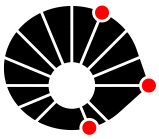

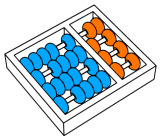

Estudo sobre Antipadrões em microsserviços e os impactos na evolução do processo de desenvolvimento de aplicações

D. H. P. Oliveira *B. B. N. França*

Relatório Técnico - IC-PFG-20-05
Projeto Final de Graduação
2020 - Agosto

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Estudo sobre Antipadrões em microsserviços e os impactos na evolução do processo de desenvolvimento de aplicações

Daniel Helu Prestes de Oliveira* Breno Bernard Nicolau de França*

Resumo

O estilo arquitetural em microsserviços, é um dos estilos mais utilizados no contexto de entrega contínua e que, devida a sua recente criação, é preciso definir formas de se identificar, de forma automatizada, padrões que representem más práticas de design e que, portanto, não devem seguidos. Este trabalho apresenta um estudo sobre antipadrões na arquitetura de microsserviços. Com base no estudo de literatura, desenvolvemos uma forma automatizada de identificar os dois anti padrões mais frequentes que são *Megaservice* e *Nanoservice*, analisando apenas o código fonte. Aplicamos a análise em um projeto *open source* chamado *SiteWhere*, que utiliza a arquitetura de microsserviços, e fomos capazes de identificar seis microsserviços, entre os 11 existentes no projeto, que apresentaram as características dos antipadrões. Esta seleção permitiria que desenvolvedores do projeto analisassem esses microsserviços e tomassem as ações que julgassem necessárias para remediar o problema.

1 Introdução

Microsserviços é o nome dado ao estilo de arquitetura no qual uma aplicação é estruturada como uma coleção de serviços altamente testáveis e manuteníveis (*maintainable*), lançados (*deployed*) independentemente, organizados em torno de capacidades de negócio e gerenciados por times pequenos. Isso permite a entrega rápida, frequente e confiável de aplicações complexas.[1]

Sendo um dos estilos mais utilizados no contexto da prática de entrega contínua, uma prática de desenvolvimento imprescindível atualmente no mercado, e com grandes organizações como Netflix e Amazon utilizando essa arquitetura, torna-se importante analisar padrões de degradação que podem ocorrer no processo de desenvolvimento de aplicações e suas consequências.

O conceito da arquitetura de microsserviços surgiu por volta de 2011 e por isso ainda estão sendo descobertos comportamentos que se repetem em diferentes projetos e que, a medida que o projeto cresce, eles podem impactar de maneira positiva ou negativa o desenvolvimento como um todo. Dito isso, esse projeto tem como objetivo o estudo de padrões considerados problemáticos (antipadrões), que ferem os princípios deste estilo de arquitetura

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

e que podem impactar negativamente a produtividade, a qualidade do produto e o desenvolvimento de um negócio. Juntamente com esse estudo teórico, o trabalho busca criar um método para identificar os antipadrões automaticamente para auxiliar os desenvolvedores a manterem as boas práticas da entrega contínua.

Esse relatório está organizado da seguinte forma: a Seção 2 apresenta a fundamentação teórica, explicando mais sobre a arquitetura de microsserviços e os três tipos de antipadrões que analisamos neste trabalho: *Megaservice*, *Nanoservice* e *Chatty Service*. A Seção 3 apresenta os métodos, mostrando como foi desenvolvido nosso estudo. Na Seção 4, apresentamos o algoritmo desenvolvido e utilizado para a identificação dos antipadrões em código-fonte. A Seção 5, apresenta e analisa os resultados obtidos a partir da execução do nosso programa em um projeto de código aberto. A Seção 6 apresenta as limitações e desafios encontrados durante o desenvolvimento do projeto. A Seção 7 traz as conclusões do nosso projeto e possíveis trabalhos que podem ser feitos futuramente. O apêndice A mostra o algoritmo que foi desenvolvido para identificar os antipadrões.

2 Fundamentação Teórica

2.1 Arquitetura de Microsserviços

Um serviço é uma aplicação que se comunica pelo uso de mecanismos como chamada de serviços web ou procedimento remoto. Na Figura 1, podemos observar a estrutura de um serviço [1], criado em torno da lógica de negócio e, para comunicação, ele utiliza uma API (*Application Programming Interface*) que expõe suas operações, as quais podem ser síncronas ou assíncronas. Ainda, o serviço pode publicar ou receber eventos de alterações de dados vindo de outros serviços. Normalmente, cada serviço possui a sua própria base de dados para evitar problemas de dependências com outros serviços. Da perspectiva de um agente externo, a API representa a interface que expõe as operações dos serviços.

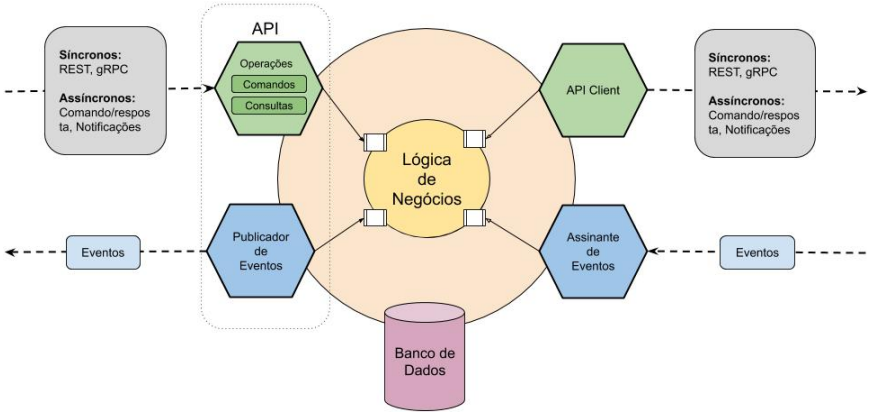

Figura 1: Estrutura de um serviço (adaptado de [1]).

A arquitetura de microsserviços é o estilo de arquitetura na qual uma aplicação é estruturada como uma coleção de serviços testáveis e manuteníveis (*maintainable*), implantados

(*deployed*) independentemente, organizados em torno de capacidades de negócio e gerenciados por times pequenos. A intenção é que essas características permitam a entrega rápida, frequente e confiável de aplicações complexas. Na Figura 2, podemos ver um exemplo de uma arquitetura de microsserviços como exemplificado no livro "*Microservice Patterns*" de Chris Richardson [2].

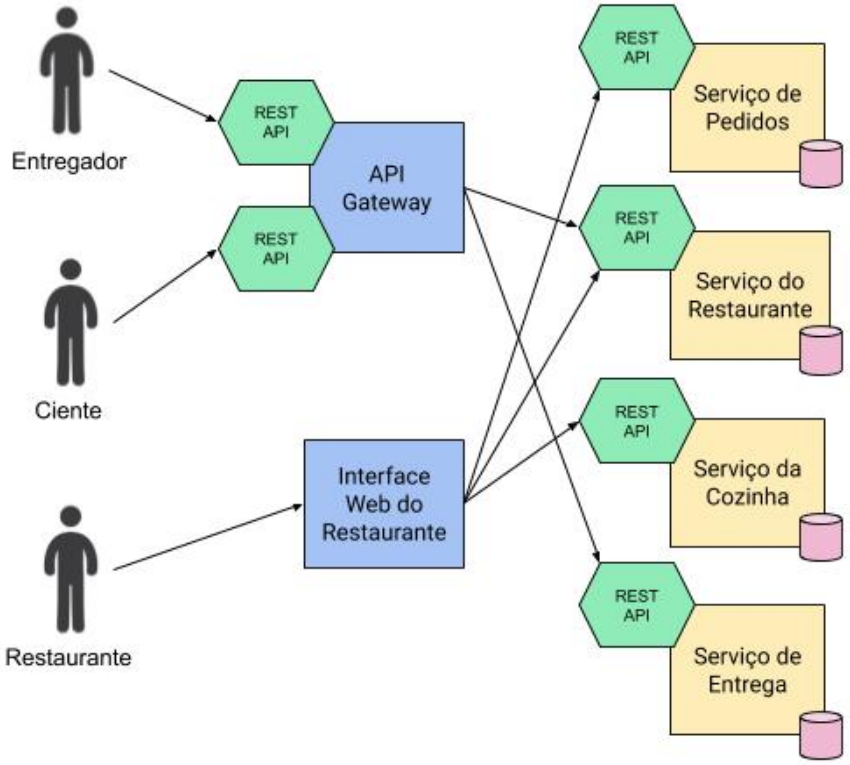

Figura 2: Exemplo de arquitetura de microsserviço (adaptado de [2]).

Na Figura 2, temos um exemplo de uma aplicação que tem como objetivo providenciar um serviço de entregas para redes de restaurante. O projeto possui quatro microsserviços que expõem seus métodos utilizando a REST API e cada um com sua própria base dados. Para os usuários se comunicarem com a plataforma são utilizados uma interface web ou um API Gateway que são capazes de redirecionar os usuários para o serviço correto de acordo com o que é requisitado.

2.2 Antipadrões

Um antipadrão é a utilização recorrente de uma solução de código ou projeto (*design*) que leva a um resultado de baixa qualidade [3]. Esse resultado ruim pode ser observado sob o ponto de vista de performance, código de difícil manutenção, ou até mesmo a falha total do projeto. Existem inúmeros trabalhos que identificaram e analisaram antipadrões em aplicações orientadas a objetos [4].

Neste trabalho, analisaremos os três antipadrões que ocorrem mais frequentemente em arquiteturas de microsserviços, conforme relatado em [5]: *Megaservice*, *Nanoservice* e *Chatty Service*. Para a análise do *Chatty Service*, devido alguns impedimentos (ver Seção 6), fomos capazes de desenvolver a identificação manual do antipadrão, mas não conseguimos fazer a implementação automatizada.

2.2.1 *Megaservice*

Megaservice, também conhecido como *Multiservice*, é um antipadrão que tem como característica um grande número de funcionalidades expostas por um serviço, que abordam diferentes tipos de capacidade de negócio, ou seja, um *Megaservice* acumula mais responsabilidades do que necessário. Devido a grande quantidade de funcionalidades, esse antipadrão torna mais difícil o desenvolvimento do código pelo fato do código se tornar demasiadamente grande e ter uma maior dependência entre si e de elementos externos, o que dificulta o lançamento de novas versões do serviço também. Na Figura 3, podemos ver um *Megaservice* que abstrai toda a lógica por trás de um comércio eletrônico, incluindo processamento de pedidos, processamento de pagamentos e controle de estoque. De forma contrária, uma boa solução de projeto deveria quebrar essas preocupações em módulos distintos.

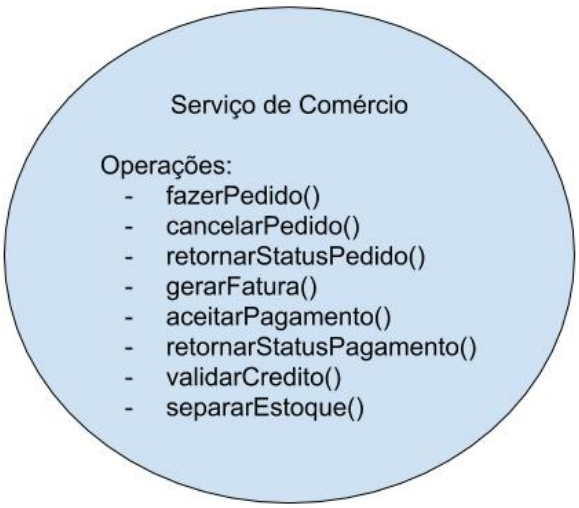

Figura 3: Exemplo de *Megaservice*

2.2.2 *Nanoservice*

Nanoservice, também conhecido como *Tiny Service*, é o antipadrão no qual o serviço possui pouquíssimas operações. Em muitos casos, o serviço oferece somente um subconjunto das operações necessárias para o modelo de negócio, o que faz com que ele precise se conectar a outros serviços para entregar o necessário. Isso incorre em um custo do processo de comunicação devido às chamadas remotas entre os serviços. Caso a única função exposta seja utilizada por uma quantidade grande de serviços, um *Nanoservice* torna-se um gargalo

ao desempenho do sistema de maneira geral. Na Figura 4, podemos observar o exemplo de um modelo negócio que se propõe a gerenciar o processamento de pedidos. Este é composto por três serviços que realizam partes diferentes do processo, porém para funcionar corretamente é necessário que todos os serviços estejam presentes. Neste exemplo, cada um desses serviços representa um *Nanoservice*.

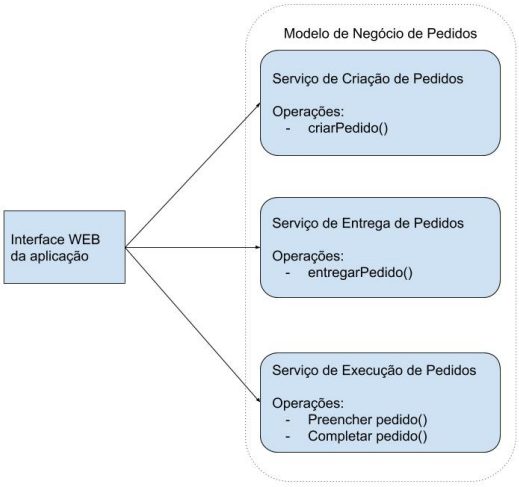

Figura 4: Exemplo de *Nanoservices*

2.2.3 *Chatty Service*

Chatty Service, também conhecido como *Chatty Web Service*, apresenta-se na forma de um serviço que expõe uma quantidade grande de funções, no entanto essas funções são muito simples, o que torna necessária a combinação da chamada de diversas funções pelos clientes desse serviço, de forma a atingir a solução requisitada pelo modelo de negócio. Esse antipadrão tem como consequência um impacto negativo na performance do sistema, uma vez que há um custo associado a cada chamada que é feita para essas funções, e fica sendo responsabilidade do cliente externo desenvolver toda a lógica necessária para adquirir a informação desejada. Na Figura 5, vemos um exemplo de microsserviço que apresenta as características do antipadrão *Chatty Service*. O microsserviço possui métodos públicos muito simples o que faz com que o agente externo precisaria fazer várias chamadas ao serviço para ser capaz de criar uma ordem de um pedido, por exemplo.

3 Métodos

O objetivo do trabalho é estudar antipadrões existentes para a arquitetura de microsserviços e desenvolver uma forma automatizada de detectá-los com base na análise estática do código-fonte. Para atingir essa meta, o trabalho teve como etapa inicial uma revisão da literatura para identificar os trabalhos relacionados ao mapeamento e detecção de antipadrões. Com

Figura 5: Exemplo de um *Chatty Service*

isso, seria possível definir quais antipadrões seriam analisados. Ainda, isso nos ajudou a direcionar o trabalho para os antipadrões que ocorriam com maior frequência e, portanto, maior relevância prática do trabalho.

Definido quais os antipadrões a analisar, estudamos suas características e métodos para a detecção deles. Com isso, criamos um algoritmo para identificá-los com base somente no código-fonte.

Após isso, procuramos um projeto de código aberto que adotasse a arquitetura de microsserviços e que possuísse, pelo menos, mais que cinco microsserviços para que pudéssemos avaliar a solução implementada em uma aplicação real.

Com isso, desenvolvemos um código em python que executasse nosso algoritmo de detecção dos antipadrões. Realizamos então uma análise automatizada e verificamos os resultados manualmente, considerando se os microsserviços retornados pelo nosso programa realmente apresentavam as características dos antipadrões estudados.

4 Desenvolvimento

Como um dos objetivos do trabalho, procuramos encontrar formas de detectar esses antipadrões, em um projeto real, de maneira automatizada. Estudando a literatura [6, 7], vimos que as seguintes heurísticas podem ser utilizadas na identificação de cada um dos antipadrões:

- *Megaservice*
  - Baixa coesão
  - Alta quantidade de operações expostas
  - Alto tempo de resposta
  - Baixa disponibilidade

- *Nanoservice*
  - Baixa quantidade de operações expostas
  - Alta dependência de outros serviços
- *Chatty Service*
  - Alta coesão
  - Baixo número de parâmetros nas operações
  - Alto número de parâmetros primitivos
  - Alta quantidade de invocação de métodos
  - Altíssimo acesso aos dados do serviço

Para este trabalho, utilizando essas heurísticas como base, focamos em encontrar uma maneira automatizada de identificar esses antipadrões utilizando apenas análise estática (com base no código-fonte). Então, desconsideramos as características que só poderiam ser calculadas em tempo de execução como por exemplo alto tempo de resposta. Ainda, para automatizar a detecção, precisamos definir o que consideramos como "alto"e "baixo"para cada um desses parâmetros. Neste caso, escolhemos a análise estatística por *Boxplots*, também conhecido como Diagrama de Caixa, com base na ideia utilizada em um outro trabalho de identificação de antipadrões em aplicações seguindo uma arquitetura orientada a serviços (SOA) [8].

O *Boxplot* permite avaliar a distribuição dos dados de forma que o quartil inferior (ou primeiro quartil) corresponde a 25% das menores medidas e o quartil superior (ou terceiro quartil) corresponde a 75% das menores medidas. Para essa análise, consideramos como "baixo"qualquer valor dentro do primeiro quartil e como "alto"qualquer valor acima do terceiro quartil. Assim, foi possível definir os seguintes algoritmos para encontrar os antipadrões no código-fonte:

- *Megaservice e* Nanoservice
  1. Contar o número de funções expostas por todos os microsserviços do projeto;
  2. Gerar o *Boxplot* para o total de microsserviços a partir da quantidade de operações de cada microsserviço;
  3. Microsserviços acima do terceiro quartil do *boxplot* são os candidatos a *Megaservice*; e os microsserviços que pertencem ao primeiro quartil são os candidatos a *Nanoservice*.
- *Chatty Service*
  1. Contar o número de funções expostas por todos microsserviços do projeto, considerando a quantidade de instruções de cada um dos métodos (devem ser contabilizados apenas os métodos com poucas instruções);
  2. Gerar o *Boxplot* da mesma forma que no caso anterior;

3. Os microsserviços que estão acima do terceiro quartil são candidatos a *Chatty Service*.

Posteriormente, buscamos encontrar um projeto de código aberto que adotasse uma arquitetura de microsserviços, com uma quantidade razoável (pelo menos cinco) de microsserviços, para que fosse possível colocar em prática essa análise. Encontramos a aplicação *SiteWhere v2.1.1* [9], um sistema *Open Source* que facilita o armazenamento, processamento e integração de dispositivos *IoT (Internet of Things)*. Essa plataforma utiliza a arquitetura de microsserviços e utiliza tecnologias como *Kubernetes* para orquestração de *containers*, *Istio* para o gerenciamento do ambiente operacional e balanceamento de carga, e *Kafka* para comunicação assíncrona entre serviços. Ela é composta de microsserviços desenvolvidos em Java utilizando o *framework Spring Boot* [10].

Para a nossa análise, focamos nos protocolos síncronos de comunicação existentes na aplicação *SiteWhere*, sendo eles o gRPC, utilizado para a comunicação entre os microsserviços da aplicação, e o REST, para a comunicação com agentes externos à plataforma. Como veremos na Seção 6, nosso programa foi capaz de identificar somente as operações expostas pelo protocolo gRPC devido algumas limitações encontradas.

Desenvolvemos um programa em python (o algoritmo pode ser visto no apêndice A) que recebe como entrada uma *string*, que representa a implementação da interface gRPC, neste caso "*implements IGrpcApiImplementation*", e a partir dela foi possível identificar as classes que expõe os métodos de cada microsserviço. Essas são as classes que passaram pela análise citada acima. Os resultados encontram-se na seção 5.

5 Resultados e Análise

Considerando a busca pelos componentes expondo os serviços (incluindo via gRPC), encontramos as seguintes classes expondo os métodos de cada microsserviço:

- sitewhere/service-instance-management/src/main/java/com/sitewhere/microservice/grpc/instance/**InstanceManagementImpl**.java
- sitewhere/service-instance-management/src/main/java/com/sitewhere/microservice/grpc/tenant/**TenantManagementImpl**.java
- sitewhere/service-instance-management/src/main/java/com/sitewhere/microservice/grpc/user/**UserManagementImpl**.java
- sitewhere/service-device-state/src/main/java/com/sitewhere/microservice/grpc/**DeviceStateImpl**.java
- sitewhere/service-event-management/src/main/java/com/sitewhere/microservice/grpc/**EventManagementImpl**.java
- sitewhere/service-asset-management/src/main/java/com/sitewhere/asset/grpc/**AssetManagementImpl**.java

- sitewhere/sitewhere-microservice/src/main/java/com/sitewhere/microservice/management/**MicroserviceManagementImpl**.java
- sitewhere/service-label-generation/src/main/java/com/sitewhere/microservice/grpc/**LabelGenerationImpl**.java
- sitewhere/service-schedule-management/src/main/java/com/sitewhere/microservice/grpc/**ScheduleManagementImpl**.java
- sitewhere/service-batch-operations/src/main/java/com/sitewhere/microservice/grpc/**BatchManagementImpl**.java
- sitewhere/service-device-management/src/main/java/com/sitewhere/microservice/grpc/**DeviceManagementImpl**.java

Executando a análise para Megaservice e Nanoservice, obtivemos os resultados mostrados na Tabela 1 e o BoxPlot apresentado na Figura 6.

Tabela 1: Número de métodos para cada microsserviço e candidatos a antipadrão.

Microsserviço	***Número de métodos expostos***	***Antipadrão***
InstanceManagement	3	Nanoservice
TenantManagement	7	Nanoservice
DeviceState	7	Nanoservice
MicroserviceManagement	8	N/A
LabelGeneration	11	N/A
ScheduleManagement	11	N/A
BatchManagement	11	N/A
AssetManagement	13	N/A
UserManagement	16	Megaservice
EventManagement	17	Megaservice
DeviceManagement	84	Megaservice

Com base na Figura 6, temos que a análise considerou como "baixos"valores menores ou iguais a 7 (arredondamento de 7,25) e "altos"os que fossem maiores ou iguais a 16 (arredondamento de 15,25).

Analisando a quantidade de métodos expostos contabilizados para os microsserviços de *DeviceManagement* e *InstanceManagement*, fica claro que estes representam os antipadrões de *Megaservice* e *Nanoservice*, respectivamente.

Já os microsserviços *TenantManagement* e *DeviceState* apresentam somente funções básicas de criação, remoção, atualização e leitura (*CRUD*), o que necessita uma análise mais detalhada por parte dos desenvolvedores do projeto para avaliar se esses microsserviços poderiam ser classificados como *Nanoservice* ou se eles contêm as funcionalidades necessárias para a abstração de negócio.

No caso do microsserviço *UserManagement*, além das funções básicas (CRUD de usuarios), identificamos também funções ligadas a parte de autenticação de usuários, o que

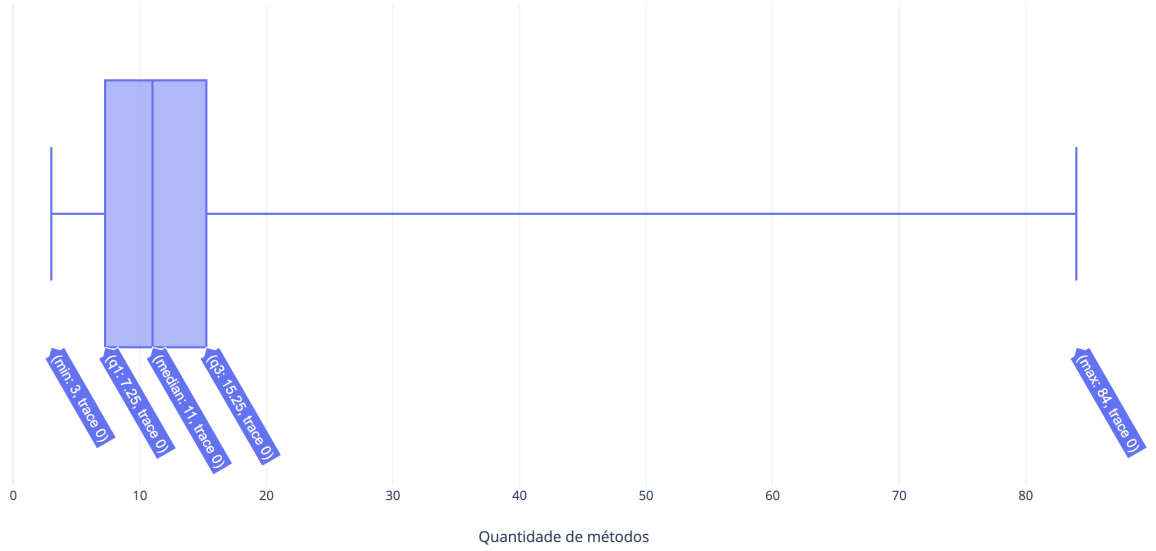

Figura 6: *BoxPlot* da contagem dos métodos expostos dos microsserviços com protocolo gRPC.

pode representar uma acumulação indevida de responsabilidades definidas para este microsserviço, caracterizando-o como um *Megaservice*. Já no caso do *EventManagement*, todos os métodos parecem pertinentes às interações básicas com as dados de eventos, o que pode significar que temos um falso positivo, que é possível dada a ausência de análise semântica no algoritmo. Ainda, é possível que esse serviço seja particionado em outros, gerando especialidades no gerenciamento de eventos. Para ambos os casos, também seria necessária investigação adicional por parte dos desenvolvedores.

Com os resultados obtidos pela a nossa análise automática, o desenvolvedor precisaria olhar somente parte de seus microsserviços para confirmar se eles possuem os respectivos antipadrões conforme apontado pelo algoritmo e para os casos em que isso é verdade, ele poderia, por exemplo, realizar as seguintes ações: para os *Megaservice*, separar as responsabilidades que não cabem àquele microsserviço em um novo microsserviço ou serviços existentes, se pertinente; para os *Nanoservice*, considerar transformar cada um deles em uma biblioteca, que seria adicionada aos serviços que o utilizam.

6 Limitações e Desafios

Ao longo do desenvolvimento desse projeto, encontramos algumas dificuldades. Como optamos pela análise estática, baseada unicamente no código-fonte, nossa análise fica dependente das escolhas estruturais do programador. Como existem diferentes formas de modelar um projeto e dificilmente é seguido um mesmo padrão de codificação em diferentes projetos, torna-se complicado desenvolver uma ferramenta capaz de detectar padrões automaticamente, porque há forte dependência de aspectos tecnológicos e do contexto de cada projeto em particular.

Além disso, é difícil ou inviável mapear os métodos expostos para suas respectivas realizações no caso destes serem definidos por interfaces. Assim, sem compilar o código não se pode decidir quais métodos devem ser analisados, por exemplo, considerando a sobrescrita de métodos, interfaces cuja implementação é decidida apenas em tempo de compilação, entre outros. Isso inviabiliza a análise para o *Chatty Web Service*, por exemplo, pois esta requer a contagem do número de instruções executadas dentro de cada método, e como podem existir chamadas a outros métodos no corpo da função, seria necessário contabilizar também as instruções presentes nas definições desses métodos, o que exigiria o mapeamento descrito anteriormente.

Pelo mesmo motivo, também não foi possível realizar a análise de antipadrões para o protocolo REST, já que os métodos eram expostos de maneira indireta com uso de funções locais que chamavam os métodos dos microsserviços. Por exemplo, como mostrado na Figura 7, temos a função *createArea* que é exposta utilizando REST API que possui somente uma linha, mas essa instrução chama outra função chamada *createArea* que é externa a classe. Para classificarmos a qual microsserviço essa função está ligada, precisaríamos identificar a classe que implementa a interface que contém esse método, o que como explicamos acima só é decidida durante a compilação do código, isso impossibilitou a análise do protocolo REST e da mesma forma do *Chatty Service*.

```
96      /**
97       * Create a new area.
98       *
99       * @param input
100      * @return
101      * @throws SiteWhereException
102      */
103     @PostMapping
104     @ApiOperation(value = "Create new area")
105     public IArea createArea(@RequestBody AreaCreateRequest input) throws SiteWhereException {
106         return getDeviceManagement().createArea(input);
107     }
```

Figura 7: Exemplo de função exposta utilizando protocolo REST.[1]

7 Conclusões e Trabalhos Futuros

Neste projeto, estudamos três antipadrões no contexto de arquiteturas em microsserviços. Abordamos os impactos negativos que os antipadrões *Megaservice*, *Nanoservice* e *Chatty Web Service* podem causar no desenvolvimento de software e estudamos maneiras de detectar esses antipadrões utilizando abordagens automatizadas.

Além disso, fomos capazes de automatizar, analisando apenas o código-fonte, a identificação de candidatos aos antipadrões *Megaservice* e *Nanoservice* nos serviços implementa-

[1] https://github.com/sitewhere/sitewhere/blob/sitewhere-2.1.1/service-web-rest/src/main/java/com/sitewhere/web/rest/controllers/Areas.java

dos no projeto *SiteWhere*, considerando os microsserviços que se comunicam utilizando o protocolo gRPC.

Dados os desafios encontrados durante o projeto, uma maneira de dar continuidade seria estudando formas de tornar essa análise mais tolerante a diferenças de estruturação do projeto/código, o que facilitaria a identificação de outros antipadrões e tornaria possível a utilização da ferramenta desenvolvida em uma gama maior de projetos. Outra possibilidade seria estudar maneiras de mapear as funções sem que seja necessário compilar o código.

Ainda, é interessante verificar a viabilidade da detecção de outros antipadrões para arquiteturas em microsserviços que não foram discutidos aqui, mas que já foram categorizados em outros estudos [5].

Referências

[1] CHRIS RICHARDSON CONSULTING INC., "What's a service - part 1?," 2020. Available at: http://chrisrichardson.net/post/microservices/general/2019/02/16/whats-a-service-part-1.html Accessed in: August, 19th, 2020.

[2] C. Richardson, *Microservices patterns*. Manning Publications Company, 2018.

[3] B. Dudney, S. Asbury, J. K. Krozak, and K. Wittkopf, *J2EE antipatterns*. John Wiley & Sons, 2003.

[4] G. Suryanarayana, G. Samarthyam, and T. Sharma, *Refactoring for software design smells: managing technical debt*. Morgan Kaufmann, 2014.

[5] J. Bogner, T. Boceck, M. Popp, D. Tschechlov, S. Wagner, and A. Zimmermann, "Towards a collaborative repository for the documentation of service-based antipatterns and bad smells," in *2019 IEEE International Conference on Software Architecture Companion (ICSA-C)*, pp. 95–101, IEEE, 2019.

[6] N. Moha, F. Palma, M. Nayrolles, B. J. Conseil, Y.-G. Guéhéneuc, B. Baudry, and J.-M. Jézéquel, "Specification and detection of soa antipatterns," in *International Conference on Service-Oriented Computing*, pp. 1–16, Springer, 2012.

[7] D. Taibi, V. Lenarduzzi, and C. Pahl, "Microservices anti-patterns: A taxonomy," in *Microservices*, pp. 111–128, Springer, 2020.

[8] F. Palma, N. Moha, G. Tremblay, and Y.-G. Guéhéneuc, "Specification and detection of soa antipatterns in web services," in *European Conference on Software Architecture*, pp. 58–73, Springer, 2014.

[9] GitHub, Inc., "Sitewhere v2.1.1," 2020. Available at: https://github.com/sitewhere/sitewhere/tree/sitewhere-2.1.1 Accessed in: August, 19th, 2020.

[10] VMware, Inc., "Spring boot," 2020. Available at: https://spring.io/projects/spring-boot Accessed in: August, 19th, 2020.

A Algoritmo

Algoritmo 1: Identificar *Megaservice* e *Nanoservice*

Entrada: *String* que identifica a implementação da interface gRPC
Saída: Microsserviços que apresentaram características dos antipadrões *Megaservice* ou *Nanoservice*

```
// Inicialização
var gRPCID = entrada;
var contador, servico;
array arquivos[], megaservices[], nanoservices[];
map servicos;
para todo arquivo presente no código-fonte faça
    se Arquivo possui gRPCID então
        arquivos.add(arquivo);
    fim
fim
para todo arquivo presente em arquivos faça
    contador = 0;
    servico = identificar qual microsserviço esse arquivo pertence;
    contador = número de métodos que são expostos no arquivo;
    se contador != 0 então
        servicos[servico] = contador;
    fim
fim
var quartis = Montar Boxplot com o conjunto servicos;
para todo servico presente em servicos faça
    se servicos[servico] ≥ quartis[3] então
        megaservices.add(servico);
    fim
fim
para todo servico presente em servicos faça
    se servicos[servico] ≤ quartis[1] então
        nanoservices.add(servico);
    fim
fim
retorna megaservices, nanoservices
```