Towards a Resilient Spatial Data Infrastructure

Helisson Luiz do Nascimento[1], **Cláudio de Souza Baptista**[1], **Fabio Gomes de Andrade**[2], **Leanderson Coelho dos Santos**[2]

[1]Universidade Federal de Campina Grande (UFCG)
Caixa Postal 58109-970-Campina Grande-PB-Brazil

[2]Instituto Federal de Ciência de Tecnologia da Paraíba (IFPB)

helisson@copin.ufcg.edu.br, baptista@computacao.ufcg.edu.br

fabio@ifpb.edu.br

leanderson.coelho@academico.ifpb.edu.br

Abstract. *Spatial Data Infrastructures (SDI) has contributed expressively to the discovery and sharing of geospatial data. However, with the big number of geo-services available in SDI catalogs, and its sparse description, it is difficult to monitor the availability during the consumption of these resources. Aiming to overcome this limitation, this paper proposes an architecture that defines resilience layers in SDI context, with a Circuit Breaker pattern implementation for retrieving data even during instability. This architecture delivers a novel way of reliable access to resources and spatial data from the SDI catalogs.*

Resumo. *Infraestruturas de Dados Espaciais (IDE) têm contribuído de forma expressiva para a descoberta e o compartilhamento de dados. Porém, com o grande número de geo-serviços disponíveis nos catálogos das IDE, e sua descrição esparsa, é difícil monitorar a disponibilidade durante o consumo desses recursos. Visando superar essa limitação, este trabalho propõe uma arquitetura que define camadas de resiliência no contexto de IDE, com uma implementação do padrão Circuit Breaker para a recuperação de dados mesmo durante instabilidade. Esta arquitetura oferece uma nova forma de acesso confiável a recursos e dados espaciais dos catálogos de SDI.*

1. Introduction

The development of Spatial Data Infrastructures (SDI) has contributed expressively to the discovery and sharing of geospatial data. The implementation of these infrastructures has, among its primary issues, political and budgetary challenges [Grus et al. 2011]. Hence, issues related to their architectures have remained in the background for a long time. Over the years, most SDIs have been implemented based on the Service Oriented Architecture (SOA) standard, following the guidelines and standards defined by the Open Geospatial Consortium (OGC). The SOA architectural approach proposes that web systems must be broken into web services focused on business logic [Krafzig et al. 2005]. The increasing number of applications based on cloud SOA has enabled developers to refine concepts and establish architectural, technical, and organizational standards for the development of service-oriented applications. SDIs based on this architectural model have been conceived as monolithic structures [Assis et al. 2019], in which a set of applications run under a single process using a single module.

Nevertheless, SOA has remained a very broad concept, interpreted in different ways by different organizations. Usually, it has been related to a group of medium complexity services, which access the same database and communicate through an ESB (Enterprise Service Bus). This has inserted bottlenecks and points of failure in web applications.

The Microservice-Based Architecture has emerged as an alternative for dealing with the challenges identified over the maturing years of the concepts encompassed in SOA [Soldani et al. 2018]. Microservices can be defined as a group of small, autonomous services that work together [Newman 2015].

In a monolithic application, if an important functionality fails, the whole application stops working, causing an availability failure. In turn, microservice-based distributed architecture prevent applications from becoming completely unavailable. However, when an application is distributed in several microservices, many problems and possibilities of failures, which do not exist for monolithic architecture, require attention and control. With the development of applications with architectures based on microservices, large companies have used the resources and standards to structure their systems, as is the example of Netflix, which developed a framework for managing resilient architectures based on microservices, the Netflix Open Source Software [OSS 2020]. Features like Hystrix, Eureka, and Zuul make up the Netflix OSS framework and can implement patterns like Circuit Breaker, Service Discovery, and API Gateway that would empower distributed applications to achieve resilience and scalability.

The Circuit Breaker pattern [Montesi and Weber 2016] is one of the primary strategies to deal with the recovery of unavailable service requests. When a resource is unavailable, the Circuit Breaker acts as a proxy, similar to a tripped Circuit Breaker, throwing the exception immediately. This exception can be handled with a function that retrieves alternative resources. The Circuit Breaker can represent an interesting way to deal with the unavailability of services, by allowing to manage the effects of unavailability.

The amount of geo-services offered in SDI catalogs makes the task of monitoring the availability of these services very complex, requiring a method of dynamic integration of geo-services that provides a resilience layer. Microservice patterns can achieve some of these resilience requirements.

Recently, some authors have developed SDIs based on a microservice architecture [Assis et al. 2019, Mena et al. 2019, Li 2019]. In their works, they have extended the capabilities of the services provided by the infrastructure in terms of scalability, better use of cloud resources, and orchestration of container instances. However, there are still important issues that have not been addressed in the literature for the development of these infrastructures, such as resilience and fault tolerance.

In an SDI, it is expected that different services provide similar features about the same place or theme. Nevertheless, when a feature that is being used becomes unavailable for some reason, the client is in charge of searching the SDI's catalog to find a service that supplies a similar feature that could replace it. Since current infrastructures do not keep information about feature similarity, as well many services are poorly described in the catalog service, this task can be quite tedious and time-consuming. This problem can be especially critical in applications such as environmental monitoring and disaster management, in which real time decisions are due.

Aiming at solving these limitations, we propose an architecture that enables the implementation of resilient spatial data infrastructures. To validate our solution, we conducted a case study based on the Brazilian National Spatial Data Infrastructure (INDE). In this paper, we present a method for adding geo-services as vertices of a resilient architecture based on microservices. Using a Circuit Breaker implementation to manage unavailable resources, this architecture provide reliable access to SDI catalogs resources.

The remainder of this paper is structured as follows. Section 2 addresses related works. Section 3 presents our SDI architecture. Section 4 focuses on some use case scenarios. Finally, section 5 concludes the paper and points out further research to be undertaken.

2. Related Work

Over the years, several authors have analyzed the way in which SDIs are being implemented from an architectural point of view. Thus, some works proposed SDI implementations using the SOA architecture as an approach to the evolution of each service that makes up the infrastructure. Friis-Christensen et al. (2006) propose an SDI prototype aimed at assessing areas of damage caused by fires using SOA. Oliveira et al. (2008) proposed the application of SOA concepts by evolving a municipality GIS to a local SDI. However, the authors concluded that the implemented model did not perform satisfactorily. Basanow et al., (2008) used SOA principles to develop an SDI for 3D data. However, the analyzed services orchestration principles did not meet the complexity ranges that the system could reach. Likewise, Barik et al. (2016) also applied SOA principles to an SDI for the tourism sector in the city of Bhubaneswar, India. The authors detailed a methodology for building their own geospatial database. However, aspects of using data services from other SDI have not been addressed.

In general, based on initiatives led by the OGC [Friis-Christensen et al. 2006], many of the implemented SDI have sought to comply with at least some of the SOA principles. However, even complying with these principles, most applications do not reach minimum current requirements on scalability [Scholten et al. 2006], performance and availability [Soldani et al. 2018].

We observe that SOA has reached an important step forward in the systems complexity, as is the case of a cloud SDI. However, in a cloud environment, the microservice architecture achieves better performance than SOA. Several authors approach this topic as an architecture to enable SDI in the cloud. [Krämer 2018], for example, introduced a native cloud GIS proposal built on a microservice-based architecture, in order to process large volumes of distributed geospatial data. [Schäffer et al. 2010] carried out a study on the feasibility of implementing a cloud SDI. The authors identified some barriers to make this transition, including budget and legal difficulties.

[Li 2019] proposed a four-layer microservices architecture for public waterway information services. The proposed four layers are: data layer; microservices layer; application layer and client layer. Although they implemented a web application based on the microservice architecture, the authors did not explore availability and resilience issues.

Another application of microservices in SDI was proposed by [Assis et al. 2019] by presenting TerraBrasilis, an infrastructure for analyzing geospatial data on deforestation. In this proposal, the authors developed an SDI optimized for data analysis using a microservice-based architecture. The proposed solution performs real-time monitoring of system services in virtualized containers [Docker 2020], which facilitates scalability, enables the availability of resources and protects the system from potential external attacks. The system uses the agility of the microservice architecture to provide a geospatial data analysis platform regarding deforestation in the Brazilian cerrado, and, therefore, is focused on a particular domain. The authors deal with fault tolerance and availability for SDI using services available in virtualized computing environments in an IaaS (Infrastructure as a Service) platform. However, a more in-depth solution for handling unavailable services that goes beyond managing instances and service states is not addressed.

When implementing a cloud SDI using the microservice architecture, several possibilities arise for deployment automation, easy integration, as well as scalability. However, distributing an application into several microservices introduces some challenges such as the low reliability of the network. In particular concerning SDI, it is also necessary to take into account the possibility of unavailability of the data services that compose the SDI catalog. Therefore, in order to fully exploit the possibilities of cloud computing under microservice-based architecture, the use of resilience and fault tolerance patterns as Circuit Breakers is of fundamental importance.

The handling of exceptions to redirect calls to services is something totally dependent on the business logic where the Circuit Breaker pattern is used [Nygard 2018]. In the context of applications such as SDI, it is important to prioritize data retrieval in order to improve the underlying decision making process.

Mena et al. (2019) apply microservice patterns to build a resilient application for geospatial data visualization. The application implements the Circuit Breaker pattern through the use of the Netflix OSS Hystrix framework [OSS 2020]. The authors isolated microservices in containers [Docker 2020], which are replicated and in case of unavailability of the microservice, the Ribbon [OSS 2020] is used as a resource to redirect calls to mirrored microservices. However, this work does not implement the OGC geo-services standards, as well the proposed architecture does not perform a scalable monitoring of geo-services availability. The work developed introduces a scalable geospatial data application, but not an SDI.

The solutions proposed in the aforementioned approaches aimed at cases of failure of a call or unavailability of a service that composes the internal architecture of the application. When dealing with the unavailability of external services, the alternative is a data return with low adaptability to maintain the user experience. Table 1 presents a comparison of the aforementioned research works.

The application of resilience patterns in the availability of spatial data resources is still an unexplored theme. The works that applied the microservices-based architecture paradigms to SDI did not address this aspect directly. Therefore, in this paper we introduce an architecture based on microservices that applies resilience standards in the

Table 1. Comparison between geospatial approaches that use service oriented architecture

	Microservice-Based Architecture Applied to SDI	Scalability	Service availability	Resilience
Friis-Christensen et al., (2006)	No	No	No	No
Oliveira et al., (2008)	No	No	No	No
Basanow et al., (2008)	No	No	No	No
R. K. Barik et al., (2016)	No	No	No	No
Li, Y.(2019)	Yes	No	No	No
Mena et al., (2019)	No	Yes	Yes	Yes
Assis et al. (2019)	Yes	No	No	Yes in a given domain
Our proposed solution	Yes	Yes	Yes	Yes

provision of geo-services.

3. The Proposed Architecture

To create a resilient application of geospatial data, it is necessary to build an infrastructure agile enough to solve situations of unavailability with low latency. Figure 1 depicts our high-level architectural model, developed from the concepts explored in the topics aforementioned.

As an SDI aggregates resources from different services, it must be aware of the availability of each of these sources. Dealing with each geo-service available in the SDI catalog as an architectural microservice enables the resilience and recovery patterns available for microservices-based architectures. Thus, we consider each geo-service that compose the SDI catalog as a point of failure within the architecture, seeking to establish strategies to mitigate possible downtime.

The features of an SDI are made available from geo-services, which are implemented based on the OGC standards. One of these standards is the Web Map Service (WMS), which renders and returns images of the available geographic resources [OGC 2020]. In our work, we focused on providing reliable access to WMS services because they are widely used in different applications and contexts. So, they serve as a main entry point for SDI spatial data consumption.

Next subsections detail the functions and the characteristics of each component of the proposed architecture.

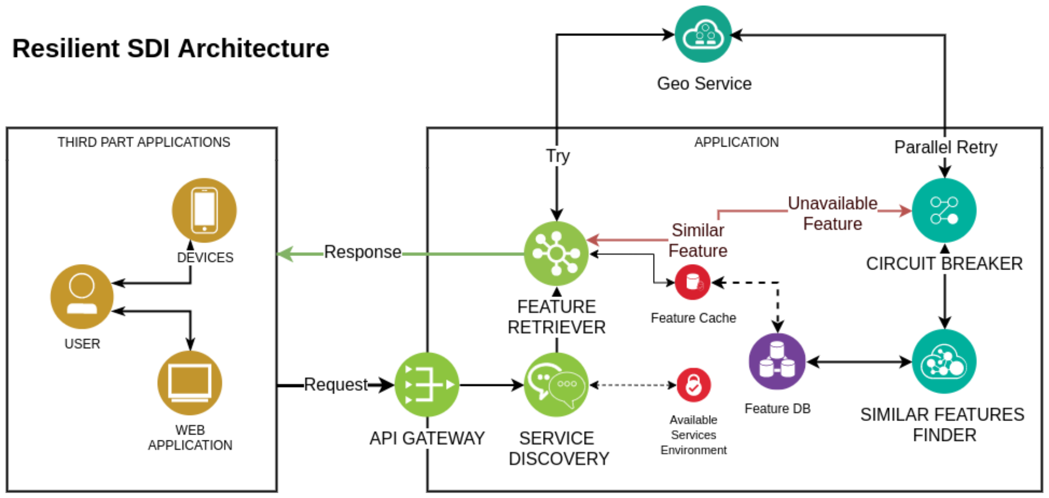

Figure 1. Our Proposed architecture.

3.1. The API Gateway

The *API gateway* performs the management of our application's REST entry-points available for public requests. This component intercepts the requests for services and redistributes them through the application, which processes the necessary actions. The importance of this resource is due to the monitoring, which are the most common paths taken by users and services that use the application. This data is useful to guide the evolution of the architecture, prioritizing the most used resources.

The *API Gateway* is also responsible for performing the load balancing, in which requests are redistributed among the available service instances. To accomplish this task, microservices are instantiated in Docker containers [Docker 2020]. Then, when the application is under a heavy load of requests, it is possible to instantiate several microservices in parallel to balance the load and distribute the requests.

Figure 2 shows the *API Gateway* flow for two different requests. The *API Gateway* checks if the service requested in the request is unavailable. The *API Gateway* accesses the *Available Services Environment* to check if the requested service is unavailable and has been replaced by another one. If applicable, the request is adjusted and forwarded to a *Feature Retriever* instance to get the data.

3.2. The Service Discovery

When an application is distributed on the network, several instability and latency problems may arise among the possibilities of failure. Then, it is always necessary to check which services are available. The primary function of a Service Discovery implementation is to listen and keep information about the microservices operating in the application. When a microservice is successfully instantiated, it is registered with service discovery to receive requests. When a service is unavailable, the Service Discovery is notified, and this service is added to the list of unavailable services until it registers again.

In our architecture, the *Service Discovery* registers the instances of available microservices and operates an *Available Services Environment*. This environment is used by

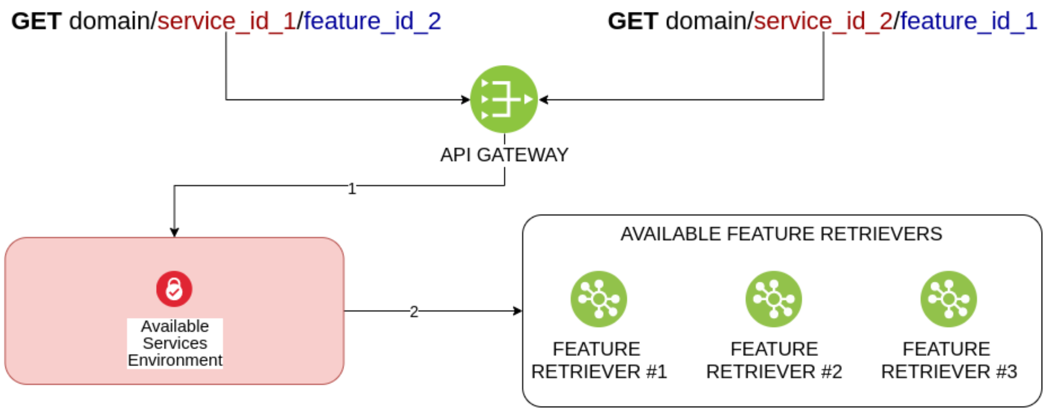
GET domain/service_id_1/feature_id_2
GET domain/service_id_2/feature_id_1
API GATEWAY
1
Available Services Environment
2
AVAILABLE FEATURE RETRIEVERS
FEATURE RETRIEVER #1
FEATURE RETRIEVER #2
FEATURE RETRIEVER #3

Figure 2. API Gateway process.

other fixed microservices of the application as an environment to check if the service that provides a requested resource can be replaced by an available service that has the same resource.

The *Available Services Environment* associates the identification of a service and feature with its similar one currently available. This strategy has been adopted because when dealing with third party services it is not possible to receive registration requests. Hence, only missing services and their available counterparts are registered.

3.3. The Feature Retriever

The *Feature Retriever* seeks to work with an interface that guarantees the safe recovery of data and features of a geo-service with maximum security. This service has only one HTTP GET route as an access point, as the goal is to simplify access to resources as much as possible and facilitate replication in different containers, since this is the main load point of the application.

In the *Feature Retriever* there is a cache layer under a database that contains information about the geo-services and feature types provided by the SDI, as it is shown in Figure 3.

The service poses a query to retrieve the requested feature formatted URL. When the *Feature Retriever* is replicated in several containers, its cache is also replicated, avoiding overloading the main database and maintaining the principle of decentralized access to data.

As is depicted in Figure 3, the *Feature Retriever* gets the access data to a geo-service and performs an attempt to retrieve the data, as requested by the user. If the request receives a successful response, the response is forwarded to the user. If an error occurs when requesting access to the geo-service, the *Feature Retriever* accesses the *Circuit Breaker* to retrieve an available geo-service that has a similar feature, similar data is returned to the user from the *Feature Retriever*.

3.4. The Circuit Breaker

The Circuit Breaker pattern is used as one of the main strategies for handling requests to unavailable services in microservices-based architectures. In this case, the *Circuit Breaker*

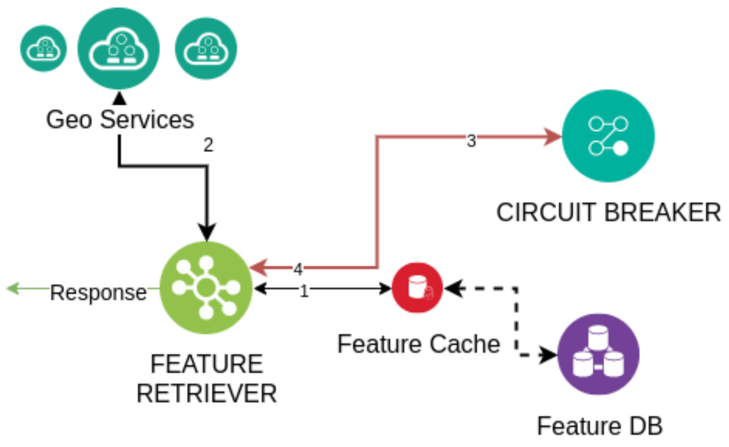

Figure 3. Feature Retriever process.

acts as a proxy, similar to an eletrical breaker that opens the circuit when it detects changes in voltage to avoid unwanted consequences.

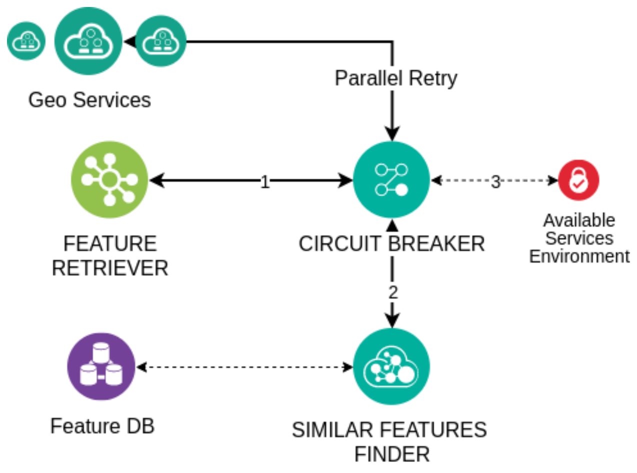

Figure 4. Circuit Breaker process.

In our work, the *Circuit Breaker* aims to keep the flow of spatial data constant for the user. Figure 4 depicts the flow of an unavailable service case. In step one, the *Feature Retriever* alerts the *Circuit Breaker* that a particular service has triggered an exception when trying to access a particular feature. In step 2, the *Circuit Breaker* asks the *Similar Feature Finder* to search for available services that have features similar to the one the user requested (the aim is to keep the user with useful information, even during the unavailability of the service that was requested). After this process, the service with the most similar and available feature is selected. In step 3, the *Circuit Breaker* inserts the available service and the feature of interest in the *Available Services Environment* associated with the service and feature of the original request. Then, alternative data is returned to the user.

If, in step 3, the *Similar Feature Finder* does not find any alternative data, the feature is labeled labeled as unavailable in the *Available Services Environment*. Then a response with a error is returned to the user.

After these steps, the *Circuit Breaker* registers in the *API Gateway* notification service, to be alerted when an user request for the unavailable service, is made again. When this next request is invoked, the circuit switches to Half-Open and a request attempt is made in parallel, without affecting the operation of the rest of the request, if the request is successful, the mention of the faulty service is removed from the *Available Services Environment*, and the circuit returns to Closed, otherwise it remains Open.

3.5. The Similar Features Finder

In our architecture, the *Similar Feature Finder* microservice is in charge of finding features that can replace a feature that became temporarily unavailable for some reason. During this process, it compares this feature to all the features provided by the infrastructure and returns the ones that have a similarity score higher than a predefined threshold. To perform this task, the service extracts three information about the unavailable feature type: the spatial extent, which is identified using its bounding-box, the temporal extent, in cases where temporal expressions can be found in its description, and theme, which is identified using its title.

An important characteristic that hindered the implementation of this microservice is that the catalog service provided by SDIs does not provide information at the level of the feature type. Hence, to overcome this limitation, we had to implement a module that extracts information from the catalog service. This process is performed in four stages. Firstly, it collects all the metadata records registered in the SDI's catalog service. In the second stage, it processes each one of these records and identifies the URL of the OGC web services (WMS and WFS) from which the data can be downloaded. Then, it accesses each one of these services to get information about the feature types they offer. Finally, a subset of the metadata describing the service and its respective feature types are stored in a local database, which is used as the source for finding similar features.

In order to find similar features, we implemented a search engine that uses a set of similarity metrics to estimate the similarity between the feature that is unavailable and each feature type stored in our database. The overall similarity between two feature types is based on three ranking values: spatial, temporal, and thematic. The spatial and temporal rankings are calculated using the approach proposed by Andrade et al. [Andrade et al. 2014]. To accomplish thematic ranking, we generated a document for each feature type containing information such as name, title, description, keywords, and some metadata about the service from which it is offered. These documents are indexed and retrieved using Apache Solr, which is a tool that provides scalable document retrieval. Whenever this tool executes a query, it returns a ranking value for each retrieved document. Then, we consider these values as the thematic ranking of the features related to these documents.

After all the ranking values are calculated, the features that got zero as the result for any of the rankings are discarded. For the remaining features, the similarity score is calculated using the average of the ranking values obtained for each dimension. Finally, the features with a similarity score higher than the threshold are selected, sorted, and returned by the microservice.

4. Case Study

To validate our approach, we implemented a case study based on the Brazilian National Spatial Data Infrastructure (INDE) [BRASIL 2008]. In this section, we demonstrate the application behavior during the open and closed states of the *Circuit Breaker*.

As shown in the top left section of Figure 5, the user makes a request to the application to get a feature of a specific service through its identifiers. In the case analyzed, the user requests a specific feature related to Public Health Equipment in Brazil.

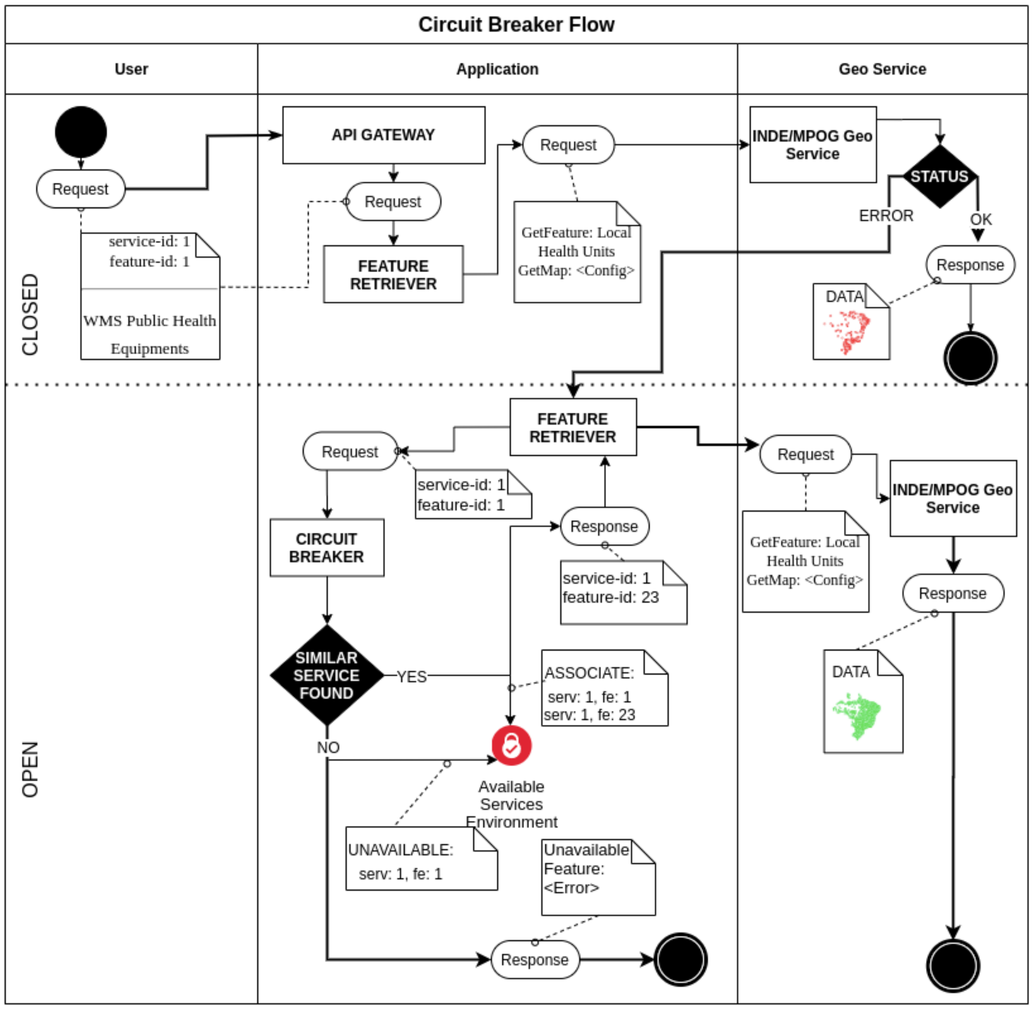

Figure 5. Request flow in our proposed fallback method.

The *API Gateway* collects the request and the filters are performed. Then, the request is forwarded to the microservice for features retrieval. The *Feature Retriever* retrieves the INDE's MPOG (Portuguese Acronym for Ministry of Planning, Budget and Management) geo-service data access, and the feature with public health equipment in Brazil. A request attempt is made, and if the response is obtained successfully, it is returned to the user.

If the *Feature Retriever* catches an error, it changes the *Circuit Breaker* state to Open. The *Feature Retriever Service* asks for a similar feature concerning Public Health Equipment in Brazil. In this case, the *Circuit Breaker* finds an available feature from the same MPOG geo-service. The founded feature has less equipment data, focusing in First Aid places, but it works well as an alternative to an error that can be exposed to the user or cause more cascading errors. The alternative feature is returned to the user. Meanwhile, the *Circuit Breaker* inserts the alternative feature into the *Available Services Environment*. Next attempts to access this feature will be redirected to the alternative feature.

If the *Circuit Breaker* does not find similar features, as shown in the lower center of the Figure 5, the feature is labeled in the *Available Services Environment* as unavailable. When any requested resource is labeled this way, the resilient SDI architecture returns an error response instantly. This reduces the response latency in the next attempts of this feature.

5. Conclusion and Future Work

This paper proposed a microservice based architecture that is able to to deal with the unavailability of geospatial data in SDI, maintaining the availability of applications that depend on this data. The proposed solution implements resilience patterns, providing a layer of reliability for users and applications that need high data availability. To make alternative data available, it relies on a microservice that evaluates the similarities between features.

As future work, we plan to implement a solution for detecting the semantic relationships between features using Natural Language Processing, including Named Entity Recognition. Such extensions would make our architecture able to perform deeper analysis of semantic relationship between features types, leading to better results in finding alternative data for fallback. We also intend to implement a new microservice for user administration to deal with user's settings for resources consumption. Moreover, we plan to develop a messaging service to provide notifications about unavailability of features consumed, as well as the feature substitutions performed by the system.

6. Acknowledgements

The authors would like to thank the Brazilian National Research Council (CNPq) for partially funding this research.

References

Andrade, F. G., de Souza Baptista, C., and Davis, C. A. (2014). Improving geographic information retrieval in spatial data infrastructures. *GeoInformatica*, 18(4):793–818.

Assis, L. F., Ferreira, K. R., Vinhas, L., Maurano, L., Almeida, C., Carvalho, A., Rodrigues, J., Maciel, A., and Camargo, C. (2019). Terrabrasilis: A spatial data analytics infrastructure for large-scale thematic mapping. *ISPRS International Journal of Geo-Information*, 8(11):513.

BRASIL (2008). Decreto no 6.666, de 27 de novembro 2008. `http://www.planalto.gov.br/ccivil_03/_ato2007-2010/2008/decreto/d6666.htm`. Accessed on 2020-11-04.

Docker (2020). Official docker documentation. `https://docs.docker.com`. Accessed on 2020-08-27.

Friis-Christensen, A., Bernard, L., Kanellopoulos, I., Nogueras-Iso, J., Peedell, S., Schade, S., and Thorne, C. (2006). Building service oriented applications on top of a spatial data infrastructure–a forest fire assessment example. In *9th AGILE International Conference—Shaping the Future of Geographic Information Science in Europe*, pages 19–127.

Grus, Ł., Castelein, W., Crompvoets, J., Overduin, T., van Loenen, B., van Groenestijn, A., Rajabifard, A., and Bregt, A. K. (2011). An assessment view to evaluate whether spatial data infrastructures meet their goals. *Computers, Environment and Urban Systems*, 35(3):217–229.

Krafzig, D., Banke, K., and Slama, D. (2005). *Enterprise SOA: service-oriented architecture best practices*. Prentice Hall Professional.

Krämer, M. (2018). *A microservice architecture for the processing of large geospatial data in the Cloud*. PhD thesis, Technische Universität.

Li, Y. (2019). Research and application of micro-services framework for public information of waterway. In *2019 International Conference on Modeling, Simulation and Big Data Analysis (MSBDA 2019)*. Atlantis Press.

Mena, M., Corral, A., Iribarne, L., and Criado, J. (2019). A progressive web application based on microservices combining geospatial data and the internet of things. *IEEE Access*, 7:104577–104590.

Montesi, F. and Weber, J. (2016). Circuit breakers, discovery, and api gateways in microservices. *arXiv preprint arXiv:1609.05830*.

Newman, S. (2015). *Building microservices: designing fine-grained systems*. " O'Reilly Media, Inc.".

Nygard, M. T. (2018). *Release it!: design and deploy production-ready software*. Pragmatic Bookshelf.

OGC (2020). Web map service standard. `https://www.ogc.org/standards/wms`. Accessed on 2020-09-01.

OSS, N. (2020). Netflix open source software. `https://netflix.github.io/`. Accessed on 2020-08-28.

Schäffer, B., Baranski, B., and Foerster, T. (2010). Towards spatial data infrastructures in the clouds. In *Geospatial thinking*, pages 399–418. Springer.

Scholten, M., Klamma, R., and Kiehle, C. (2006). Evaluating performance in spatial data infrastructures for geoprocessing. *IEEE Internet Computing*, 10(5):34–41.

Soldani, J., Tamburri, D. A., and Van Den Heuvel, W.-J. (2018). The pains and gains of microservices: A systematic grey literature review. *Journal of Systems and Software*, 146:215–232.