International Journal of Natural Sciences: Current and Future Research Trends (IJNSCFRT)
ISSN (Print) , ISSN (Online)
© International Scientific Research and Researchers *Association*
https://ijnscfrtjournal.isrra.org/index.php/Natural_Sciences_Journal/index

Differences Between Service-Oriented Architecture and Microservices Architecture

Lendina Rushani[a]*, Festim Halili[b]

[a]*University of Tetova, Faculty of Natural Sciences and Mathematics, Department of Informatics, Tetovo, 1200, North Macedonia*
[b]*University of Tetova, Faculty of Natural Sciences and Mathematics, Department of Informatics,Tetovo, 1200, North Macedonia*
[a]*Email: l.rushani3202022@unite.edu.mk* , [b]*Email: festim.halili@unite.edu.mk*

Abstract

Over decades the industry demands different changes on software design and architecture. The increasing complexity of applications, changes of requirements and several other evolutions caused an industrial shift of architectures. In the past, architectures like CORBA, Java RMI and SOA ensured an answer to the above mentioned problems, however now it seems that even SOA has its successor, the microservices architecture. After analyzing and reading this paper, the readers will be familiar and have solid information about service-oriented architecture, microservices, their differences and similarities, challenges, advantages and disadvantages and their implementation.

Keywords: services; architectures; microservices; SOA; service characteristics; architecture characteristics.

1. Introduction

With the increased complexity and the need for potent applications, the monolithic architecture is no longer the best choice because it affects the performance and how the application responds to changes. To deal with the limitations of the monolithic architecture, developers have adopted the *single responsibility principle* which says: gather together those things that can change for the same reasons and separate those things that can change for different reasons. For this reason, service-oriented architecture and microservices architecture began to be used by enabling developers to build applications that consist of services that run in their own environment but are independently deployable. The usage of service-oriented architecture started to increase on the mid 2000's by taking over the information technology industry. A big number of companies started adopting this new architecture style with the purpose of bringing better functionality reuse to the organization and to secure a better communication between companies among each other and among their customers.

----

* Corresponding author.

Also, there appeared several practices in order to implement this architecture. Unfortunately, companies learned the hard way that SOA was a big, expensive and complicated architecture that takes a long time to design and implement which often results in failed projects. This was the reason why the usage of SOA started decreasing. Today, microservices architecture is taking over the information technology industry by being able to address problems that are related to big monolithic applications and by developing highly scalable and modular applications. Both SOA and microservices suggest decomposing the system into services that are available over a network and can be integrated across heterogeneous platforms. In both approaches, services work together and share the same goal about the functionality of the overall system but the path to achieve that goal is different [1]. Even though SOA and MSA are both considered *service-based architectures*, which means that they are architectural patterns that rely on *services* as the primary architecture component that is used to perform business and non-business functionality; they differ from each other but also share many similar characteristics [2]. Creating a new product is very risky and choosing the right architecture ensures an essential step towards success. The architecture should be chosen by the developers based on the characteristics and objectives of the product that is being developed. In general, the architecture depends on how large and diverse the product that you are developing is. If you are working on more diverse and larger projects, then it is suggested to work with the SOA approach because it supports integration between heterogeneous applications and messaging protocols via an enterprise-service bus (ESB). Meanwhile, small environments e.g. web and mobile applications do not need a very potent communication layer and can be developed with the microservices architecture [3].

2. Service based architecture

All service-based architectures are generally distributed architectures, namely the service components can be accessed remotely through some sort of remote-access protocols such as Simple Object Access Protocol (SOAP), Remote Method Invocation (RMI), Microsoft Message Queuing (MSMQ), Java Message Service (JMS), Representational State Transfer (REST) and .NET Remoting. Distributed architectures enable more advantages over monolithic and layered-based architectures. These advantages include: separation of components, better scalability and better control of development, testing and deployment; another advantage of distributed architecture is that they enable the development of more loosely coupled and modular applications. The components of a distributed architecture allow better change control and easier maintenance, which leads to more responsive and potent applications. On service-based architecture we often come across the term *modularity*. Modularity refers the practice of encapsulating components of the application into self-contained services that can be designed, developed, tested and deployed while being independent from other components and services of the applications. Modular architectures also prefer and support rewriting over maintenance; this allows refactoring and replacing the architectures into smaller pieces as the business grows and gets more complex. Apart from advantages, distributed architectures have disadvantages too. The main disadvantage of distributed architecture is the complexity and cost. Complexity includes maintaining service contracts, maintaining unresponsive or unavailable services, choosing the right remote-access protocol, securing remote services and managing distributed transactions [4].

2.1 Service contracts

Service contracts are agreements between a customer or client and a person or company who will provide services; it is an agreement between a service and a service customer. These contracts specify the inbound and outbound data along with the contract format. Service contracts are a very essential part of service-based architectures that should be treated with a special attention. Creating and maintaining a service contract is a difficult task that should be treated with seriousness because they define important characteristics such as: information about both parties, outline of service and work, timeline of services, defaults terms, warranties and much more [5]. In service-based architectures there are two types of service contract models:

- Service-based contracts
- Consumer-driven contracts

The difference between service-based and consumer-driven contracts is the degree of collaboration. On service-based contracts the only owner of the contract is the service. The service is able to evolve and change the contract without considering the other party (the service consumer). With this model the service customers are forced to adopt new service contract changes whether or not they need or want these changes.

On the other hand, consumer-driven contracts work on a closer relationship between the service and its consumers. On this service model, the service and consumers collaborate together so that the needs of consumers are taken into account while creating service contracts. Generally, it is required that the service knows who its consumers are and how the service is used by every consumer. The service consumers have the opportunity to suggest service contract changes, which may be rejected or adapted by the service. The ideal form of creating consumer-driven contracts is: if a service consumer wants a change he delivers tests that suggest that change to the service owner. The service owner then executes those tests to see if the change breaks another's service consumers. If the change and has no effect on other consumers then it is accepted, in opposite it is rejected.

Another important element of service contracts is the *contract versioning*. Contract versioning is important because at some point in the future the contract might change, for this reason it should be planed from the very beginning of the development process because even if you think that you will not need it – eventually you will. The degree of changes is very dependable on how they will affect each service consumer and the compatibility supported by the service about contract changes. With contract versioning we are able to roll out new features that include contract changes, and at the same time we are able to provide compatibility for the consumers that still use previous contracts. Contract versioning is performed with the help of strategies. Several open source and commercial frameworks can help on managing and implementing those strategies. Strategies are implemented with the help of techniques. There are two basic techniques for implementing your own custom contract-version strategy:

- Homogeneous versioning – involves the usage of contract version numbers in the same service contracts. An example of this technique is the XML-based contract that represents an order for some goods with a contract version number 1.0. If a newer version 1.1 that contains an additional field that provides delivery instructions in the event where customer is not at home when the order is delivered is released, then the original contract (version 1.0) can remain backward compatible by making the new delivery instructions field optional.

- Heterogeneous versioning – This technique supports multiple types of contracts and is closer to the consumer-driven contracts. On homogeneous contracting as new features are introduced, there are introduced new contracts that support those new features.

The main goal of contract versioning is to provide backward compatibility. Also, maintaining a mindset that services must support multiple versions of a contract allows the team to quickly deploy new features and other changes without the fear of breaking the existing contracts with other service consumers.

2.2. Service availability

Availability and responsiveness of a service are two elements that are common to all service-based architectures. Both of these elements relate with the ability of the service consumer to communicate with a remote service, but they have slightly different meanings and are addressed by consumers in different ways. The *availability* refers to the ability of a remote service to accept requests in a timely manner, while the *responsiveness* refers to the ability of the service consumer to receive a timely response from the service. The way you handle errors about availability and responsiveness are different; since service availability is related to service connectivity, there is not much a service consumer can do except to retry the connection for a set number of times or queue the request for later processing if possible. On the other hand, responsiveness is much more difficult to address because once you successfully send a request to a service, you have to give an answer for the questions: how long should you wait for a response, is the service slow or did something happen in the service which prevents the response from being sent?

2.3. Service availability

Knowing that services are accessed remotely in service-based architectures, it is essential to be sure that the service consumer is allowed to have access to a particular service. Depending on the service that the consumer wants to access, authentication and authorization might be needed. With *authentication* it is decided whether the service consumer can connect to the service through sign-in credentials using a username and a password. Sometimes authentication is not the best solution because the fact that service consumers can connect to a service does not mean that they can access all the features in that service. *Authorization* refers to whether or not a service consumer is allowed to access specific business features within a service.

3. Service-oriented architecture

Since services are the basic building blocks of service-oriented architecture, firstly we have to understand the services in order to understand the architecture better. Services have existed since the civilization itself; in fact any person that is performing a task in support for others is providing a service. Any group of individuals collectively performing a task in support of a larger task also delivers a service. A service is a self-contained unit of software that is able to perform a specific task through a published API and it is part of a service contract. A service has three components: an interface - that defines how a service provider will perform requests from a service consumer, a contract – that defines how the service provider and the service consumer interact, and the implementation – that is the actual service code itself [6]. The interface and the implementation of a service are separated from each other, for this reason a service provider can execute a request without knowing how the service performs; the service consumer only worries about consuming the service. Services are reusable (reusability depends on how the services are designed), stateless, non-context specific and can be dynamically discovered across the system, enterprise or in the cloud. These characteristics enables services to be loosely coupled which results in new applications. Services can be created by writing code or they can be derived from existing IT assets. An important characteristic of services is granularity. A coarse-grained service includes more functionality than a fine-grained service which includes less functionality. The granularity of one service depends on its purposes and a poorly grained service will result in low reuse possibilities. In a service-oriented architecture, services can be combined with other services of the network with the help of service orchestration. Service orchestration helps on creating higher-level composite services and applications. The services of SOA are divided into two categories: Business Services and Infrastructure Services.

Business Services also known as Application Services are services that perform specific business functions and are required for successfully completing a business process. These services can be used for developing composite services and business applications that automate business processes. Business Services can be categorized in:

- Entity Services – are data-centric components of the enterprise system. Entity Services are responsible for exposing the information stored in databases.
- Capability Services– are action-centric components that implement business capabilities. Capability Services are coarse-grained and provide generic business capabilities.
- Activity Services – just like capability services are action-centric components that implement business capabilities. Unlike capability services, they are fine-grained and provide more specific business capability.
- Process Services – they enable integrating entity, capability and activity services through service orchestration with the purpose of creating composite business services. Also, process services contain the business logic.

Infrastructure Services secure the technical functionality that is required for business implementation in service-oriented architecture, but they do not add any business value. Infrastructure services are part of centrally managed infrastructure components such as Enterprise Service Bus. These services can be categorized in:

- Communication Services – have the responsibility to transport messages both within and without the enterprise.
- Utility Services – provide other technical capabilities that are not related to message transfer.

For programming and building services you can use different implementation technologies, but the two common implementation mediums are SOAP-based Web services and RESTful services. The conflict between what businesses really needed and what IT industry delivered continued to exist until the model and processes were transformed into a new paradigm. That paradigm was Service-Oriented Architecture. Today, business leaders and IT together determine how the business should operate and work to make it a reality with the help of SOA; they also determine a strategy that both liberates the business from IT and allows IT to create maintainable, extensible and compliant systems. Service-oriented architecture does not have a single definition, but it is efined in several ways. For example; The Open Group defines it as an architectural style that supports service orientation, OASIS defines it as a paradigm for organizing and utilizing distributed capabilities that are under the control of different ownership domains, The Object Management Group defines it as an architectural style for a community of providers and consumers of services with the purpose of achieving mutual value, and IBM defines it as a style of architecture that treats software components as a set of services [7].

A design paradigm is an approach for designing solution logic. While building distributed solution logic the design approach focuses on the “separation of concerns” theory. This theory states that a larger problem is more effectively solved when decomposed into a set of smaller problems or concerns. This gives you the option of partitioning solution logic into capabilities, each of which is designed to solve an individual concern. Related capabilities can be grouped into units of solution logic. There are different design paradigms. What distinguishes SOA from other paradigms is the manner which it carries out the separation of concerns and how it shapes the individual units of solution logic with specific characteristics [8]. Service-oriented architecture is an architectural design pattern that is based on separate pieces of software that provide functionality as services to other applications through protocols. It is a collection of services that communicate with each other; this communication can include transferring simple data or coordinating services to achieve an activity. With the help of SOA principles and methodologies you can design and develop a software product in the form of interoperable services. These services are well-defined functionalities that are built as software components which can be reused and distributed for different purposes. The SOA approach enables building IT systems that allow businesses to increase existing assets and manage the inevitable changes required to support the business [9]. One of the most important aspects of SOA is that it is a business approach and methodology as much as it is a technological aproach and methodology. This means that SOA enables businesses to take business decisions that are supported by technology, instead of taking business decisions that are determined by technology. One of the most important characteristics of SOA is that you do not “throw away” the software assets or components that you use everyday, but they are designed on a form that will allow you to use them, reuse them and keep reusing them. In fact, SOA extends the idea of reuse not only to web services but also to business services. SOA is a form that enables the business to move, change, partner and reinvent itself easily. That form is achieved with the help of standarts; specifically, using industry standard interfaces and creating business services without any dependency that will allow the business to be more flexible, to change its business model and to reorchestrate itself.

3.1 Delivering a SOA – Framework and Strategies

Many of SOA definitions indicate that the arrangement and relationship between services and modules should be loosely coupled rather than tightly coupled because this will allow easier change and customization of services based on needs and requests; the disadvantage is that this leads towards a plethora of definitions and approaches to SOA implementation. For this reason, there appears the need for a framework. We need a unifying model and framework for building applications based on SOA where multiple services will be able to interact with each other or a business application will reuse some of the services. A framework should include metadata that describe various important features of SOA, how these features can be arranged, the libraries and locations of services, service contracts and the service provider – whether internal or external. An SOAIF is a general-purpose infrastructure platform that enables developers and business analysts to create, deploy, manage, and change processes within and across the enterprise. SOAIFs have unique requirements at both the tools and infrastructure levels that are not typically provided by any single current technology or platform. A SOAIF focuses on internal and cross-enterprise processes, helps organizations for streamlining operations, reduces costs and increases responsiveness. Specifically, it provides general-purpose, service-based distributed capabilities that deliver operational efficiencies, easier application development and deployment, less expensive application integration and faster response to changing business requirements.

For delivering SOA projects you have to pass through a few phases that are included in the lifecycle of a SOA delivery project. The lifecycle is simply compromised of a series of steps that have to be completed to construct the services for a service-oriented solution.

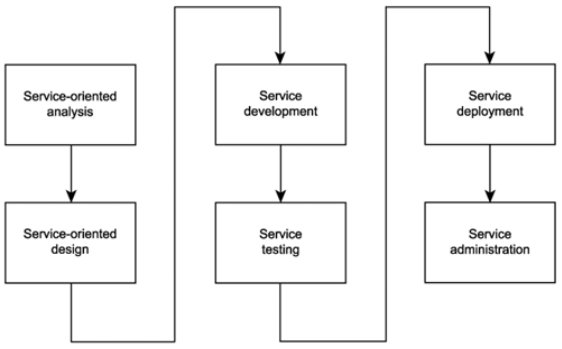

Figure 1: Phases of a SOA delivery lifecycle

- Service-oriented analysis – it is the initial stage of the delivery lifecycle where we determine the potential scope of our SOA. On this phase we map the service layers and model the individual services as service candidates that compromise a preliminary SOA [10].
- Service-oriented design – after deciding about what we will build, we have to determine how it should be built. The second phase on the lifecycle is a heavily standards-driven phase that incorporates industry conventions and service-oriented principles into the service designing process. On this phase the service designers have to establish the hard logic boundaries that are encapsulated by services.
- Service development – this is the building or construction phase. On this phase you have to choose the programming language and development environment that will determine the physical form of services and orchestrated business processes in accordance with their design.
- Service testing – as we mentioned above, services can be reused and composed in unforeseeable situations, that is why they have to be tested prior to deployment.
- Service deployment – on this stage you install and configure distributed components, service interfaces and any associated middleware products onto production servers.
- Service administration - the last phase takes place after the services are deployed. On this phase you take care of application management issues, and have to find answers about several question e.g. how will the service usage be monitored, how will the messages be traced and managed, how will performance errors be detected.

After building the individual services you have to organize them in a way that will: accommodate the preferences referring to which types of service layers we want to deliver, support a transition toward a standardized SOA that fulfills the requirements and coordinate the delivery of the application, business and process services. We know that the success of a service-oriented solution is determined by the fulfillment of the expected requirements within a given budget and deadline. To reach the wanted expectations we have to choose a strategy that will be based on the organization's priorities to establish the correct balance between the long-term migration goals and the fulfillment of short-term requirements. There are three strategies that help us on reaching the expectations in different manners:

- Top-down strategy – this strategy demands more of an initial investment because it introduces an upfront analysis stage focused on the creation of the service inventory blueprint. This strategy encourages the creation of an organization's existing building logic and requires all the business processes to become service-oriented. With the help of top-down strategy we are able to create numerous reusable business and application services. Top-down strategy is achieved by completing these steps: defining relevant ontology, aligning relevant ontology, performing service-oriented analysis, performing service-oriented design, developing services, testing services and deploying services.
- Bottom-up strategy – is focused on the fulfillment of immediate business requirements and the prime objective of the project. On this strategy the services are built and modeled to encapsulate application logic to serve the immediate requirements of the solution. This strategy avoids the extra cost, effort and time required to deliver services, has shorter lifespan and requires more frequent maintenance. Bottom-up strategy is achieved by completing these steps: modeling application services, designing application services, developing application services, testing services and deploying services.
- Agile (meet-in-the-middle) strategy – the challenge of this strategy is to find balance on incorporating service-oriented design principles into business analysis environments without having to wait before integrating web services technologies into technical environments. To achieve this objective we have to define a new process that allows the business-level analysis to occur concurrently with service design and development. Agile strategy is more complex that the previous two strategies because it needs to fulfill two opposing sets of requirements [11].

3.2 SOA Advantages

SOA offers the following advantages:

- Flexibility, ease of access and increased customer satisfaction.
- Loosely coupled applications and location transparency – it allows enterprises to easily add new services and upgrade the existing services.
- Seamless connectivity of applications and interoperability – this gives you the opportunity to increase business agility and to respond on demand.
- Alignment of IT around the needs of business – IT is considered the technology that provides value to business operations.
- Reuse of existing assets and applications – this helps on reducing costs, reducing development time and

reducing time to market.

- Better scalability and graceful evolutionary changes - this is enabled due to the reusability of services and their ability to develop applications independently and in parallel [12].

3.3 SOA Disadvatages

Despite the advantages, SOA has the following disadvantages:

- Complexity – in SOA, the interactions between objects and services are often rich and complex.
- Connectivity – a majority of distributed architectures do not work over wide-area and intermitten networks.
- Evolutionary development - building and updating services is fine. However, if applications continually require additional functionality and these requests are granted, the entire system may become unstable.
- Since services can invoke other services, each service needs to validate *completely* every input parameter. This has negative implications by way of response time and overall machine load.
- A bug or corruption introduced in a well-used service takes out the entire system, not just a single application.

4. Microservices architecture

For years we have been trying to find better ways for building systems by learning from what has come before, by adopting new technologies and by observing how technology companies operate with the purpose of creating IT systems that make both their customers and developers happy. Thanks to their modularity and flexibility that is offered to both development and operational team, microservices have gained a lot of attention in the IT industry. In the past, applications were generally developed with monolithic approaches. On monolithic approaches one code-base is used to run an entire application; this is less complicated than distributed approaches but running these applications is very challenging. Monolithic applications were difficult to change because of the high coupling nature and larger size. These obstacles created the need to search new developing approaches. Microservices are one of the solutions that help on solving the above-mentioned issues. Microservices have become popular in the last years along with the spread of DevOps practices and technologies like Kubernetes or Docker. The usage of microservice architecture has increased especially after 2014 and this can be verified by observing the industry where we conclude that microservices have been far superior when compared to other models. Microservice architecture is a style of engineering highly automated and evolvable software systems made up of capability-aligned microservies [13]. Microservices are a set of services that are designed to work together with the purpose of forming an application. On microservices each service is built to execute a single task [14]. Netflix, Amazon, The Guardian, LinkedIn and other well-known companies have evolved their applications towards a microservice architecture because it reduces the needed time for putting a new feature in operation. Big companies like the above-mentioned prefer this architectural style because they are more interested in improving the ability to change rather than finding a universal pattern or process. Microservice architectural style is defined as an approach for developing a single application as a set of small services, each running in its own process and communicating with the help of standardized interfaces and protocols. MSA focuses on specific aspects e.g. componentization of small lightweight services, agile applications, usage of infrastructure automation with continuous delivery features, decentralized data management and decentralized governance among services

[15]. The main purpose of microservices is to use autonomous units that are isolated from each other and coordinate them into a distributed infrastructure with the help of a lightweight container technology. The isolation of units and business functionalities while using microservices is very important because it enables independence during the development and deployment of each microservice, and it also optimizes their autonomy and ability to replace. Usually the adoption of this architectural style relies in adopting agile practices and this reduces the time between implementing a change in the system and transferring this change to the production environment. The small size of microservices makes the process of deployment and replacement much easier; in fact the smaller the service the more you maximize the benefits and downsides of microservice architecture. This allows companies to accomplish tasks that would have been very difficult or even impossible with the monolithic approach; for example a single service with a higher traffic can be scaled while the rest of services are left at the same size, this will help the businesses on reducing the cost of running services. On MSA we focus our service boundaries on business boundaries by making it obvious where code lives for a given piece of functionality. Also by keeping the service focused on the set boundaries we avoid the temptation for it to grow too large and all the difficulties that come with it. Services communicate with each other via network calls to enforce separation between services and to avoid the risks of tight coupling. They need to be able to change independently of each other, and be deployed without requiring consumers to change. In fact, while developing applications with MSA you have to be careful while deciding what will the services expose and what should they hide. This is important because if there is too much sharing then your consuming services will become coupled to the internal representations which will cause autonomy decrease and will require more cooperation with consumers while making changes.

Main concepts and principles of microservices are: Microservices are ideal for big systems – microservices are designed to solve problems of big systems. Size is a relative measure and it is not easy to quantify the difference between small, normal and big system. In fact the developers were not concerned about the size of the system but they were worried about what would happen on the situations where the system was too big. The developers identified that the systems grow in size despite the boundaries that we initially define. Shortly, new problems appear due to system scale and microservice architecture helps on solving these problems.

Microservice architecture is goal oriented – microservices focus on a goal rather than a solution. The purpose of microservice architecture is to try and reach a goal with the help of a particular approach. There may be a set of common characteristics that derive from this architectural style, but the focus of all these characteristics is to solve problems and achieve wanted goals.

Microservices focus on replaceability – the ability to replace is the main advantage of microservices. The idea of replacing a component rather than maintaining existing components is what makes this architectural style special.

While developing applications with the microservices architecture you are free to use different technologies. This will allow you to pick the right tool for each service and not use a more standardized, one-size-fits-all approach that often might end up being a failure. If one part of the system needs to improve its performance, you are free to use a different technology that is better able to achieve the required performance level. With microservices, you are also able to adopt the technology more quickly and to understand how new advancements may help you. One of the biggest barriers on trying out and adopting new technology is the risks associated with it. Within a monolithic application, if you want to try a new programming language, database, or framework, any change will impact a large amount of your system. With a system that is made of multiple services, you have multiple places where you can try out a new piece of technology [16].

4.1 Designing Microservices Systems

While developing applications in the microservices style you will need to conceptualize the design as much more than isolated individual design; which means that thinking in services term it is not enough because you have to consider how all aspects of the system can work together in order to create an emergent behavior. Emergent behaviors are the ones that are greater than the sum of their parts and for a microservices application this includes the runtime behavior that emerges when we connect individual services together and the organizational behavior that gets you there. A microservices system includes all of the elements about your organization that are related to the application e.g. the structure of your organization, the people who work there, the way they work and the outputs they produce are all important system factors. Equally important are runtime architectural elements such as service coordination, error handling, and operational practices. There is the additional challenge that all of these elements are interconnected— a change to one part of the system can have an unforeseen impact on another part. For example, a change to the size of an implementation team can have a profound impact on the work that the implementation team produces. From what we mentioned you can conclude that all the interconnected parts make the system very complex and it is difficult to predict the results that derive from a change on the system. For this reason scientists have developed a model. This model allows them to understand the system more accurately and to predict the behavior of a system. A microservices design model consists of five elements:

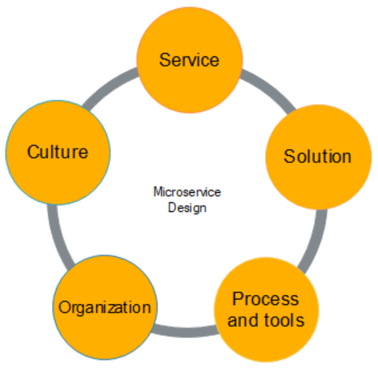

Figure 2: Microservice system design model

Service - exposes an application programming interface (API) and services collaborate with the help of those APIs. Implementing well-designed microservices and APIs are essential to a microservice architecture. If you can get the design, scope, and granularity of your service just right, you will be able to induce complex behavior from a set of components that are simple.

Solution – solution architecture is derived from the individual service design elements because it represents a macro view of your solution. When designing a particular microservice your decisions are bounded by the need of producing a single output—the service itself. So, when designing a solution architecture your decisions are bounded by the need of coordinating all the inputs and outputs of multiple services.

Process and tools - The system behavior is also a result of the processes and tools that workers in the system use to do their job. In the microservices system this usually includes tooling and processes related to software development, code deployment, maintenance, and product management. Choosing the right processes and tools is an important factor in producing good microservice system behavior. For example, adopting standardized processes like DevOps and Agile or tools like Docker containers can increase the changeability of your system.

Organization – how we work and how we communicate is also important. Organizational design includes the structure, direction of authority, granularity and composition of teams. Many of the companies that have had success with microservices architecture highlight their organizational design as a key ingredient.

Culture - is perhaps the most intangible yet may also be the most important. We can define culture as a set of values, beliefs, or ideals that are shared by all of the workers within an organization. Organizational culture is important because it shapes all of the atomic decisions that people within the system will make. Culture is a context-sensitive feature of your system. What works in Japan may not work in the United States and what works in a large insurance firm may not work at an e-commerce company. Also, culture is difficult to measure. There exist surveys and models for measurement but many businesses evaluate the culture in a more instinctual way via daily interactions with team members, team product and customers.

When put together all of these design elements form the microservices system. They are connected with each other and a change to one element can have a meaningful and sometimes unpredictable impact on other elements.

4.2 Microservices architecture advantages

Microservices architecture offers the following advantages:

- Faster release of functionality - this is enabled because it is not needed to wait until it is possible to release the whole system. Bringing changes into production rapidly is a priority for any business. The more an application is broken down into smaller components, the easier it is to deal with changes [17].
- Easier to build and maintain applications - Applications become easier to build and maintain when they are split into a set of smaller fragments. Managing the code also becomes easier because each microservice is, in fact, a separate part of code. MSA allows each service to be deployed, rebuilt, re-deployed and managed independently.
- Flexibility – is enabled when breaking a system into smaller components. The more the software is loosely coupled, the easier it is to engage with open source, provided that there is a license. If a company needs some functionality to be incorporated into the systems from open source components then it has to decompose the system into smaller loosely coupled components.
- Independent scaling – only the parts of the application that need to be scaled up can be assigned with required

resources and this results in the efficient use of resources.

- Security – it is easier to focus on security by putting more sensitive services into more protective zones.
- Each service can be built using the best and most appropriate tool for the task – it is possible to move parts of the system to the cloud. A company may decide to put some components on the cloud to be managed by specialized competencies. Whether or not the decision is to use multiple technologies in a system, there is a possibility to do it if needed.
- Redundancy – Usually, it is assumed that redundancy should be avoided. In a microservices design, redundancy is a classic way of increasing resilience.

4.3 Mircroservices disadvantages

Despite the advantages, MSA has the following disadvantages:

- Communication between services is complex – since everything is now an independent service, you have to carefully handle requests traveling between the modules. In such scenario, developers may be forced to write extra code.
- Microservices are more expensive – services will need to communicate with each other which results in a lot of remote calls. These remote calls result in higher costs associated with network latency and processing.
- Difficulties on small companies – microservices are great for large companies but can be slower to implement and too complicated for small companies who need to create and iterate quickly.
- Interface is more critical – Each microservice has its own API. You can easily make changes to a microservice without impacting the external systems interacting with it, but if you change the API any application using that microservice will be affected.

5. Differences between SOA and MSA

At first glance SOA and MSA sound very similar, but they both are different form a traditional, monolithic architecture where every service has its responsibility. Service-oriented architecture and microservices architecture are both scalable and agile approaches, however they have some important differences.

5.1 Differences in development

In both approaches we use different and several programming languages and tools, this enables the development team to have a diversity of technologies. The development process can be organized in multiple teams. In SOA every team has to know about the common communication mechanism, but on microservices architecture the services can operate and be deployed independently of other services [18].

5.2 Differences in service characteristics

The main difference in service characteristics is the service taxonomy. Microservices have very limited service taxonomy and they consist of two service types: functional and infrastructure services. Functional services are accessed externally and their role is to support specific business operations and functions; while infrastructure services support nonfunctional tasks e.g. security, performance, authentication, authorization and monitoring. Infrastructure services are not exposed to the outside world but they are treated as private shared services that are available only internally to other services. Service taxonomy in SOA is different. Within this approach there can be any number of services, but the architecture patterns defines four basic service types: business services - abstract, high-level, coarse-grained services that define the core business operations that are performed at the enterprise level. They are represented through XML or BPEL [19]. Enterprise services - concrete, enterprise-level, coarse-grained services that implement the functionality defined by business services, application services – they are also concrete, enterprise-level, coarse-grained services that implement the functionality defined by business services. They provide specific business functionality that is not found at the enterprise level. And infrastructure services – as in microservices architecture, these services support nonfunctional tasks.

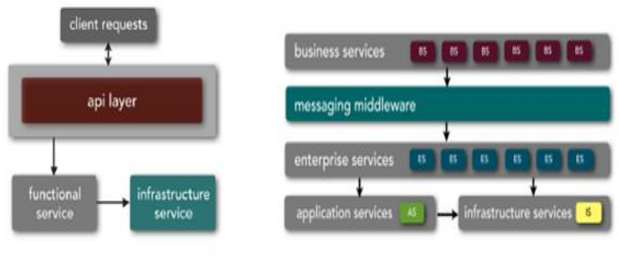

Figure 3: Service taxonomy in MSA (left) and SOA (right)

Another difference is the service ownership and coordination. In SOA there are different service owners for every type of services, for example: the owners of business services are business users, the owners of enterprise services are shared service teams or architects, application services are owned by application development teams and infrastructure services are owned by either application development team or infrastructure services team. In SOA, in order to create or maintain a single business request you have to coordinate with a lot of groups e.g. business users must be consulted about the abstract business services, shared services teams must be consulted about the enterprise services created to implement the business services, application development teams must be coordinated so that enterprise services can invoke lower-level functionality, and infrastructure teams must be coordinated to ensure nonfunctional requirements are fulfilled. In MSA, in order to complete a business request there is needed a little or no coordination at all. On cases where a little coordination is needed, it is done quickly and efficiently through small development teams. In fact, service ownership and coordination is important because it affects the effort and time that is needed for developing, testing, deploying and maintaining applications. MSA has a priority thanks to the small size of services and their minimal coordination.

From the service perspective, a big difference between these two approaches is the service granularity. Microservices are small, fine-grained services that are generally single-purpose and complete a specific task. In SOA services can range in size anywhere from small application services to very large enterprise services. In fact, it is common to have a service within SOA that represents a large product or even a subsystem. In MSA the service functionality tends to be very small, sometimes implemented through only one or two modules; in SOA, services tend to include much more business functionality, sometimes implemented as complete subsystems.

The difference in granularity affects the service's scope and functionality, and also the management of performance and transaction.

5.3 Differences in architecture characteristics

On the architecture topology of SOA and MSA we can find differences on sharing components. Service-oriented architecture is focused on a share-as-much-as-possible architecture style, while microservices architecture is focused on a share-as-little-as-possible architecture style. Component sharing is one of the most important elements in SOA because it is what enterprise services are all about. For example; a retail company that has to process orders uses customer-management system, warehouse management system and order fulfillment system. Each of these systems has its own order service. If we want to update an order we need a special business logic that has to be replicated across several applications in the enterprise because each system has its own database. Among applications it might be needed additional verification and coordination. SOA solves this problem with the help of enterprise-level shared services. To solve this problem you have to create a centrally shared order service so that every application can share the same processing logic that is associated with updating the order. This order service is smart enough to know on which database to go to retrieve and update order data for each system, at the same time to synchronize the data among all three systems. In other words, the order is represented not by one database, but by a combination of three databases.

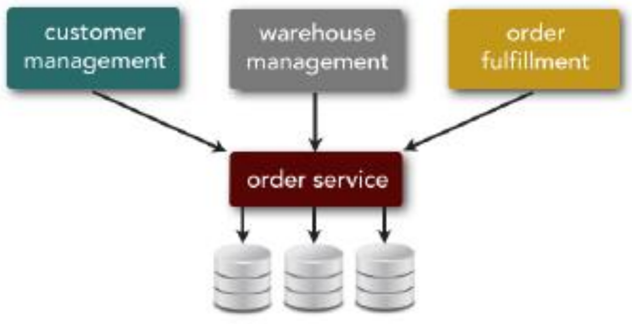

Figure 4: Service component sharing on SOA

This architecture style solves the problems related to the duplication of business functionality, but on the other hand leads to tightly coupled components and increases the overall risk that is associated with change. MSA and share-as-little-as-possible architecture style use a domain-driven design called a bounded context which refers to the coupling of a service and its data as a closed unit with minimal dependencies. You can minimize dependencies and achieve the bounded context by replicating common functionality across services. A service that is designed this way is very self-contained and exposes a well defined interface and contract. MSA makes the service maintaining easier thanks to the lack of dependent modules that allows the service independent change, and also the deployment is easier because there is less risk that a change made on one service will affect other services.

Another difference that is included in the architecture topology is the service orchestration and choreography. In order to understand the difference you have to understand the meaning of these terms. Service orchestration indicates the coordination of a number of services with the help of a centralized mediator e.g. a service consumer. The role of a mediator is to coordinate all the service calls that are needed for completing a business transaction. Service choreography indicates the coordination of a number of services without the help of a centralized mediator. On service choreography one service calls another service, which may call another service and so on, performing a service chaining. SOA uses both orchestration and choreography to process a request. Processing is done on this way: messaging middleware manages the orchestration by calling several enterprise services that are based on a business service request while choreography is used to call application services and infrastructure services. Microservices architecture prefers service choreography because it lacks a centralized component; but the actual objective of microservices is to minimize the service choreography because it can lead to high coupling. High coupling is risky because if a service is not available it can affect the overall application. To avoid high coupling on service choreography you can combine fine-grained services into a more coarse-grained service so that if a fine-grained service is shared among several services you can keep it separately or add that common functionality to each coarse-grained service.

Accessing remote services is performed in different ways. The main difference on accessing these types of services is that SOA has no restrictions and microservices rely on REST. In fact, the capability of SOA on handling several kinds of remote-access protocols is one of the main differences between these approaches. This ability of SOA comes by the messaging middleware component that supports any number of remote protocols and allows transformation from one protocol to another. The limited number of technologies used in microservices affects its simplicity and ease of usage. As we mention above, microservices are focused on REST and simple messaging but that does not mean that you cannot use other protocols such as SOAP or .NET but they will not be needed since most of the services are REST-based or messaging based.

5.4 Differences in architecture capabilities

Scope and size is one of the main differences between these two architecture styles. SOA has an enterprise scope, while the microservices architecture has an application scope. The prefix "micro" in microservices refers to the granularity of the internal components, meaning they have to be significantly smaller than what SOA tends to be. In fact the fine-grained nature of microservices affects the scope and decreases its size. Service components within microservices generally have to accomplish a single purpose. On the other hand, in SOA services usually include much more business functionality and they are often implemented as complete subsystems.

Heterogeneous interoperability refers to the ability to integrate with systems implemented in different programming languages and platforms. SOA supports the usage of multiple heterogeneous protocols, while microservices try to create a more simple architecture by reducing the number of choices. So if you are developing an application where there is needed integration of several types of services with the help of several types of protocols, you should rely towards SOA; but if all the services can be accessed through the same protocol then you should rely towards microservices architecture.

5.5 Which is best for you?

There is no definite answer to this question. In some cases SOA is the best option, in others it will not be as effective as microservices. Service-oriented architecture and microservices architecture are run in the cloud, which increases the flexibility for developing and deploying applications, but choosing the right approach depends on the business needs and requirements. Both of the approaches have its own advantages and disadvantages but for determining which one will work best you have to analyze the kind of the project that your team is working on. Regardless of which approach you are going to use, you will have development diversity because they can be developed in different technology stacks. You also have to determine how large and diverse the application you are developing is; for larger and more diverse environments you can use service-oriented architecture because it supports integration between heterogeneous applications and messaging protocols with the help of an enterprise-service bus (ESB). On the other hand, while developing web applications and mobile applications that do not need a powerful communication layer you can use microservices architecture. Service-oriented architecture is preferred when developing complex applications, applications that require interaction with one another and applications that are constantly developing. Microservices architecture is preferred when developing smaller web-based applications, projects where a developer wants to have as much control as possible and when you want to bring the application to the market as soon as possible. It is not easy to tell which architecture is less complex because they both require developers to deal with distributed systems and other challenges. Also, while choosing the right architecture you should asses the capacity, skills and ability to adapt the architecture.

5.6 Solving differences between SOA and MSA

The ideal IT architecture would be the one that is able to combine a composable service architecture and the organizational principles behind microservices. In other words, businesses should be approaching microservices and SOA, rather than microservices vs SOA [20]. That can be achieved by being organized around an application network. An application network can combine small, composable and reusable services designed around particular business capabilities that are connected via APIs. Also, an application network allows many people both inside and outside the enterprise to have controlled access to business data and to reusable microservices. It makes it easier for anyone in the organization to create a useful application, set of data, or an API by creating a particular experience through a composable service approach, and then exposing that value to the network. If the service is available beyond the scope of the project, then the provider's service is exposed to the network and can be used in other projects too.

References

[1] Cerny T., Donahoo M.J., Pechanec J., "Disambiguation and Comparison of SOA, Microservices and Self-Contained Systems", International Conference on Research in Adaptive and Convergent Systems, 2017, pp. 228-235

[2] Richards M., "Microservices vs. Service-Oriented Architecture", O'Relly Media Inc., Sebastopol, CA, 2016. pp.1

[3] IBM Cloud Team, IBM Cloud, "SOA vs. Microservices: What's the difference", September 2020, [https://www.ibm.com/cloud/blog/soa-vs-microservices], Accessed January 18, 2021.

[4] O'Reilly., "Microsevices vs. Service-Oriented Architecture by Mark Richards", [https://www.oreilly.com/library/view/microservices-vs-service-oriented/9781491975657/ch01.html], Accessed January 20, 2021.

[5]ContractsCounsel, "Service Contract", [https://www.contractscounsel.com/t/us/service-contract], Accessed 20 January, 2021.

[6]MuleSoft, "Services in SOA", [https://www.mulesoft.com/resources/esb/services-in-soa#:~:text=What%20is%20a%20service%20in,%2C%20a%20contract%2C%20and%20implementation.&text=These%20characteristics%20enable%20services%20to,designed%20according%20to%20SOA%20principles.], Accessed January 21, 2021.

[7] Erickson J., Siau K., "Web Services, Service-Oriented Computing and Service-Oriented Architecture: Seperating Hype from Reality", Journal of Daabase Management, pp.42-54, 2008.

[8] Erl T., "Service-Oriented Archittecture. Analysis and Design for Services and Microservices", Arcitura Education Inc., 2017

[9] Hurwitz J., Bloor R., Baroudi C., Kaufman M., "Service Oriented Architecture for Dummies", Wiley Publishing Inc., Indianapolis, Indiana, 2007

[10]Flylib, "SOA delivery lifecycle phases", [https://flylib.com/books/en/2.365.1/soa_delivery_lifecycle_phases.html], Accessed January 23, 2021.

[11]Flylib, "The agile strategy", [https://flylib.com/books/en/2.365.1/the_agile_strategy.html], Accessed January 23, 2021[12]Mahmood Z., "The Promise and Limitations of Service Oriented Architecture", International Journal of Computers, Issue 3, Volume 1, pp. 74-78, 2007.

[12]Mahmood Z., "The Promise and Limitations of Service Oriented Architecture", International Journal of Computers, Issue 3, Volume 1, pp. 74-78, 2007.

[13]Nadareishvili I., Mitra R., McLarty M., Amundsen M., "Microservice Architecture: Aligning Principles, Practices and Culture", O'Reilly Media Inc., Sebastopol, 2016, pp.6

[14]Hamzehloui M. S., Sahibuddin S., Ashabi A., "A Study on the Most Prominent Areas of Research in Microservices", International

Journal of Machine Learning and Computing, Vol.9, No.2, pp.242-247, April 2019.

[15]Francesco D. P., Lago P., Malavolta I., "Research on Architecting Microservices: Trends, Focus and

Potential for Industrial Adoption", IEEE International Conference on Software Architecture, pp.21-30, 2017.

[16]Newman S., "Building Microservices", O'Reilly Media Inc., Sebastopol, 2015.

[17] Elfatatry A.,"Microservices: A Review of the Costs and the Benefits", The Fifth International Conference on Advances and Trends in Software Engineering, 2019.

[18]Arsov K., "Microservices vs. SOA – Is There Any Difference at All", November 2017, [https://kikchee.medium.com/microservices-vs-soa-is-there-any-difference-at-all-2a1e3b66e1be], Accessed January 26, 2021.

[19]Tuli S., "Microservices vs SOA: What's the difference", DZone, 16 May 2018, [https://dzone.com/articles/microservices-vs-soa-whats-the-difference], Accessed January 26, 2021.

[20]MuleSoft, "The differences between microservices and SOA", [https://www.mulesoft.com/resources/api/microservices-soa], Accessed January 27, 2021.