REST (Representational State Transfer) architecture for enterprise web application development

Luis Miguel Alamilla Hernández, Alejandro Hernández Cadena, José Ney Garrido Vázquez, José Ángel Jesús Magaña, José Manuel Gómez Zea

Tecnológico Nacional de México/Instituto Tecnológico de Villahermosa; Carretera Villahermosa - Frontera Km. 3.5 Ciudad Industrial Villahermosa, Tabasco, México. C.P. 86010.

Resumen

Actualmente, las empresas dedicadas a desarrollar aplicaciones distribuidas tienden a enfrentarse con ciertas incertidumbres al momento de elegir la arquitectura para implementar sus aplicaciones de desarrollo, por lo tanto, en sus inicios implementaron una arquitectura tradicional y creen que algo que les funcionó anteriormente les dará resultado. Entonces llevan su proyecto con una arquitectura monolítica, pero tiempo después la aplicación tiene la necesidad de crecer y se dan cuenta que el tipo de arquitectura seleccionada no fue la adecuada, ya que no pueden manejar los fallos del sistema eficientemente y no permite escalabilidad y modularidad.

Ante esta situación surgió SOA como una mejora de las aplicaciones monolíticas. Aunque en muchos aspectos SOA es más sencilla que una arquitectura monolítica, conlleva a un riesgo de cambios en cascada en todo el entorno si las interacciones de los componentes no se comprenden claramente, situación que para una organización conlleva a cuantiosas pérdidas si no se planea bien. Con respecto a REST en una organización, incorpora mejoras sustanciales lo que permite descomponer las arquitecturas tradicionales en partes más pequeñas, los servicios que implementa una arquitectura de microservicios usando un marco de mensajería común, es API RESTful. En donde cada servicio es independiente, permitiendo que un servicio pueda ser reemplazado, mejorarlo o abandonarlo, sin afectar a los demás servicios de la arquitectura.

Abstract

Currently, companies dedicated to developing distributed applications tend to face certain uncertainties when choosing the architecture to implement their development applications, therefore, in the beginning, they implemented a traditional architecture and believe that something that worked for them before will work for them. Then they take their project with a monolithic architecture, but sometime later the application has the need to grow and they realize that the type of architecture selected was not the right one, since it cannot handle system failures efficiently and does not allow scalability and modularity.

Faced with this situation, SOA emerged as an improvement to monolithic applications. Although in many aspects SOA is simpler than a monolithic architecture, it entails a risk of cascading changes throughout the environment if the interactions of the components are not clearly understood, a situation that for an organization leads to considerable losses if it is not well planned. Concerning REST in an organization, it incorporates substantial improvements that allow traditional architectures to be broken down into smaller parts. The services that a microservices architecture implements using a common messaging framework are RESTful APIs. Where each service is independent, allowing a service to be replaced, improved, or abandoned, without affecting the other services in the architecture.

Palabras claves: *REST, RESTful, SOA, monolithic applications, web service.*

1. INTRODUCTION

Today, technology advances by leaps and bounds, thus demanding new forms of communication between the different applications of companies since its main objective is to provide a quality product and service. Companies are looking to optimize the activities in their processes with the new technologies that the industry offers, however, this imposes a considerable investment for such implementation.

At a business level, both in the private and public sector, software development is carried out to meet the needs of automation of internal processes, this development has followed the trends imposed by the platform, programming language or by the experience of the development area. This results in the implementation of traditional or monolithic construction system.

A bad choice in the architecture now of the delivery of the systems, represents dissatisfaction in the final client and for the organizations considerable nonnegotiable losses.

Organizations are interested in providing robust and scalable systems. As business processes are carried out in complex technological information spaces, the integration of existing information systems becomes an important basis for the technical implementation of business processes.

2. ANALYSIS OF MONOLITHIC APPLICATIONS FOR THEIR MIGRATION TO SOA-BASED ARCHITECTURES

A computer application is a software that allows the user to perform one or more types of work, there are types of applications:

- Monolithic Applications.
- Client/Server applications.
- 2, 3, N layer applications.
- Distributed Applications.

Monolithic applications are those that the three parts form a whole and are executed in the same computer: User interface, Logic or business rules, and Data management. [1]

A monolithic architecture is a single unit, which is composed of three parts: data access also called the Model layer, a user interface such as a view, and the controller which communicates the database layer with the interface layer. [8]

Monolithic applications employ technology that limits the availability of tools that are indispensable for deploying the system.

This architecture is designed to be considered autonomous, its components linked in the code are deployed within a single compiler in the programming language where the application is made, forced to be interconnected and dependent on itself for its functionality.

From a business point of view, the design was considered one of the first software applications to be developed with important advantages such as single file execution, single deployment, and ease of development. By implementing this architecture, the development of monolithic applications was less expensive, so focus on single-file deployment functionality.

For a company when choosing to implement a monolithic architecture, it must take into account the business logic and consider that it is not convenient to tie the application to a single unit, but rather decide on an architecture that allows scalability, interoperability, and ease of implementation.

To put in context today, developing a CRM (Customer Relationship Management) with monolithic architecture where you have complex functionality and business logic in growth, where you must make deployment of all the development by the slightest change to a component, involves a high risk of errors and/or failures in the deployment and updating of the application, therefore it is done less frequently, as it requires more test cycles and communication between development teams. Also, the quality of code maintenance is not guaranteed, and it is very difficult to find dependencies between different modules.

In particular, a company, wanting to make a functional change, will face the problem of deployment and will have its developers attentive to the failures, as well as coexist with the contingencies until the corrections are made and an update is deployed.

Using an alternative architecture that allows scalability, flexibility, interoperability, autonomy, and efficiency to the multifunctional management is possible since the Service Oriented Architecture (SOA) has a lot to contribute.

3. SERVICE ORIENTED ARCHITECTURE (SOA) TO DESIGN AND IMPLEMENT WEB SERVICES

Now, with the great apogee in which the Internet is found, the term Web Services is very common as software artifacts from which more complex applications can be built.

According to the National Institute of Standards and Technology (NIST), the Web Service is a layered architecture consisting of (1) the Web Services layer, (2) the Web Service framework layer, and (3) the Web Server layer. According to Mokbel and Jiajin (2008), the objective of designing security architecture is to summarize the security details at the level of the business logic message. The ISO-WSP (Web Services Platforms) is an information flow architecture that decomposes the WSP into two parts and runs in different protection domains: (1) the T-WSP: handles the security of confidential data and (2) the U-WSP: contains extensive code that provides the normal functionality of the WS (Singaravelu, Wei and Pu, 2008). [2]

According to the W3C, a Web Service is: "A software application identified by a URI, whose interface and links are capable of being defined, described and discovered as XML artifacts. A web service supports direct interaction with other software agents using XML-based messages exchanged through Internet-based protocols".

Web Services allow linking, using different protocols, any kind of technology or service on the web, making legacy applications communicate with other applications of an organization or any third party.

The implementation of these components, the Web Services, has become massive, is an important point of value for the organizations, since there converge several business cores by which vital information for the organizations is transferred, from personal data, financial information, to all type of information that can be typified as confidential. [7]

For many companies, building their systems with a web service-oriented architecture makes it easier to publish the system to the world, as well as being used internally by their developers. This statement allows each team to choose the development tools they want to use to build their projects without affecting the link with other systems. [7]

The intention to compose a Web Service to be used in any organization in the environment or any other person has made possible the creation of languages and formal standards of definition of the same. However, the high abstraction for manufacturing and specification makes it very difficult to understand. It should be emphasized that a Web Service is an interface to the outside world of a software application that can be attacked or violated.

Every Web Service has a specification that provides the necessary information to invoke it. One of the best-known description standards is WSDL (Web Service Definition Language) [3]. The WSDL specification is an XML language, having defined rules where each Web Service component is specified.

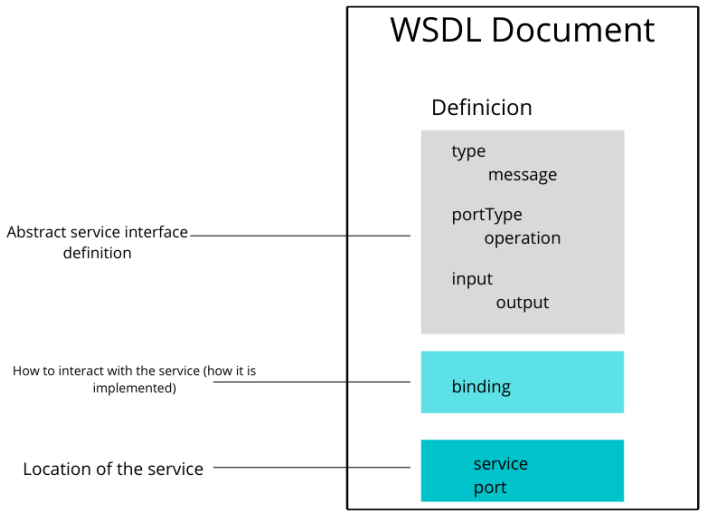

Figure 1. WSDL document

As shown in illustration 1, just as the WSDL file is useful for a software developer or anyone who can implement it to make use of the web service it describes, it also has the possibility of giving information to unwanted agents or even exposing vulnerabilities in the web service. Considering that there are tools that generate WSDL automatically for a Web Service, with a high level of attention to the information that is published is not exempt from low control. Such control is very important for those Web Services that belong to banks, online shopping services, among others.

Competing companies can learn the know-how and manage to copy the design to offer similar and competitive services. But it's not just about competition, security attacks such as information espionage, customer impersonation, command injection, and denial of service are also possible as attackers can learn about the data exchanged and the invocation patterns of WSDL documents. While the readability of service descriptions makes web services recognizable, it also contributes to service vulnerability [4].

3.1 SOA versus microservices architecture

SOA is a style of Software Architecture based on the definition of reusable services, with well-defined public interfaces, where service providers and consumers interact in a decoupled way to carry out business processes. It is based on four basic abstractions: services, application frontend, service repository, and service bus. A service consists of an implementation that provides business logic and data, a service contract, the restrictions for the consumer, and an interface that physically exposes the functionality. Application frontends consume

services by forming business processes. A service repository stores the service contracts and the service bus interconnects the application frontend and the services. [10]

The two most used architectural approaches for remote APIs are service-oriented architecture (SOA) and microservices architecture. SOA is the older of the two and began as an enhancement of monolithic applications. Instead of using a single application that does everything, you can use several applications that provide different functions and have no direct connection, all thanks to an integration pattern, such as an enterprise service bus (ESB). [9]

However, in many respects' SOA is much easier than a monolithic architecture, carrying a risk of cascading changes to the environment as long as the interactions of the components are not completely understood. This additional difficulty may be present in several of the problems that SOA aims to solve.

However, given the complex and changing nature of business requirements in the context of today's organizations, it is recommended that an incremental iterative approach be followed as a way of dealing with these changes, and with a strong focus on the generation of intermediate products as a way of obtaining key products in the advancement of the business that provides visibility over the development. [11]

4. MICROSERVICE-ORIENTED ARCHITECTURE

Microservice architectures are similar to SOA standards in that services are specialized and not directly connected. But they also break down traditional architectures into smaller parts. Microservice architecture services use a common messaging framework, such as RESTful's APIs. They use RESTful APIs to communicate with each other, without the need for complex data conversion operations or additional integration layers. Using RESTful APIs allows and even encourages the faster distribution of new functions and updates. Each service is independent. A service can be replaced, enhanced, or dropped, without affecting the other services in the architecture. This lightweight architecture optimizes distributed or cloud resources and supports the dynamic scalability of individual services. [9]

4.1. Features implemented by a REST architecture

REST is not a standard; it clearly defines architectural principles to be followed when implementing web applications or services. However, REST is based on standards for its implementation: HTTP, XML. REST services have the following characteristics:

- The most important operations that allow the manipulation of resources are four: GET to consult and read; POST to create; PUT to edit; DELETE to delete.
- The use of hypermedia to allow the client to navigate through the different resources of a REST API through HTML links.
- The REST API responses are usually JSON regardless of the language used.

4.2 Restrictions defining RESTful APIs

APIs are RESTful if they comply with 6 fundamental restrictions of a RESTful system:

ISSN: 2007-4786

1) Client-server architecture: this restriction keeps the REST architecture composed of clients, servers, and resources, managing the requests or requests with HTTP.
2) Stateless: for this restriction, the content of the clients is never stored in the server between requests, therefore the information about the state of the session remains in the client.
3) Cache capacity: client-side caching eliminates the need and consumption of memory when making client-server interactions.
4) Layered systems: client-server interactions achieve additional layered mediations, which allow for other functionalities such as load balancing, shared caches, or the same security.
5) On-demand code: servers can extend the functions of a client by transferring executable code.
6) Uniform interface: defines a generic interface to manage each client-server interaction in a uniform manner, which separates and simplifies the architecture. However, it is fundamental to the design of RESTful APIs to include 4 aspects:

    a. Identification of resources in requests: resources are identified in requests and these are separated from the representations that are returned to the client.
    b. Managing resources through representations: customers can receive files that represent resources. These representations must have sufficient information to be able to be modified or deleted.
    c. Self-describing messages: Each message returned to the client contains sufficient information to describe how the information should be processed.
    d. Hypermedia is the application status engine: after accessing a resource, the REST client must be able to discover through hyperlinks all other actions that are currently available.

These limitations may seem too many, but they are much simpler than a previously defined protocol. Therefore, RESTful APIs are becoming more common than SOAP APIs.

In recent years, the OpenAPI specification has become a common standard for defining RESTful APIs. OpenAPI establishes a language-independent way for developers to design REST API interfaces, allowing users to understand them with minimal effort. [9]

5. GRAPHQL, THE REST API REPLACEMENT?

GraphQL is a technology developed by the company Facebook initially in 2012 internally and then decide to publish for free use worldwide in 2015. This initiative is developed due to the need identified by the company to redesign the data search model to an API that will not take much effort on the part of the server to prepare the data and on the part of the client to transform it and adapt it to use. [12]

GraphQL and REST are two architectural mechanisms that are on different levels since GraphQL is a technology and REST is an architectural style. According to Eizinger (2017), there are three possible approaches to comparing these two mechanisms:

1) Generalize GraphQL and raise it to the level of an architectural style.
2) Specialize REST, that is, create a technology that follows the restrictions of the style perfectly.
3) Use only the architectural principles of each mechanism for comparison.

ISSN: 2007-4786

GraphQL requires an interface provided by a server, which must process queries and return a set of data specified by the client. It can use an API gateway in which access to external systems can be encapsulated [14]. The query service is an instance that runs separately from the client and can be considered as a web service. The GraphQL server allows the client to send more powerful queries than would be possible with another standard such as SOAP or RESTful.

It is a strongly typed query language developed by Facebook, which provides a flexible syntax to describe the data requirements and interactions to create applications [15], and later launched the service as a draft language specification [16]. The language was conceived for several general objectives, such as reducing the data transparency overhead about REST-like web service models, in terms of both the amount of data unnecessarily transferred, and the number of separate queries required to do so; as well as reducing errors caused by invalid queries from the customer and supporting an evolving data model without the need for API versions.

6. WHY IMPLEMENT REST ARCHITECTURE IN A COMPANY

There is currently no application or project that does not have an API REST for generating services from an application. Many companies generate business thanks to REST and its APIs.

Organizations and companies like Twitter, Amazon, Facebook, Google, among many others, work with REST. Currently, small companies are starting to integrate them into their platforms, as they have realized that they help them to increase their profits, allowing them to improve the quality and functionality of their platforms. Since, the use of REST architecture in companies, provide multiple benefits when improving the organization's processes. Without REST, all horizontal growth would be virtually impossible. This is because REST is the most logical, efficient, and common standard in the creation of APIs for Internet services.

However, in graph 1 we can see the strength that REST has concerning SOA and GraphQL, however, we can say in favor of GraphQL that it is more recent and therefore has less popularity. But regarding SOA, which is an older architecture than REST and taking as a starting point the year 2004 until today, objectively we can say that between 2004 and 2008 the use of REST and SOA was very equal, and clearly, we can see that although REST was recent and seemed to be the fashion, it was not here to stay and it has been demonstrated since 2008 when it was taken off from SOA and companies started to implement REST to their applications and since that date REST is shown and has been shown as the best alternative for the development of business applications.

However, it is worth mentioning that today while REST is still strong, SOA is losing interest, and below we have GraphQL that is gaining strength for implementation in developments.

ISSN: 2007-4786

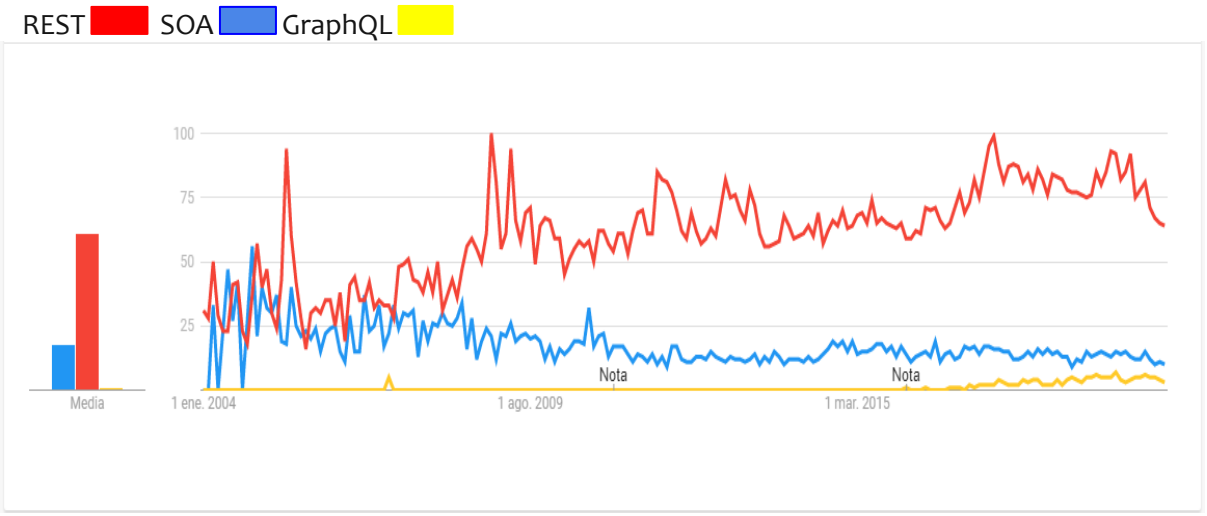

Figure 2. Comparative REST versus SOA and GraphQL

It is worth mentioning that REST has been gaining strength at an impressive speed and more with the arrival of NodeJS and NoSQL databases like MongoDB. However, with the advent of the Internet of Things (IoT) more and more devices are being connected to the Internet that needs to be integrated, making this a great opportunity for REST to continue with that strength that was mentioned above.

CONCLUSIONS

Many big organizations still implement SOA and do not invest in architecture with higher performance and integration measures like REST because the change brings a series of modifications and time, and a company, if it knows that there is something that works for them, does not put its business at risk, even though it knows that REST brings great benefits already mentioned.

Most clients prefer the REST standard. Many companies have invested and bet on SOAP, but it is evident that the developers prefer, in many cases, a simpler form of data manipulation like what REST offers.

When REST appears, it has a greater acceptance, that is why it will depend on the changing technology. Also, it is much easier to find web content and services that are based on the REST architecture. Although implementing a REST web service is a little easier than using SOA.

An implementation following the REST architectural style in an API has a faster response time than an API created with GraphQL technology. As we know REST imposes that the cache should be used to avoid making unnecessary requests to the server and thus increase the speed of the interactions between the client and the server, while GraphQL as a technology does not naturally offer this middleware.

Do not discard that very shortly we will talk more about GraphQL as opposed to REST since the performance that GraphQL provides us is considered.

REFERENCES

[1] Ma. Carmen Gastalver Robles (2017). UF1757 - Información y gestión operativa de la compraventa internacional, p 331.
[2] Singaravelu, L., Wei, J., y Pu, C. (2008). A Secure Information Flow Architecture for Web Services. SCC '08 Proceedings of the 2008 IEEE International Conference on Services, p. 182-189.
[3] WSDL Specification for W3C https://www.w3.org/TR/wsdl.
[4] Pananya Sripairojthikoon, Twittie Senivongse. "Concept-Based Readability Measurement and Adjustment for Web Services Descriptions". ICACT Transactions on Advanced Communications Technology (TACT) Vol. 3, Issue 1, January 2014.
[5] P. Bianco, K. Rick y M. Paulo, Software Engineering Institute, 2007. [En línea]. Available: http://resources.sei.cmu.edu/library/asset-view.cfm?assetid=8443
[6] M. Vera, Intelligence Bussines, 2014. [En línea]. Available: http://www.i2btech.com/blog-i2b/tech-deployment/que-se-entiendepor-soa-y-cuales-son-sus-beneficios/.
[7] Bernal, M. F. C. (2017). Asegurar web services no es cuestión de costos. Ciencia, Innovación y Tecnología, 3, 105-109.
[8] Pacheco Laje, J. L. (2018). Estudio comparativo entre una arquitectura con microservicios y contenedores dockers y una arquitectura tradicional (monolítica) con comprobación aplicativa (Doctoral dissertation, Universidad de Guayaquil. Facultad de Ciencias Matemáticas y Físicas. Carrera de Ingeniería En Sistemas Computacionales).
[9] Red Hat. (2020). Qué son las API y para qué sirven. Recuperado el 07 de junio de 2020 de https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces
[10] Delgado, A., González, L., & Piedrabuena, F. (2006). Desarrollo de aplicaciones con enfoque SOA (Service Oriented Architecture). Reportes Técnicos 06-16.
[11] Krafzig, D. Banke, K. Slama, D. "Enterprise SOA, Service Oriented Architecture Best Practices" Prentice Hall, 2005, ISBN 0-13-146575-9.
[12] Byron, L. (2015) GraphQL: A data query language. [Artículo oficial de Facebook acerca de GraphQL]. Recuperado en https://code.facebook.com/posts/1691455094417024/graphql-a-data-querylanguage/ el 07 de junio de 2017.
[13] Eizinger, T. (2017) API Design in Distributed Systems: A Comparison between GraphQL and REST. [Trabajo de Maestría para optar por el título de Gran Maestre de ciencia en ingeniería]. Universidad de ciencias aplicadas Fachhochschule Technikum Wien, Vienna, Austria.
[14] Fowler, M. (2003). Patterns of enterprise application architecture. Addison-Wesley.
[15] GraphQL. (2016). Introduction to GraphQL. Recuperado el 07 de junio de 2020 de https://graphql.org/learn/.
[16] GraphQL. (2018). GraphQL Specification Versions. Recuperado el 07 de junio de 2020 de https://graphql.github.io/graphql-spec/June2018/.

Correo electrónico autor: *alamilla-96@hotmail.com*