Common Data Model for the Microservices of a Radiopropagation Tool

Adrián Valledor
Computer Science Dept.
Universidad de Alcalá
Madrid, Spain
e-mail: adrian.valledor@uah.es

Marcos Barranquero
Computer Science Dept.
Universidad de Alcalá
Madrid, Spain
e-mail: marcos.barranquero@uah.es

Juan Casado
Software Engineer at Starleaf
Starleaf, Building 7
United Kingdowm
e-mail: juan.ballesteros@starleaf.com

Josefa Gómez
Computer Science Dept.
Universidad de Alcalá
Madrid, Spain
e-mail: josefa.gomezp@uah.es

Abdelhamid Tayebi
Computer Science Dept.
Universidad de Alcalá
Madrid, Spain
e-mail: hamid.tayebi@uah.es

***Abstract*—This paper presents the development and improvement of a part of a Web simulation tool for radio propagation of 2D and 3D geospatial data. In particular, a fraction of its architecture, based on microservices, is shown. With our study, we encountered the need to use a common data model that allows the managment of the data throughout the tool. To solve this, some of the possible solutions to this problem are presented, such as the GraphQL Application Programming Interface (API) or the use of REpresentational State Transfer (REST) APIs and the use of Docker together with microservices. Finally, the implementation of a model supported by geometry specifications is provided as a solution and we conclude with the results obtained together with future work plans.**

Keywords*—*Microservices, radiopropagation tool, REST APIs, data model, GraphQL.

I. Introduction

In recent years, there has been a need, either because of time or because of the inability of maintenance during the software lifecycle, to migrate from the old monolithic systems to current microservices models. This is based on models in which the application was a single atomic unit, but complex and difficult to maintain and grow over time [1]. Therefore, nowadays the Service Oriented Architecture (SOA) model stands out, known for its great modularity and communication with other models. With this also comes the need to establish correct protocols for their communication with other models. At the moment, a microservice can be defined as ”a small application that can be deployed independently, scaled independently and tested independently and that has a single responsibility” [1]. Despite all the benefits shown in the use of microservices, they also brings some problems to the table. One of them can be the inconsistency of the data model, since the microservices can be independent and, therefore, they may not share the same specifications. In this case, the improvement of a radio propagation simulation tool is being developed [2]. This Web simulation tool will allow the display of geospatial data through an interactive map in 2D and in 3D urban environments [3]. The tool uses several empirical and semi-empirical models for the computation of radiopropagation. In addition, it displays terrain-related information, e.g., height values, population density, terrain type or any other raster input that the algorithms may require. It also allows to represent and visualize the simulation results on the map in 2D for empirical methods or 3D for deterministic methods. Additionally, it allows the optimization of antenna positioning, by means of genetic algorithms, to provide the best coverage in a given area. In particular, the solution adopted for the development of this tool will be shown through the definition of a specification structure adapted to this need.

The remainder of this paper is organized as follows: Section II gives a short overview of work related to the idea to be put forward. Section III shows the devised solution of the specification structure as a data model. Section IV presents another solution related to microservices and Docker. Section V summarises the advantages and disadvantages of implementing this model. Finally, Section VI concludes the article and gives an outlook on future work.

II. State of Art

Among the solutions that can be envisaged to solve the problem of the shared data model are the following.

A. GraphQL

GraphQL is a query-based language, in a JavaScript Object Notation (JSON) like format, for APIs and runtime that checks existing data. It provides a complete description of the data in its API, allowing users to request what they need [4]. Among its highlights is that in addition to getting the properties of a particular resource, it provides its references. This allows for quick queries, even if it relies on more limited network connections. Another point in favour is that it allows continuous adaptation to the types of data we need, i.e., if we need to add or remove specific fields, it would not be necessary to modify the existing queries.

On the contrary, this is not a suitable solution for the data model that is required for the simulation tool, as it may present one of the following problems:

- Performance problems, allowing the user to execute impermissible queries.
- It presents excessive complexity to solve this problem, making subsequent maintenance difficult.
- It has only one endpoint, making it difficult to use caching.
- Difficult error handling regarding other structures such as REST.

B. *REST APIs*

REST APIs represent a set of architectural principles that fits the specific needs of each application as defined by Dr. Roy Fielding et al. [5]. This provides a high level of flexibility and freedom for development of a microservices architecture. In addition, it must meet a number of requirements:

- Uniform interface: all requests must be the same, regardless of their origin.
- Decoupling of client-server, client and server applications must be independent of each other.
- Without status: each application must contain all the information necessary to process the query.
- Cacheability: resources must be able to be cached on the client or server side.
- Layered architecture: calls and responses will pass through different layers.
- Code on demand: in some cases responses may contain executable code.

REST APIs operation is based on communication through HTTP requests [6] that execute database functions, generally CRUD (Create, Read, Update and Delete).

In conclusion, the use of both GraphQL and the REST architecture is excessive or can become complex with respect to the development that is desired in the long term. For example, the need to provide a data model prior to GraphQL to be able to start working with it or the need to develop a larger architecture to be able to apply REST. For all these reasons, a better solution is proposed, in this case, the development of an own data model in the form of specifications.

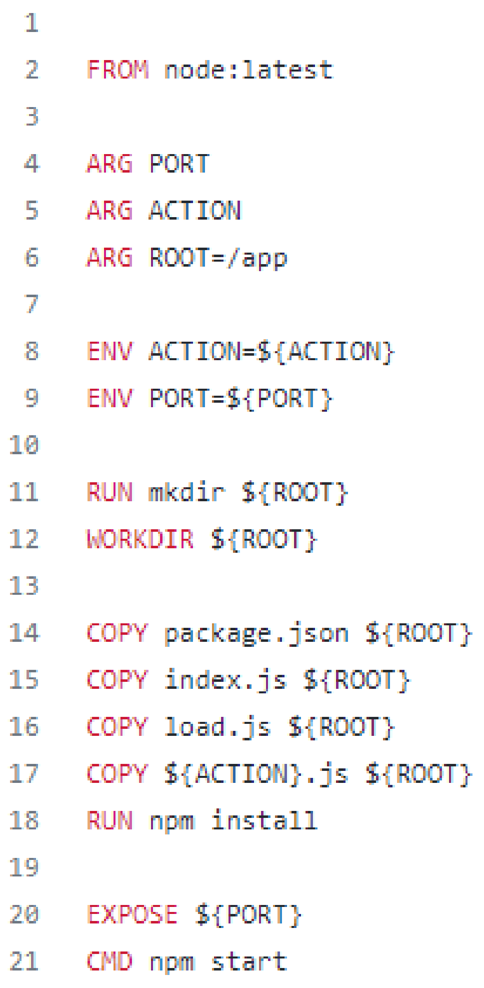
III. Microservices with Docker

Another solution related to microservices is the use of Docker [7]. In this case, a small Dockerfile is designed that uses a couple of environment variables to define the port and the action to be performed. In this case, the action will be to launch one microservice or another depending on whether it is involved in the operation or not.

One of the benefits of using this framework is the possibility of implementing load balancing by launching or stopping microservices depending on their need.

Another great benefit of using Docker is that it will allow microservices to be scalable, easily upgradable and independently deployable.

As it can be seen from the code in Figure 1 referring to the Dockerfile, through this small development in Docker, it is possible to load the microservices components and code dynamically. That allows to have the same interfaces and code for all of the microservices, and loading only the part of the code relevant to that microservice encapsulated in one container. For the case shown in Figure 1, it is used with servers of different types depending on whether they are necessary or not.



```

FROM node:latest

ARG PORT
ARG ACTION
ARG ROOT=/app

ENV ACTION=${ACTION}
ENV PORT=${PORT}

RUN mkdir ${ROOT}
WORKDIR ${ROOT}

COPY package.json ${ROOT}
COPY index.js ${ROOT}
COPY load.js ${ROOT}
COPY ${ACTION}.js ${ROOT}
RUN npm install

EXPOSE ${PORT}
CMD npm start
```

Fig. 1. Dockerfile to launch a microservice.

To denote which elements will be loaded into the container, operating system environment variables with different values common to servers, such as the port or server name, are used. By reading these attributes in code, the code to be copied into the container is determined, thus loading only what is necessary for that microservice.

IV. Solution Specifications

At this point, we propose an application with an architecture divided into a front-end and a back-end. In the back-end is the set of servers that provide the microservices. The microservices, in turn, are responsible for providing the necessary data both to show the users of the application and to carry out internal operations, for example, obtaining building heights, optimising antennas locations or calculating the radiopropagation, among others. On the other hand, there is a front-end in charge of representing graphically in the browser the data collected from the servers in the form of geometries. As can be guessed, a common data model is needed that is recognised by both sides of this structure. In this way, through a single model, the different parties involved can communicate without the need to add extraneous elements or dependencies.

As a solution to the data model, it was decided to create a common data type that will be used transversally across the different microservices of the application and that has been defined as ”Specification”. This Specification is based on the common properties that all geometry is considered to have, geometry being a fundamental structure in the application. The Specification will have a similar structure to JSON type files and is defined by the type of geometry it represents, as well as the coordinates of the points that compose it. See, for example, if one wishes to represent the geometry of a rectangle, a specification will be generated in JSON format indicating the type of geometry, in this case rectangle, together with the coordinates of the four points that compose it. In this way, this specification can be shared and recognised in the same way in the different microservices that make up the application.

V. Advantages and disadvantages

Having seen some of the different alternatives available, such as GraphQL or REST designs, together with the proposed solution, a set of advantages and disadvantages about them can be obtained.

The advantages include the following:

- Use of a common data model.
- No external dependencies.
- Easy to maintain structure.
- It provides a simple overview of the tool.

On the contrary, it has the following disadvantages:

- It does not have the possibility of being reusable in other developments.
- It has too concrete a design focused on the current tool.

VI. Conclusion

This work proposed the implementation of a common data model for radiopropagation tool. To do so, other models such as GrapQL or REST architectures have been discussed, reaching the conclusion of defining the Specification model, achieving a model recognised by the whole structure of its system. Moreover, it can be seen how beneficial it is for a framework like this tool the importance of maintaining an architecture through microservices.

The use of microservices has allowed the elimination of external dependencies, as well as the possibility of reusing them in the future and ensuring better maintenance over time. Plus, it adds modularization to the project, providing scalability and load balance between the front-end and the back-end. Parallel to the implementation of the microservices, it was necessary to use or develop a correct data model common to all of them in order to allow efficient communication. Along with this, current alternatives to this data model have been considered, but they do not fit with the situation of the tool being developed, either because of their size or because of future dependencies.

Once this problem has been solved, we propose as future work, the improvement of the user interface, as well as the relationship of this with the different libraries that allow the representation of maps in the browser. In addition, we intend to make use of the specifications designed in the data model, so that everything is communicated and perfectly integrated in the tool. The mentioned improvement will be based on the use of the OpenLayers [8] library, which allows maps to be represented in a browser and different operations to be carried out. Together with it, the React [9] framework will be used to transmit through forms, the coordinates that are intended to be represented on the map, which in turn will be represented by the data model of the specifications presented in this article. In this way, the data will be kept accessible throughout the entire structure of the tool.

Acknowledgement

This work was supported by the program “Programa de Estímulo a la Investigación de Jóvenes Investigadores” of Vice rectorate for Research and Knowledge Transfer of the University of Alcala and by the Comunidad de Madrid (Spain) through project CM/JIN/2021-033.

References

[1] J. Thönes, ”Microservices,” in IEEE Software, vol. 32, no. 1, pp. 116-116, Jan.-Feb. 2015, doi: 10.1109/MS.2015.11.

[2] A. Tayebi, J. Gomez, F. Saez de Adana, O. Gutierrez, and M. Fernandez de Sevilla, ”Development of a Web-Based Simulation Tool to Estimate the Path Loss in Outdoor Environments using OpenStreetMaps [Wireless Corner],” IEEE Antennas and Propagation Magazine, vol. 61, no. 1, pp. 123-129, Feb. 2019, doi: 10.1109/MAP.2018.2883088.

[3] F. Saez De Adana, J. Gómez, A. Tayebi, and J. Casado, ”Applications of Geographic Information Systems for Wireless Network Planning”, Artech, 2020.

[4] GraphQL — A query language for your API. [Online]. Available from: http://www.Graphql.org. June, 2022.

[5] R. Fielding and R. Taylor, ”Principled design of the modern Web architecture”. ACM transactions on Internet technology, 2(2), pp.115–150, 2002, doi: 10.1145/514183.514185.

[6] J. Gómez, A. Tayebi, and J. Casado, “On the use of Websockets to maintain temporal states in stateless applications”, in The 15th International Conference on Internet and Web Applications and Services ICIW 2020, pp. 21-24.

[7] X. Wan, X. Guan, T. Wang, G. Bai, and B. Choi, ”Application deployment using Microservice and Docker containers: Framework and optimization”. Journal of Network and Computer Applications, 119, pp.97-109, 2018, doi: 10.1016/j.jnca.2018.07.003.

[8] O. Zabala-Romero, E. Chassignet, J. Zabala-Hidalgo, P. Velissariou, H. Pandav, and A. Meyer-Baese, “OWGIS 2.0: Open source Java application that builds web GIS interfaces for desktop and mobile devices”, SIGSPATIAL’14: Proceedings of the 22sn ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, 2014, pp. 311-320, doi:10.1145/2666310.2666381.

[9] C. M. Novac, O. C. Novac, R. M. Sferle, M. I. Gordan, G. Bujdoso, and C. M. Dindelegan, ”Comparative study of some applications made in the Vue.js and React.js frameworks”. 16th International Conference on Engineering of Modern Electric Systems (EMES), 2021, pp. 1-4, doi: 10.1109/EMES52337.2021.9484149.