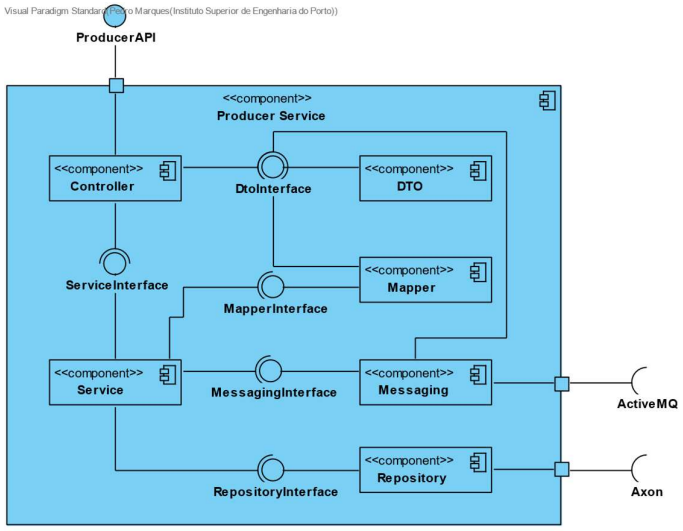



Figura 35 - Vista lógica do componente serviço produtor (protótipo mediador)

O novo serviço adicionado ao protótipo mediador, serviço mediador, presente na figura 36 encontra-se dividido em duas camadas:

- Camada de mensagem: responsável pela receção de mensagens recolhidas com recurso a Apache Camel que implementa o componente ActiveMQ. Também é responsável pelo envio de mensagens através do componente ActiveMQ disponibilizado pelo Apache Camel;
- Camada de serviço: responsável pela aplicação de certas regras de negócio.

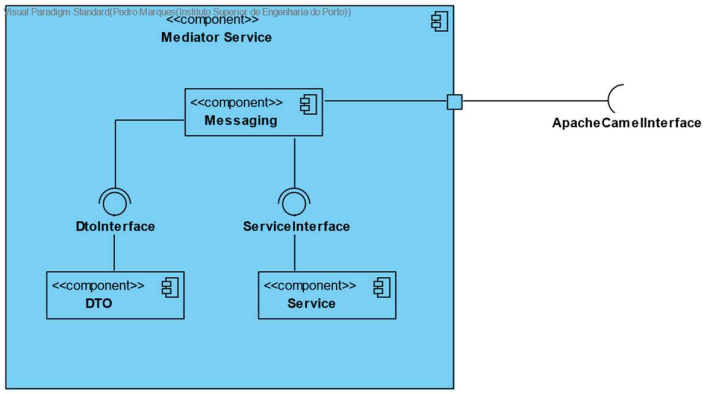



Figura 36 - Vista lógica do componente serviço mediador (protótipo mediador)

Por fim evidencia-se a vista de processo para o requisito de gestão de sensores. Através da figura 37 evidencia-se que o processo é idêntico ao do protótipo inicial, existindo a diferença que neste novo protótipo existe o armazenamento dos eventos recebidos, após o envio de mensagem para o *broker* de mensagens.

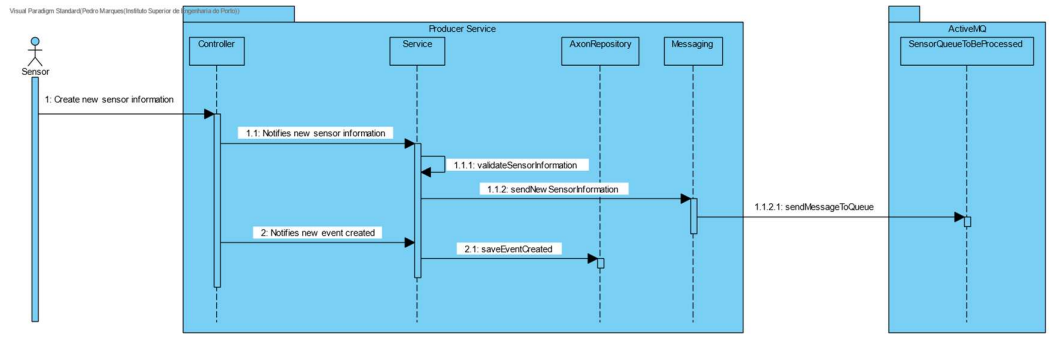



Figura 37 - Vista de processo referente ao requisito de gestão de sensores (parte 1) (protótipo mediador)

Outra diferença face ao protótipo inicial, é que neste novo protótipo a mensagem após ser colocada no *broker* pelo serviço produtor, esta é analisada pelo serviço mediador. Este serviço mediador aplica as regras de negócio e toma a decisão de envio da mensagem para o serviço consumidor, como se observa na figura 38. A parte consumidora é semelhante ao do protótipo inicial, pelo que foi tomada a decisão de não se representar a vista de processo correspondente.

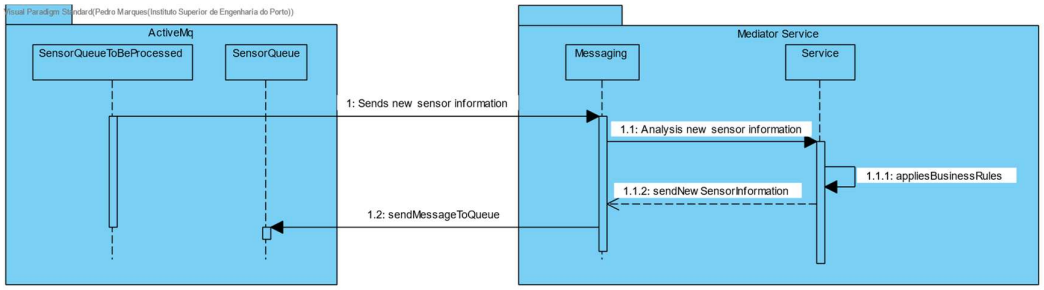



Figura 38 - Vista de processo referente ao requisito de gestão de sensores (parte 2) (protótipo mediador)

O protótipo baseado no modelo ator possui o componente de registo de informação, que agrega parte das responsabilidades presentes nos microsserviços produtor e consumidor do protótipo inicial, que se encontra dividido em quatro camadas principais, evidenciado na figura 39:

- Camada controladora: responsável pela gestão dos pedidos *REST* recebidos. Realiza a gestão do ator principal responsável pela concretização do caso de uso em questão;
- Camada de ator: responsável pela criação de outros atores e pela invocação da camada de serviço;
- Camada de serviço: responsável pela aplicação de regras de negócio e também pelo armazenamento dos eventos produzidos e pela atualização de estado dos conceitos de negócio com recurso à camada de repositório;
- Camada de repositório: responsável pela gestão dos dados de negócio, isto é o armazenamento dos eventos recebidos e pela atualização de estado referente aos conceitos de negócio.

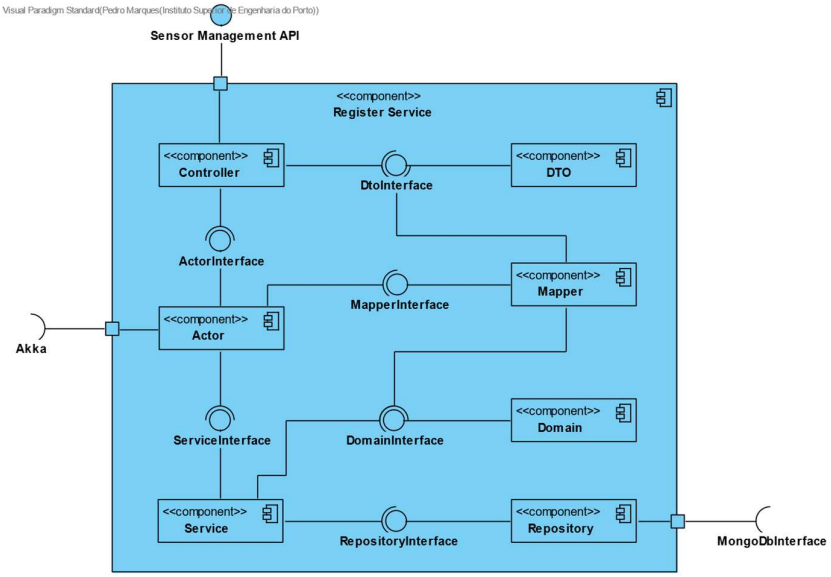



Figura 39 - Vista lógica do componente registo de informação (protótipo atores)

Na figura 40 é percetível a vista de implementação referente ao componente de registo de informação, assim torna-se mais explicita a relação entre os diferentes componentes arquiteturais.

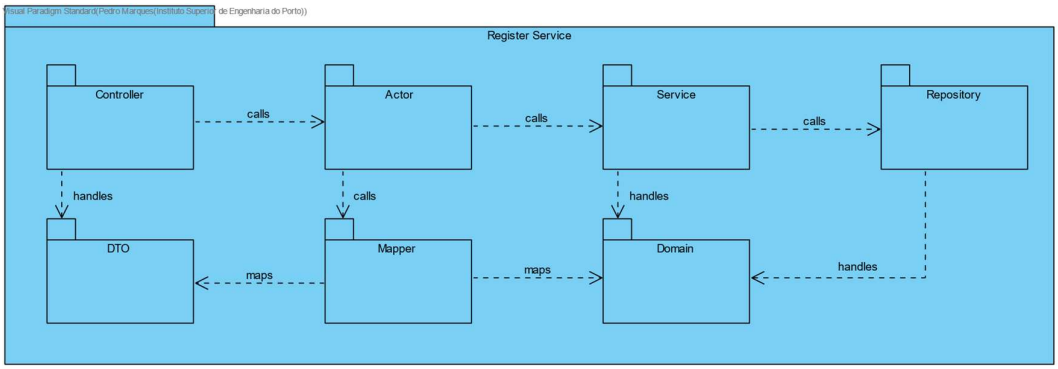



Figura 40 - Vista de implementação do componente registo de informação (protótipo atores)

Finaliza-se com a evidência referente à vista de processo para o requisito de gestão de sensores, presente na figura 41. Aqui percebe-se que após receber informação referente a um sensor, a camada controladora procede à criação de novos atores que efetuam a gestão do registo e da validação dos dados recebidos, recorrendo da camada de serviço para a concretização das tarefas.

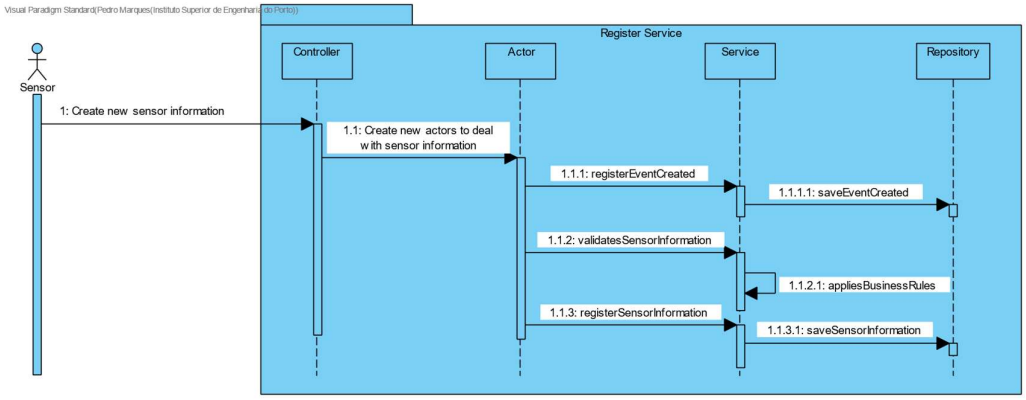



Figura 41 - Vista de processo referente ao requisito de gestão de sensores (protótipo atores)

### 4.6.5 Nível de código

Por último, o nível de código do modelo C4 é explorado. O requisito de gestão de sensores foi escolhido para se concretizar com mais detalhe a lógica endereçada no nível de componente.

A vista de processo referente ao protótipo inicial, presente nas figuras 42 e 43, evidencia os detalhes da implementação referente às aplicações típicas da empresa. Salienta-se a utilização de *mappers*, que permitem a transformação de diferentes tipos de dados/objetos e a correta separação de responsabilidades ao longo das diferentes camadas aplicacionais.

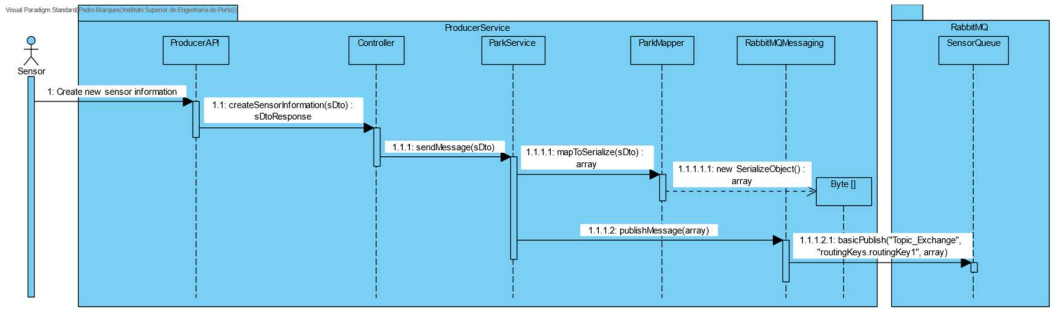



Figura 42 - Vista de processo referente ao requisito de gestão de sensores no nível de código (parte 1) (protótipo inicial)

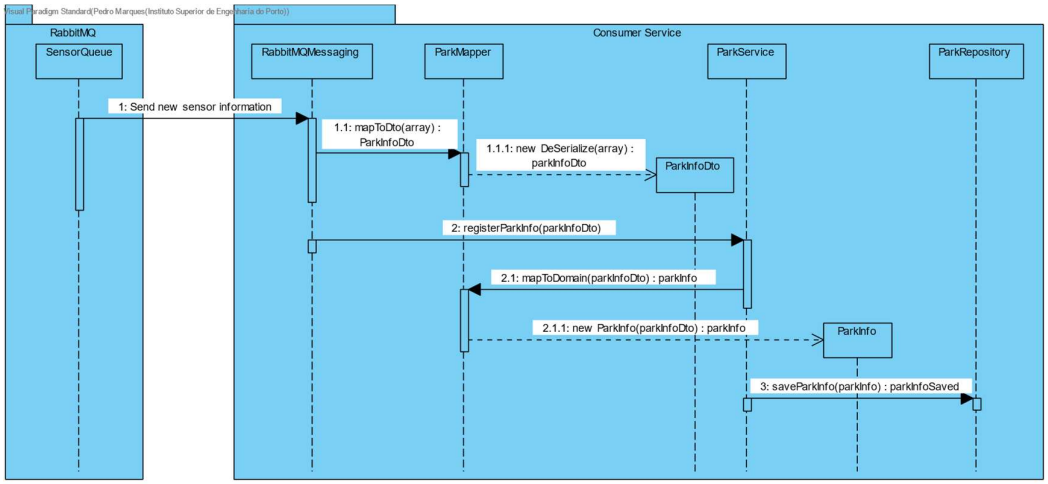



Figura 43 - Vista de processo referente ao requisito de gestão de sensores no nível de código (parte 2) (protótipo inicial)

De seguida, apresenta-se a vista de processo referente ao protótipo mediador, presente nas figuras 44, 45 e 46. Realça-se que neste protótipo a parte produtora, presente na figura 44, faz uso do mecanismo de *event sourcing*, pelo que possui uma camada de repositório responsável pelo armazenamento dos eventos despoletados. A parte mediadora deste protótipo, presente na figura 45, responsável pela intermediação de mensagens entre parte produtora e consumidora, é acedida através da camada de *messaging* com recurso à utilização dos métodos disponibilizados pelo Apache Camel.

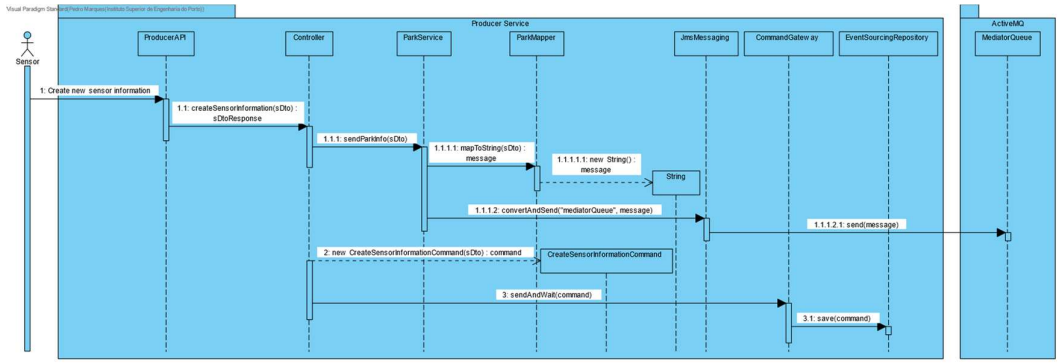



Figura 44 - Vista de processo referente ao requisito de gestão de sensores no nível de código (parte 1) (protótipo mediador)

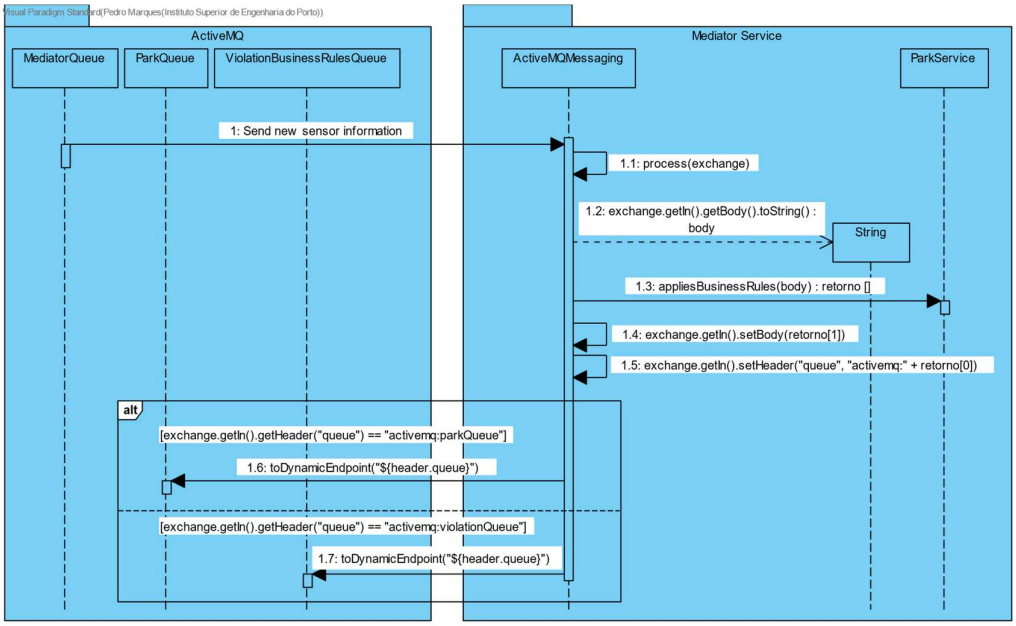



Figura 45 - Vista de processo referente ao requisito de gestão de sensores no nível de código (parte 2) (protótipo mediador)

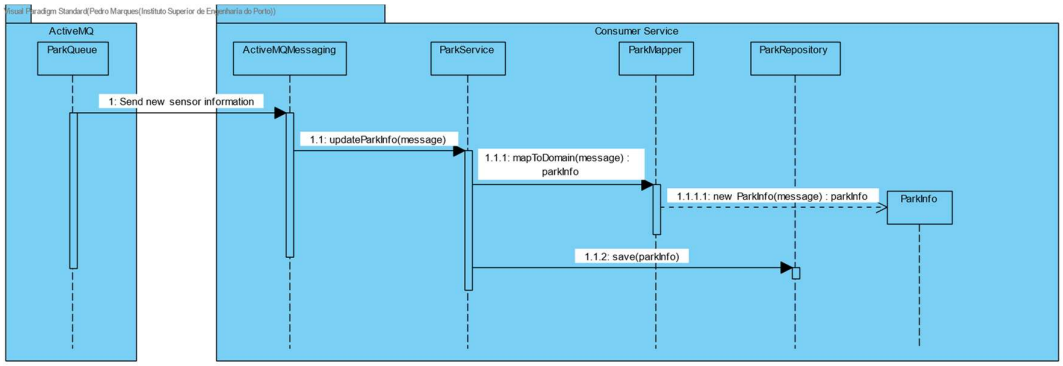



Figura 46 - Vista de processo referente ao requisito de gestão de sensores no nível de código (parte 3) (protótipo mediador)

Finaliza-se com a vista de processo referente ao protótipo baseado no modelo ator, presente na figura 47. Torna-se percetível a utilização das funcionalidades disponibilizadas pelo Akka na camada aplicacional de atores. Cada ator presente nessa camada pode despoletar outro ator, ou então pode instanciar serviços convenientes para se fazer cumprir a lógica do requisito funcional em questão.

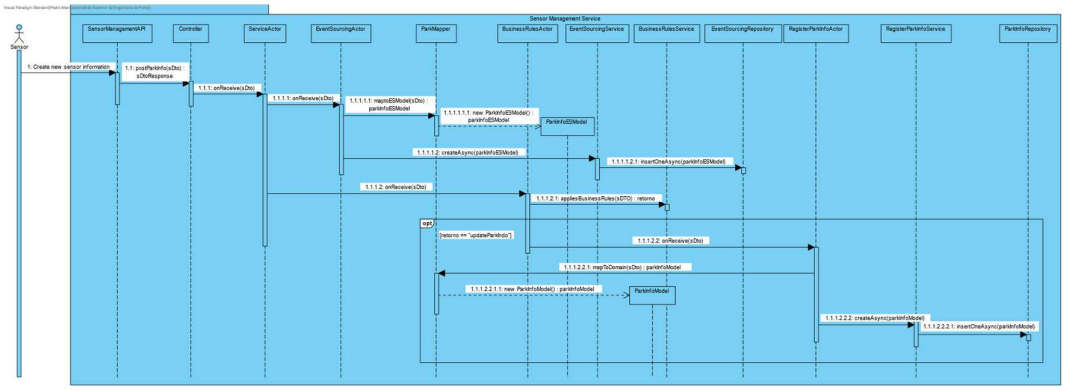


Figura 47 - Vista de processo referente ao requisito de gestão de sensores no nível de código (protótipo atores)

## 4.7 Sumário

Sumariza-se que o negócio de estacionamento de veículos foi descrito, com recurso a requisitos funcionais e não funcionais. Adicionalmente as diferenças arquiteturais existentes entre os protótipos: 1. baseado na topologia *broker* (representativo das aplicações típicas da empresa); 2. baseado na topologia mediadora e 3. baseado no modelo ator foram expostas através de diagramas arquiteturais. Através deste processo tornou-se mais fácil a perceção das alterações a efetuar às aplicações típicas da empresa, para que estas consigam contemplar novas abordagens arquiteturais orientadas a eventos.

# 5 Implementação da solução

Este capítulo pretende expor as principais partes de implementação referentes aos protótipos desenvolvidos. Primeiramente, a configuração de cada um dos protótipos é exposta, através da lista de dependências primárias e do diagrama de implantação. Seguidamente, os detalhes de implementação são evidenciados, através da exposição dos padrões de software adotados e das particularidades das *frameworks* utilizadas. Por fim, os testes realizados são sintetizados, assegurando assim a qualidade geral dos protótipos implementados.

## 5.1 Configuração dos protótipos

### 5.1.1 Protótipo baseado na topologia broker (protótipo inicial)

A tabela 24 agrega as principais dependências referentes ao protótipo inicial, representativo das aplicações típicas da empresa. Utilizaram-se, sempre que possível, as versões estáveis mais recentes, tendo em conta a compatibilidade a existir entre as diferentes dependências a utilizar.

Tabela 24 - Dependências principais do protótipo inicial

| Dependência | Descrição | Versão |
|---|---|---|
| C# | Linguagem de programação adotada. | 10.0 |
| .NET | Versão estável da *framework* usada. Esta versão é a mais recente compatível com a versão de C# escolhida. | 6.0 |
| RabbitMQ.Client | Biblioteca cliente oficial para C#. *Broker* de mensagens adotado. | 6.4.0 |

| | | |
|---|---|---|
| MassTransit.RabbitMQ | *Framework open-source* para .NET. Permite abstrair a lógica necessária para se conseguir fazer uso de brokers de mensagens. | 8.0.8 |
| Swashbuckle.AspNetCore | Ferramentas Swagger para documentação de *APIs*. | 6.2.3 |
| OpenTracing | Esta biblioteca é uma implementação em .NET da *API* de OpenTracing. | 0.12.1 |
| Jaeger | Dependência que permite monitorizar sistemas distribuídos. | 1.0.3 |
| App.Metrics.AspNetCore | *Framework web open-source* que permite a recolha de métricas e a respetiva exposição das mesmas. | 4.0.0 |
| MongoDB.Driver | Driver oficial para MongoDB. | 2.18.0 |

O diagrama de implantação presente na figura 48 pretende sintetizar a visão de implantação estática referente ao protótipo inicial, onde se observam os diferentes nós e as suas relações. O diagrama foi dividido em dois pacotes, o pacote de serviços e o de dependências. O pacote de serviços contém os dois microsserviços implementados, que necessitam dos nós presentes no pacote de dependências para o seu correto funcionamento.

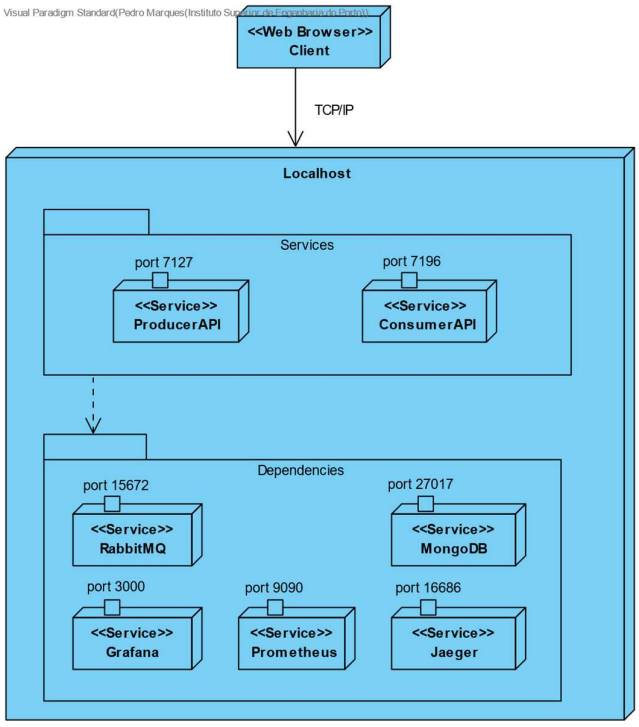



Figura 48 - Diagrama de implantação do protótipo inicial

### 5.1.2 Protótipo baseado na topologia mediadora

Na tabela 25 sumariza-se as principais dependências referentes ao protótipo mediador. Utilizaram-se sempre que possível as versões estáveis mais recentes, tendo em conta a compatibilidade que tem de existir entre as diferentes dependências a utilizar.

Tabela 25 - Dependências principais do protótipo mediador

| **Dependência** | **Descrição** | **Versão** |
|---|---|---|
| Java | Linguagem de programação adotada. | 17 |
| Spring Boot | *Framework* usada compatível com a versão da linguagem de programação. | 2.5.4 |
| Axon Framework | *Framework* java *open-source* que facilita a construção de microsserviços orientados a eventos. Permite a aplicação de padrões *CQRS* e *event sourcing*. | 4.5.3 |
| OpenTracing | Dependência que permite a utilização da *API* de OpenTracing. A ferramenta Jaeger foi adotada sendo possível a monitorização dos microsserviços. | 3.3.1 |
| Resilience4j | Biblioteca que disponibiliza mecanismos de tolerância a falhas. O mecanismo de *circuit breaker* foi adotado. | 1.5.0 |
| Micrometer | Conjunto de bibliotecas que permitem a captura e a exposição de métricas, através de diferentes ferramentas. A ferramenta Prometheus disponibilizada foi adotada. | 1.7.3 |
| ActiveMQ | Broker de mensagens adotado. | 2.5.4 |
| Derby | Conector relacional usado para persistência de eventos despoletados. | 10.14.2.0 |
| Apache Camel | Conector Apache Camel que permite a adição de *routes* mediadoras. Permite a utilização do conector ActiveMQ. | 3.19.0 |
| MongoBB | Conector não relacional usado para persistência de dados. | 2.7.5 |

O diagrama de implantação presente na figura 49 pretende sintetizar a visão de implantação estática referente ao protótipo mediador, onde se observam os diferentes nós e as suas relações. O diagrama foi dividido em dois pacotes, o pacote de serviços e o de dependências. O pacote de serviços contém os três microsserviços desenvolvidos, que necessitam dos nós presentes no pacote de dependências para o seu correto funcionamento.

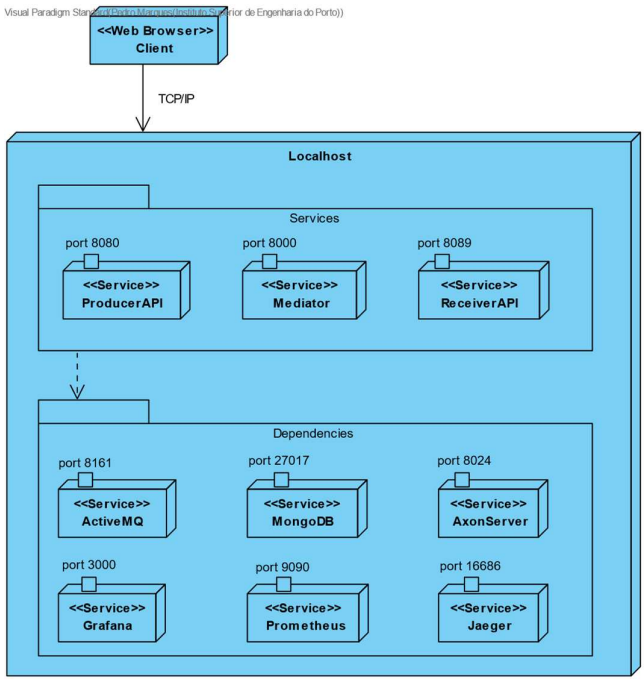



Figura 49 - Diagrama de implantação do protótipo mediador

### 5.1.3 Protótipo baseado no modelo ator

A tabela 26 sumariza as principais dependências referentes ao protótipo baseado em atores. Utilizaram-se sempre que possível as versões estáveis mais recentes, tendo em conta a compatibilidade que tem de existir entre as diferentes dependências a utilizar.

Tabela 26 - Dependências principais do protótipo baseado em atores

| **Dependência** | **Descrição** | **Versão** |
|---|---|---|
| C# | Linguagem de programação adotada. | 10.0 |
| .NET | Versão estável da *framework* usada. Esta versão é a mais recente compatível com a versão de C# escolhida. | 6.0 |
| Akka.NET | Biblioteca *open-source* para C# que permite a implementação do modelo ator. | 1.4.49 |
| App.Metrics.AspNetCore | *Framework web open-source* que permite a recolha de métricas e a respetiva exposição das mesmas. | 4.3.0 |
| Jaeger | Dependência que permite monitorizar sistemas distribuídos. | 1.0.3 |
| MongoDB.Driver | Driver oficial para MongoDB. Permite a persistência de eventos e de dados de negócio. | 2.19.0 |

| | | |
|---|---|---|
| Open Tracing | Esta biblioteca é uma implementação em .NET da *API* de OpenTracing. | 0.12.1 |
| Swashbuckle.AspNetCore | Ferramentas Swagger para documentação de *APIs*. | 0.12.1 |

O diagrama de implantação presente na figura 50 pretende sintetizar a visão de implantação estática referente ao protótipo baseado em atores, onde se observam os diferentes nós e as suas relações. O diagrama foi dividido em dois pacotes, o pacote de serviços e o de dependências. O pacote de serviços contém os dois microsserviços desenvolvidos, que fazem uso dos nós presentes no pacote de dependências para o seu correto funcionamento.

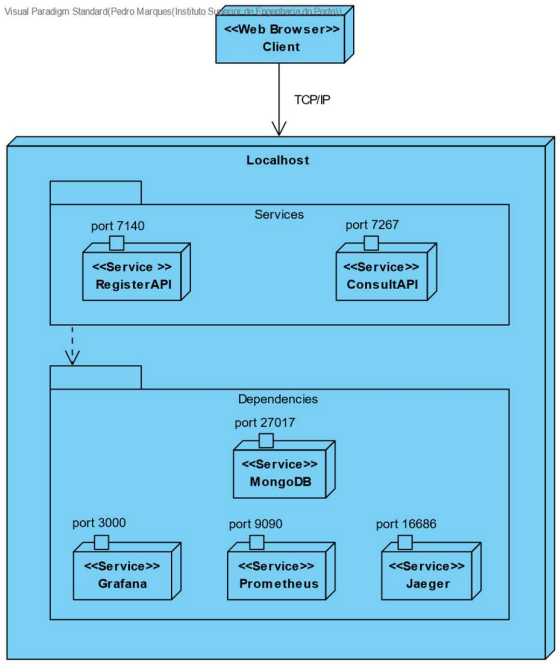



Figura 50 - Diagrama de implantação do protótipo baseado em atores

## 5.2 Uso de padrões

Esta secção pretende expor os principais detalhes de implementação referentes ao uso de padrões nos protótipos implementados.

### 5.2.1 CQRS

Na secção 2.3.2 o padrão *CQRS* foi introduzido e no capítulo 4 os protótipos foram concebidos considerando as exigências que este padrão de software despoleta.

A abordagem da divisão dos sistemas em duas partes, a parte de comandos e de consultas, permitiu o desenvolvimento de protótipos tendo em conta o princípio da separação de responsabilidades.

O protótipo mediador possui estruturas de escrita e de leitura diferenciadas. Quando o microsserviço *ProducerAPI* recebe informação proveniente de sensores de estacionamento, via protocolo *HTTP*, este utiliza a sua estrutura de escrita para persistência do evento que recolheu. Adicionalmente o corpo da mensagem referente ao sensor de estacionamento é disponibilizado num mecanismo de comunicação assíncrona. O microsserviço *Mediator* analisa a mensagem, recolhida do broker de mensagens, com recurso a Apache Camel. Por fim, a mensagem alcança o microsserviço *ReceiverAPI* que procede à atualização da base de dados de consultas de informação relativa a estacionamentos. O microsserviço *ReceiverAPI* oferece aos utilizadores uma estrutura de dados de consulta.

O excerto de código 1 expõe uma estrutura de comando presente no protótipo mediador, utilizada para criar informação proveniente de sensores de estacionamento.

```
 9 @Data
10 @Builder
11 public class CreateSensorInformationCommand {
12
13     @TargetAggregateIdentifier
14     private String sensorId;
15     private String tempoRecolha;
16     private String estacionamento;
17     private Boolean estado;
18 }
```

Código 1 – Estrutura de comando para criar informação de estacionamento

Após a construção da estrutura de comando, presente no excerto de código 1, torna-se necessário instanciar a mesma com a informação proveniente do sensor de estacionamento, presente nas linhas 50 até 56 do excerto de código 2. Depois a interface *CommandGateway* é utilizada permitindo que os componentes enviem comandos aguardando o resultado da execução, com recurso ao método *sendAndWait*, presente na linha 57 do excerto de código 2.

```
50          CreateSensorInformationCommand createSensorInformationCommand =
51                  CreateSensorInformationCommand.builder()
52                          .sensorId(UUID.randomUUID().toString())
53                          .tempoRecolha(park.TempoRecolha)
54                          .estacionamento(park.Estacionamento)
55                          .estado(park.Estado)
56                          .build();
57          String result = commandGateway.sendAndWait(createSensorInformationCommand);
```

Código 2 – Instanciação da estrutura de comando e respetivo envio do mesmo via CommandGateway

A construção do protótipo baseado em atores também teve em atenção o princípio da separação e segregação de responsabilidades. Quando o microsserviço *RegisterAPI* obtém informação proveniente de sensores de estacionamento, via protocolo *HTTP*, este utiliza a sua estrutura de escrita para persistência do evento com recurso ao assincronismo do ator *EventSourcingActor*. Posteriormente, o mesmo microsserviço despoleta outro ator,

denominado de *RegisterParkInfoActor*, que procede à persistência dos dados do sensor de estacionamento, utilizando uma estrutura de dados de escrita diferenciada face à utilizada no processo de *event sourcing*. O microsserviço *ConsultAPI* fornece aos utilizadores uma estrutura de dados de consulta de informação relativa a sensores de estacionamento.

### 5.2.2 Event Sourcing

Na secção 2.3.3 o padrão *event sourcing* foi anunciado como um padrão relevante de aplicação numa arquitetura de microsserviços, e no capítulo 4 os protótipos foram concebidos tendo em conta os requisitos que este padrão exige.

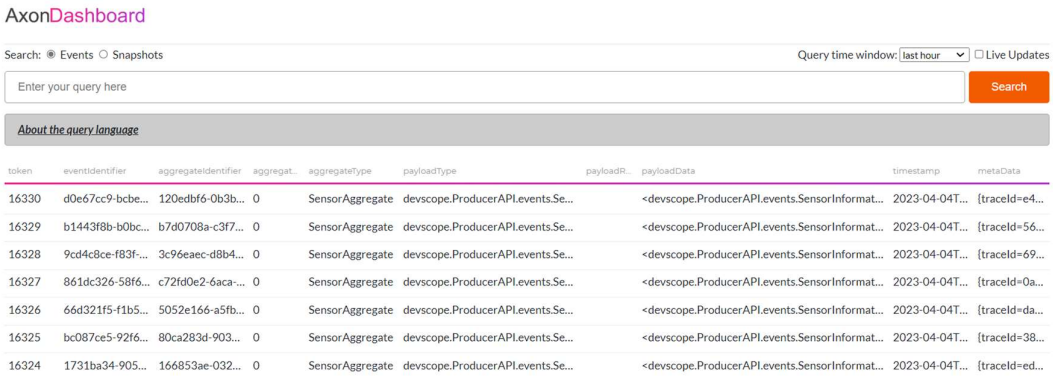
A abordagem de se persistir qualquer evento que é processado pelos protótipos, através de eventos de domínio imutáveis, foi adotada nos microsserviços de ordem produtora, ou seja, aqueles que possuem a faceta de escrita.

A *framework* Axon foi utilizada no protótipo mediador onde se definiu o objeto de domínio, identificado como um agregado na terminologia da *framework,* presente no excerto de código 3, que agrega a informação de cada evento. No construtor do agregado, presente nas linhas 24 até 32 do excerto de código 3, é injetado o comando que possui os dados referentes ao evento recebido, depois o evento em si é criado originando o objeto representativo *SensorInformationCreatedEvent*. O agregado possui também um manipulador de eventos gerados pelo mesmo, presente nas linhas 38 até 43 do excerto de código 3, que possui os dados do corpo do evento despoletado.

```
14 @Aggregate
15 public class SensorAggregate {
16
17     @AggregateIdentifier
18     private String sensorId;
19     private String tempoRecolha;
20     private String estacionamento;
21     private Boolean estado;
22
23     @CommandHandler
24     public SensorAggregate(CreateSensorInformationCommand createSensorInformationCommand) {
25         //You can perform all the validations
26         SensorInformationCreatedEvent sensorInformationCreatedEvent =
27                 new SensorInformationCreatedEvent();
28
29         BeanUtils.copyProperties(createSensorInformationCommand,sensorInformationCreatedEvent);
30
31         AggregateLifecycle.apply(sensorInformationCreatedEvent);
32     }
33
34     public SensorAggregate() {
35     }
36
37     @EventSourcingHandler
38     public void on(SensorInformationCreatedEvent sensorInformationCreatedEvent) {
39         this.sensorId = sensorInformationCreatedEvent.getSensorId();
40         this.tempoRecolha = sensorInformationCreatedEvent.getTempoRecolha();
41         this.estacionamento = sensorInformationCreatedEvent.getEstacionamento();
42         this.estado = sensorInformationCreatedEvent.getEstado();
43     }
44 }
```

Código 3 - Objeto de domínio para event sourcing utilizando a framework Axon

Na figura 51 expõe-se o *dashboard* disponibilizado pelo servidor Axon para consulta dos eventos de domínio, registados na parte de escrita do protótipo mediador.




| token | eventIdentifier | aggregateIdentifier | aggregat... | aggregateType | payloadType | payloadR... | payloadData | timestamp | metaData |
|---|---|---|---|---|---|---|---|---|---|
| 16330 | d0e67cc9-bcbe... | 120edbf6-0b3b... | 0 | SensorAggregate | devscope.ProducerAPI.events.Se... | | <devscope.ProducerAPI.events.SensorInformat... | 2023-04-04T... | {traceId=e4... |
| 16329 | b1443f8b-b0bc... | b7d0708a-c3f7... | 0 | SensorAggregate | devscope.ProducerAPI.events.Se... | | <devscope.ProducerAPI.events.SensorInformat... | 2023-04-04T... | {traceId=56... |
| 16328 | 9cd4c8ce-f83f-... | 3c96eaec-d8b4... | 0 | SensorAggregate | devscope.ProducerAPI.events.Se... | | <devscope.ProducerAPI.events.SensorInformat... | 2023-04-04T... | {traceId=69... |
| 16327 | 861dc326-58f6... | c72fd0e2-6aca-... | 0 | SensorAggregate | devscope.ProducerAPI.events.Se... | | <devscope.ProducerAPI.events.SensorInformat... | 2023-04-04T... | {traceId=0a... |
| 16326 | 66d321f5-f1b5... | 5052e166-a5fb... | 0 | SensorAggregate | devscope.ProducerAPI.events.Se... | | <devscope.ProducerAPI.events.SensorInformat... | 2023-04-04T... | {traceId=da... |
| 16325 | bc087ce5-92f6... | 80ca283d-903... | 0 | SensorAggregate | devscope.ProducerAPI.events.Se... | | <devscope.ProducerAPI.events.SensorInformat... | 2023-04-04T... | {traceId=38... |
| 16324 | 1731ba34-905... | 166853ae-032... | 0 | SensorAggregate | devscope.ProducerAPI.events.Se... | | <devscope.ProducerAPI.events.SensorInformat... | 2023-04-04T... | {traceId=ed... |

Figura 51 - Excerto da lista de eventos armazenada no servidor Axon

Na figura 52 evidencia-se a estrutura de um evento de domínio que para além do corpo do evento, *payloadData*, destacam-se o identificador do evento, *eventIdentifier*, o identificador do agregado, *aggregateIdentifier*, e o tempo de recolha do evento, *timestamp*.

| Name | Value |
|---|---|
| token | 16330 |
| eventIdentifier | d0e67cc9-bcbe-4395-90cc-ab24eb456eeb |
| aggregateIdentifier | 120edbf6-0b3b-48d7-b42d-a7c8cb14f7f1 |
| aggregateSequenceNumber | 0 |
| aggregateType | SensorAggregate |
| payloadType | devscope.ProducerAPI.events.SensorInformationCreatedEvent |
| payloadRevision | |
| payloadData | <devscope.ProducerAPI.events.SensorInformationCreatedEvent><sensorId>120edbf6-0b3b-48d7-b42d-a7c8cb14f7f1</sensorId><tempoRecolha>16/11/2022 03:52:11</tempoRecolha><estacionamento>Spot 56, Avenida da Liberdade, Porto, Porto</estacionamento><estado>true</estado></devscope.ProducerAPI.events.SensorInformationCreatedEvent> |
| timestamp | 2023-04-04T19:27:16.079Z |
| metaData | {traceId=e4b59f1b-df4e-4765-bf8a-4f467c14104f, correlationId=e4b59f1b-df4e-4765-bf8a-4f467c14104f} |

Figura 52 - Estrutura de um evento

No protótipo baseado em atores não foi utilizada nenhuma *framework* de aplicação de *event sourcing*. Mas, foi construída uma estrutura de dados, idêntica à do agregado do excerto de código 3, que permitiu a persistência de eventos recebidos na parte de escrita do protótipo numa base de dados não relacional em MongoDB.

### 5.2.3 Circuit Breaker

A aplicação de *circuit breaker* permite a criação de sistemas de microsserviços resilientes, através de mecanismos que permitem limitar o impacto de falhas nos serviços e de latências existentes (Dineshchand, 2022). Pelo que, o objetivo principal da aplicação de *circuit breaker* é o de evitar qualquer falha em cascata num sistema (Dineshchand, 2022).

A figura 53 evidencia os diferentes estados existentes na aplicação de um *circuit breaker*, bem como as transições entre estados plausíveis de se realizarem. Um *circuit breaker* permanece no

estado fechado quando o sistema que se pretende invocar está disponível e uma resposta adequada é recebida por parte do invocador do serviço em questão (Dineshchand, 2022). Caso o número de invocações falhadas ao sistema que se pretende invocar exceda um valor configurável, o *circuit breaker* alcança o estado aberto. Neste estado aberto, o *circuit breaker* retorna um erro sem efetuar a chamada ao serviço pretendido (Dineshchand, 2022). Após atingido o limite temporal configurado, o *circuit breaker* transita automaticamente do estado aberto para o estado meio aberto, com o intuito de verificar se o problema com o serviço que se pretende invocar ainda persiste (Dineshchand, 2022). Caso a invocação ao serviço falhe, neste estado meio aberto, o *circuit breaker* é colocado outra vez no estado aberto. Contrariamente, no caso de a invocação ser realizada com sucesso, o *circuit breaker* é colocado no estado fechado, ou seja, o sistema retoma ao normal funcionamento.

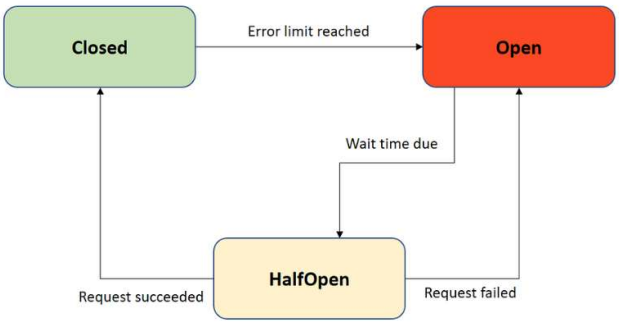



Figura 53 - Estados existentes na aplicação de circuit breaker (Dineshchand, 2022)

A configuração necessária para se fazer uso do mecanismo de *circuit breaker*, no protótipo mediador, é evidenciada no excerto de código 4. Salienta-se a definição da taxa de erro de 50%, caso seja atingida o sistema passa automaticamente para o estado aberto, e a definição do valor 5 como número mínimo de invocações no estado fechado para posteriormente se avaliar a transição de estado. Definiu-se, complementarmente, o valor de 5 segundos como duração máxima que o sistema pode encontrar-se no estado aberto, e o valor de 3 como o número máximo de invocações ao sistema no estado meio aberto.

```
resilience4j.circuitbreaker.instances.circuitBreakerMessageBroker.registerHealthIndicator=true
resilience4j.circuitbreaker.instances.circuitBreakerMessageBroker.eventConsumerBufferSize=10
resilience4j.circuitbreaker.instances.circuitBreakerMessageBroker.failureRateThreshold=50
resilience4j.circuitbreaker.instances.circuitBreakerMessageBroker.minimumNumberOfCalls=5
resilience4j.circuitbreaker.instances.circuitBreakerMessageBroker.automaticTransitionFromOpenToHalfOpenEnabled=true
resilience4j.circuitbreaker.instances.circuitBreakerMessageBroker.waitDurationInOpenState=5s
resilience4j.circuitbreaker.instances.circuitBreakerMessageBroker.permittedNumberOfCallsInHalfOpenState=3
resilience4j.circuitbreaker.instances.circuitBreakerMessageBroker.slidingWindowSize=10
resilience4j.circuitbreaker.instances.circuitBreakerMessageBroker.slidingWindowType=COUNT_BASED
```

Código 4 - Excerto do ficheiro application.properties referente à configuração do circuit breaker

Após definição do *circuit breaker* torna-se possível aplicar as configurações a métodos e a classes de código. No excerto de código 5 exemplifica-se a anotação a usar, assim o método em questão implementa o *circuit breaker*, denominado de *circuitBreakerMessageBroker*, e caso exista algum erro de invocação o método responsável pela gestão da falha, denominado de *messageBrokerFallback*, é acionado.

```
38    @CircuitBreaker(name = "circuitBreakerMessageBroker", fallbackMethod = "messageBrokerFallBack")
```

Código 5 - Anotação de circuit breaker

## 5.3 Akka.NET

Na secção 2.2.2.5 foi sintetizado o modelo conceptual ator, inclusive realizou-se uma breve descrição da *framework* orientada a atores Akka.NET. Os detalhes de implementação referentes à *framework* são abordados nesta secção.

Atores em C# são implementados através da extensão da classe *UntypedActor* e através da implementação do método *OnReceive* (Petabridge, 2017b). *Props* é uma classe de configuração que permite a especificação de opções para a criação de atores (Petabridge, 2017b). Um ator é criado através do método *ActorOf*, passando uma instância de *Props*, disponível nas classes *ActorSystem* e *ActorContext* (Petabridge, 2017b). Caso se utilize o método da classe *ActorSystem* um ator de topo é criado, supervisionado pelo ator guardião fornecido pelo sistema de atores, por outro lado caso se use o método da classe *ActorContext* um ator filho é criado (Petabridge, 2017b).

No excerto de código 6 evidencia-se parte da implementação do ator *ServiceActor*, responsável pela gestão dos atores, referente ao protótipo baseado em atores. Esse ator no seu construtor procede à inicialização de dois atores filhos, com recurso ao método *ActorOf* da classe *ActorContext*, presente nas linhas 18 até 23 do excerto de código 6. Nas linhas 25 até 32 do excerto de código 6 o método *OnReceive* é exposto, aqui os atores responsáveis pela realização de *event sourcing* e de aplicação de regras de negócio são invocados com recurso ao método *Tell*. O método *Tell* despoleta automaticamente, de uma forma assíncrona, o método *OnReceive* de cada ator invocado.

```
 9    public class ServiceActor : UntypedActor
10    {
11
12        //Child actor dedicated to event sourcing
13        private readonly IActorRef eventSourcingActor;
14
15        //Child actor dedicated to business rules application
16        private readonly IActorRef businessRulesActor;
17
          0 references
18        public ServiceActor(ParkInfoESService arg1, ParkService arg2, ParkInfoService arg3)
19        {
20            this.eventSourcingActor = Context.ActorOf(Props.Create<EventSourcingActor>(arg1));
21
22            this.businessRulesActor = Context.ActorOf(Props.Create<BusinessRulesActor>(arg2, arg3));
23        }
24
          0 references
25        protected override void OnReceive(object message)
26        {
27            Console.WriteLine($"Message received: {message}");
28            //EventSourcingActor actor invocation
29            eventSourcingActor.Tell(message);
30            //BusinessRulesActor actor invocation
31            businessRulesActor.Tell(message);
32        }
```

Código 6 - Excerto da classe de código do ator ServiceActor

## 5.4 Estratégia de supervisão

Em sistemas baseados em atores o conceito de supervisão descreve uma relação de dependência entre diferentes atores, onde o ator supervisor atribui tarefas aos atores subordinados e, consequentemente, deve responder às suas eventuais falhas (Petabridge, 2017a). Caso um ator subordinado detete uma falha, este suspende-se a si próprio e a todos os seus subordinados e envia uma mensagem para o seu ator supervisor, existindo a sinalização da falha (Petabridge, 2017a). Dependendo da lógica de atores implementada e da natureza da falha detetada, o ator supervisor pode optar pelas seguintes opções (Petabridge, 2017a):

- Retomar o subordinado, persistindo o seu estado interno acumulado;
- Reiniciar o subordinado, apagando o seu estado interno acumulado;
- Parar o subordinado de uma forma permanente;
- Escalar a falha para o próximo supervisor na hierarquia, existindo assim uma falha.

O excerto de código 7 evidencia um exemplo de uma estratégia de supervisão adotada, mais especificamente no ator *ServiceActor* referente ao protótipo baseado em atores, a aplicar quando um ator supervisionado retornar uma falha. Para esta estratégia definiu-se 10 como o número máximo de tentativas, presente na linha 38 do excerto de código 7, durante 1 minuto, presente na linha 39. Caso esse valor seja atingido o ator supervisionado é parado de imediato. Nas linhas 40 até 49 do excerto de código 7 realiza-se o mapeamento da exceção, detetada pelo ator supervisionado, numa diretiva de supervisão. Salienta-se que nessa estratégia de supervisão caso a exceção retornada não estiver descrita nas linhas 44 e 45 do excerto de código 7, o ator supervisionado é parado de imediato.

```
34      //Supervision strategy to deal with eventual errors in the child actors execution
        0 references
35      protected override SupervisorStrategy SupervisorStrategy()
36      {
37          return new OneForOneStrategy(
38              maxNrOfRetries: 10,
39              withinTimeRange: TimeSpan.FromMinutes(1),
40              localOnlyDecider: ex =>
41              {
42                  return ex switch
43                  {
44                      NullReferenceException ne => Directive.Restart,
45                      SocketException se => Directive.Resume,
46                      _ => Directive.Stop
47                  };
48              }
49              );
50      }
```

Código 7 - Estratégia de supervisão adotada no ator ServiceActor

## 5.5 Testes

Com o intuito de se garantir o correto funcionamento dos protótipos desenvolvidos, estes foram sujeitos a diferentes níveis de teste. A figura 54 evidencia os níveis de teste habitualmente usados em projetos de software. Primeiramente foram efetuados testes unitários, seguidos dos testes de integração, por fim efetuaram-se testes de sistema e de aceitação.

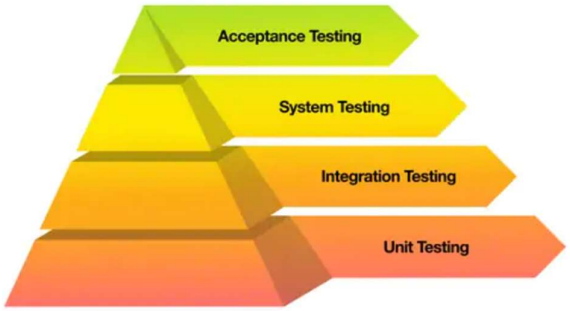



Figura 54 - Níveis de teste de software – imagem de (Paspelava, 2021)

### 5.5.1 Testes Unitários

Neste nível é possível testar unidades individuais ou componentes de um sistema. O objetivo deste nível de teste é verificar cada unidade individual, através do seu isolamento, e de seguida realizar os testes que demonstrem que cada componente consegue dar resposta aos requisitos definidos para uma determinada funcionalidade (Reqtest, 2014). Através deste nível de teste é possível a deteção de erros e de *bugs* numa fase inicial do processo de desenvolvimento de um produto de software (Reqtest, 2014).

A ferramenta XUnit foi usada para se efetuar os testes aos protótipos desenvolvidos em C# com a *framework* .NET. No excerto de código 8 evidencia-se um teste efetuado ao método *mapToESModel,* do protótipo baseado em atores, com recurso a XUnit, salienta-se a palavra reservada *Fact* que é usada pelo *test runner* executar o método de teste *mapToESModelTest*.

```
11      [Fact]
        0 references
12      public void mapToESModelTest()
13      {
14          ParkInfo park = new ParkInfo();
15          park.Estacionamento = "137 Rua infante D.Henrique, Porto, Porto";
16          park.Estado = true;
17          park.TempoRecolha = new DateTime(2022, 12, 13, 15, 45, 55);
18
19          ParkInfoESModel parkInfoESModel = ParkMapper.mapToESModel(park.ToString());
20
21          Assert.NotNull(parkInfoESModel);
22          String estacionamentoExpected = " 137 Rua infante D.Henrique, Porto, Porto ";
23          Assert.Equal(parkInfoESModel.estacionamento, estacionamentoExpected);
24          String estadoExpected = " True";
25          Assert.Equal(parkInfoESModel.estado, estadoExpected);
26          String tempoRecolhaExpected = " 12/13/2022 3:45:55 PM ";
27          Assert.Equal(parkInfoESModel.tempoRecolha, tempoRecolhaExpected);
28      }
```

Código 8 - Excerto da classe de testes ParkMapperTests

A ferramenta JUnit foi utilizada para se efetuar os testes ao protótipo mediador. Adicionalmente a *framework* Mockito foi adotada para se conseguir isolar os componentes devidamente, assim tornou-se possível efetuar testes unitários com maior rigor. O excerto de código 9 evidencia um exemplo de instanciação dos objetos que possuem a anotação *@Mock*.

```
@BeforeEach
public void setUp() {
    MockitoAnnotations.openMocks(this);
    this.service = new EstacionamentoService(estacionamentoRepo);
}
```

Código 9 - Inicialização de objetos Mock

A *framework* Mockito disponibiliza métodos que permitem simular o retorno de diferentes funções e/ou métodos, assim foi possível, por exemplo, isolar a camada de repositório da camada de serviço do protótipo mediador, observável nas linhas 41 e 45 do excerto de código 10. Na linha 52, do excerto de código 10, o método que se pretende testar, *atualizarEstacionamento*, é invocado recorrendo aos métodos simulados sempre que este invoca métodos de componentes externos à camada de serviço.

```
35  @Test
36  public void atualizarEstacionamentoTest() {
37
38      //Repository methods mock setup
39      List<Estacionamento> estacionamentos = new ArrayList<Estacionamento>();
40      estacionamentos.add(new Estacionamento(new EstacionamentoId("12 Rua de testes, Porto, Porto"), true, new TempoRecolha("16", "11", "2022", "03", "52", "11")));
41      doReturn(estacionamentos).when(estacionamentoRepo).findByEstacionamentoId(isA(String.class));
42      Estacionamento estacionamentoSaved = new Estacionamento(new EstacionamentoId("12 Rua de testes, Porto, Porto"),
43                                                              true,
44                                                              new TempoRecolha("16", "11", "2022", "03", "52", "11"));
45      doReturn(estacionamentoSaved).when(estacionamentoRepo).save(isA(Estacionamento.class));
46
47      //Call to method that is intended to be tested
48      String estacionamento = "12 Rua de testes, Porto, Porto";
49      String estado = "true";
50      String tempoRecolha = "16/11/2022 03:52:11";
51      String message = "Estacionamento-> " + estacionamento + " ;Estado-> " + estado + " ;TempoRecolha-> " + tempoRecolha + ";";
52      Estacionamento estacionamentoRetornado = this.service.atualizarEstacionamento(message);
53
54      //Asserts to test the output obtained by the execution of atualizarEstacionamento of the EstacionamentoService class
55      assertThat(estacionamentoRetornado.getIdentificador().getIdentificador()).isEqualTo(estacionamento);
56      assertThat(estacionamentoRetornado.getEstado()).isEqualTo(true);
57      assertThat(estacionamentoRetornado.getTempoRecolha().getDia()).isEqualTo("16");
58      assertThat(estacionamentoRetornado.getTempoRecolha().getMes()).isEqualTo("11");
59      assertThat(estacionamentoRetornado.getTempoRecolha().getAno()).isEqualTo("2022");
60      assertThat(estacionamentoRetornado.getTempoRecolha().getHora()).isEqualTo("03");
61      assertThat(estacionamentoRetornado.getTempoRecolha().getMinuto()).isEqualTo("52");
62      assertThat(estacionamentoRetornado.getTempoRecolha().getSegundo()).isEqualTo("11");
63  }
```

Código 10 - Teste unitário atualizarEstacionamentoTest

### 5.5.2 Testes de Integração

Testes de integração são realizados com o objetivo de se conseguir testar a interação entre diferentes partes de um sistema, assim consegue-se perceber se as partes funcionam corretamente quando integradas/combinadas (Reqtest, 2014).

No excerto de código 11 apresenta-se um exemplo representativo dos testes de integração efetuados, mais concretamente um teste efetuado ao ator responsável pela gestão do registo da informação de estacionamento, referente ao protótipo baseado em atores. Observa-se na linha 30, do excerto de código 11, que no momento da instanciação do ator *RegisterParkInfoActor*, o serviço de registo de informação, *dbService*, é injetado. Assim, torna-se possível averiguar a correta integração entre a camada de ator e de serviço, terminando o teste com a verificação do estado do ator, presente nas linhas 35 e 36 do excerto de código 11, após finalização da execução de lógica assíncrona do ator em teste.

```
21      [Fact]
        0 references
22      public void CreateRegisterParkInfoActorTest()
23      {
24          ParkInfo park = new ParkInfo();
25          park.Estacionamento = "Rua infate D.Pedro, Porto, Porto";
26          park.Estado = true;
27          park.TempoRecolha = new DateTime(2022, 12, 14);
28
29          var actorSystem = ActorSystem.Create("test-actor-system");
30          var actor = actorSystem.ActorOf(Props.Create<RegisterParkInfoActor>(dbService));
31          actor.Tell(park.ToString());
32
33          Thread.Sleep(1000);
34          Boolean expectedStatus = false;
35          var actorAfterExecution = actorSystem.ActorSelection(actor.Path);
36          Assert.Equal(actorAfterExecution.Anchor.IsNobody(), expectedStatus);
37      }
```

Código 11 - Teste de integração CreateRegisterParkInfoActor

### 5.5.3 Testes de Sistema

Aquando da realização de testes de sistema pretende-se testar uma aplicação como um todo, para se conseguir averiguar se a qualidade geral de um produto de software consegue satisfazer os requisitos inicialmente recolhidos (Reqtest, 2014). Nesta abordagem de teste os desenvolvedores realizam os testes sem saberem a organização interna de código, detalhes de implementação e caminhos internos, ou seja, o sistema é visto como uma caixa preta (Paspelava, 2021).

Para se conseguir alcançar esta abordagem de teste a ferramenta Postman foi utilizada, aqui foi possível a definição de coleções organizadas por diferentes pacotes, observáveis na figura 55.

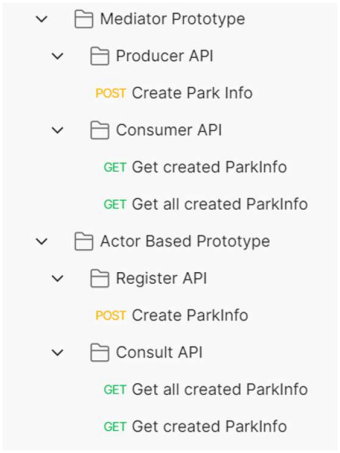



Figura 55 - Organização dos testes em coleções e pacotes

No excerto de código 12 apresentam-se os testes efetuados ao pedido *POST* do serviço produtor referente ao protótipo mediador, salienta-se que estes são realizados com recurso à linguagem Javascript. Aqui verifica-se o código de resposta, o tempo de resposta e o conteúdo do retorno da informação referente a estacionamentos.

```
pm.test("Status code is 200", function () {
    pm.response.to.have.status(200);
});
pm.test("Response time is less than 100ms", function () {
    pm.expect(pm.response.responseTime).to.be.below(100);
});
pm.test("Assert park info data", function () {
    var jsonData = pm.response.json();
    pm.expect(jsonData.Estacionamento).to.eql("Spot 56, Avenida da
        Liberdade, Porto, Porto");
    pm.expect(jsonData.TempoRecolha).to.eql("16/11/2022 03:52:11");
    pm.expect(jsonData.Estado).to.eql(true);
});
```

Código 12 - Teste de sistema ao serviço produtor do protótipo mediador

A figura 56 relata o resultado da execução automática dos testes efetuados ao protótipo mediador.

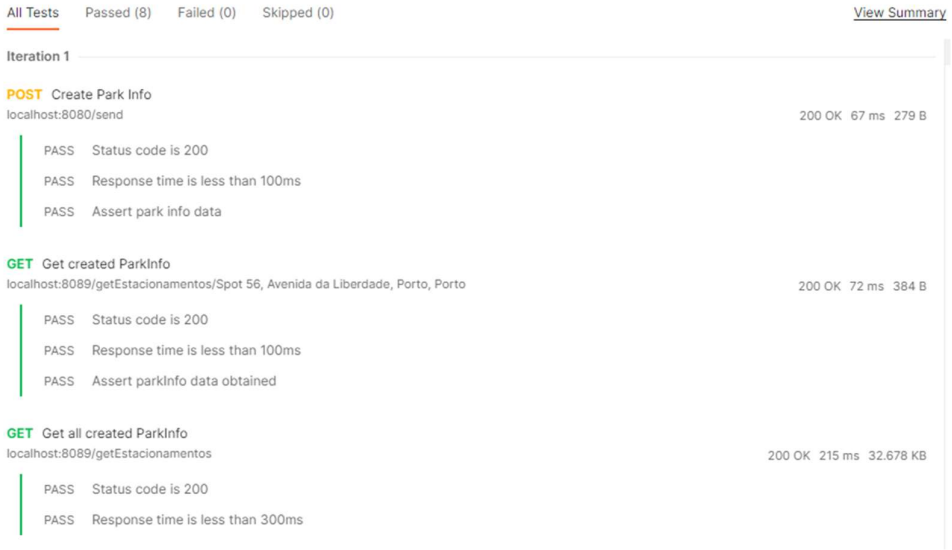



Figura 56 - Execução dos testes de sistema referentes ao protótipo mediador

### 5.5.4 Testes de Aceitação

Realizam-se testes de aceitação com o objetivo de avaliar se o sistema consegue satisfazer os requisitos de negócio e também para se perceber se este está preparado para o processo de implantação (Reqtest, 2014). Existem métodos variados para se realizar este tipo de testes, sendo a realização de cenários escritos uma abordagem plausível de concretização (Reqtest, 2014).

Os testes de aceitação realizados permitiram testar a efetividade da comunicação assíncrona, a implementação dos mecanismos de tolerância a falhas e a construção do rastreamento presente ao longo da execução dos diferentes protótipos.

O primeiro teste realizado ao protótipo mediador, presente na tabela 27, permitiu testar a comunicação assíncrona presente entre os diferentes serviços, testar o rastreamento existente no protótipo e acompanhar o estado do *circuit breaker* existente.

Tabela 27 – Teste de pedido de registo de informação de sensores realizado com sucesso (protótipo mediador)

| **Etapa** | **Descrição** |
|---|---|
| Cenário | Através do serviço *ProducerAPI* criar um pedido de registo de informação de estacionamento proveniente de um novo sensor de estacionamento. O pedido possui informação sobre o identificador, data e hora de recolha e estado de ocupação de um novo lugar de estacionamento. |
| Critério | A ferramenta de monitorização Jaeger regista o pedido *HTTP POST* efetuado ao serviço *ProducerAPI*. |
| | O serviço *MediatorService* recebe uma mensagem proveniente da fila *MediatorQueue*, colocada nesta pelo serviço *ProducerAPI*. |
| | O serviço *ConsumerAPI* recebe uma mensagem proveniente da fila *ParkQueue*, colocada nesta pelo serviço *MediatorService*. |
| | O mecanismo de *circuit breaker* mantém-se no estado fechado durante a execução do cenário exposto. |
| | A ferramenta de monitorização Grafana apresenta as métricas recolhidas durante a execução do cenário exposto. |

O segundo teste realizado ao protótipo mediador, presente na tabela 28, permitiu testar em detalhe a transição de estados referente ao *circuit breaker* do protótipo em questão.

Tabela 28 - Teste de pedido de registo de informação de sensores com o broker de mensagens indisponível (protótipo mediador)

| **Etapa** | **Descrição** |
|---|---|
| Cenário | Através do serviço *ProducerAPI* criar um pedido de registo de informação de estacionamento proveniente de um sensor de estacionamento, após suspender a execução do *broker* de mensagens ActiveMQ. Depois, realizar duas vezes o mesmo pedido, mantendo o *broker* de mensagens indisponível. Por fim, ativar a execução do *broker* de mensagens e enviar o mesmo pedido, após passarem 30 segundos. |
| Critério | Atingida a taxa de 50% referente ao rácio de pedidos falhados, o *circuit breaker* transita para o estado aberto. |
| | A mensagem que contém informação de estacionamento não é disponibilizada na fila *MediatorQueue*. |

| | |
|---|---|
| | Após 5 segundos o *circuit breaker* transita de estado aberto para o estado meio aberto. |
| | Antes de passarem os 30 segundos e de se realizar duas vezes o mesmo pedido, o *circuit breaker* permanece no estado meio aberto. |
| | Após se alcançar os 30 segundos, o mesmo pedido é realizado e o *circuit breaker* transita para o estado fechado. |
| | A mensagem que possui informação de estacionamento é disponibilizada na fila *MediatorQueue*. |
| | A ferramenta de monitorização Grafana apresenta as métricas referentes aos diversos estados que o *circuit breaker* transitou durante a execução do cenário em questão. |

O primeiro teste realizado ao protótipo baseado no modelo ator, presente na tabela 29, permitiu testar a comunicação assíncrona presente entre os diferentes atores, testar o rastreamento existente no protótipo e acompanhar o estado da estratégia de supervisão existente.

Tabela 29 – Teste de pedido de registo de informação de sensores realizado com sucesso (protótipo atores)

| **Etapa** | **Descrição** |
|---|---|
| Cenário | Através do serviço *RegisterAPI* criar um pedido de registo de informação de estacionamento proveniente de um sensor de estacionamento. O pedido possui informação sobre o identificador, data e hora de recolha e estado de ocupação de um determinado lugar de estacionamento. |
| Critério | A ferramenta de monitorização Jaeger regista o pedido *HTTP POST* efetuado ao serviço *RegisterAPI*. |
| | O ator supervisor *ServiceActor* é criado juntamente com os atores supervisionados *EventSourcingActor* e *BusinessRulesActor*. |
| | Os atores realizam o processamento correspondente, não necessitando de recorrer das estratégias de supervisão para deteção e tratamento de falhas, e são encerrados. |
| | A ferramenta de monitorização Grafana apresenta as métricas recolhidas durante a execução do cenário exposto. |

O segundo teste realizado ao protótipo baseado no modelo ator, presente na tabela 30, permitiu testar em detalhe a invocação da estratégia de supervisão presente neste protótipo.

Tabela 30 - Teste de pedido de registo incompleto de informação de sensores (protótipo atores)

| **Etapa** | **Descrição** |
|---|---|
| Cenário | Através do serviço *RegisterAPI* criar um pedido de registo de informação de estacionamento proveniente de um sensor de estacionamento. O pedido possui informação de data e hora de recolha e estado de ocupação referentes a um lugar de estacionamento, mas não possui informação identificadora (identificador) do respetivo lugar de estacionamento. |
| Critério | A ferramenta de monitorização Jaeger regista o pedido *HTTP POST* efetuado ao serviço *RegisterAPI*. |
| | O ator supervisor *ServiceActor* é criado juntamente com os atores supervisionados *EventSourcingActor* e *BusinessRulesActor*. |
| | A estratégia de supervisão, presente no ator *ServiceActor*, é acionada. Esta estratégia mantém o estado atual dos atores filhos que continuam a processar eventuais novas mensagens. |
| | Os atores resumem o processamento correspondente e são encerrados. |
| | A ferramenta de monitorização Grafana apresenta as métricas recolhidas durante a execução do cenário exposto. |

## 5.6 Sumário

Terminado o processo de implementação da solução conclui-se que os protótipos baseados na topologia mediadora e no modelo ator conseguiram implementar todos os requisitos funcionais e não funcionais identificados no processo de engenharia de requisitos. Salienta-se que os requisitos de desempenho foram alcançados através da adoção de *CQRS* e *event sourcing*. Os requisitos de disponibilidade através da adoção de *circuit breakers* e de estratégias de supervisão. No que diz respeito à escalabilidade a adoção de diferentes mecanismos de comunicação assíncrona, desde *brokers* de mensagens até atores assíncronos, permitiu endereçar o atributo de qualidade em questão. Por fim, a adoção de ferramentas de monitorização de sistemas permitiu contemplar os requisitos de monitorabilidade.

# 6 Experimentação e Avaliação

Este capítulo pretende descrever a abordagem de experimentação e avaliação aplicada nos diferentes protótipos. Previamente, na secção 2.2.4 foram identificadas diferentes métricas e após o processo de análise de requisitos identificaram-se desempenho, escalabilidade, disponibilidade e monitorabilidade como atributos de qualidade fundamentais. Para se avaliar a concretização destes atributos de qualidade nos protótipos a abordagem *Goals, Questions, Metrics* (*GQM*) foi adotada.

## 6.1 Goals, Questions, Metrics

A abordagem *GQM* é uma técnica orientada a metas, utilizada na Engenharia de Software, para se conseguir medir produtos de software de uma forma objetiva. Primeiramente o objetivo é delineado, depois este é refinado através da formulação de questões, e por fim as métricas são definidas devendo providenciar informação para se responder às questões (Solingen & Berghout, 1999). Sumariza-se que a abordagem *GQM* define as métricas numa perspetiva *top-down*, por outro lado analisa e interpreta os dados medidos através de uma perspetiva *bottom-up*, como se observa na figura 57.

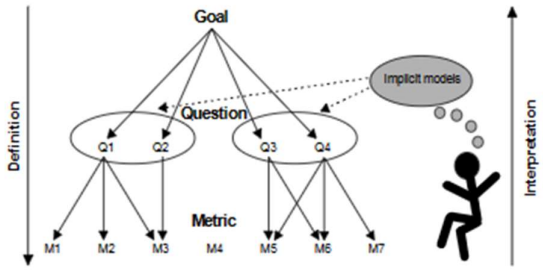



Figura 57 - Abordagem GQM - imagem de (Solingen & Berghout, 1999)

A aplicação da abordagem, evidenciada na tabela 31, onde primeiramente foi definido o objetivo a atingir, depois a coluna central da tabela expõe as questões a responder, e termina-se com a exposição das métricas, na última coluna da mesma, a aplicar a cada protótipo.

Tabela 31 - Aplicação de GQM

| **Objetivo** | **Questões** | **Métricas** |
|---|---|---|
| Explorar a harmonização do desempenho, tolerância a falhas, escalabilidade e monitorabilidade em aplicações com microsserviços orientados a eventos com o intuito de se encontrar a solução mais adequada. | Qual o efeito que uma solução baseada na topologia mediadora comparativamente a uma solução baseada no modelo ator tem no desempenho? | Crescimento da fila de mensagens |
| | | Utilização de memória RAM |
| | | Utilização de CPU |
| | | Tempo de resposta |
| | | Taxa de transferência |
| | Qual o efeito que uma solução baseada na topologia mediadora comparativamente a uma solução baseada no modelo ator tem na tolerância a falhas? | Número de pedidos concluídos com sucesso |
| | | Tráfego em rede |
| | | Deteção de falhas |
| | | Latência |
| | Qual o efeito que uma solução baseada na topologia mediadora comparativamente a uma solução baseada no modelo ator tem na escalabilidade? | Número de dependências síncronas de cada microsserviço |
| | | Frequência de uso de cada microsserviço |
| | | Escalonamento horizontal / vertical |
| | Qual o efeito que uma solução baseada na topologia mediadora comparativamente a uma solução baseada no modelo ator tem na monitorabilidade? | Geração de dados monitorizados |
| | | Armazenamento de dados monitorizados |
| | | Apresentação de dados monitorizados |

Com o intuito de se conseguir avaliar os protótipos concebidos e implementados foram recolhidos valores de métricas referentes ao protótipo inicial, protótipo este que simboliza as aplicações típicas da empresa, presentes na secção 6.3. Assim, tornou-se possível a definição de uma escala de avaliação que irá permitir classificar cada protótipo de acordo com os valores

das métricas que apresentar para cada atributo de qualidade, presente na secção 6.3. Em 6.4 e 6.5 reporta-se os valores das métricas recolhidos, durante o processo de experimentação e avaliação, referentes aos protótipos baseados na topologia mediador e no modelo ator, respetivamente.

Entende-se que é possível obter harmonização se os valores das métricas, para cada atributo de qualidade, alcançarem pelo menos o parâmetro suficiente na escala de avaliação. Avaliação esta sintetizada em 6.8. Sendo que cada atributo de qualidade possui diversas métricas, alcançou-se uma fórmula de avaliação a aplicar a cada protótipo, presente na seção 6.9. Assim foi possível a obtenção de um valor representativo de cada atributo de qualidade, para cada protótipo, a considerar: desempenho, escalabilidade, disponibilidade e monitorabilidade.

## 6.2 Obtenção de valores de métricas

Após a definição das métricas que permitem avaliar os protótipos desenvolvidos, torna-se fundamental expor como os valores destas são obtidos. Para tal, a configuração das ferramentas que permitem obter os valores das métricas é exposta, realçando-se as métricas endereçadas por cada ferramenta em particular.

### 6.2.1 Apache Jmeter

Apache Jmeter é uma ferramenta *open-source* construída em Java que permite efetuar testes de carga, conseguindo-se assim medir o desempenho, por exemplo, das aplicações em teste (JMeter, 2011). Inicialmente a ferramenta foi desenvolvida para se testar aplicações Web, contudo, atualmente, já suporta outras funções de teste (JMeter, 2011).

Para se conseguir avaliar os protótipos decidiu-se realizar um teste de carga com 250 sensores com um *loop count* de 3, isto é, cada um dos sensores vai enviar 3 mensagens de estado. A figura 58 apresenta a configuração das propriedades da *thread*.

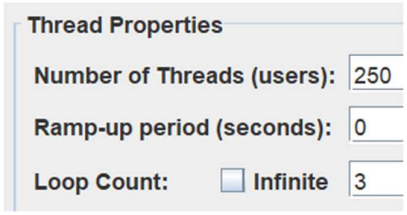



Figura 58 - Configuração das propriedades da thread utilizada no teste de carga

Com o objetivo de se automatizar e diversificar as mensagens de estado, enviadas por cada sensor em teste, um ficheiro *CSV (Comma-separated values)* com informação proveniente de sensores foi construído. Na figura 59 está presente um excerto de uma linha desse ficheiro CSV.

| 1 | testID | method | expectedCode | estacionamento | estado | horario |
|---|---|---|---|---|---|---|
| 2 | Test 1 | POST | 200 | Estacionamento 1238 | TRUE | 15/11/2022 03:52:11 |

Figura 59 - Excerto de uma linha do ficheiro de teste CSV

Após definição e construção do ficheiro *CSV*, utilizado no teste de carga, configurou-se na ferramenta Jmeter o caminho para este ficheiro, o nome das variáveis presentes no ficheiro para a ferramenta considerar os valores das colunas em questão e por fim o delimitador utilizado no ficheiro. Estas configurações estão presentes na figura 60.

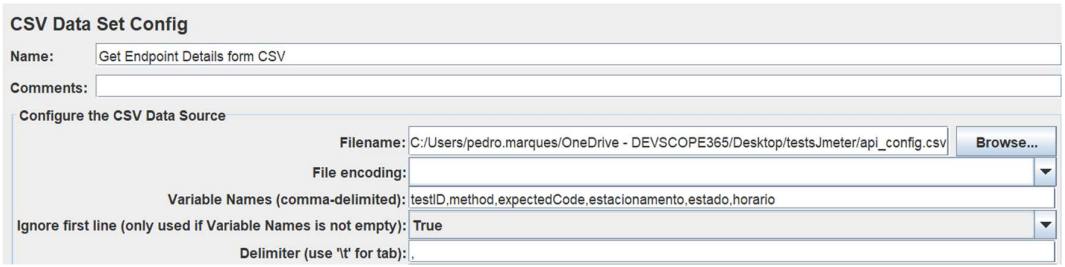



Figura 60 - Configuração do ficheiro CSV utilizado no teste de carga

Depois configurou-se o pedido *HTTP* a realizar por cada *thread*, ou seja, cada sensor. Aqui utiliza-se o valor das variáveis *method*, estacionamento, estado e horário presentes no ficheiro *CSV*. A figura 61 exemplifica a configuração do pedido *HTTP POST* efetuado no teste de carga ao protótipo mediador.

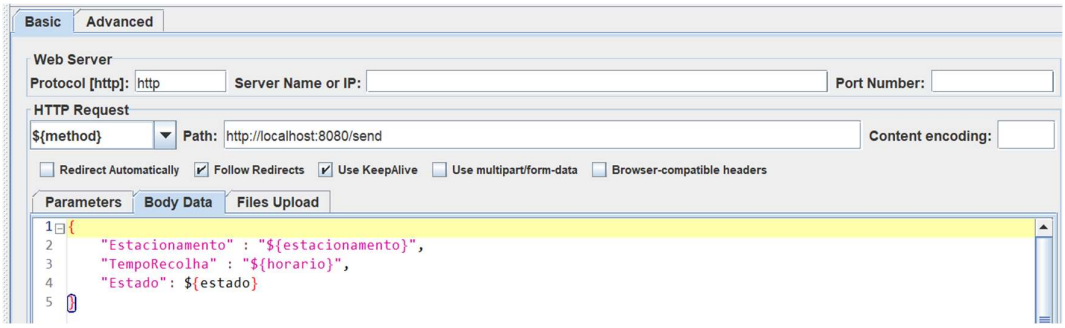



Figura 61 - Configuração do pedido HTTP a realizar ao endpoint do protótipo mediador

Após a execução do teste de carga é possível a recolha de valores de métricas como a **taxa de transferência**, o **tempo de resposta por pedido**, o **crescimento da fila de mensagens**, o **número de pedidos efetuados com sucesso**, a **latência** e o **tráfego enviado e recebido em rede**. Valores estes disponibilizados sobre a forma de relatório na ferramenta Jmeter, a figura 62 agrega os diferentes relatórios utilizados.

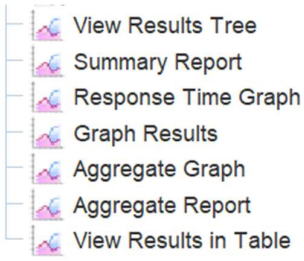



Figura 62 - Agregação dos diferentes relatórios que possuem os valores das métricas recolhidos

### 6.2.2 OpenTracing

Open Tracing é uma *API* independente de fornecedor que permite aos desenvolvedores integrarem rastreamento distribuído no código base das suas aplicações. Rastreamento distribuído é um método usado, especialmente em sistema de microsserviços, para se caracterizar, monitorizar e depurar aplicações (OpenTracing, 2019b). Empresas produtoras de sistemas de rastreamento suportam Open Tracing como uma abordagem padrão de se instrumentar rastreamento distribuído (SentinelOne, 2021).

*Trace* em Open Tracing, visto como um gráfico acíclico dirigido, é definido implicitamente pelos seus *spans* (OpenTracing, 2019a). Um *span* agrega o seguinte estado (OpenTracing, 2019a):

- Nome de operação;
- Tempo de início e de fim;
- Conjunto de zero ou mais *Span Tags* do tipo chave-valor, onde as chaves devem de ser alfanuméricas e os valores podem ser alfanuméricos, booleanos ou tipos numéricos;
- Conjunto de zero ou mais *Span Logs* do tipo chave-valor com um carimbo de tempo associado, onde as chaves devem de ser alfanuméricas e os valores podem ser de qualquer tipo;
- *SpanContext* que agrega qualquer estado dependente da implementação do Open Tracing e também itens de bagagem;
- Referências para zero ou mais spans relacionados;

Para se conseguir utilizar as potencialidades disponibilizadas pelo Open Tracing a ferramenta Jaeger foi adotada, permitindo assim a monitorização dos protótipos desenvolvidos.

### 6.2.3 Jaeger

Jaeger é um sistema de rastreamento distribuído, *open-source*, desenvolvido pela Uber Technologies. A ferramenta permite analisar propagação de contexto distribuído, monitorizar transações distribuídas e realizar análise de causa raiz e de dependência de serviço (Jaeger, 2023b).

A figura 63 sintetiza a arquitetura do Jaeger, onde o Jaeger Collector recebe dados provenientes de aplicações monitorizadas e persiste estes mesmos dados diretamente na sua base de dados (Jaeger, 2023a). O componente Jaeger Collector utiliza uma fila em memória que permite dar resposta a picos de tráfego de curto prazo (Jaeger, 2023a).

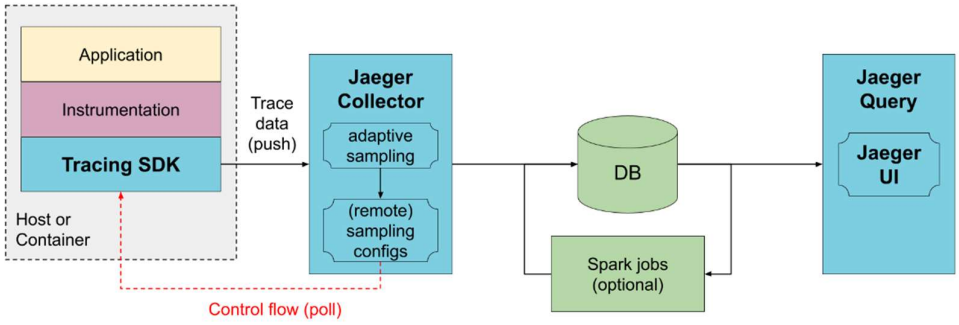



Figura 63 - Arquitetura da ferramenta Jaeger – imagem de (Jaeger, 2023a)

O excerto de código 13 evidencia a configuração necessária para se fazer uso da *API* de Open Tracing juntamente com a ferramenta Jaeger, nos protótipos implementados em C#. Após se adicionar a instrumentação Open Tracing, presente na linha 81 do excerto de código 13, a configuração do *tracer* é realizada, presente nas linhas 84 até 94 do mesmo excerto de código, permitindo assim adicionar futuramente ao código base da aplicação a criação e a propagação de *spans*. Destaca-se a linha 86 do excerto de código 13, onde é definido o protocolo *UDP* como o protocolo de comunicação entre a aplicação monitorizada e o servidor Jaeger.

```
80      //Open Tracing configuration for Jaeger
81      services.AddOpenTracing();
82      services.AddSingleton<ITracer>(sp =>
83      {
84          var serviceName = sp.GetRequiredService<IWebHostEnvironment>().ApplicationName;
85          var loggerFactory = sp.GetRequiredService<ILoggerFactory>();
86          var reporter = new RemoteReporter.Builder().WithLoggerFactory(loggerFactory).WithSender(new UdpSender())
87              .Build();
88          var tracer = new Tracer.Builder(serviceName)
89              // The constant sampler reports every span.
90              .WithSampler(new ConstSampler(true))
91              // LoggingReporter prints every reported span to the logging framework.
92              .WithReporter(reporter)
93              .Build();
94          return tracer;
95      });
```

Código 13 - Configuração Jaeger Open Tracing em C#

No excerto de código 14 encontra-se um exemplo de criação de um *span* responsável pela monitorização de um serviço, que realiza a gestão de envio de mensagens assíncronas.

```
33                  //Open Tracing scope setup
34                  using var scope = _tracer.BuildSpan("Envio RabbitMQ").StartActive(true);
35                  //Rabbit MQ send Messages
36                  RabbitMQClient client = new RabbitMQClient(_configuration, _connection);
37                  client.SendMessage(parkInfo);
38                  //Scope Termination
39                  scope.Span.Finish();
```

Código 14 - Utilização de Open Tracing em C# .NET

Os excertos de código 15 e 16 revelam a configuração necessária em Java para se conseguir fazer uso das funcionalidades de rastreamento disponibilizadas pelo Jaeger, com recurso a Open Tracing.

```
29    //Created for OpenTracing with Jaeger
30    @Bean
31    public JaegerTracer jaegerTracer() {
32        return new io.jaegertracing.Configuration("producerAPI")
33                .withSampler(new io.jaegertracing.Configuration.SamplerConfiguration().withType(ConstSampler.TYPE)
34                .withParam(1))
35                .withReporter(new io.jaegertracing.Configuration.ReporterConfiguration().withLogSpans(true))
36                .getTracer();
37    }
```

Código 15 - Configuração Jaeger Open Tracing em Java Spring Boot

```
6 opentracing.jaeger.http-sender.url=http://localhost:14268/api/traces
```

Código 16 - Excerto do ficheiro application.properties com a configuração Jaeger Open Tracing em Java Spring Boot

Através da adoção da ferramenta Jaeger foi possível a obtenção de valores de métricas de **frequência de uso** e **número de dependências síncronas de cada microsserviço**.

### 6.2.4 Prometheus

Prometheus disponibiliza um conjunto de mecanismos, *open-source*, de monitorização e de alertas. Esta ferramenta permite a recolha e o armazenamento de métricas, sendo que cada valor armazenado possui informação de data/hora de recolha juntamente com o nome da métrica e pares chave-valor intitulados de rótulos (Prometheus, 2023).

Na figura 64 apresenta-se a arquitetura do Prometheus dividida pelos seus principais componentes. Destaca-se a facilidade de recolha de informação através do método *pull* do protocolo *HTTP* e a possibilidade de descoberta de alvos de uma forma estática ou dinâmica.

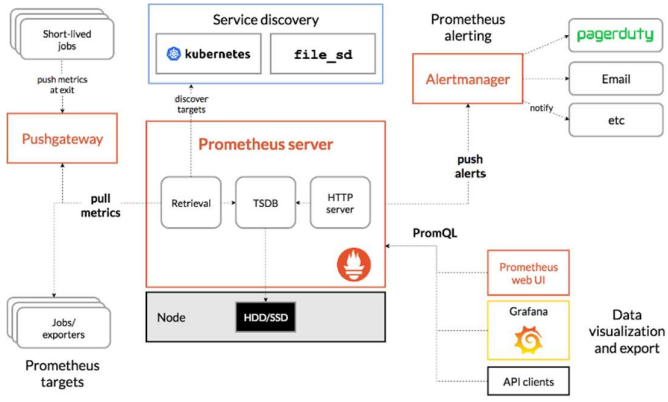



Figura 64 - Arquitetura da ferramenta Prometheus – imagem de (Prometheus, 2023)

No excerto de código 17 apresenta-se a configuração de um alvo, ou seja, um *job*, relativamente ao serviço produtor do protótipo mediador. Através da definição de diferentes alvos estáticos,

o servidor Prometheus recolhe a informação das métricas disponibilizadas no endereço, denominado de *metrics_path*, definido para cada alvo.

```
- job_name: "producerAPI-mediator"
  metrics_path: /actuator/prometheus
  static_configs:
    - targets: ["localhost:8080"]
      labels:
        application: "ProducerAPI Mediator"
```

Código 17 - Excerto de configuração do servidor Prometheus

O excerto de código 18 evidencia as configurações necessárias, referentes aos protótipos que utilizam a *framework* .NET, para se conseguir expor *endpoints* de apresentação de métricas recolhidas.

```
14  public static IHostBuilder CreateHostBuilder(string[] args) =>
15      Host.CreateDefaultBuilder(args)
16      //Metrics configuration
17          .UseMetricsWebTracking()
18          .UseMetrics(options =>
19          {
20              options.EndpointOptions = endpointsOptions =>
21              {
22                  endpointsOptions.MetricsTextEndpointOutputFormatter = new MetricsPrometheusTextOutputFormatter();
23                  endpointsOptions.MetricsEndpointOutputFormatter = new MetricsPrometheusProtobufOutputFormatter();
24                  endpointsOptions.EnvironmentInfoEndpointEnabled = false;
25              };
26          })
```

Código 18 - Configuração de endpoint em .NET a ser usado pelo Prometheus

O excerto de código 19 apresenta um excerto do ficheiro applications.properties referente à configuração necessária, do protótipo que utiliza a *framework* Spring Boot, para conseguir expor um *endpoint* de apresentação de métricas recolhidas.

```
7 management.endpoints.web.exposure.include=prometheus
```

Código 19 - Excerto do ficheiro application.properties com a configuração de endpoint em Spring Boot a ser usado pelo Prometheus

A correta configuração dos *endpoints* permitiu o registo de valores de métricas como a **utilização dos recursos de CPU** e de **memória RAM**. Complementarmente, a ferramenta permitiu rever certos valores de métricas já identificados e recolhidos pela ferramenta Jmeter. Por outro lado, identificaram-se diversas outras métricas, mas não se revelou interessante a sua exploração para o processo de avaliação e experimentação exposto ao longo deste capítulo.

### 6.2.5 Grafana

A ferramenta Grafana é uma solução *open-source*, que permite de uma forma interativa visualizar dados referentes a métricas recolhidos por outras ferramentas, como por exemplo o Prometheus (Shivang, 2021). Grafana permite auxiliar no estudo, na análise e na monitorização de dados em períodos configuráveis. Assim, torna-se possível a perceção do comportamento

do utilizador e da aplicação de uma forma visual, com recurso a gráficos, ao longo do tempo (Shivang, 2021).

Na tabela 32 agregam-se os *dashboards* utilizados na ferramenta Grafana, com o intuito de se conseguir visualizar de uma forma gráfica as métricas recolhidas em todos os protótipos.

Tabela 32 - Dashboards Grafana utilizados

| **Nome do dashboard** | **Descrição** | **Código/ Referência** |
|---|---|---|
| App Metrics – Web Monitoring – Prometheus | Permite a visualização de dados de métricas capturados pela *framework* AppMetrics usada nos protótipos implementados em C#. | 2204 |
| Resilience4j | Permite a visualização do estado de um sistema que implemente o mecanismo de *circuit breaker*. | (Karlsen, 2021) |
| Spring Boot APM Dashboard | Permite a visualização de dados de métricas capturados no protótipo implementado em Java. | 12900 |
| Windows Node | Permite a visualização de dados de métricas, referentes ao desempenho de uma máquina em particular, recolhidas pelo serviço wmi_exporter. | 2129 |

Na figura 65 mostra-se parte do *dashboard* que permite visualizar algumas das métricas recolhidas ao longo da execução do protótipo baseado em atores. Na interface superior deste *dashboard* é possível a escolha do serviço que se pretende monitorizar, mais concretamente no campo *application*, bem como a escolha da fonte de informação de dados monitorizados, no campo *datasource*.


Figura 65 - Dashboard de monitorização de certas métricas

Esta ferramenta permitiu a **apresentação gráfica dos dados gerados e armazenados** pela ferramenta Prometheus.

## 6.3 Protótipo baseado na topologia broker (protótipo inicial)

O teste de carga realizado com 250 sensores com um *loop count* de 3 e a correta configuração das ferramentas de monitorização, presentes na secção 6.2, permitiram a obtenção de valores para as métricas dos atributos de qualidade desempenho, escalabilidade, disponibilidade e monitorabilidade referentes ao protótipo simbólico das aplicações típicas da empresa. Adicionalmente a escala de avaliação foi construída consoante os valores das métricas que este protótipo apresentou.

### 6.3.1 Desempenho

O teste de carga foi executado em 6 segundos, obtendo-se o valor de 0% representativo da percentagem de erros ocorridos durante o teste em questão, permitindo a recolha de métricas como o **crescimento da fila de mensagens**, a **taxa de transferência** e o **tempo de resposta por pedido**. A ferramenta Grafana permitiu a visualização de métricas como a **utilização de CPU** e a **utilização de memória RAM** aquando do respetivo teste de carga efetuado. A tabela 33 sumariza os valores das métricas obtidos após a realização do teste de carga ao protótipo inicial.

Tabela 33 - Valores das métricas de desempenho para o protótipo inicial

| **Métrica** | **Valor** | **Valor Inicial** | **Valor Final** |
|---|---|---|---|
| Utilização de CPU | Variação de Idle: 0,61 | Idle: 6,20 | Idle: 6,81 |
| Utilização de memória RAM | Variação: 100MB | Memória livre: 8,8GB | Memória livre: 8,7GB |
| Taxa de transferência | 173 pedidos por segundo | - | - |
| Tempo de resposta médio por pedido | 1,2 segundos | - | - |
| Crescimento da fila de mensagens | Variação: 750 mensagens | 0 mensagens | 750 mensagens |

A tabela 34 revela a escala de avaliação a ser usada para se medir o desempenho dos protótipos.

Tabela 34 - Escala de avaliação do desempenho de um sistema

| **Métrica/Escala** | **Insuficiente** | **Suficiente** | **Bom** |
|---|---|---|---|
| Utilização de CPU | Variação de Idle superior a 1 | Valor de Idle entre 0,50 e 1 | Valor de Idle inferior a 0,50 |
| Utilização de memória RAM | Superior a 250MB | Entre 100MB e 250MB | Inferior a 100MB |
| Taxa de transferência | Inferior a 150 pedidos por segundo | Entre 150 e 200 pedidos por segundo | Superior a 200 pedidos por segundo |
| Tempo de resposta médio por pedido | Superior a 1,5 segundos | Entre 1 e 1,5 segundos | Inferior a 1 segundo |
| Crescimento da fila de mensagens | Inferior a 750 mensagens | 750 mensagens | - |

### 6.3.2 Escalabilidade

Relativamente ao atributo de qualidade escalabilidade a ferramenta Jaeger permitiu endereçar as métricas relevantes relativamente a este atributo.

Começando com a métrica de **frequência de uso**, a tabela 35 enumera o número de pedidos e o número de invocações a outros microsserviços relativamente a cada microsserviço do protótipo representativo das aplicações típicas da empresa.

Tabela 35 - Frequência de uso de cada microsserviço para o protótipo inicial

| **Microsserviço** | **Número de pedidos** | **Número de invocações a outros microsserviços** |
|---|---|---|
| Producer Service | 1 | 0 |
| Consumer Service | 2 | 0 |
| Total | 3 | 0 |

Concluiu-se que o rácio entre o número de pedidos efetuados ao sistema e o total de pedidos, síncronos e assíncronos, efetuados a todo o sistema é de 60%, realçando-se que a comunicação entre microsserviços é efetuada por mecanismos assíncronos.

Seguidamente endereça-se a métrica de **número de pedidos síncronos**, a tabela 36 enumera o número de operações e o número de operações síncronas de cada microsserviço do protótipo inicial.

Tabela 36 - Número de operações de cada microsserviço para o protótipo inicial

| **Microsserviço** | **Número de operações** | **Número de operações síncronas** |
|---|---|---|
| Producer Service | 2 | 1 |
| Consumer Service | 3 | 2 |
| Total | 5 | 3 |

Os resultados reforçam que o rácio entre o número de pedidos síncronos e o número total de pedidos é de 60% para o protótipo inicial. O valor é efetivamente elevado, mas realça-se que somente os pedidos de leitura e um pedido de escrita é que são efetuados sincronamente.

Avançando para o **escalonamento horizontal e vertical** é necessário garantir que o funcionamento do sistema não seja comprometido apesar do tamanho deste sofrer alterações. A possibilidade de adoção de *event sourcing* permite assegurar tal cenário, pois permite o reprocessamento de eventos, conseguindo-se alcançar um estado uniforme entre todas as instâncias.

A tabela 37 revela a escala de avaliação a ser usada para se medir a escalabilidade dos protótipos.

Tabela 37 - Escala de avaliação da escalabilidade de um sistema

| **Métrica/Escala** | **Insuficiente** | **Suficiente** | **Bom** |
|---|---|---|---|
| Frequência de uso | Superior a 61% | Entre 51% e 60% | Inferior a 50% |
| Número de pedidos síncronos | Superior a 61% | Entre 51% e 60% | Inferior a 50% |
| Escalonamento horizontal/vertical | Não implementa event sourcing | Implementa event-sourcing | - |

### 6.3.3 Disponibilidade

No que diz respeito ao atributo de qualidade disponibilidade, o mesmo teste de carga efetuado para se avaliar o desempenho de uma solução permitiu a obtenção de métricas de disponibilidade. O valor de **número de pedidos efetuados com sucesso**, de **latência** e de **tráfego em rede** foram obtidos através da ferramenta Jmeter. A tabela 38 agrega os valores das métricas recolhidos no teste de carga efetuado ao protótipo inicial. A métrica de **tolerância a falhas** também foi identificada, sendo possível de alcançar através de implementação de estratégias de supervisão e/ou *circuit breaker*.

Tabela 38 - Valores das métricas de disponibilidade para o protótipo inicial

| **Métrica** | **Valor** |
|---|---|
| Número de pedidos efetuados com sucesso | 750 |
| Latência máxima | 5 segundos |
| Tráfego enviado em rede | 37,39Kb |
| Tráfego recebido em rede | 32,32Kb |

A tabela 39 revela a escala de avaliação a ser usada para se medir a disponibilidade dos protótipos.

Tabela 39 - Escala de avaliação da disponibilidade de um sistema

| **Métrica/Escala** | **Insuficiente** | **Suficiente** | **Bom** |
|---|---|---|---|
| Número de pedidos efetuados | Inferior a 750 pedidos | 750 pedidos | - |
| Latência máxima | Superior a 5 segundos | Entre 4 e 5 segundos | Inferior a 4 segundos |
| Tráfego enviado em rede | Superior a 50Kb | Entre 30Kb e 50Kb | Inferior a 30Kb |
| Tráfego recebido em rede | Superior a 50Kb | Entre 30Kb e 50Kb | Inferior a 30Kb |
| Tolerância a falhas | Não implementa *circuit breaker* nem estratégia de supervisão | Implementa *circuit breaker ou estratégia de supervisão* | Implementa *circuit breaker* e estratégia de supervisão |

### 6.3.4 Monitorabilidade

Para se conseguir garantir a monitorabilidade de um sistema é necessário realizar a **geração de dados**, com o intuito de se obter *logs* e métricas referentes ao sistema em análise. Depois é necessário o **armazenamento de dados** para que possam ser consultados posteriormente. Estas duas etapas são alcançadas através da adoção da ferramenta Prometheus. Por fim, relativamente à **apresentação de dados**, as ferramentas Grafana e Jaeger permitem o consumo e o processamento dos dados armazenados sendo possível a monitorização, através de variados monitores e vistas, de um sistema. Na tabela 40 apresenta-se os valores das métricas de monitorabilidade para o protótipo inicial.

Tabela 40 - Valores das métricas de monitorabilidade para o protótipo inicial

| **Métrica** | **Valor** |
|---|---|
| Geração de dados monitorizados | Possui |
| Armazenamento de dados monitorizados | Possui |
| Apresentação de dados monitorizados | Possui |

A tabela 41 revela a escala de avaliação a ser usada para medir a monitorabilidade dos protótipos.

Tabela 41 - Escala de avaliação da monitorabilidade de um sistema

| **Métrica/Escala** | **Insuficiente** | **Suficiente** |
|---|---|---|
| Geração de dados monitorizados | Não possui | Possui |
| Armazenamento de dados monitorizados | Não possui | Possui |
| Apresentação de dados monitorizados | Não possui | Possui |

## 6.4 Protótipo baseado na topologia mediadora

O teste de carga realizado com 250 sensores com um *loop count* de 3 e a correta configuração das ferramentas de monitorização, presentes na secção 6.2, permitiram a obtenção de valores para as métricas dos atributos de qualidade desempenho, escalabilidade, disponibilidade e monitorabilidade referentes ao protótipo baseado na topologia mediadora.

### 6.4.1 Desempenho

O teste de carga foi efetuado em 5 segundos, obtendo-se o valor de 0% representativo da percentagem de erros ocorridos durante o teste em questão, consequentemente os valores das métricas de desempenho registados encontram-se presentes na tabela 42.

Tabela 42 - Valores das métricas de desempenho para o protótipo mediador

| **Métrica** | **Valor** | **Valor Inicial** | **Valor Final** |
|---|---|---|---|
| Utilização de CPU | Variação de Idle: 0,11 | Idle: 7,11 | Idle: 7,22 |
| Utilização de memória RAM | Variação: 20MB | Memória livre: 9,01GB | Memória livre: 8,81GB |

| | | | |
|---|---|---|---|
| Taxa de transferência | 194 pedidos por segundo | - | - |
| Tempo de resposta médio por pedido | 1,027 segundos | - | - |
| Crescimento da fila de mensagens | Variação: 750 mensagens | 0 mensagens | 750 mensagens |

### 6.4.2 Escalabilidade

O número de pedidos e o número de invocações a outros microsserviços relativamente a cada microsserviço encontram-se representados na tabela 43.

Tabela 43 - Frequência de uso de cada microsserviço para o protótipo mediador

| **Microsserviço** | **Número de pedidos** | **Número de invocações a outros microsserviços** |
|---|---|---|
| Producer Service | 1 | 0 |
| Mediator Service | 0 | 0 |
| Consumer Service | 2 | 0 |
| Total | 3 | 0 |

Concluiu-se que o rácio entre o número de pedidos efetuados ao sistema e o total de pedidos efetuados a todo o sistema é de 42,8%, permitindo assim endereçar a métrica de frequência de uso, destaca-se que a comunicação entre microsserviços é efetuada por mecanismos de comunicação assíncronos.

A tabela 44 evidencia o número de operações e o número de operações síncronas referentes a cada microsserviço. O rácio entre o número de pedidos síncronos e o número total de pedidos é de 42,8%. Concluindo-se que a métrica de número de pedidos síncronos contém um número inferior face ao número de pedidos assíncronos existentes neste protótipo.

Tabela 44 - Número de operações de cada microsserviço para o protótipo mediador

| **Microsserviço** | **Número de operações** | **Número de operações síncronas** |
|---|---|---|
| Producer Service | 2 | 1 |
| Mediator Service | 2 | 0 |

| | | |
|---|---|---|
| Consumer Service | 3 | 2 |
| Total | 7 | 3 |

Relativamente à métrica de escalonamento horizontal/vertical o padrão *event sourcing* foi aplicado neste protótipo. Assim garante-se que o funcionamento do sistema não é prejudicado apesar do tamanho deste poder vir a sofrer alterações, conseguindo-se alcançar um estado uniforme entre todas as instâncias. Complementarmente a possibilidade de existência de múltiplos serviços consumidores para a mesma fila de mensagens, através de consumo concorrente, é algo que reforça a escalabilidade da solução (ActiveMQ, 2019b).

### 6.4.3 Disponibilidade

O mesmo teste de carga realizado para se avaliar o desempenho permitiu também a recolha de valores referentes às métricas de disponibilidade, presentes na tabela 45. Relativamente à métrica de tolerância a falhas o mecanismo *circuit breaker* foi implementado. Mecanismo este, explicado em 5.2.3, onde de uma forma automática existe a deteção de falhas referentes à comunicação entre microsserviços, despoletando o mecanismo em questão para o modo aberto, resultando numa proteção adicional ao sistema.

Tabela 45 - Valores das métricas de disponibilidade para o protótipo mediador

| **Métrica** | **Valor** |
|---|---|
| Número de pedidos efetuados com sucesso | 750 |
| Latência máxima | 2,480 segundos |
| Tráfego enviado em rede | 41,34Kb |
| Tráfego recebido em rede | 37,63Kb |

### 6.4.4 Monitorabilidade

Por fim endereça-se o atributo de qualidade monitorabilidade. Através do uso das ferramentas de monitorização Prometheus, Grafana e Jaeger foi possível encaminhar as métricas de criação, armazenamento e apresentação de dados monitorizados, presentes na tabela 46.

Tabela 46 - Valores das métricas de monitorabilidade para o protótipo mediador

| Métrica | Valor |
|---|---|
| Geração de dados monitorizados | Possui |
| Armazenamento de dados monitorizados | Possui |
| Apresentação de dados monitorizados | Possui |

## 6.5 Protótipo baseado no modelo ator

O mesmo processo de avaliação e experimentação evidenciado nos protótipos anteriores, protótipo baseado na topologia *broker* e protótipo baseado na topologia mediadora, foi aplicado ao protótipo baseado no modelo ator. Nas secções 6.5.1, 6.5.2, 6.5.3 e 6.5.4 apresentam-se os valores das métricas recolhidos para os atributos de qualidade desempenho, escalabilidade, disponibilidade e monitorabilidade, respetivamente, referentes ao protótipo em questão.

### 6.5.1 Desempenho

O teste de carga foi concluído em 4 segundos, obtendo-se o valor de 0% representativo da percentagem de erros ocorridos durante o teste em questão, permitindo a recolha de valores de métricas referentes ao atributo de qualidade desempenho, presentes na tabela 47.

Tabela 47 - Valores das métricas de desempenho para o protótipo baseado no modelo ator

| Métrica | Valor | Valor Inicial | Valor Final |
|---|---|---|---|
| Utilização de CPU | Variação de Idle: 0,04 | Idle: 7,26 | Idle: 7,30 |
| Utilização de memória RAM | Variação: 15MB | Memória livre: 9,52GB | Memória livre: 9,37GB |
| Taxa de transferência | 250 pedidos por segundo | - | - |
| Tempo de resposta médio por pedido | 0,526 segundos | - | - |
| Crescimento da fila de mensagens | Variação: 750 mensagens | 0 mensagens | 750 mensagens |

### 6.5.2 Escalabilidade

Na tabela 48 agrega-se o número de pedidos e o número de invocações a outros microsserviços por microsserviço.

Tabela 48 - Frequência de uso de cada microsserviço para o protótipo baseado em atores

| **Microsserviço** | **Número de pedidos** | **Número de invocações a outros microsserviços** |
|---|---|---|
| Register Service | 1 | 0 |
| Consult Service | 2 | 0 |
| Total | 3 | 0 |

Alcança-se um rácio de 42,8% referente ao número de pedidos efetuados ao sistema sobre o número total de pedidos efetuados a todo sistema. Na tabela 49 apresenta-se o número de operações e o número de operações síncronas referentes a cada microsserviço. O rácio entre o número de pedidos síncronos e o número de pedidos é de 42,8%. Reforça-se o elevado número de pedidos assíncronos presentes no microsserviço *Register Service*, providenciados pela utilização das funcionalidades referentes ao modelo ator, através do uso da *framework* Akka.NET.

Tabela 49 - Número de operações de cada microsserviço para o protótipo baseado no modelo ator

| **Microsserviço** | **Número de operações** | **Número de operações síncronas** |
|---|---|---|
| Register Service | 5 | 1 |
| Consult Service | 2 | 2 |
| Total | 7 | 3 |

Sobre a métrica de escalonamento horizontal/vertical, este protótipo implementa o padrão *event sourcing*. Consequentemente o funcionamento do sistema não é comprometido com as possíveis variações de tamanho do mesmo, sendo possível de se alcançar um estado uniforme entre todas as instâncias. Outro aspeto que reforça a escalabilidade desta solução é a presença de atores assíncronos, que de uma forma isolada aplicam a lógica correspondente, recorrendo de uma quantia residual de recursos de hardware como se concluiu na secção de desempenho 6.5.1.

### 6.5.3 Disponibilidade

Relativamente ao atributo de qualidade disponibilidade o teste de carga foi efetuado permitindo o registo dos valores das métricas, observáveis na tabela 50. A estratégia de supervisão, mencionada na secção 5.4, adotada neste protótipo permitiu a deteção e o tratamento de falhas que podem vir a ocorrer durante a execução da solução. Através deste mecanismo foi possível de uma forma informada a tomada de decisão relativamente ao fluxo de informação, aquando da deteção de uma falha, a seguir pelo protótipo.

Tabela 50 - Valores das métricas de disponibilidade para o protótipo baseado no modelo ator

| **Métrica** | **Valor** |
|---|---|
| Número de pedidos efetuados com sucesso | 750 |
| Latência máxima | 3,731 segundos |
| Tráfego enviado em rede | 54,08Kb |
| Tráfego recebido em rede | 46,75Kb |

### 6.5.4 Monitorabilidade

A respeito do atributo de qualidade monitorabilidade através do uso das ferramentas de monitorização, previamente mencionadas na subsecção 6.2, foi possível endereçar as métricas correspondentes. As métricas a considerar com o respetivo valor estão representadas na tabela 51.

Tabela 51 - Valores das métricas de monitorabilidade para o protótipo baseado no modelo ator

| **Métrica** | **Valor** |
|---|---|
| Geração de dados monitorizados | Possui |
| Armazenamento de dados monitorizados | Possui |
| Apresentação de dados monitorizados | Possui |

## 6.6 Testes de Hipóteses

Através da ferramenta Jmeter foram obtidos diversos valores de medições, tendo sido considerados valores médios e máximos referentes a certas métricas de avaliação dos protótipos, como por exemplo a métrica de tempo médio de resposta e de latência máxima. Ou seja, foi aplicada estatística descritiva que não permite concluir se as diferenças encontradas

nessas medições são significativas ou não. Pelo que, testes de hipóteses foram formulados para se perceber se as diferenças encontradas nas amostras são casuais ou significativas (Alvarenga, 2018).

As amostras recolhidas são emparelhadas, pois os dados foram obtidos através da execução do mesmo teste de carga nos diferentes protótipos. Concluindo-se que o teste mais adequado caso exista normalidade na distribuição é o teste paramétrico *t-test* para amostras emparelhadas (STHDA, 2016b), senão deve-se aplicar o teste não paramétrico de *Wilcoxon* para amostras emparelhadas (STHDA, 2016c).

Primeiro torna-se necessário averiguar se existe normalidade na distribuição. A normalidade pode ser avaliada através de inspeção visual, recorrendo a gráficos de densidade ou quantil-quantil, ou através de testes de significância (STHDA, 2016a). A inspeção visual usualmente não é confiável, existindo a necessidade de realização de testes de significância, recorrendo por exemplo do método *Shapiro-Wilk's* (STHDA, 2016a). Salienta-se a existência do teorema do limite central que assume que caso a amostra possua pelo menos 30 elementos é possível ignorar a distribuição dos dados, pois esta tende a ser normal, e utilizar testes paramétricos (STHDA, 2016a).

R é um software que permite a computação estatística e de gráficos, pois possui uma ampla variedade de técnicas estatísticas e gráficas, apresentando-se com um software livre compatível com diferentes plataformas e sistemas operativos (Foundation, 2001). O software R foi escolhido para a realização dos testes de hipóteses, pois o autor da dissertação possui conhecimentos nesse ambiente de desenvolvimento.

### 6.6.1 Tempo de resposta por pedido

A experiência controlada realizada através do teste de carga, efetuado aos protótipos, permitiu verificar o efeito da intervenção efetuada ao nível do tempo de resposta por pedido. Assim, três amostras referentes aos valores de tempo de resposta foram recolhidas:

- Amostra A: possui valores de tempo de resposta referentes ao protótipo baseado na topologia *broker*;
- Amostra B: possui valores de tempo de resposta referentes ao protótipo baseado na topologia mediadora;
- Amostra C: possui valores de tempo de resposta referentes ao protótipo baseado no modelo ator.

#### 6.6.1.1 Teste de hipóteses para as amostras A e B

As amostras A e B possuem mais de 30 valores, pelo que é possível aplicar o teorema do limite central. Contudo foi efetuada inspeção visual, com recurso ao gráfico quantil-quantil, presente na figura 66, onde se desenhou a correlação entre as amostras e a sua distribuição normal. Em (1) apresenta-se a fórmula de aplicação do teste de normalidade visual, no software R.

$$qqnorm(amostraA - amostraB)\ (1)$$

Concluindo-se que a distribuição aparenta estar normalmente distribuída, pois os pontos do gráfico, presentes na figura 66, tendem a aproximar-se ao longo da linha diagonal reta.

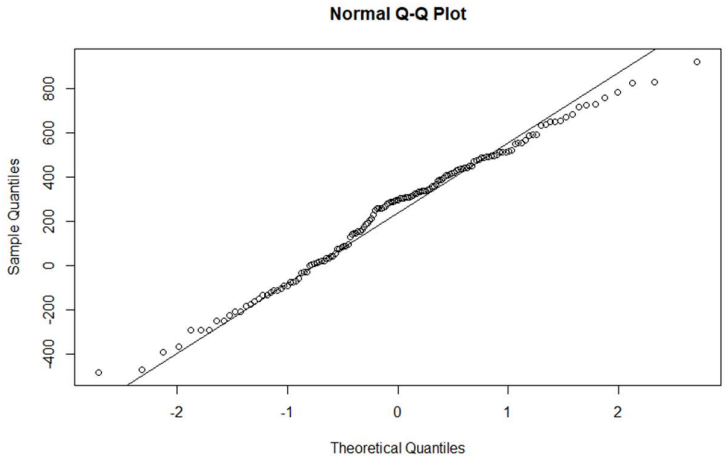



Figura 66 - Gráfico quantil-quantil para análise de normalidade

Após realização da análise visual foi efetuado um teste de significância de normalidade, para se concluir com um grau de certeza mais elevado que a distribuição é normal, com recurso ao método *Shapiro-Wilk's* com as seguintes hipóteses:

- H0: os dados são normalmente distribuídos;
- H1: os dados não são normalmente distribuídos.

Em (2) apresenta-se a fórmula de aplicação do teste de significância de normalidade no software R, tendo sido obtido o valor de *p-value* igual a 0,1274. Conclui-se que não se pode rejeitar a hipótese nula, pois o valor de *p-value* é superior a 0,05, podendo-se assumir que existe normalidade na distribuição.

$$shapiro.test(amostraA - amostraB)\ (2)$$

Comprovada a normalidade da distribuição, define-se as hipóteses de teste para o teste paramétrico *T-test*:

- H0: não há variações de tempo de resposta;
- H1: há variações de tempo de resposta.

Em (3) apresenta-se a fórmula de aplicação do teste paramétrico para amostras emparelhadas, tendo sido obtido o valor de *p-value* inferior a $2{,}2*10^{-16}$. Pode-se então rejeitar a hipótese nula, pois *p-value* é inferior a 0,05, e concluir que o tempo médio de resposta antes da intervenção é significativamente diferente do tempo médio de resposta após a intervenção.

$$t.test(amostraA, amostraB, paired = TRUE)\ (3)$$

#### 6.6.1.2 Teste de hipóteses para as amostras A e C

As amostras A e C possuem mais de 30 valores pelo que o teorema do limite central foi aplicado, assumindo-se a normalidade da distribuição.

As hipóteses de teste para o teste paramétrico *T-test* são as seguintes:

- H0: não há variações de tempo de resposta;
- H1: há variações de tempo de resposta.

Em (4) apresenta-se a fórmula de aplicação do teste paramétrico para amostras emparelhadas, tendo sido obtido o valor de *p-value* inferior a $2{,}2*10^{-16}$. Pode-se então rejeitar a hipótese nula (*p-value* < 0,05) e concluir que o tempo médio de resposta antes da intervenção é significativamente diferente do tempo médio de resposta após a intervenção.

$$t.test(amostraA, amostraC, paired = TRUE) \ (4)$$

#### 6.6.1.3 Teste de hipóteses para as amostras B e C

As amostras B e C possuem mais de 30 valores pelo que o teorema do limite central foi aplicado, assumindo-se a normalidade da distribuição.

As hipóteses de teste para o teste paramétrico *T-test* são as seguintes:

- H0: não há variações de tempo de resposta;
- H1: há variações de tempo de resposta.

Em (5) apresenta-se a fórmula de aplicação do teste paramétrico para amostras emparelhadas, tendo sido obtido o valor de *p-value* inferior a $2{,}2*10^{-1}$ . Pode-se então rejeitar a hipótese nula (*p-value* < 0,05) e concluir que o tempo médio de resposta é significativamente diferente em ambas as intervenções realizadas.

$$t.test(amostraB, amostraC, paired = TRUE) \ (5)$$

### 6.6.2 Latência por pedido

A experiência controlada realizada através do teste de carga, efetuado aos protótipos, permitiu verificar o efeito da intervenção efetuada ao nível da latência. Assim, três amostras referentes aos valores de latência foram construídas:

- Amostra A: possui valores de latência referentes ao protótipo baseado na topologia *broker*;
- Amostra B: possui valores de latência referentes ao protótipo baseado na topologia mediadora;
- Amostra C: possui valores de latência referentes ao protótipo baseado no modelo ator.

#### 6.6.2.1 Teste de hipóteses para as amostras A e B

As amostras A e B possuem mais de 30 valores pelo que o teorema do limite central foi aplicado, assumindo-se a normalidade da distribuição.

As hipóteses de teste para o teste paramétrico *T-test* são as seguintes:

- H0: não há variações de latência;
- H1: há variações de latência.

Em (6) apresenta-se a fórmula de aplicação do teste paramétrico para amostras emparelhadas, tendo sido obtido o valor de *p-value* inferior a $2{,}2*10^{-16}$. Pode-se então rejeitar a hipótese nula (*p-value* < 0,05) e concluir que a latência média antes da intervenção é significativamente diferente da latência média após a intervenção.

$$t.test(amostraA, amostraB, paired = TRUE) \ (6)$$

#### 6.6.2.2 Teste de hipóteses para as amostras A e C

As amostras A e C possuem mais de 30 valores pelo que o teorema do limite central foi aplicado, assumindo-se a normalidade da distribuição.

As hipóteses de teste para o teste paramétrico *T-test* são as seguintes:

- H0: não há variações de latência;
- H1: há variações de latência.

Em (7) apresenta-se a fórmula de aplicação do teste paramétrico para amostras emparelhadas, tendo sido obtido o valor de *p-value* inferior a $2{,}2*10^{-16}$. Pode-se então rejeitar a hipótese nula (*p-value* < 0,05) e concluir que a latência média antes da intervenção é significativamente diferente da latência média após a intervenção.

$$t.test(amostraA, amostraC, paired = TRUE) \ (7)$$

#### 6.6.2.3 Teste de hipóteses para as amostras B e C

As amostras B e C possuem mais de 30 valores pelo que o teorema do limite central foi aplicado, assumindo-se a normalidade da distribuição.

As hipóteses de teste para o teste paramétrico *T-test* são as seguintes:

- H0: não há variações de latência;
- H1: há variações de latência.

Em (8) apresenta-se a fórmula de aplicação do teste paramétrico para amostras emparelhadas, tendo sido obtido o valor de *p-value* igual a 0,8009. Pode-se então rejeitar a hipótese H1 (*p-value* > 0,05) e concluir que a latência média não é significativamente diferente em ambas as intervenções realizadas.

$$t.test(amostraB, amostraC, paired = TRUE) \ (8)$$

## 6.7 Testes de Stress

Através da realização de testes de *stress* pretende-se sobrecarregar um sistema, com o intuito de se encontrar o ponto de rutura do mesmo. Partiu-se dos valores utilizados no teste de carga presente nas secções 6.3, 6.4 e 6.5, ou seja, 250 sensores com um *loop count* de 3, incrementando-se, sucessivamente, o valor de *loop count* numa unidade em cada teste de *stress* realizado. A tabela 52 sumariza o nível de completitude dos testes de *stress* realizados aos protótipos, evidenciando-se o ponto de rutura dos mesmos. Conclui-se que o ponto de rutura do protótipo baseado na topologia *broker* é com 1250 pedidos, do protótipo baseado no modelo ator é com 1750 pedidos e do protótipo baseado na topologia mediadora é com 2250 pedidos.

Tabela 52 – Nível de completitude dos testes de stress efetuados aos protótipos

| Número de sensores | Loop Count | Número de pedidos | Protótipo Broker | Protótipo Mediador | Protótipo Ator |
|---|---|---|---|---|---|
| 250 | 3 | 750 | Superou | Superou | Superou |
| 250 | 4 | 1000 | Superou | Superou | Superou |
| 250 | 5 | 1250 | Não superou | Superou | Superou |
| 250 | 6 | 1500 | - | Superou | Superou |
| 250 | 7 | 1750 | - | Superou | Não superou |
| 250 | 8 | 2000 | - | Superou | - |
| 250 | 9 | 2250 | - | Não superou | - |

A tabela 53 agrega os tempos de execução dos testes de *stress* realizados em cada protótipo. Adicionalmente, quando o protótipo não consegue superar a carga de pedidos a considerar, o número de mensagens perdidas é expresso. Por exemplo, no caso do protótipo baseado na topologia *broker*, este perde 204 mensagens quando é sujeito a 1250 pedidos.

Tabela 53 - Tempos de execução dos testes de stress realizados aos protótipos

| Número de sensores | Loop Count | Número de pedidos | Tempo de execução do Protótipo Broker | Tempo de execução do Protótipo Mediador | Tempo de execução Protótipo Ator |
|---|---|---|---|---|---|
| 250 | 3 | 750 | 6 segundos | 5 segundos | 4 segundos |
| 250 | 4 | 1000 | 8 segundos | 9 segundos | 7 segundos |

| | | | | | |
|---|---|---|---|---|---|
| 250 | 5 | 1250 | 41 segundos (com perdas de 204 mensagens) | 11 segundos | 26 segundos |
| 250 | 6 | 1500 | - | 17 segundos | 30 segundos (com perdas de 181 mensagens) |
| 250 | 7 | 1750 | - | 24 segundos | - |
| 250 | 8 | 2000 | - | 28 segundos | - |
| 250 | 9 | 2250 | - | 32 segundos (com perdas de 48 mensagens) | - |

## 6.8 Categorização e sintetização dos resultados

Após a recolha dos valores das métricas referentes aos três protótipos realizou-se a comparação dos resultados alcançados, para tal os valores das métricas foram categorizados segundo a escala de avaliação exposta ao longo da secção 6.3. A tabela 54 expõe os resultados alcançados.

Tabela 54 - Resultados da avaliação de cada protótipo por métrica

| **Métrica/Protótipo** | **Inicial** | **Mediador** | **Ator** |
|---|---|---|---|
| Utilização de CPU | Suficiente | Bom | Bom |
| Utilização de memória RAM | Suficiente | Bom | Bom |
| Crescimento da fila de mensagens | Suficiente | Suficiente | Suficiente |
| Tempo médio de resposta por pedido | Suficiente | Suficiente | Bom |
| Taxa de transferência | Suficiente | Suficiente | Bom |
| Deteção de falhas | Insuficiente | Suficiente | Suficiente |
| Tráfego enviado em rede | Suficiente | Suficiente | Insuficiente |
| Tráfego recebido em rede | Suficiente | Suficiente | Suficiente |
| Número de pedidos concluídos com sucesso | Suficiente | Suficiente | Suficiente |
| Latência | Suficiente | Bom | Bom |
| Frequência de uso | Suficiente | Bom | Bom |

| | | | |
|---|---|---|---|
| Número de dependências síncronas | Suficiente | Bom | Bom |
| Escalonamento horizontal/vertical | Insuficiente | Suficiente | Suficiente |
| Geração de dados monitorizados | Suficiente | Suficiente | Suficiente |
| Armazenamento de dados monitorizados | Suficiente | Suficiente | Suficiente |
| Apresentação de dados monitorizados | Suficiente | Suficiente | Suficiente |

Relativamente ao desempenho concluiu-se que o protótipo baseado na topologia mediadora apresenta resultados melhorados nas métricas de utilização de recursos de hardware, face ao protótipo representativo das aplicações típicas da empresa. Por outro lado, o protótipo baseado no modelo ator apresenta resultados aperfeiçoados face ao protótipo baseado na topologia mediadora nas métricas de tempo médio de resposta por pedido e de taxa de transferência.

Sobre o atributo de qualidade disponibilidade os protótipos baseados na topologia mediadora e no modelo ator apresentam um valor melhorado na métrica de deteção de falhas face ao protótipo representativo das aplicações típicas da empresa, devido aos mecanismos de deteção que implementam. No que diz respeito à métrica de tráfego enviado em rede, o protótipo baseado no modelo ator apresenta um valor mais elevado pois mais tráfego é enviado em rede face aos outros protótipos. Sobre a latência, os protótipos baseados na topologia mediadora e no modelo ator apresentam valores enriquecidos nesta métrica face ao protótipo inicial.

Em relação ao atributo de qualidade escalabilidade concluiu-se que os protótipos baseados na topologia mediadora e no modelo ator apresentam valores melhorados em todas as métricas face ao protótipo representativo das aplicações típicas da empresa.

No que concerne aos valores das métricas de monitorabilidade todos os protótipos implementados apresentam a capacidade de gestão de dados monitorizados, alcançando-se assim o valor máximo da escala utilizada para se avaliar as métricas deste atributo de qualidade.

## 6.9 Sumário

Terminado o processo de categorização e comparação dos valores das métricas referentes aos protótipos, tornou-se necessária a construção de uma fórmula (1) que permite de uma maneira objetiva alcançar um valor que represente a concretização de cada atributo de qualidade em cada um dos protótipos.

$$Número\ de\ Suficientes + 1.5 * Número\ Bons - Número\ Insuficientes \quad (1)$$

Na tabela 55 agregam-se os valores resultantes da aplicação da fórmula de avaliação alcançada. Resultando assim um valor que simboliza o nível de concretização de cada atributo de qualidade nos diferentes protótipos, bem como um valor global, que agrega todos os atributos de

qualidade, que permite a seleção do protótipo mais relevante para o objetivo traçado na aplicação de *GQM*.

Tabela 55 – Valores resultantes da aplicação da fórmula aos protótipos

| **Atributo de Qualidade/Protótipo** | **Inicial** | **Mediador** | **Ator** |
|---|---|---|---|
| Desempenho | 5,0 | 6,0 | 7,0 |
| Disponibilidade | 3,0 | 5,5 | 3,5 |
| Escalabilidade | 1,0 | 4,0 | 4,0 |
| Monitorabilidade | 3,0 | 3,0 | 3,0 |
| Todos os atributos anteriores | 12,0 | 18,5 | 17,5 |

Concluiu-se que o protótipo baseado no modelo ator possui o valor, igual a 7,0, referente ao atributo de qualidade desempenho superior face aos outros protótipos. Consequentemente, uma solução baseada no modelo ator comparativamente a uma solução baseada na topologia mediadora, para o problema de negócio em questão e para as métricas identificadas, possui um efeito mais positivo quando se possui requisitos de desempenho.

Relativamente à disponibilidade o protótipo baseado na topologia mediadora apresenta o valor, igual a 5,5, mais elevado. Consequentemente, uma solução baseada na topologia mediadora comparativamente a uma solução baseada no modelo ator, para o problema de negócio em questão e para as métricas identificadas, possui um efeito mais positivo quando se possui requisitos de disponibilidade.

Do ponto de vista da escalabilidade os protótipos baseados na topologia mediadora e no modelo ator apresentaram ambos o valor de 4,0. Consequentemente, as soluções baseadas na topologia mediadora e no modelo ator, para o problema de negócio em questão e para as métricas identificadas, possuem ambas o mesmo efeito quando se possui requisitos de escalabilidade.

Por fim, sobre a monitorabilidade todos os protótipos possuem o valor 3,0 correspondente ao atributo de qualidade em questão. Consequentemente, as soluções baseadas na topologia mediadora e no modelo ator, para o problema de negócio em questão e para as métricas identificadas, possuem ambas o mesmo efeito quando se possui requisitos de monitorabilidade.

Sumariza-se que o protótipo baseado na topologia mediadora é o que mais satisfaz o objetivo delineado na aplicação de *GQM*, por apresentar o valor final mais elevado, referente à agregação dos valores de todos os atributos de qualidade, igual a 18,5.

# 7 Conclusão

Neste capítulo os resultados atingidos referentes ao projeto de tese desenvolvido são expostos, com base nos objetivos traçados no capítulo de introdução. Depois as contribuições são expressas. De seguida, as ameaças detetadas relativas ao trabalho são evidenciadas. Finaliza-se com o trabalho futuro a ponderar realizar, minimizando-se assim as ameaças identificadas, com o objetivo de se enriquecer ainda mais o projeto.

## 7.1 Resultados

Na secção 1.3 o objetivo deste projeto de tese foi delineado, definindo-se complementarmente tarefas a atingir no desenvolver deste trabalho. Os resultados atingidos alcançados através da concretização das tarefas, presentes em 1.3, permitiram concluir sobre o grau de cumprimento das tarefas. Para cada tarefa apresenta-se como é que esta foi endereçada e realizada:

- Investigar interpretações/abordagens a considerar no processo de desenvolvimento de arquiteturas de microsserviços orientadas a eventos: a revisão de literatura realizada no capítulo de estado de arte permitiu a identificação de desafios, padrões e topologias que surgem no momento do desenvolvimento de arquiteturas de microsserviços orientadas a eventos (presente na secção 2.2.1). Adicionalmente casos práticos que aplicam uma arquitetura de microsserviços orientada a eventos foram expostos (presente na secção 2.2.3);
- Investigar tecnologias relevantes aquando da implementação de arquiteturas de microsserviços orientadas a eventos: através da mesma revisão de literatura identificaram-se e descreveram-se tecnologias relevantes que permitem a implementação de diferentes topologias de microsserviços orientadas a eventos (presente na secção 2.2.2). *Frameworks* que permitem a implementação do modelo

conceptual ator, proporcionando a construção de sistemas orientados a eventos, foram também identificadas e exploradas (presente na secção 2.2.2);

- Investigar métricas utilizadas para se avaliar arquiteturas de microsserviços orientadas a eventos: recorrendo da mesma revisão de literatura foram identificadas métricas e opiniões a considerar no momento de avaliação de arquiteturas de microsserviços orientadas a eventos (presente na secção 2.2.4);
- Conceber diferentes protótipos segundo uma arquitetura de microsserviços orientada a eventos: terminado o processo de revisão de literatura, presente no capítulo de estado de arte, foi tomada a decisão de desenvolvimento de duas soluções de microsserviços orientadas a eventos (presente no capítulo 3). A primeira através da implementação da topologia mediadora, com recurso a Apache Camel, e a segunda através da aplicação de uma *framework* orientada a atores, mais concretamente o Akka.NET. A arquitetura das aplicações típicas da empresa foi exposta, evidenciando-se as alterações a efetuar a esta para se conceber as duas soluções orientadas a eventos anteriormente mencionadas (presente no capítulo 4);
- Implementar diferentes protótipos segundo uma arquitetura de microsserviços orientada a eventos: concluído o processo de conceção, dos protótipos segundo arquiteturas de microsserviços orientadas a eventos, os principais detalhes de implementação referentes aos protótipos concebidos e implementados foram expostos (presente no capítulo 5);
- Avaliar protótipos implementados que seguem uma arquitetura de microsserviços orientada a eventos: findo o processo de implementação dos protótipos, a configuração das ferramentas de monitorização que permitem a obtenção de valores de métricas foi sintetizada (presente na seção 6.2). A recolha desses valores permitiu a avaliação da concretização de cada atributo de qualidade em cada protótipo (presente nas secções 6.3, 6.4 e 6.5);
- Comparar os protótipos implementados que seguem uma arquitetura de microsserviços orientada a eventos: Após o processo de avaliação de cada um dos protótipos desenvolvidos, estes foram comparados entre si, identificando-se os pontos fortes bem como as limitações de cada um destes (presente nas secções 6.8 e 6.9).

## 7.2 Contribuições

Nos últimos anos a quantidade de dados criados, capturados, copiados e consumidos globalmente pelos sistemas de software tem vindo a aumentar expressivamente de ano para ano. Espera-se que esses sistemas se apresentem como altamente disponíveis, escaláveis, resilientes e com elevado desempenho. Pelo que, a procura por sistemas distribuídos tem crescido e cada vez mais empresas adotam o estilo arquitetural de microsserviços.

O trabalho de mestrado desenvolvido é uma contribuição considerável para as empresas que pretendem explorar abordagens orientadas a eventos conjugadas com arquiteturas de microsserviços. O processo de conceção de arquiteturas de microsserviços orientadas a eventos

foi exposto, explorando-se as duas principais topologias (*broker* e *mediator*) e uma abordagem orientada a atores para o negócio de estacionamento de veículos.

O trabalho de implementação desenvolvido, composto pelos três protótipos arquiteturalmente diferentes, encontra-se disponível no repositório Bitbucket do autor desta dissertação (Marques, 2023). Este repositório possui comentários nas partes de implementação mais relevantes, e complementarmente apresenta um guia de como executar os diferentes protótipos e como os utilizar devidamente. Assim, torna-se possível a outros investigadores explorarem e continuarem o trabalho iniciado neste projeto de mestrado.

Conclui-se que a principal contribuição é o entendimento da temática das arquiteturas de microsserviços orientadas a eventos, desde o processo de recolha de informação sobre o tema até ao processo de implementação e avaliação dos diferentes protótipos alcançados. Apesar de os protótipos contemplarem o negócio de estacionamento de veículos, os conceitos principais arquiteturais podem ser replicados noutras áreas de negócio.

## 7.3 Ameaças à validade

Durante o processo de desenvolvimento do projeto de tese foram detetadas adversidades, apresentando-se como possíveis ameaças à validação da solução exposta neste documento. As ameaças identificadas são as seguintes:

- Um ambiente de teste totalmente isolado não foi desenvolvido. Pelo que, a validade de alguns testes efetuados, sobretudo de desempenho e de disponibilidade, pode ter sido colocada em causa por influência de fatores presentes no ambiente de teste utilizado para se avaliar os protótipos desenvolvidos.
- Cada protótipo desenvolvido apresenta um número reduzido de microsserviços, não tendo sido efetuados cenários com um número mais expressivo de microsserviços. Pelo que, os resultados atingidos poderiam não ser os mesmos em protótipos com um número superior de microsserviços.
- O trabalho desenvolvido contempla a conceção e implementação de diferentes abordagens de microsserviços orientados a eventos, sendo que para cada abordagem foi escolhida uma tecnologia para a respetiva implementação. Pelo que, para se conseguir generalizar de uma forma mais rigorosa os resultados alcançados, seria necessário para cada abordagem (*broker*, mediadora e ator) explorada possuir a implementação desta segundo diferentes tecnologias.

## 7.4 Trabalho futuro

O trabalho presente neste documento pretendeu responder aos objetivos delineados na secção 1.3 do capítulo introdução. Contudo, certos pontos adicionais seriam interessantes de se

explorar, minimizando-se assim as ameaças detetadas e ao mesmo tempo acrescentando-se valor ao projeto desenvolvido. Os pontos a considerar em trabalho futuro são os seguintes:

- Alcançar um ambiente de teste isolado;
- Realizar os mesmos testes efetuados anteriormente, mas agora no novo ambiente de teste;
- Diversificar a presença de outros testes de carga a efetuar aos protótipos;
- Desenvolver outros protótipos, segundo as mesmas topologias/abordagens, mas com recurso a tecnologias diferentes;

# Referências


ACM. (2018). Code of Ethics. Retrieved February 17, 2023, from https://www.acm.org/code-of-ethics#h-3.6-use-care-when-modifying-or-retiring-systems

ActiveMQ, A. (2019a). How do durable queues and topics work. Retrieved January 5, 2023, from https://activemq.apache.org/how-do-durable-queues-and-topics-work

ActiveMQ, A. (2019b). Multiple consumers on a queue. Retrieved April 21, 2023, from https://activemq.apache.org/multiple-consumers-on-a-queue

Akka. (2017). What is Akka | Akka.NET Documentation. Retrieved December 12, 2022, from https://getakka.net/articles/intro/what-is-akka.html

Al-Twajre, B. A. (2019). Performance Analysis of Messages Queue in the Different Actor System Implementation. *2019 11th International Scientific and Practical Conference on Electronics and Information Technologies, ELIT 2019 - Proceedings*, 127–131. https://doi.org/10.1109/ELIT.2019.8892329

Aley, J. (2022). The benefits and challenges of event-driven architecture | InfoWorld. Retrieved January 23, 2023, from https://www.infoworld.com/article/3669414/the-benefits-and-challenges-of-event-driven-architecture.html

Alvarenga, H. (2018). Teste t de Student no R. Retrieved June 14, 2023, from http://rstudio-pubs-static.s3.amazonaws.com/408638_6679293d4c7a415eaebe00faa3aea0cb.html

Ambre, T. (2020). Architectural considerations for event-driven microservices-based systems. Retrieved January 23, 2023, from https://developer.ibm.com/articles/eda-and-microservices-architecture-best-practices/

Arambarri, F., Dennis, D., Dahan, U., Sherer, T., & Hallihan, R. (2023). PADRÃO CQRS - Microsoft Learn. Retrieved June 5, 2023, from https://learn.microsoft.com/pt-pt/azure/architecture/patterns/cqrs

Arambarri, F., Hallihan, R., Ganji, S., Leskis, A., & Sherer, T. (2023). Padrão de fornecimento do evento - Microsoft Learn. Retrieved June 5, 2023, from https://learn.microsoft.com/pt-br/azure/architecture/patterns/event-sourcing

Bellemare, A. (2020). *Building event-driven microservices : leveraging organizational data at scale*. O'Reilly Media. Retrieved from https://books.google.com/books/about/Building_Event_Driven_Microservices.html?id=GbJIzQEACAAJ

Belliveau, P., Griffin, A., & Somermeyer, S. (2002). *The PDMA ToolBook 1 for New Product Development - Google Livros*. Retrieved from https://books.google.pt/books?hl=pt-PT&lr=&id=kqX5EvT2U8AC&oi=fnd&pg=PA5&dq=fuzzy+front+end+effective+methods+tools+and+techniques+pdf&ots=8Lpk97vMha&sig=4aNVvJLa32b-I2YzWSu-SzM6p7o&redir_esc=y#v=onepage&q&f=false

Bos, O. (2020). The Engineers Guide to Event-Driven Architectures: Benefits and Challenges | by Oskar uit de Bos | The Startup | Medium. Retrieved January 23, 2023, from https://medium.com/swlh/the-engineers-guide-to-event-driven-architectures-benefits-and-challenges-3e96ded8568b

Bos, O. uit de. (2021a). The Event-Carried State Transfer pattern. Retrieved June 5, 2023, from https://itnext.io/the-event-carried-state-transfer-pattern-aae49715bb7f

Bos, O. uit de. (2021b). The event notification pattern. Retrieved June 5, 2023, from https://medium.com/geekculture/the-event-notification-pattern-a62d48519107

Brown, S., & Betts, T. (2018). The C4 Model for Software Architecture. Retrieved January 10, 2023, from https://www.infoq.com/articles/C4-architecture-model/

Camel, A. (2009). Home - Apache Camel. Retrieved December 12, 2022, from https://camel.apache.org/

Camel, A. (2021a). Components :: Apache Camel. Retrieved December 12, 2022, from https://camel.apache.org/components/next/

Camel, A. (2021b). What is Camel? :: Apache Camel. Retrieved December 12, 2022, from https://camel.apache.org/manual/faq/what-is-camel.html

Cerny, T., Donahoo, M. J., & Trnka, M. (2017). Contextual Understanding of Microservice Architecture: Current and Future Directions. *RACS '17: Proceedings of the International Conference on Research in Adaptive and Convergent Systems*, 228–235. https://doi.org/10.1145/3129676.3129682

Dennis, D., Dahan, U., & Sherer, T. (2022). Padrão de Publisher-Subscriber - Microsoft Learn. Retrieved June 5, 2023, from https://learn.microsoft.com/pt-pt/azure/architecture/patterns/publisher-subscriber

Dineshchand, G. R. (2022). What is Circuit Breaker in Microservices? | by Dineshchandgr - A Top writer in Technology | Javarevisited | Medium. Retrieved April 5, 2023, from https://medium.com/javarevisited/what-is-circuit-breaker-in-microservices-a94f95f5e5ae

Fielding, R. T. (2000). *Architectural Styles and the Design of Network-based Software Architectures*. Retrieved from https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Foundation, R. (2001). R: What is R? Retrieved June 14, 2023, from https://www.r-project.org/about.html

Fowler, M. (2006). Event Collaboration. Retrieved June 5, 2023, from https://martinfowler.com/eaaDev/EventCollaboration.html

Fowler, M. (2017). What do you mean by "Event-Driven"? Retrieved December 10, 2022, from https://martinfowler.com/articles/201701-event-driven.html

Fu, G., Zhang, Y., & Yu, G. (2021). A Fair Comparison of Message Queuing Systems. *IEEE Access*, *9*, 421–432. https://doi.org/10.1109/ACCESS.2020.3046503

García-Jiménez, F. J., Martínez-Carreras, M. A., & Gómez-Skarmeta, A. F. (2010). Evaluating open source Enterprise Service Bus. *Proceedings - IEEE International Conference on E-Business Engineering, ICEBE 2010*, 284–291. https://doi.org/10.1109/ICEBE.2010.12

Gordesli, M., & Varol, A. (2022). Comparing Interservice Communications of Microservices for E-Commerce Industry. *10th International Symposium on Digital Forensics and Security, ISDFS 2022*. https://doi.org/10.1109/ISDFS55398.2022.9800784

Gosewehr, F., Wermann, J., Borsych, W., & Colombo, A. W. (2018). Apache camel based implementation of an industrial middleware solution. *Proceedings - 2018 IEEE Industrial Cyber-Physical Systems, ICPS 2018*, 523–528. https://doi.org/10.1109/ICPHYS.2018.8390760

Gotin, M., Lösch, F., Heinrich, R., & Reussner, R. (2018). Investigating performance metrics for scaling microservices in CloudIoT-environments. *ICPE 2018 - Proceedings of the 2018 ACM/SPEC International Conference on Performance Engineering*, *2018-March*, 157–167. https://doi.org/10.1145/3184407.3184430

Ingham, J. (1999). *What is an Agent?* Retrieved from https://www.researchgate.net/publication/2461558_What_is_an_Agent

Jaeger. (2023a). Architecture — Jaeger documentation. Retrieved April 6, 2023, from https://www.jaegertracing.io/docs/1.43/architecture/

Jaeger. (2023b). Jaeger documentation. Retrieved April 6, 2023, from https://www.jaegertracing.io/docs/1.43/

Jayawardene, san D. (2021). 4+1 Architectural view model in Software. Retrieved January 10, 2023, from https://medium.com/javarevisited/4-1-architectural-view-model-in-software-ec407bf27258

JMeter, A. (2011). Apache JMeter - Overview. Retrieved May 24, 2023, from https://jmeter.apache.org/index.html

Karlsen, D. J. M. (2021). resilience4j/grafana_dashboard.json at master · resilience4j/resilience4j · GitHub. Retrieved April 6, 2023, from https://github.com/resilience4j/resilience4j/blob/master/grafana_dashboard.json

Katsifodimos, A., & Fragkoulis, M. (2019). Operational stream processing: Towards scalable and consistent event-driven applications. *Advances in Database Technology - EDBT*, *2019-March*, 682–685. https://doi.org/10.5441/002/EDBT.2019.86

Kuhn, R. (2015). akka-meta/ComparisonWithOrleans.md at master · akka/akka-meta · GitHub. Retrieved December 12, 2022, from https://github.com/akka/akka-meta/blob/master/ComparisonWithOrleans.md

Li, S., Zhang, H., Jia, Z., Zhong, C., Zhang, C., Shan, Z., … Babar, M. A. (2021). Understanding and addressing quality attributes of microservices architecture: A Systematic literature review. *Information and Software Technology*, *131*, 106449. https://doi.org/10.1016/J.INFSOF.2020.106449

Lv, H., Zhang, S., Zhang, X., Guo, Q., Dong, J., & Zhao, Y. (2020). The actor model based

distributed fault tolerant control system, 1591–1594. https://doi.org/10.1109/ITAIC49862.2020.9338801

Marques, P. (2023). Event Driven Microservices Architectures — Bitbucket Repository. Retrieved June 6, 2023, from https://bitbucket.org/PedroMarques1180873/event_driven_microservices_architectures/src/master/

Microsoft. (2022). Orleans overview | Microsoft Learn. Retrieved December 12, 2022, from https://learn.microsoft.com/en-us/dotnet/orleans/overview

Mulesoft. (2015). What is Mule ESB? | MuleSoft. Retrieved December 12, 2022, from https://www.mulesoft.com/resources/esb/what-mule-esb

Newman, S. (2021). *Building Microservices Designing Fine-Grained Systems*. (O. Media, Ed.). Retrieved from https://books.google.pt/books?hl=pt-PT&lr=&id=ZvM5EAAAQBAJ&oi=fnd&pg=PT8&dq=Building+Microservices+Designing+Fine-Grained+Systems+Sam+Newman&ots=uhcgaxbJUu&sig=3-LKWn-VNEQ1ZX87QOuuSYLKMUI&redir_esc=y#v=onepage&q=Building Microservices Designing Fine-Grain

Oliveira Rocha, H. F. (2022). *Practical Event-Driven Microservices Architecture Building Sustainable and Highly Scalable Event-Driven Microservices* (1st ed.). Apress Berkeley, CA. https://doi.org/10.1007/978-1-4842-7468-2

OpenTracing. (2019a). OpenTracing specification. Retrieved April 6, 2023, from https://opentracing.io/specification/

OpenTracing. (2019b). What is Distributed Tracing? Retrieved April 6, 2023, from https://opentracing.io/docs/overview/what-is-tracing/

Osterwalder, A. (2004). *The Business Model Ontology: A Proposition in a Design Science Approach - Alexander Osterwalder - Google Books*. Retrieved from https://books.google.pt/books/about/The_Business_Model_Ontology.html?id=wunyZwEACAAJ&redir_esc=y

Parulkar, S. (2020). The importance of event-driven architecture in the digital world. Retrieved January 23, 2023, from https://www.redhat.com/en/blog/importance-event-driven-architecture-digital-world

Paspelava, D. (2021). 4 Levels of Software Testing | Test Levels in Software Testing. Retrieved April 5, 2023, from https://www.exposit.com/blog/4-levels-software-testing-how-develop-reliable-product/

Petabridge. (2017a). Supervision | Akka.NET Documentation. Retrieved April 5, 2023, from https://getakka.net/articles/concepts/supervision.html#one-for-one-strategy-vs-all-for-one-strategy

Petabridge. (2017b). UntypedActor API | Akka.NET Documentation. Retrieved April 5, 2023, from https://getakka.net/articles/actors/untyped-actor-api.html

Petersen, K., Feldt, R., Mujtaba, S., & Mattsson, M. (2008). Systematic Mapping Studies in Software Engineering. *12th International Conference on Evaluation and Assessment in*

*Software Engineering, EASE 2008*. https://doi.org/10.14236/EWIC/EASE2008.8

Petersen, T., Arambarri, F., Sperker, H., & Urnun, M. (2022). Padrão de Request-Reply assíncrona - Microsoft Learn. Retrieved June 5, 2023, from https://learn.microsoft.com/pt-br/azure/architecture/patterns/async-request-reply

Prajapati, A. (2021). AMQP and beyond. *2021 International Conference on Smart Applications, Communications and Networking, SmartNets 2021*. https://doi.org/10.1109/SMARTNETS50376.2021.9555419

Prometheus. (2023). Overview | Prometheus. Retrieved April 6, 2023, from https://prometheus.io/docs/introduction/overview/

Reqtest. (2014). Levels of Testing - Understand the Difference b/w Different Levels & Types. Retrieved April 5, 2023, from https://reqtest.com/testing-blog/different-levels-of-testing/

Richards, M. (2019). 2. Event-Driven Architecture - Software Architecture Patterns [Book]. Retrieved December 10, 2022, from https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437/ch02.html

Rudrabhatla, C. K. (2018). Comparison of Event Choreography and Orchestration Techniques in Microservice Architecture. *International Journal of Advanced Computer Science and Applications*, *9*(8), 18–22. https://doi.org/10.14569/IJACSA.2018.090804

Saaty, T. L. (1980). The analytic hierarchy process: planning. *Priority Setting. Resource Allocation, MacGraw-Hill, New York International Book Company*, 287. Retrieved from https://books.google.com/books/about/The_Analytic_Hierarchy_Process.html?id=Xxi7AAAAIAAJ

Schirgi, T., & Brenner, E. (2021). Quality Assurance for Microservice Architectures. *Proceedings of the IEEE International Conference on Software Engineering and Service Sciences, ICSESS*, *2021-August*, 76–80. https://doi.org/10.1109/ICSESS52187.2021.9522227

Sein, M. K., Henfridsson, O., Purao, S., Rossi, M., & Lindgren, R. (2011). Action design research. *MIS Quarterly: Management Information Systems*, *35*(1), 37–56. https://doi.org/10.2307/23043488

SentinelOne. (2021). What Is OpenTracing? Everything You Need to Get Started. Retrieved April 6, 2023, from https://www.sentinelone.com/blog/what-is-opentracing/

Shivang. (2021). What Is Grafana? Why Use It? Everything You Should Know About It - Scaleyourapp. Retrieved April 6, 2023, from https://scaleyourapp.com/what-is-grafana-why-use-it-everything-you-should-know-about-it/

Solingen, R. van., & Berghout, E. (1999). The goal/question/metric method : a practical guide for quality improvement of software development, 199.

STHDA. (2016a). Normality Test in R - Easy Guides - Wiki - STHDA. Retrieved June 14, 2023, from http://www.sthda.com/english/wiki/normality-test-in-r

STHDA. (2016b). Paired Samples T-test in R - Easy Guides - Wiki - STHDA. Retrieved June 14, 2023, from http://www.sthda.com/english/wiki/paired-samples-t-test-in-r

STHDA. (2016c). Paired Samples Wilcoxon Test in R - Easy Guides - Wiki - STHDA. Retrieved June 14, 2023, from http://www.sthda.com/english/wiki/paired-samples-wilcoxon-test-in-r

Stopford, B. (2018). *Designing even-driven systems : concepts and patterns for streaming services with Apache Kafka*. O'Reilly Media, Inc. Retrieved from https://www.oreilly.com/library/view/designing-event-driven-systems/9781492038252/

Strategyzer. (2017). Value Proposition Canvas – Download the Official Template. Retrieved January 3, 2023, from https://www.strategyzer.com/canvas/value-proposition-canvas

Surantha, N., Utomo, O. K., Lionel, E. M., Gozali, I. D., & Isa, S. M. (2022). Intelligent Sleep Monitoring System Based on Microservices and Event-Driven Architecture. *IEEE Access*, *10*, 42055–42066. https://doi.org/10.1109/ACCESS.2022.3167637

Taylor, P. (2022). Total data volume worldwide 2010-2025 | Statista. Retrieved May 24, 2023, from https://www.statista.com/statistics/871513/worldwide-data-created/

Taylor, T. (2020). Apache, Spring and Mule: Explore 3 top integration frameworks | TechTarget. Retrieved December 12, 2022, from https://www.techtarget.com/searchapparchitecture/tip/Apache-Spring-and-Mule-Explore-3-top-integration-frameworks

Tiwari, K. K. (2019). Actor Model in Nutshell. Actor Model is a conceptual concurrent… | by Krishna Kumar Tiwari | Medium. Retrieved December 12, 2022, from https://medium.com/@KtheAgent/actor-model-in-nutshell-d13c0f81c8c7

Trends, G. (2023). event driven microservices - Explorar - Google Trends. Retrieved January 3, 2023, from https://trends.google.pt/trends/explore?date=2019-11-15 2022-12-15&q=event driven microservices

Vale, G., Correia, F. F., Guerra, E. M., De Oliveira Rosa, T., Fritzsch, J., & Bogner, J. (2022). Designing Microservice Systems Using Patterns: An Empirical Study on Quality Trade-Offs. *Proceedings - IEEE 19th International Conference on Software Architecture, ICSA 2022*, 69–79. https://doi.org/10.1109/ICSA53651.2022.00015

Vyas, S., Tyagi, R. K., Jain, C., & Sahu, S. (2021). Literature review: A comparative study of real time streaming technologies and Apache Kafka. *Proceedings - 2021 4th International Conference on Computational Intelligence and Communication Technologies, CCICT 2021*, 146–153. https://doi.org/10.1109/CCICT53244.2021.00038

Warren, G., Wenzel, M., Dahan, U., & Pine, D. (2022). Asynchronous message-based communication - Microsoft Learn. Retrieved June 5, 2023, from https://learn.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/asynchronous-message-based-communication

Wolniak, R. (2017). The history of the QFD method. *Scientific Papers of Silesian University of Technology. Organization and Management Series*, *2017*(100), 553–564. https://doi.org/10.29119/1641-3466.2017.100.42

Zeithaml, V. A. (1988). Consumer Perceptions of Price, Quality, and Value: A Means-End Model and Synthesis of Evidence. *Journal of Marketing*, *52*(3), 2–22. https://doi.org/10.1177/002224298805200302

Zhelev, S., & Rozeva, A. (2019). Using microservices and event driven architecture for big data stream processing Deep learning model for object detection AIP Conference Using Microservices and Event Driven Architecture for Big Data Stream Processing, *2172*, 20001. https://doi.org/10.1063/1.5133587

Zhong, Y., Li, W., & Wang, J. (2019). Using event sourcing and CQRS to build a high performance point trading system. *ACM International Conference Proceeding Series*, 16–19. https://doi.org/10.1145/3317614.3317632

# Anexo 1 Valor

O conceito *perceived value* endereça a opinião do cliente relativamente a um produto, sendo por norma influenciada pela forma como o produto corresponde às necessidades e expetativas do cliente. Salienta-se que diferentes clientes podem ter uma perceção de valor diferente, para o mesmo produto. O conceito é derivado da avaliação de utilidade de um determinado produto, consoante as perceções do que é recebido e do que é dado (Zeithaml, 1988).

Na tabela 56 apresentam-se os benefícios e sacrifícios para o cliente. O cliente para este projeto de mestrado é a empresa Devscope, atuante em diferentes áreas de negócio, que tem evidenciado dificuldade no balanceamento de requisitos de desempenho, disponibilidade e escalabilidade.

Tabela 56 - Benefícios e sacrifícios para o cliente

| **Benefícios** | **Sacrifícios** |
|---|---|
| Perceção de diferentes abordagens de microsserviços orientadas a eventos.<br><br>Perceção da aplicabilidade das provas de conceito tendo em conta os requisitos de desempenho e de escalabilidade, tendo em consideração a tolerância a falhas.<br><br>Perceção de métricas que permitem averiguar a qualidade do software produzido. | Custo da resistência de mudança de abordagem.<br><br>Adaptação a tecnologias não usuais no dia-a-dia do cliente. |

Assim, conclui-se que a realização deste projeto possui benefícios e sacríficos para o cliente.

## Proposta de valor

Através da proposta de valor pretende-se dar a conhecer o conjunto de produtos e serviços que despoletam valor para os clientes (Osterwalder, 2004).

Com o objetivo de se descrever o valor produzido para o cliente, a ferramenta *Value Proposition Canvas* foi adotada (Strategyzer, 2017), assim foi possível definir os perfis do cliente de uma forma precisa e visualizar o valor criado pelo produto a oferecer.

O perfil do cliente é alcançado através da concretização dos seguintes pontos:

- *Gains*: aqui descreve-se os objetivos que os clientes gostariam de atingir;
- *Pains*: aqui descreve-se os riscos e obstáculos relativamente aos objetivos a alcançar;
- *Customer jobs*: aqui descreve-se o que é que os clientes estão a tentar alcançar no seu dia-a-dia.

A criação de valor é alcançada através da concretização dos seguintes pontos:

- *Gains Creators*: aqui descreve-se como é que o produto cria ganhos ao cliente;
- *Pain Relievers*: aqui descreve-se como é que o produto alivia os receios do cliente;
- *Products and Services*: aqui enumeram-se os produtos a entregar ao cliente.

A figura 67 evidencia a proposta de valor *canvas* alcançada, através do preenchimento dos tópicos anteriormente referidos.

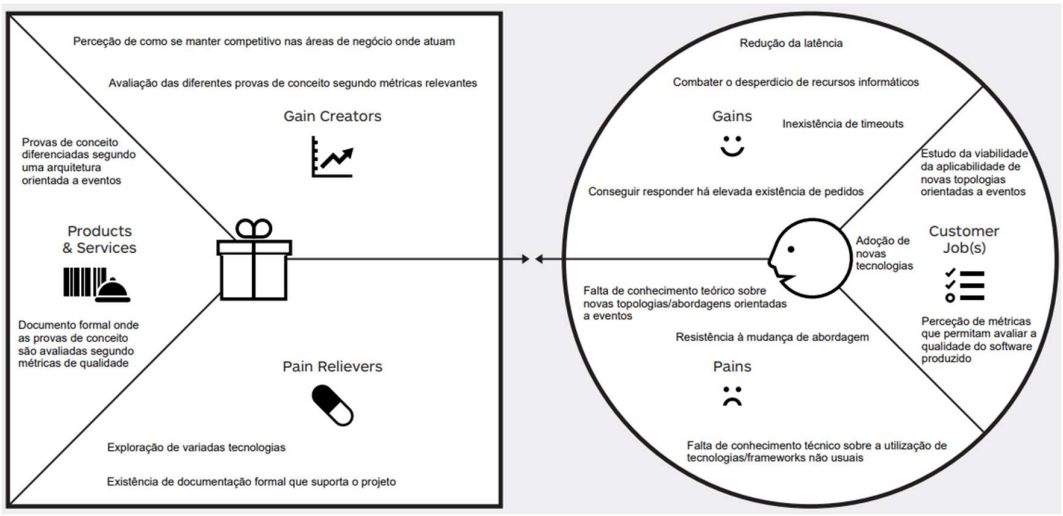



Figura 67 - Proposta de valor canvas alcançada