Peta International Journal of Social Science and Humanity

Peta Research Publication

www.PIJSSh.peta-research.com

Online ISSN 2830-540X

Microservices and Web-Services: A Review

Saddam Meshaal[1,*], Anwar Saif[1]

[1]Faculty of Computer and Information technology, Sana'a University, Yemen

Abstract

Microservices and web services are two architectural styles SOA. From theory point of view both are SOA styles having shared principals and concerns but in more detailed specifications a lot of differences exists. As application developer I need a clear and breve overview that summarize the concerns about each of web services and microservices starting from top level theory aspects ending with the most accurate details, as a complementary aspect of this review; a detailed review about implementation protocols of web services SOAP & REST (which can be used with microservices as well). As Microservice is the latest architectural style of SOA, this also work reviewed the most important challenges of implementing microservices over web services and reviewed comparison (from aspect of loose coupling) between microservices and web services declaring that microservices is more loosed in coupling.

Keywords: SOA, microservices, Web services, loose coupling, REST, SOAP.

Received on 14 Ja 2023, accepted on 27 Feb 2023, published on 08 Mar 2023

1. Introduction

During the last two decades, application architecture had the most changes towards implementing the best practices for SOA features. Although SOA principles are stable - nothing new was added- but the ways implementing these principals are evolving due to our ability to learn and optimize our solutions to serve our needs as best as it can and with lowest cost it can be. To have a clear view of change history of application architectural style, the bellow three figures (fig1, fig2 and fig3) illustrate that.

*Corresponding author

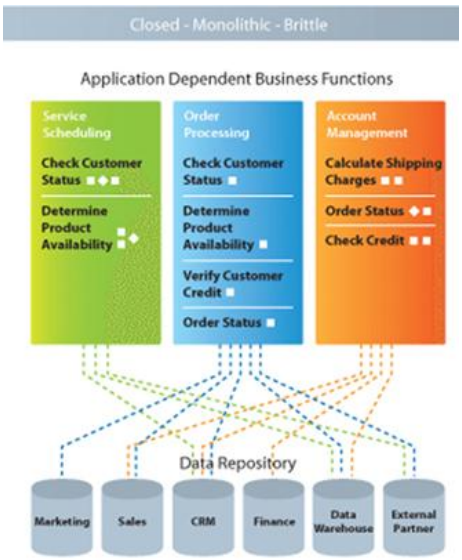

Figure 1. Applications Architecture Before SOA (Globisch et al., 2019)

Before SOA applications was developed as monolith where the whole application is one unit of code so if one functionality is needed in two applications it's needed to be rewritten in both applications, so duplicated code and functions exists therefore modification and upgrade was very time consuming operation. Example in fig.1illustrate check customer status function in three dependent applications, the function was duplicated in the three applications each function query data from database and return the result. Fig.2 describe SOA Architecture where all services are declared separately in layer called business layer connected to lower layer called database (data repository) layer and this layer (business layer) has higher layer called application layer (composite applications) which manages all applications and relation of each application to its services. SOA has been a solution for challenges of distributed systems in its early era (Raj & Sadam, 2021).

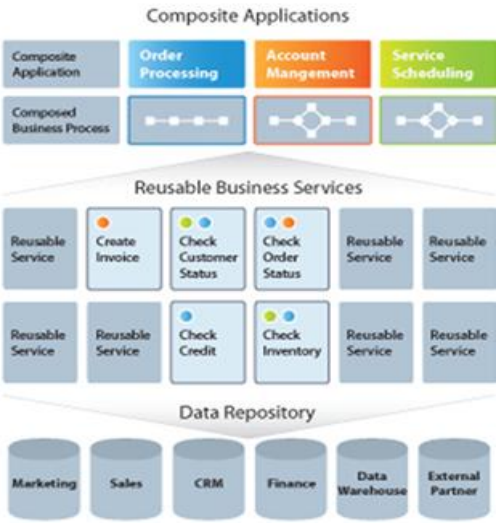

Figure 2. Applications Architecture After SOA (Globisch et al., 2019)

Microservices is an approach to developing a single application as a suite of small services, each running in its own process and communicating with lightweight mechanisms (Raj & Sadam, 2021). Fig.3 view Microservice architecture and debate the huge change that happened to systems architecture. Although a lot advantages of this architecture (approach) there are also some disadvantages that developer needs to take into account whether to adopt in their application projects or not.

Figure 3. Example of Microservice Architecture

Web services and microservices both are modular, services-oriented application architectures implementing SOA principals. When web services concerned essentially to connecting to separated application together via web service, microservices go away behind that trying to make the application itself a composition of autonomous separated services achieving less coupling between those services and enhancing the reusability of components higher

Finally, this work is divided into three sections, in section one we had an introduction about SOA and its history of change and the differences between web services and microservices, in section two we listed the related works and in section three the methodology of this work and discussion of related works. Finally, in section four conclusion and references.

2. Related Works

SOA become a field of interest in the last two decades for many researchers (Raj & Sadam, 2021). Vinay Raj and Ravichandra Sadam in their work “Evaluation of SOA-Based Web Services and Microservices Architecture Using Complexity Metrics” using Empirically Based Architecture Evaluation (EBAE) have evaluated the complexity in architecture when using microservices and web services architectures resulting that microservices are less complex than web services.

Nicola Dragoni et.al in their work “Microservices: Yesterday, Today, and Tomorrow” (Dragoni et al., 2017) reviewed the history of microservices starting from microservices past going into its current situation and the future, discussing the key features and challenges of adopting microservices architecture.

Anshu Soni and Virender Ranga in their work "API Features Individualizing of Web Services REST and SOAP" (Soni & Ranga, 2019) compared both REST and SOAP web services protocols as practical protocol for implementing web services and microservices as well. Unal et.al in their work "Automated API Testing" (Unal et al., 2016) discussed the ability to have automated testing of API to overcome the problems of one side modification and ensure functionality, reliability, security and delivery of business logic

3. The Framework

In this paper we have reviewed three papers discussing SOA (Service Oriented Architecture) from different levels of work starting from early stages of architecture engineering which is the analysis phase of application that needs to address the requirements and determine the appropriate architecture which was discussed in the first paper "Evaluation of SOA-Based Web Services and Microservices Architecture Using Complexity Metrics" that discussed web services and microservices from complexity metrics beside different aspects that concern application developer in early stages of projects. As Microservice is new architecture and developers may have confusion about it, it is important to have full vison about it, so this work summarize a comprehensive work about it which is "Microservices: Yesterday, Today, and Tomorrow" (Dragoni et al., 2017) which covered all aspects developer needs to have about that. Because we need deeper knowledge about technical and practical works for implementing SOA architecture; the third paper discussed the protocols can be used for implementing web services (and microservices as well). Paper named "API Features Individualizing of Web Services REST and SOAP" (Globisch et al., 2019) discussed REST architectural style based and SOAP protocol which is aimed to complete the circle of knowledge related to SOA.

4. Discussion

4.1 First Case

In "Evaluation of SOA-Based Web Services and Microservices Architecture Using Complexity Metrics" (Raj & Sadam, 2021) paper researchers consider that software architects are in chaos after the evolution of microservices, this chaos is whether to adopt microservices over web services or not. The authors discussed microservices and web services in terms of coupling principal of SOA as they claim no one has investigated this empirically. The researchers proposed "a service graph-based approach" to analyze and evaluate the effectiveness of microservices architecture when compared with web-services from loose coupling perspective. Based on this work "evolution of designed architecture should be done not only at the beginning of the project but also during the implementation and maintenance phase" (Raj & Sadam, 2021).

Before discussion of their imperial examples and their results; it is important to review some important principals-based researchers:

1. coupling is measured by the level of dependency each service has on other services.
2. If the coupling between the services increase, the complexity of the architecture also increases.
3. Coupling is a crucial factor among all the principals. The relation between loose coupling and others SOA shown in fig.4 where: (a) Nature of loose coupling minimizes dependencies between services therefore service autonomy is achieved. (b) Availability for reuse opportunity is increased because loose coupling frees tight

dependencies between services. (c) when services are loosely coupled service statelessness and scalability are achieved

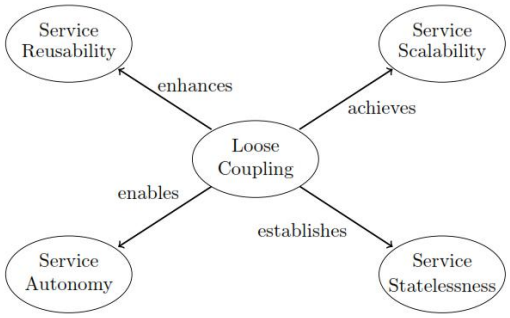

Figure 4. Relation between Loose Coupling and other SOA principals

The empirical study of this search is based on two applications that was already designed using web services and they redesigned those applications using microservices architecture. In this work we will view one case study which is vehicle management system.

Vehicle Management System:

The system allows customers to select, customize and identify vehicles using a secure web interface. The author in old web service base application has identified the services from the web based application and framed an SOA system.

Table 1. List of services with Cs & RCs values of web services-based applications

Service #	Service Name	Interacting service	CS value	RCS value
1.	Config Service	2,3,4,5,6,8	6	0.75
2.	Part service	1,4,5,6,8	5	0.62
3.	Product service	1,4,5,6,8	5	0.62
4.	Compare service	1,2,3,8	4	0.5
5.	Intensives and Pricing service	1,2,3,8	4	0.5
6.	Dealer and Inventory service	1,2,3,7,8	5	0.62
7.	Lead service	6,8	2	0.25
8.	User Interface Client	1,2,3,4,5,6,7	7	0.87

The author of the empirical study redesigned the vehicle management system using microservices approach. Based on the Single Responsibility Principle (SRP), author identified different individual tasks and structured them as microservices**.** The number of services and the interactions between the services has increased, making the services independent of other services

Table 2. List of services with Cs & RCs values of microservices-based applications

Service #	Service Name	Interacting service	CS value	RCS value
1.	Config Service	2,3,4,5,6,7,9,10,12	9	0.75
2.	Part service	1,4,5,6,10,12	6	0.5
3.	Product service	1,4,5,6,10,12	6	0.5
4.	Compare service	1,2,3,10,12	5	0.41

5.	Intensives service	1,2,3,6,12	5	0.41
6.	Pricing service	1,2,3,5,10,12	6	0.5
7.	Dealer service	1,9,10,11,12	5	0.41
8.	Get-A-Quote service	11,12	2	0.16
9.	Dealer locator service	1,7,10,12	4	0.33
10.	Inventory service	1,2,3,4,6,7,9,12	8	0.67
11.	Lead processor service	7,8,12	3	0.25
12.	User Interface Client	1,2,3,4,5,6,7,8,9,11,11	11	0.91

Coupling service value (CS) for a service is calculated by number of services that needs to interact with. Relative Coupling Service (RCS) for a service is calculated by the number of services that needs to interact with divided by the total number of services in the system. Finally the result of this study is that with designing the system with microservices approach; the architecture of the system had less coupling service and had less coupling service factor as illustrated in fig.5 which means that microservices offer more availability more reusability and more statelessness and scalability

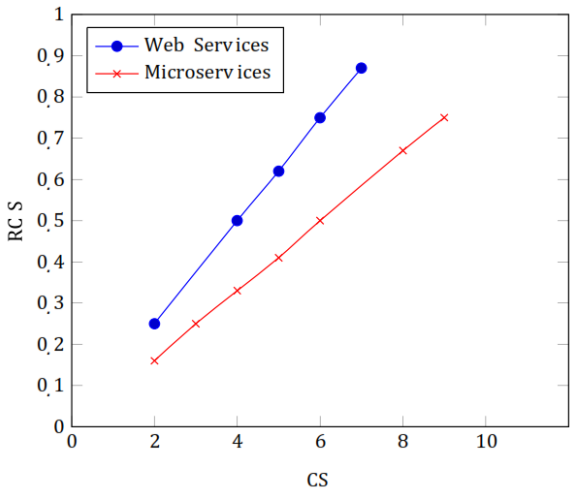

Figure 5. Comparison between web service and microservices approaches of case study.

4.2 Second Case

Nicola Dragoni et.al in their paper “Microservices: Yesterday, Today, and Tomorrow” did a historical review of systems architecture starting with monoliths architecture where the whole application modules cannot be executed independently and listed the most relevant problems of monoliths and listed their solutions that SOA offers. Some of those issues are: (1) due to monoliths complexity it’s difficult to maintain and evolve in large-size ones. Microservices solved this issue because are independent services so developers can test and investigate their functionality in isolation with respect to the rest of the system. (2) monoliths suffer from “dependency hell” in which application is one stone of code, so adding or modifying libraries results in inconsistent systems. Microservices cope with this issue through gradual transitions to new versions of microservices and this fosters continuous integration and eases software maintenance. (3) any change in one module of monolith requires rebooting the whole application. Since microservices are small in size any change in module requires only reboot to

microservices of that module. (4) monoliths limit scalability whereas in microservices scalability is enhanced since each module is consists of group of services that can be deployed/disposed as needed. (5) monoliths represent a technology lock-in where developers are bounded to use the same language and framework of the original application. Microservices by contrast can be developed using different programming languages and in different frameworks but constrains related communication (media, protocols, data encoding) exists.

Yesterday:

Architecture is what allows systems to evolve and it's concerned with providing a bridge between system functionality and requirements for quality attributes that the system has to meet. In this section the authors provide an overview on software architecture from the early days to the advent of microservices. The problem of large-scale software development was first experienced around the 1960s. the spike of interest in software architecture increases the number of existing software architecture patterns. Attention to separation of concerns has led to the emergence of the component-based software engineering (CBSE) before the appearance of Object-oriented Programming (OOP).

Today:

Microservices architecture appeared lately and was first introduced in 2011and become now a trend in software architecture, which emphasis the design and development of highly maintainable, scalable and loosely coupled software by taking modularity to the next level. Microservices different from SOA in the following: A) Size- microservices emphasis dividing services to the smallest possible units. B) Bounded Context- in which related functionalities are combined into a single business capability. C) independency-Each service in microservices architecture is operationally independent from other services.

Tomorrow:

In this section the authors see that microservices are so recent that we can consider their exploration to have just begun. The authors also see that the greatest strength of microservices comes from pervasive distribution (even the internal components of the software are autonomous services) and its greatest weakness is that programming distributed systems is harder than monoliths. This weakness beside to other weaknesses i.e. how can we manage changes to a service that may have side effect on other services that communicate with? Also the security threats that are higher potentiality because of network communication is essential part of the application work for each service.

The authors discussed in detail the challenges of microservices as they called it "pitfalls we need to keep in mind when programming with microservices". A) interfaces- we may have problems when dealing with different languages and passing data between different frameworks. In other words, what ensure that the service is called as it must be especially some languages come without specification language or computability checker of microservices like JavaScript. B) behavioral specifications and choreographies- if two services engage in session and started executing incomputable I/O. C) Greater surface attack area- monoliths communicate via internal communication therefore attack is limited to OS running that application, whereas microservices application may be broken down into multiple OS over the network what makes surface of attack greater.

4.3 Third Case

After the previous knowledge we had in the it's important to complete the knowledge circle about web services and microservices by discussing little deeper technical work about Representational State Transfer Protocol (REST) web services architectural style and Simple

Object Access Protocol (SOAP) web services protocol. Anshu Soni and Virender Ranga in their paper "API Features Individualizing of Web Services: REST and SOAP" compared REST and SOAP web services based on number of parameters (response time, architecture, security, reliability and efficiency). From theatrical aspect the difference between REST and SOAP is that REST is an architectural style for developing web services whereas SOAP is an underling protocol for developing web services. REST have number of constraints that control this architecture, those constraints are (client-server, stateless, cache, uniform interface, layered system and code on demand) while SOAP protocol has a certain message format that contains: A) envelope characterize start and end of the message format. B) Header contains optional data like authentication information. C) body which contains request and response information in xml format. D) Fault contains information about errors occurs while sending message.

The authors developed a web service in both SOAP protocol and REST style and compared many parameters and conclude that: Response time of REST is less than SOAP API which is approximate 4ms to 7ms. With the enhancement of number of API, SOAP takes approximate 1MB to 2MB more memory usage than REST as SOAP has heavy payload than REST and also takes more CPU time. REST is less coupled to a server which is an advantage over SOAP. SOAP is more reliable because it has built in successful/retry logic. SOAP has WS-Security feature while REST is limited to HTTPs but we can add additional methods to enhance security of REST **.** REST is easier to understand and develop**,** lightweight, works with any format while SOAP is restricted to XML only

5. Conclusion

In this work we have reviewed different aspect about web services and microservices that help applications developers deciding whether to move to microservices over web services or not depending on loose coupling principal. We conclude that loose coupling affect the rest principals of SOA; increase statelessness, scalability and availability for reuse of services. Microservices are less coupling than web services which means it's better for SOA principals' achievement. Implementing microservices over web services has also many pitfalls that we need to take into account: problems of dealing with different languages (interfaces), behavioural specifications and choreographies, and greater surface of attack area.

Finally, a comparison between SOAP web service protocol and REST web service style was conducted resulting that response time of REST is less than SOAP 4ms to 7ms and with increased number of APIs SOAP takes 1MB to 2MB more memory than REST because that SOAP has heavy payload regarding to message envelop and SOAP use XML while REST use JSON messaging format. SOAP is more reliable because it has its built in successful/retry logic. SOAP is more secure regarding to its WS-Security feature while REST limited to HTTPs with ability to add additional security methods. REST is lightweight, easier to understand and develop, and work with any format while SOAP is restricted to XML.

References

Dragoni, N., Giallorenzo, S., Lafuente, A. L., Mazzara, M., Montesi, F., Mustafin, R., & Safina, L. (2017). Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, 195-216.

Globisch, J., Plötz, P., Dütschke, E., & Wietschel, M. (2019). Consumer preferences for public charging infrastructure for electric vehicles. *Transport Policy*, *81*, 54-63.

Raj, V., & Sadam, R. (2021). Evaluation of SOA-based web services and microservices architecture using complexity metrics. *SN Computer Science*, *2*(5), 374.

Soni, A., & Ranga, V. (2019). API features individualizing of web services: REST and SOAP. *International Journal of Innovative Technology and Exploring Engineering*, *8*(9), 664-671.

Unal, A., Nayak, M., Mishra, D. K., Singh, D., & Joshi, A. (2016). *Smart Trends in Information Technology and Computer Communications: First International Conference, SmartCom 2016, Jaipur, India, August 6–7, 2016, Revised Selected Papers* (Vol. 628). Springer.