Akash Singhal

MODERNIZATION OF ENTERPRISE SYSTEMS

Master Thesis

Examiners: Professor Kari Smolander
Ilkka Viitanen

Supervisors: Professor Kari Smolander
ABSTRACT

Lappeenranta University of Technology
School of Engineering Science
Software Engineering
Master's Programme in Software Engineering and Digital Transformation

Akash Singhal

MODERNIZATION OF ENTERPRISE SYSTEMS

Master’s Thesis

2020

51 pages, 19 figures, 2 tables

Examiners: Professor Kari Smolander
Ilkka Viitanen

Keywords: Enterprise Architecture, Service Oriented Architecture, Microservices, Enterprise Integrations

A growing trend in the digital transformation of Enterprises has led to an increase in the modernization of enterprise architecture. Service-Oriented Architecture (SOA) and Microservices (MSA) are the latest trends in software architecture; however, they have their advantages and disadvantages. This thesis explores the strategies to modernize enterprise systems such as Monolithic and Legacy applications to modern systems using a trade-off of SOA, MSA, and Enterprise Integration Patterns (EIP). Based on design science this research contributes an artifact to be used as a reference when modernizing enterprise architecture.
ACKNOWLEDGEMENTS

This thesis is part of the Master of Science (Technology) degree with Major stream as Software engineering at LUT University. This was a great learning experience overall and firstly; I would like to thank my Thesis supervisor Prof. Kari Smolander for his guidance and support throughout the research. My work and the topic of the thesis is based on the architecture course part of my master’s curriculum. Besides, it is my area of interest and is close to my working experience over the years. Prof. Kari was supportive of the timeline of this research.

This research project took 8 months and is done in cooperation with CGI Suomi Oy, and I would like to thank CGI for giving me an opportunity to study the existing challenges and problems in the enterprise. This gave motivation and direction for the research to find a novel solution. Big thanks to my colleagues Ilkka Viitanen, Miika Lindfors, Jani-Jukka, Ari Vilkman and Ari Saapunki for their constant support and help.

Lastly, I would like to thank my family for their support; this would not have been possible without my family. My family also extends their wishes to me from India, as I live and study abroad.

March 10, 2020

Akash Singhal
# TABLE OF CONTENTS

1 INTRODUCTION ........................................................................................................... 3
   1.1 BACKGROUND ....................................................................................................... 3
   1.2 STRUCTURE OF THE THESIS ............................................................................. 4
   1.3 PROBLEM STATEMENT ......................................................................................... 5

2 RESEARCH METHOD .................................................................................................. 7
   2.1 RESEARCH QUESTIONS ...................................................................................... 10

3 OBJECTIVES AND CURRENT SOLUTIONS ............................................................ 11
   3.1 CHALLENGES OF ENTERPRISE SYSTEMS ....................................................... 11
   3.2 MONOLITHIC APPLICATIONS & ARCHITECTURE ........................................... 13
   3.3 INTEGRATIONS IN LEGACY AND MONOLITHIC APPLICATIONS ................. 15

4 MODERNIZATION OF ENTERPRISE SYSTEMS .................................................... 18
   4.1 SERVICE-ORIENTED ARCHITECTURE (SOA) ............................................... 18
   4.2 MICROSERVICES ARCHITECTURE (MSA) ..................................................... 21
   4.3 SOA VS MSA ..................................................................................................... 22
   4.4 MIGRATE FROM MONOLITHS TO MICROSERVICES ................................... 26
   4.5 ENTERPRISE INTEGRATIONS AND PATTERNS ............................................. 30

5 ARCHITECTURAL DESIGN AND ARTIFACT ......................................................... 34
   5.1 PROJECT REQUIREMENTS .................................................................................. 34
   5.2 ARCHITECTURE DESIGN ................................................................................... 35
      5.2.1 Architectural Significant Requirements ............................................... 37
      5.2.2 Design Decisions ....................................................................................... 38
   5.3 FUNCTIONAL VIEW OF THE ARTIFACT ....................................................... 40

6 DEMONSTRATION OF SOLUTION ........................................................................... 42

7 EVALUATION ............................................................................................................. 45

8 DISCUSSION .............................................................................................................. 47

9 CONCLUSIONS ........................................................................................................ 48

10 REFERENCES ........................................................................................................... 49
<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>ASR</td>
<td>Architectural Significant Requirements</td>
</tr>
<tr>
<td>BI</td>
<td>Business Intelligence</td>
</tr>
<tr>
<td>CI</td>
<td>Continuous Integration</td>
</tr>
<tr>
<td>CD</td>
<td>Continuous Deployment</td>
</tr>
<tr>
<td>CRM</td>
<td>Customer Relationship Management</td>
</tr>
<tr>
<td>DWH</td>
<td>Data ware House</td>
</tr>
<tr>
<td>DS</td>
<td>Design Science</td>
</tr>
<tr>
<td>EIP</td>
<td>Enterprise Integration Pattern</td>
</tr>
<tr>
<td>EAI</td>
<td>Enterprise Application Integration</td>
</tr>
<tr>
<td>EA</td>
<td>Enterprise Architecture</td>
</tr>
<tr>
<td>FTP</td>
<td>File Transfer Protocol</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext transfer protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext transfer protocol secure</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>IoT</td>
<td>Internet of Things</td>
</tr>
<tr>
<td>MSA</td>
<td>Microservices Architecture</td>
</tr>
<tr>
<td>REST</td>
<td>Representational State Transfer</td>
</tr>
<tr>
<td>SFTP</td>
<td>Secure File Transfer Protocol</td>
</tr>
<tr>
<td>SOA</td>
<td>Service-Oriented Architecture</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Socket Layer</td>
</tr>
<tr>
<td>5G</td>
<td>Fifth Generation</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

1.1 Background

Enterprise systems have become an essential part of everyday life. As the internet grows exponentially, it has made possible to integrate systems in real-time. The development of software is a complex human activity where developers play an important role. Driven by several developments, the microservices architecture has appeared as a trend and emergent in designing large software systems as compared to the monolithic applications. Several organization tends to move away from monolithic and legacy stack of applications and move towards microservices-oriented architectures. (Mazlami, et al., 2017)

In legacy as well as modern software systems, integrations play an important role to serve as the backbone of the data communication between systems. To respond to the dynamic landscape of enterprise systems, how they scale, integrate and communicate with each other, enterprise systems should also evolve. Enterprises systems seamlessly share information within the organization internally and externally with third party systems.

The landscape also consists of so-called legacy and monolithic applications that comes with its own set of problems. They are difficult to modify and grow (Namiot & Sneps-Sneppe, 2014). Typically, maintenance is expensive and as the application grows, it degrades the performance and bloats the system. Enterprises consist of several heterogeneous applications and have point-to-point connections that result in spaghetti. The danger of running into an unmanageable ‘spaghetti’ of integrations between applications as their number increases, raising the risk of side effects with each new addition or modification to the portfolio (Meloa, et al., 2008).

Service-Oriented Architecture and Microservices are the latest trends in software architecture however; they too have their own set of problems and drawbacks. As part of the evolving architectural runway, enterprise systems go under changes, however not all legacy systems can be replaced as they perform business-critical operations. Most organizations usually have a large monolithic application and have split it over time. An
enterprise that has been in the business for long has a diverse landscape of old and new enterprise systems. Systems with enterprise data are rarely isolated, data is crucial for businesses to stay competitive in the market.

Namiot & Seps-Sneppe (2014) talks about “no one size fits all”. This research is focusing on the problem of transforming, modernizing, integrating the landscape of enterprise applications and designing an approach to modernizing the enterprise application landscape by using hybrid approaches of SOA and Microservices. It is evident that software ages and to be able to create a sustainable, scalable and maintainable application landscape, applications must be continuously modernized to the current needs and support new technologies.

1.2 Structure of the thesis

This thesis is segregated into several parts. The next chapters focus on the Introduction of the problem and methods used in this research. Chapter 4 talks about the objectives related to the research and current solutions, discussion related to literature and knowledge available on developing the artifact. Chapters 5 and 6 mainly focus on the design and use of artifact to solve the problem at hand also including project details, requirements, architecture design, and simplification and proposed solution related to the case. The last chapter of this thesis is focused on the evaluation of the artifact and discussion followed by conclusions of the research.
1.3 Problem Statement

Enterprises systems usually consist of large monolithic and legacy applications and seamlessly share information within the organization internally and externally with third party systems. As systems grow and complexity increases, organizations find it hard to manage the application lifecycle (Sadi & Yu, 2014). In this context, Henttonen, et al., (2007) says extensibility is “the ability to extend a software system with new features and components without loss of functionality or qualities specified as requirements”. The monolith architecture is typically subject to the risks of coupling, deployment models and implementation in such architecture has several challenges. Moreover, the maintenance and operations of the distributed monolith systems are not only complex but also unsustainable as we move towards more and more complex needs.

As more complex business and development processes prevail, the monolith architecture slows down the entire ecosystem of delivery and operations of a large enterprise system. Challenge continues, as monolithic and legacy software is hard to upgrade and customize. Software aging and digital transformation of enterprise application is reality (Hunold, et al., 2008). Enterprise landscape consists of several applications with different lifecycles and is built on a technology stack dating back to the 1980s. These applications are mainly monoliths and have a lot of point-to-point legacy connections and integrations, which have grown to become so-called spaghetti and bulky considering the amount of data volume increase over time. Modernization of integrations is also important and to be able to support Web Services and APIs is a minimal requirement for any enterprise application.

SOA and Microservices architecture aims to help organizations transform their legacy systems (Pahl and Jamshidi, 2016). Several important problem areas exist in this research, the first and most important is digital transformation and modernization of enterprise systems landscape, which includes the monolithic, legacy applications and integrations within the enterprises. A solution is needed to effectively provide a strategy and design architecture that may be useful to atomize the problems faced by enterprises in this modern era of digitalization. The value proposition from the solution aims to design a hybrid architecture that helps the digital transformation of monolithic applications keeping
in mind sustainability and integration needs to run business-critical applications. Secondly, can enterprise applications be modified or redeveloped, so that the new generation application can use the advantages of SOA and microservices to overcome the challenges faced by organizations to run their business application via legacy applications?

Quality parameters such as (Scalability, performance, etc.), coordination and integrations are also key aspects to the problem of modern enterprise architecture, which needs to be addressed when transforming the enterprise landscape. Service-oriented and microservices architecture tends to address these problems but they too have their own set to drawbacks and challenges. The overall solution is important for several organizations as they face major roadblocks when moving from mission-critical legacy applications to a modern system (Shadija et.al, 2017). This study aims to help the organizations find a better strategy and a reference point when modernizing enterprise architecture. Preferably, a trade-off that could be referred to while designing a new enterprise system, keeping in mind both service-oriented architecture and microservice architecture.

In this thesis, an approach to modernize and integrate enterprise applications is to be designed for a case company, which has several monolithic enterprise applications. The task involves studying the existing system landscape, review literature surrounding the research problem and designing/proposing the new solution that meets the needs of a modern enterprise architecture. When designing the solution has to also deal with challenges such as integrations with legacy systems, network slowness, heterogeneous data streams from systems and ever-changing application landscape. The solution can be cloud-based and traditional on-site customer premises-based. The main goal of the research is to find an approach suitable for the modernization of enterprise architecture, i.e. existing strategies and recommend how the applications can be modernized using a trade-off of both service-oriented architecture and microservice architecture.
2 RESEARCH METHOD

This research aims at the creation of an artifact and/or design theory as a means to improve the current state of practice as well as existing research knowledge (Baskerville, et al., 2018). According to (Hevner, et al., 2004) the problem definition will be used to develop an artifact that can effectively provide a solution. Due to the complex nature of the problem, the research focuses on the development of artifacts the creation of new knowledge through the design of novel or innovative artifacts.

In general, design science research aims to solve the problem under analysis with the help of an artifact. It is relativistic by nature, and therefore results are related to the context of the study. This research aims to start from a problem, discussing the need for a solution and then building an artifact. The new artifact is based on the ideas, which are at least partly based on the earlier existing knowledge and research papers. (Peffers, et al., 2007) explains the process of Design research. The paper explains the Design science research process (DSRP) model, which is shown in Fig. 1.

![Figure 1: Design Science Process Model (Peffers, et al., 2007)](image-url)
The essence of the design science process model is critical to this research done, the journey starts from the problem identification and motivation to develop and demonstrate the artifact. Problem centered approach has been used in this research followed by other phases of the research process. This design science research emphasis the design process to solve the observed problem; create the reference artifact, which can be applied to solve the problem in question and other similar problem cases in enterprises. The enterprise applications are part of everyday business this research tends to help many organizations. It tends to contribute to research, evaluate design and artifact, based on how well it solves the problem and shows how well it fulfills the objectives of the research. (Hevner, et al., 2004) mentioned that DS research should address important and relevant problems.

Here are the Design-Science Research Guidelines (Table 1) followed during this research.

<table>
<thead>
<tr>
<th>Guideline</th>
<th>Description (Hevner, et al., 2004)</th>
<th>What was done in the research</th>
</tr>
</thead>
<tbody>
<tr>
<td>Activity 1. Problem identification and motivation</td>
<td>Focus on defining the specific research problem and justify the value of a solution. It can include the knowledge of the state of the problem and can also include the importance of its solution</td>
<td>This phase explores to identify the problem and understanding the pain areas of the enterprise systems and its architecture in a large organization. The justification for the need for the solution is also defined clearly at the beginning of the study.</td>
</tr>
<tr>
<td>Activity 2. Define the objectives for a solution</td>
<td>Defining the objectives for a solution. Also, include the existing solutions available for the state of the problem, which could be potentially helpful in design activity.</td>
<td>A detailed study is done as part of this research to define the objectives focusing on challenges, current solutions, and integration aspects to create a novel artifact. The objectives of this research are mainly qualitative.</td>
</tr>
<tr>
<td>Activity 3. Design and development</td>
<td>Create the artifact; a design research artifact can be any designed object in which a research contribution is embedded in the design. This</td>
<td>Design activity starts from a requirements-gathering process, in which a diverse set of applications go under the transformation process. It also takes advantage of existing</td>
</tr>
</tbody>
</table>
activity includes determining the artifact’s desired functionality and its architecture and then creating the actual artifact. Knowledge. This process was iterative to improve the artifact as it is evaluated during the implementation phase to check the efficiency. The result of this phase is a novel artifact, which addresses the problems and the objectives of the research.

<table>
<thead>
<tr>
<th>Activity 4. Demonstration</th>
<th>Demonstrate the use of the artifact to solve one or more instances of the problem.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>This activity in the research demonstrates the use of the artifact to address the objectives of the study. The utility of the artifact is strongly linked to the project that is has led to the creation of the reference artifact and strategy.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity 5. Evaluation</th>
<th>This activity focuses on the measurement of the artifact, i.e. how well the artifact supports a solution to the problem.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Proof of concept used to evaluate the artifact and how well it solves the problem in question. When evaluating the artifact there were three iterations done to evaluate and test the solution.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Activity 6. Communication</th>
<th>Communication of the problem, artifact and its utility/novelty and its efficiency for researchers and other organizations.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>The thesis communicates the importance of the problem, the artifact and its novelty and utility in large enterprise architectures. There is not much research in the area as this is still an evolving topic. The artifact can be very useful for architects and organizations to support their architectural needs.</td>
</tr>
</tbody>
</table>

Table 1: Design-Science Research Guidelines and usage in the Research process
2.1 Research Questions

Based on the research guidelines, the researcher defines a clear set of research questions. This research is based on the goal to find what kind of approaches and strategies can be adopted to modernize the enterprise systems landscape from architecture and integration point of view. A novel artifact, which supports the solution of the problem situation and answers the research questions. Research questions with sub-research questions based on the research objectives mentioned below.

The main research question is:

- What kind of approach is suitable for the modernization of the enterprise systems landscape?

The main research question is divided into sub-research questions as follows:

- What are the different kinds of existing strategies in the modernization of enterprise monolithic applications?
- How can enterprise applications be modified or redeveloped, so that the new generation application can use the advantages of SOA and microservices? In addition, how to manage communications and integrations?

It is important to explore the area related to modernize the enterprise systems architecture using modern concepts, the target is overall digital transformation and enables enterprises to support current and next-generation customer needs. Understanding the problems and challenges faced by an enterprise to maintain its enterprise architecture and continuously modernize their application to serve business and customer requirements. Enterprises systems usually consist of complex large applications and seamlessly share information within the organization internally and externally with third party systems. As systems grow and complexity increases, organizations find it hard to manage the application lifecycle with the existing architecture. There are some existing strategies like SOA and MSA, however, enterprises struggle to find the right strategies to move towards migration and modernization along with a trade-off between SOA and MSA while keeping in mind the integration needs and possibilities.
3 OBJECTIVES AND CURRENT SOLUTIONS

This section defines the objectives for a solution, it explains the objectives from the dimension of problem area and information about what can be possible to do within the defined timeframe of this study. The objectives of this research are mainly qualitative as the solution produced is a novel artifact that is expected to support solutions to problems mentioned in section 2. Also later in the section, knowledge about the problem situation and currently available solutions are mentioned.

To answer the research questions, the main objectives of this research are further divided into smaller objectives:

1. Understanding the problems and challenges faced by an enterprise to maintain its enterprise architecture and continuously modernize their application to serve business and customer requirements.
2. To understand the current enterprise landscape in the problem statement, and how SOA and Microservices have to offer to revamp the aging and monolith systems.
3. What are existing ways to modernize applications and integrations: What is SOA, Microservices, How do they differ, what are the benefits of using SOA, and Microservices based architecture in enterprises.
4. How are coordination, communications, and integrations handled in modernized applications?

3.1 Challenges of Enterprise Systems

Enterprise systems are generally application software packages which have different operational and business needs they are very critical for business do none and do their day-to-day operations but the customers Enterprise systems are of different types Many of the Enterprise applications are built of large and monolithic architectures the Legacy applications are difficult to maintain and they are difficult to scale. Software aging and Integrations are other problems in Enterprise systems to digitally transform the systems we have to make a lot of change. Some of the challenges of Enterprise systems are that enterprises are typically built of several heterogeneous applications.
Heterogeneity is one key area where the enterprises have to handle a lot of complexity and changing landscape in business and technology is another pain area for enterprises. In a paper by (Endrei, et al., 2004) writers argue, "Most enterprises today contain a range of different systems, applications, and architectures of different ages and technologies. Integrating products from multiple vendors and across different platforms were almost always a nightmare". These heterogeneous applications interact with each other using point-to-point connections and these point-to-point connections results in spider webs called spaghetti. Additionally, monolithic applications are difficult to modify and grow, when in the maintenance phase, changes and features added to the system bloats it hence making the applications slow and difficult to serve and customer’s overtime. Technology and processes change frequently and this results in software aging and bloating the performance problems and slowness in monoliths is another typical problem this not only creates challenges in software development and maintenance of these Legacy applications it also causes a lot of other problems.

In addition, when we want to transform these systems into the new and digital era, inter-application communications and integrations are a big challenge. Legacy systems are typically old, they do not support Enterprise Integration patterns, and communication channels credibility and maintainability are typically not possible in the latest stages of the software development life cycle and maintenance cycle. Enterprises software systems need more and more data integrations to become relevant to serve business needs. Before the era of web services and API, this was traditionally done using file transfers or batch jobs, which runs based on a timely frequency using a typical cron or scheduler process in the system. As the landscape changes and business needs become agile, more complex, there is a need for real-time data integration, which becomes a big challenge for enterprises. Enterprise Integration means connecting systems, companies, and/or people. The definition is quite wide and can include many things. Typically, integration mainly deals with systems sharing data or making requests to create, remove or change data (Hohpe & Woolf, 2012).
3.2 Monolithic Applications & Architecture

A monolithic software application is typically a tier application were consists of a combination of the presentation layer, business logic and data layer as shown in Figure 2. These applications serve the requirement of end process flow and business transactions needed.

![Inside a Monolithic Application](image)

Figure 2: Monolithic Application

Modularity is not common is such monolithic applications. Typically, monolithic applications are constraint by deployment models and implementation functionalities in a system as they are deployed together. For scaling capabilities, the monolithic applications are deployed as multiple instances but fundamentally, they run as one single process to achieve tasks as shown in Figure 4.

A monolithic application built on monolithic architecture as shown in Figure 3, all features and functions of the application reside in one code base and deployed together as one bundle using a single platform (Richardson, 1996).
Several challenges of legacy and monolithic applications, which were found as part of the literature study:

- Legacy and large Monolithic applications are difficult to modify and grow (Namiot & Sneh-Sneppe, 2014).
- Typically, maintenance is expensive and as the application grows, it degrades the performance and bloats the system. In addition, the legacy code of applications in sometimes undocumented and the original developers have left the development team years ago (Hunold, et al., 2008).
- The monolith architecture is typically subject to the risks of coupling, usually, deployment models and implementation in such architecture has several challenges. Moreover, the maintenance and operations of the distributed monolith systems are not only complex but also unsustainable as we move towards more and more complex needs (Newman, 2019).
- As more complex business and development processes prevail, the monolith architecture slows down the entire ecosystem of delivery and operations of a large enterprise system (Namiot & Sneh-Sneppe, 2014).
3.3 Integrations in Legacy and Monolithic Applications

Typically, in enterprise integration of data and information is important. As the enterprise grows and demand for services based on data increases, the role of integration grows exponentially. In legacy as well as modern software systems, integrations play an important role to serve as the backbone of the data communication between systems. Enterprises consist of several heterogeneous applications and have point-to-point connections that result in so-called “spaghetti” as shown in Figure 5. The danger of running into an unmanageable “spaghetti” of integrations between applications as their number increases, raising the risk of side-effects with each new addition or modification to the portfolio.

![Diagram of Corporate data and networks](image)

Figure 3: Understanding enterprises (Ross et. al. 2006)

A simple way of understanding this phenomenon is where a system with not so logical flow of information. Any component of the enterprise may be connected to any other
component typically on-demand quick implementation. Almost all the components in the enterprise systems share a dependency on many or most of the other parts in some way or the other. Usually, such dependencies also do not make significant impacts on functionality. Usually, in such enterprise architecture, every component depends on many others and even a small change to one of the components breaks several other components a.k.a cascading impacts when modifying a component. Similarly, fixing one problem introduces many more. High levels of design integrity are unlikely to be found in software systems that are developed in the same fashion as shantytowns (Robinson & Gout, 2007).

Enterprises are generally built with a huge number of systems to serve the IT needs, often these applications are either customized product/tailor-made applications or a legacy application. It is likely possible that it is a combination of all the above. This makes it a very complex and chaotic environment for enterprises to maintain and grow the systems in the future; it is simply not sustainable as the changes in the business processes are so rampant. Hence breaking large and heavy systems to multiple applications is a fair argument that gives an edge to the enterprise when thinking about change management, growth, and modification of the systems (Umapathy et.al. 2017). Systems then are split into multiple applications; however, the business does not work like that. The users are likely to have their way of handling and changing business processes, it is highly unlikely that a business process designer knowns about the number, architecture of enterprise IT solutions. For example, a simple order for some typical product usually goes through several systems before it can be Fulfilled or delivered. There are systems like billing, inventory, payment, CRM, customer care involved, however from a customer point of view it is one single order journey process.

Another challenge seems to be the use of applications in different enterprise and business processes (Robinson & Gout, 2007). The components of the applications are used and integrated with other applications, which bring complexity when moving towards changes and modifications to an application. A typical component and application landscape is shown in Figure 6 explains how the components are spread across the applications and business processes. Commonly, applications are also playing a critical
role each serving several business processes. This makes it hard to maintain and operate the systems, as one change simply is complex enough to be implemented, costs are high, and impacts seem to be unprecedented. This makes the system hard to grow and update as time goes by. There are high dependencies involved when making changes to the applications and their components.

Figure 4: Independent and Overlapping Nature in Software Systems (Robinson & Gout, 2007)
4 MODERNIZATION OF ENTERPRISE SYSTEMS

In this section, we talk about the current solutions available for the problem at hand. Based on previous studies, there are several strategies on how to approach solving the challenges of enterprises. In general, the problems related to monoliths, enterprise challenges and integration methods leading to spaghetti are in some way addressed in Service-oriented architecture and microservices architecture. Both SOA and Microservices architecture suggests the decomposition of systems into services (Cerny, et al., 2018). Service is an encapsulation of components and modules into a single interface where is end goal is to provide a business function. Services are a more common way to plan and develop a further concept called the service layer, which enables the enterprises to use standard industry-wide accepted protocols that ease the access of data and functions thus enabling higher interoperability of the system.

4.1 Service-Oriented Architecture (SOA)

An SOA is an enterprise-scale IT architecture for linking resources on demand. It is a design paradigm in which application is modularized as a set of loosely coupled components and the interaction happens using services. The interaction here may be between the individual components or from any outside client. There may be an interaction between two services exchanging data or performing some common tasks. When we say loosely coupled, it means that two modules coordinate with each other without any need to know about each other's underlying implementations and changes in any module do not require major changes in other modules. One more thing about SOA is that it is not all about services but it is a design principle focusing on policies, practices, and frameworks by which we ensure the right services are provided and consumed. (Sprott & Wilkes, 2004).

Service-Oriented Architecture comprises of a number of services, which are associated with the business process and needs of IT services that mutually realizes the needs of an organization’s goals. (Arsanjani, 2004) Further SOA supports a good framework and guidelines to make communication and integration with different software
components and it has a centralized governance model that ensures the transactional and data consistency. Several scaling mechanisms are also available to make sure that the services defined can be scaled up in implemented enterprise SOA. It also makes business changes easier than in monoliths. However, the centralized nature of the integration layer, which handles the communication, integration, routing, coordination is highly prone to creating a single point of failure. Typically, SOA is more about the orchestration of services. Heavy operations and monitoring are needed for the integration layer and other components. A new service is easy to configure and existing services can be maintained in a centralized manner to ensure a better operational capacity and smooth change management process.

Microservices architecture, on the other hand, relies more on independent small components, which can rely on self-coordination or orchestration. Infrastructure in an enterprise is subject to change. Every time starting from scratch is not an option. Using SOA, we provide granularity to our system. We can add new services or make changes to existing ones with minimum or no efforts. Moreover, the design principle of SOA is so flexible that it can be incorporated in most of the enterprise's legacy systems and can help transform monolithic systems to SOA based systems. SOA defines the next generation of software architecture by embracing the challenges brought by emerging business and technology needs. SOA is targeted at providing more efficient business solutions through a new software architecture paradigm (Bih, 2006). An example of an application based on service-oriented architecture is shown in Figure 7, which explains how the application is hierarchical and comprised of services such as business, data access, and integration.
Design Principles of SOA can be applied when designing a system architecture based on SOA. Let us introduce the design principles that will help us to understand better the design of service-oriented architecture. It is also important to understand the principles as the architectural artifact could be evaluated based on the design principles of SOA. Interoperability is a natural by-product of applying fundamental principles and features of service-oriented architecture, which will be the evaluating factor for the architecture later in the evaluation phase.
4.2 Microservices Architecture (MSA)

Microservices are a variant of SOA in which an application is structured into a small set of trivial services that are often self-contained and serves a sole business use case. Microservices, recommends decomposition of services into smart entities despite the fact there should be modest routing mechanisms and no global governance model as prominent in SOA. The database is also segregated to each service in microservices as shown in Figure 8.

![Monolithic Architecture vs Microservices Architecture](image)

Figure 6: Monolith vs Microservices Architecture (Fowler & Lewis, 2014)

This obviously leads to a path towards a high decoupling and service autonomy. (Cerny, et al., 2017). On a global level, there is no need for services to agree on contracts. Since service-oriented architecture helps to build processes over the services at the integration level, it makes it easier and cost-effective to make changes in the ecosystem. However, this
flexibility comes with its drawbacks as it binds the services to a single layer of integration and creates interdependency in deployments as well as change management.

In addition, often services are liable for their key business processes and its management along with integrations to other services. There are, however, other viewpoints to consider here. Service-oriented Architectures often bring alone a complicated collection of ways and protocols essential for integrations, business transactions, overall security, etc., across services. This leads to being heavy deployment operation all linked to each other and if something fails all have to be rolled back. This characteristic resonates with the large monolith deployment model. However, in microservices, it is possible to make independent services and use heterogeneous protocols that keep low coupling and enables faster deployments. It is possible to do integrations and coordination with other services using light integration frameworks. Dependencies in the component and services for deployment may exist however there is a low rate of dependencies as there is no centralized governance, therefore the objective is to enable deployments of the services which are independent in nature (Cerny, et al., 2017).

Microservices are typically a variant of SOA and are based on independent deployment model and fine-grained, loosely coupled services which use lightweight communication protocols. (Jamshidi, et al., 2018). It is responsible for maintaining simple processes, overall business context and its viewpoint over particular information, which could lead to duplication of data across microservices. The inter-service dependencies, which tend to exist among services, are kept low due to the isolated nature or context and non-global governance mechanism.

4.3  SOA vs MSA

The basic ideology for SOA and microservices remains the same i.e., break a system into smaller components that are capable of communicating with each other. In general, both methodologies support the notion of several independent services to complete the given process, function. These independent services can be quite smaller or fine-grained in terms of size if it has to be a microservice but the goal remains the same. Design and
implementation of services differ dramatically, for example, SOA aims to provide modularity and reusable components that enable smooth development and maintaining of the services as well. The focus is to design the decomposition of the system into simple and business-relevant services that deliver value. It also promotes and supports centralized management of integration and orchestration with smart routing methods for the entire system, which is done using the service layer of the system providing governance and so-called centralized management of services.

Microservices architecture, on the other hand, suggests de-centralizing the management of integrations and focus more on choreography (defined later in section 4.5) and less on orchestration (defined later in section 4.5), keeping the focus on services, which are fine-grained and independent that can be easily deployed. The integration and so-called centralized management of services are discouraged in microservices architecture unlike in service-oriented architecture. Preference to simple routing and communication without central and global governance as noticed in service-oriented architecture, hence prominent and higher autonomy in the services. However, on the contrary, microservices are then liable to manage and deliver value to the business processes along with choreography, communication, and interaction with other microservices.

As microservices bring autonomy they also are not preferred way to go if there is a need for high data sharing and data integrity needs. Atomicity is not easy to maintain in a microservices architecture. Atomicity is the state if all steps in a transaction compete or transaction is rolled back. There are, however, other perspectives to consider. In SOA, the complexity involved with web service protocols to do operations, and communications make it difficult, and since the business processes are typically supported by the integration layer; it makes high coupling for services to one context. However this does enable flexibility while changing the processes (Cerny, et al., 2017). Figure 9 shows how the application architecture has evolved from monolithic applications to SOA based applications and to Microservices Architecture based applications. There seem to be a clear pattern evident in i, ii and iii sub-sections of the figure (Márquez, et al., 2018).

Firstly, in a monolithic application (i): the components of the application are tightly coupled and work as one unit. Secondly, SOA application (ii) have a service-based
approach communicating together via a protocol, while sharing a common database. Lastly, Microservices application (iii) has a unique viewpoint that suggests each service to have its own database and services communicate over lightweight protocols.

Figure 7: Evolution of application architecture (Márquez, et al., 2018)

The deployment model of SOA and MSA also seem to have fundamental differences, which have been shown in Figure 10. It can be inferred that the deployment models have evolved to be more complex in nature as we move towards Microservices architecture.
Another key difference is that business rules are defined at services level in MSA, whereas in SOA they are defined at the integration layer.

Figure 8: SOA vs Microservices deployment model (Cerny, et al., 2018)

In monolithic applications and SOA applications, the deployment models are simple. Therefore, due to the heavy and complex nature of deployment in MSA, there is a need for
automation in deployment processes like Continuous Integration and Continuous Deployments (CI and CD).

(Cerny, et al., 2018) research highlights key differences in Microservices architecture and Service Oriented Architecture. A summary of the differences is listed in Table 2 below.

<table>
<thead>
<tr>
<th>Concern</th>
<th>Microservices Architecture</th>
<th>Service-Oriented Architecture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deployments</td>
<td>Individual services are deployed</td>
<td>Monolithic deployment model, as a whole deployment</td>
</tr>
<tr>
<td>Teamwork</td>
<td>Microservices are typically managed by individual teams</td>
<td>Services, integrations, and UI are managed by individual teams</td>
</tr>
<tr>
<td>Integration technique</td>
<td>Simple and primitive</td>
<td>Complex integrations</td>
</tr>
<tr>
<td>Technology</td>
<td>Heterogeneous</td>
<td>Homogeneous</td>
</tr>
<tr>
<td>Cloud-native</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Management/Governance</td>
<td>Distributed</td>
<td>Centralized</td>
</tr>
<tr>
<td>Data Storage</td>
<td>Each service has own storage</td>
<td>Shared</td>
</tr>
<tr>
<td>Scalability</td>
<td>High</td>
<td>Low</td>
</tr>
<tr>
<td>Communication model</td>
<td>Prefers Choreography</td>
<td>Orchestration</td>
</tr>
<tr>
<td>Business rules location</td>
<td>Service level</td>
<td>at Integration level</td>
</tr>
</tbody>
</table>

Table 2: SOA vs MSA key differences (Cerny, et al., 2018)

### 4.4 Migrate from Monoliths to Microservices

Monolithic and legacy applications in enterprises often need transformation and migration to new technologies and architecture simplification process. A trend to migrate towards service-oriented and microservice architectures. Microservices, in general, are not the best solutions for all problems. Migrating to a microservices architecture is a decision that should be well discussed and there must be clear evidence on the benefits, which will come
with this migration. However, a key challenge remains “How to split the monolithic and legacy applications to microservices?” Typically the code in a monolithic application is tightly coupled which makes it difficult in the de-coupling process.

While several traditional patterns for migration and techniques exist, there is a lack of formal models and tools to support the migration process from monoliths to microservices (Mazlami, et al., 2017). While migrating from a monolithic application to microservices there is no such thing as a big bang. Small portions of the application can be initially identified based on the domain model and business prioritization. It is important to understand what is easy to migrate and bring high business value in doing so. To understand this concept as a whole advantage we can use a 2-d graphical representation where ease of decomposition and benefits of decomposition are mapped as shown in figure 12 a. It helps to prioritize the service decomposition. An example of prioritization for the service decomposition is shown in Figure 12 b.

![Graphical representation](image)

**Figure 9.a**: Utility to prioritize component decomposition (Newman, 2019)
Newman’s example shows how to prioritize component decomposition by mapping the components of the system based on increasing ease and benefit of decomposition. The candidates with the highest ease and benefit of decomposition are potential candidates for service decomposition. This is an area to start with. Once the prioritization has been done, it is easy to understand which domains of the application can be used to start the transformation process. One very simple example of migrating from monolith to a microservice is described in figure 12 b. Here the monolith application contains a set of domain functions where the payroll module is identified based on high ease and benefit of decomposition. Hence we can see the AS-IS and TO-BE functional architecture of the system.

Migrating production domain functions need a seamless experience to minimize the impacts to the business continuity and change management process. Hence, it is in the best interest to keep the existing service in a monolith until the change management process is approved and a new microservice for Payroll is in production. While migrating to new service it is important to take note of upstream and downstream dependencies and coordination. In this example shown in Figure 13, the existing call to the payroll in a
monolith is re-directed via proxy to the new Payroll service and the new microservice takes care of the downstream system 'User Notifications' (Newman, 2019). Figure 13 shows how the components look before the extraction of Payroll service and after the extraction towards Microservices.

![Diagram showing before and after extraction of Payroll service](image)

**Figure 11:** Example, AS-IS, and TO-BE when migrating to microservices (Newman, 2019)

Another aspect when migrating to service-oriented or microservices is migration strategies. Usually, an incremental migration strategy is preferred by large organizations. Fowler (2018) says, "If you do a big-bang rewrite, the only thing you're guaranteed of is a big bang." Hence, it is critical to understand that there is no such thing has migrated from a monolith to microservice or service-oriented architecture overnight. Incremental migration policy is adopted and services are extracted one at time. An incremental approach limits the amount of impact on key business operations.

It is important to note that the extraction of a microservice cannot be considered complete until it is in production and being actively used. There is also an element of cost and complexity when moving to a new architecture altogether. It is much easier to split the migration into several phases and iterations. Any transition to a microservice architecture should bear these principles in mind. Often irreversible and inter-organization changes are avoided and more of reversible changes are preferred when selecting the services and change to migrate as those can be rolled back with minimal efforts (Newman, 2019).
4.5 Enterprise Integrations and Patterns

In this section, we discuss the ways of communication and integration patterns between the services. Based on the existing studies, SOA and microservices are related to each other to some extent as the architectures suggest decomposing of systems. Integration has been a key driver for the systems in the last decades. Now the main question arises on how services interact with each other and with other third-party systems and services.

Traditionally there is two way of managing the communication, centralized or distributed pattern. The decomposed service tends to be a reusable component that performs the required business operations within a small domain and context. Typically, service includes a sub-section of the domain and its interface, which can be used independently over the network giving freedom to communicate effectively. The communication patterns with a centralized nature are called service orchestration whereas decentralized nature is termed as service choreography. The key difference in how services integrate and communicate with each other. In most cases, inter-organization communication is done via service choreography and a more preferred approach in a microservices architecture.

On the other hand, service, SOA prefers a central communication and integration pattern, which enables communication within the organization as well as inter-organization, service orchestration, is preferred approach in service-oriented architecture (Newman, 2015). Service orchestration expects a centralized business process, coordinating activities over different services and combining the outcomes. Figure 14 depicts how services interact with each other in an orchestration mode of interaction.
On the other hand, the centralized node for communication and interaction between services is missing in choreography. Service choreography defines the rules and fundamentals of integrations within the services itself. Services can control the logic and way of communication with other services and components. Hence, it is not centralized and there is no single point of failure. Figure 15 depicts how services interact with each other in a choreography mode (Cerny, et al., 2018).
Services are a crucial part of SOA and MSA. Traditionally enterprises have been using service orchestration with the global governance model to ensure the business process integrations. SOA ensures easy migration and composition of services enabling higher use of existing features as well and building new services based on the existing ones. The services for external systems such as third-party or other business units can also be provisioned using service-oriented architecture. The major difference in Service orchestration and choreography is explained in Figures 14 and 15.

An orchestration that also resonates with SOA believes in teams managing the services and the integration team is an independent entity that takes care of the coordination and process integration as a whole. When a change request comes for the introduction of a business process or change to an existing one pointing to an impact in the interfaces, integrations that then need changes in all components of the system. There is a high level of cooperation needed to get this change to production as the deployment involves redeploying several components and services in SOA, rollbacks are more challenging. However, in Service choreography and microservices architecture guidelines suggest each service takes care of own changes and integrations. This leads to simpler cooperation and easy self-contained deployments which are easy to make and easy to rollback.

Integrations between applications, systems, and services are here to stay. No matter how enterprise architecture evolves, there is a high demand for integrations. In this section, we also talk about how important and critical it is to handle the integration within the enterprise and outside the enterprises. Legacy and modern application continue to exist and those must be integrated as well along with the integrations of homogenous applications. Modern enterprises have an IT application stack of heterogeneous applications, which have a different lifecycle, and technology stack.

Therefore, the integration of these applications into a unified set of business processes has emerged as a priority. Due to the complexity of integrations in enterprise applications, the field of enterprise integration patterns has been used to solve the problems and use it as a reference when designing the enterprise integrations and architecture. It provides a good mechanism to share data, processes, and business functions. Due to the competitive nature
of the enterprises and the macro-environment, business tends to strive for first to market thus keeping the development cycle as short as possible, while minimizing the cost of the whole solution.

Enterprise integration patterns (EIP) have existed for over 15 years. The journal article by Zimmermann, et al., (2015), mentions that the more things change, the more they stay the same, and Enterprise Integration Patterns (EIP) remains relevant even today. SOA drove the adoption of ESBs, which incorporate and are fundamentally based on the enterprise integration patterns. Due to the shift towards a microservices architecture, Microservices as well need integrations to other applications, processes and business functions, so integration and messaging solutions will be needed for those as well. Bogner, et al., (2018) expresses that “Patterns capture expertise that is timeless. Skills learned through patterns remain applicable even as the products and technologies evolve” Therefore, in the research, I explore the possibility to use both SOA and MSA to its advantages and find a trade-off that can enable enterprises to benefit from both strategies as a hybrid architecture design that can solve the challenges of modern enterprise architecture. EIP continues to serve as a key reference point for integration needs.
5 ARCHITECTURAL DESIGN AND ARTIFACT

The objective of this section is to create an artifact where the research contribution is embedded in the design. The section of the research includes the desired functionality and its architecture and followed by the creation of the artifact. It is a typical project, which starts from a requirements-gathering process, in which a diverse set of applications go under the transformation process. The result of this section is to produce novel artifact, which addresses the problems and the objectives of the research. In general, the artifact could be applied to solve similar kind of problems in other enterprises.

5.1 Project Requirements

Project is based on company X (real name not used due to security and privacy rules) and its HR & Time management systems, main monoliths with limited capability of modern integrations. There is a high demand from product managers to have real-time and modern solutions that are easy to develop, deploy and maintain in the future keeping in mind scalability and high availability of the services. Mobile solutions rarely possible due to limitations in legacy systems/technologies. The new architecture should also enable the mobile solutions of the business-critical applications.

Another aspect is part of non-functional requirements: the performance and interoperability of applications. In current enterprise landscape due to bad design and architecture, some of the integrations do not seem to work as there are several unknown dependencies which run in the background and database querying in some processes can go up to several pages, which makes it impossible to get the required data in time, prevents scaling, reporting and other integrations. This is a pain area for the product managers, as many customers expect that the services are fast and reliable. Therefore, the scope for integrations and scalability is growing rapidly, but enterprise applications not ready to meet customer’s expectations. This need to change and clear documentation and knowledge about APIs are needed to enable development within the product and with third-party integrations. The product portfolio is highly complex, unstructured and documented poorly due to development over the years consisting of old and new technologies. The codebase has been modified and updated over time causing bloating and
accumulation of code blocks that are not even needed anymore. The technology used in the applications is diverse and difficult to maintain and further develop. For example, one of the applications is built starting the 1980s and is still being modified and used. The challenge possesses a threat that the applications need a product ecosystem to continue the product lifecycle. It is simply not efficient to move features and fixes from development to production due to high manual efforts and lack of automation. To be able to sustain the future growth of the product portfolio it is vital to transform the architecture into a modern one, which can help set up a base for product development and maintenance.

5.2 Architecture Design

Based on the problem statement described above, this phase of the research focuses on designing simplified enterprise architecture, with the migration of old applications and integration pattern implementation for the whole enterprise architecture landscape. It also takes advantage of existing knowledge related to the design of architectures, enterprise integrations and migration of monolithic applications to service-oriented applications or microservices. Due to the competitive nature of the business domain, the old applications and product offerings of the enterprise simply are lagging in several aspects such as customer experience, market share, and innovation.

To solve the problem case, the plan is to use the (Hofmeister, et al., 2007) process of designing architectures. The process itself involves several phases and stages as described in Figure.16. The initial phase of the process focuses on Architectural analysis guides to create architecturally significant requirements (ASRs); it is completely based on context and architectural concerns. Architectural synthesis and evaluation are other important phases of the process. It typically answers the problems and shapes the architecture design solution that arise based on the ASRs. To validate the decision decisions make in the process are then evaluated in Architectural evaluation, which ensures validation of the artifact (Hofmeister, et al., 2007).
The process of architecture design is the main goal and contribution for this research and the artifact produced will be a reference design pattern that can be used to solve the problem case by utility nature of the artifact. To explore the Architectural Analysis as phase one of the model for software engineering design as described by (Hofmeister, et al., 2007), we have to understand the architectural concerns and context. The concerns are described above in the project landscape in section 5.1. These are mainly high-level concerns, as the low-level concerns go deep into the problem of architectural design, which was built over the decades using easy and fast decisions that were made to deliver the solutions quick and dirty way. This has led to the so-called spaghetti of network and architecture where no one knows anymore what is happening, how it happens, and where it happens exactly.

In the current architecture, systems have several parts mostly monolithic with interactions with each other via direct database connections to other applications; file transfer jobs, web services and so on. This has been shown in Figure 17 hiding sensitive details and names have been masked to keep the. There are also several 3rd party integrations point to point, at data and application layer based on traditional file transfers with data model based on flat files such as CSV which are not shown in the context view due to privacy and security regulations.
5.2.1 Architectural Significant Requirements

Architecturally significant requirements (ASRs) are defined as “a requirement upon a software system which influences its architecture” (Obbink et al., 2002). However, not everything is relevant to the architecture synthesis process and evaluation. Conversely, not all ASRs will have originally been expressed as requirements: they may arise from other architectural concerns or the system context. The architecturally significant requirements (ASR) is the state of how to measure the achievement of the goal and indicate how the goal constrains software architecture (Hofmeister, et al., 2007).

In this analysis, the following ASRs were laid down based on the requirements, context, and concerns.

1. Monoliths should be split into several sub-modules or services to effectively develop and maintain them.
2. End Users should have a seamless experience in mobile and web UI with the new architecture.
3. The application should be available on different platforms and mobile clients.
4. Monolithic applications are based on business context.
5. Unwanted interaction and integrations outside the context area should be limited.
6. Integrations to legacy and third-party systems should also be handled using an integration layer that does not influence the microservices.
7. All services should have their database and data integrity must be maintained.
8. Integrations to legacy and third-party systems should have the capability to withstand fault tolerance and queue mechanism. Message-based integrations are preferred wherever possible to ensure, message delivery.
9. Personal information should be processed within the application in a highly secure way.
10. Microservices should be easy to scale and the deployment model should be simple and automated.
11. The application should be able to handle the increasing load and should have high availability in case some of the servers fail to work.

5.2.2 Design Decisions

To achieve the desired goals and align based on architectural significant requirements (ASRs) there are design approaches used to fulfill one or more ASRs and concerns. These are then applied to the architectural view of the system. These approaches usually depend on basic software engineering knowledge and enterprise architecture principles. It often involves applying heuristics as well. Therefore, the strategies and approaches adopted in an architectural view are called “design decisions” (Hofmeister, et al., 2007).

Here are the design decisions made during the process.

1. Some of the monolithic and legacy applications designated for end-users were selected as candidates for splitting to microservices so that it is effective to develop and maintain the solutions.
2. To make the user experience alike and seamless in the UI, it was agreed to use the mobile-first UI approach to keep the elements similar in mobile and web user-interface. Most of the components of the web and mobile UI can be
shared. Mobile and Web client enables users to use the system with much higher freedom.

3. All microservices have their own/semi-own database and data integrity must be maintained.

4. To ensure the service availability, there is a provision of load balancers and auto-scaling for all microservices, API gateway, UI applications, and Integration layer.

5. Enterprise Integration Patterns continues to play a critical role in integrations/communication between the applications due to the existence of old and legacy applications as it eases migration (Claus & Pooyan, 2016) and (Bogner, et al., 2018) suggests EI patterns bring expertise in integrations. All interactions and integrations outside the context area are limited to the API gateway or Integration layer. Integrations to legacy and third-party systems are handled using the integration layer as it needs transforming and which does not influence the microservices. In addition, it enables the capability to withstand fault tolerance and queue mechanism. Message-based integrations are preferred wherever possible to ensure, message delivery.

6. To ensure secure processing and data flow all information should be processed within the application in a highly secure way, hence secure protocols like traffic over SSL are used, HTTPS, SFTP and so on. Use of Pipelines and Continuous Deployments to keep the deployments automated, and use of auto-scaling to ensure requests are honored if the load increases.

7. Providing a secure API Gateway to the applications ensuring Authorization and Authentication. All requests authenticate via an authentication layer via tokens at the API gateway.

Overall, the design decisions enable the use of a Hybrid pattern of architecture using EIP, SOA, and microservices to transform the enterprise application landscape. API gateway is preferred to manage the APIs.
5.3 Functional View of the Artifact

Based on the above architecture significant requirements and design decisions the following reference artifact is created which can be referred to, in the same kind of transformation projects. This view in Figure 18 consists of functional elements, responsibilities, and interfaces. It also explains the primary interactions.

![Functional View Diagram](image)

**Figure 16: Functional view of the Artifact**

On a high level, the functional view can explain the functional requirements and design decisions taken based on ASRs. FLATS is the flat file transfer communication between the systems. SOAP-based and REST-based Web Services API are used to manage the communication between the systems. It enables exchanging structured information in standard formats between the systems, which are understood if the support exists. Here
based on the SOA and MSA principles, a system is split into several services connected to an API gateway. Web UI and Mobile applications consume the same API endpoints to provide seamless offerings. A lightweight service layer handles Enterprise integrations, which is also responsible for interactions with legacy systems and other monolithic applications. The interaction and communication between the microservices can leverage both service orchestration and choreography. In most cases, for large and complex enterprises service orchestration is always better since it gives the advantages of message queues, fault tolerance, and central control and monitoring capabilities. However, to keep the properties of MSA, coordination between microservices is preferred if the context as the same and data is isolated. Legacy applications and traditional monolithic web applications can use enterprise integration patterns, which makes it easier to integrate and deliver business processes efficiently. This resonates with the Hybrid pattern of architecture using EIP, SOA, and MSA to take advantage of all.

Another key aspect of Microservices architecture, which has been taken into use while designing the overall architecture, is an API gateway. As the services grow reasonably in microservices architecture, it becomes difficult to manage and maintain them without a centralized control layer. The services are usually loosely coupled, they need to be secured and managed so that access is fast and easy. A client can connect to one gateway to access all the services needed, this simplifies the overall network architecture as well. Routing and Security are key aspects of using the API gateway. All requests to different microservices are handled in a secure way, which is common to all types of requests. It runs in front of microservices and acts as a gatekeeper. As the use of API and services grows, there is also a strong demand to have some kind of API Lifecycle management, which is now a crucial part of the API gateway. In some cases where API evolves to new versions based on feature iterations and application lifecycle, the goal is to keep the newer versions of API that can support backward compatibility and offer customers to use different versions of API when needed. Monitoring and Analytics is an important aspect of microservices, measurement of metrics is key to understand the performance and service usage. Since all requests go through API gateway, its obvious place. It allows rate limiting and throttling features too. Some other features are input validation and response transformation if needed. Due to the microservices architecture principles, the decoupled nature of services result in several smaller resources, but the API gateway helps to manage these services.
6 DEMONSTRATION OF SOLUTION

The objective of this section is to demonstrate the use of the artifact to solve the above-described project landscape problem of enterprise architecture, therefore the utility of the artifact is strongly linked to the project that is has led to the creation of the reference artifact and strategy, which can be used in enterprises facing same challenges in their architectural goals. The artifact is created based on the research and applies well in this case. Hence based on the project requirements, and created artifact Figure.19 here is the proposed solution, which uses the artifact created in the design science research.

Some other requirements and decision decisions are taken into account during the demonstration of the solution using the created artifact.

1. Service availability should have high availability of more than 99.8%
2. Analytics reports should be available from Legacy, 3rd party and new microservices.
3. Analytics Platform enables the reports to be shown in a user interface, however, the data must be fed to the analytics platform. Here the platform is based on Azure cloud and Power BI. Data from modern microservices is directly moved to the analytics platform storage, however, the challenge remains where the reports are needed from legacy and third-party systems. Hence, the integration layer ensures data delivery to the Analytics platform from legacy and third-party systems after an optional transformation.

The utility of the artifact as shown in Figure 19 is, the use of modern strategies and design decisions to the architecture, and overall the process transforms the enterprise architecture and enables it to use the capabilities of SOA, MSA, and EIP. To keep the user experience seamless the frontend capabilities of the applications are planned to renew to modern web applications, which can be built on the concept of the mobile-first user interface. This expands the product offering to all kinds of devices such as mobile, tablets and web desktops. Microservices are independent however; some grouping has been done based on the product domain and offerings.
The idea is also to keep some shared microservices such as user service, which can be utilized among all services. Data for the Analytics platform is directly sent from the microservices database, which then processes the reports needed to be published at the API gateway for UI applications to consume.

Figure 17: Demonstration of the solution: Proposed Architecture based on the artifact
Figure 19 also demonstrates that other third parties and legacy applications use the lightweight service bus to feed the data to the analytics platform via service layer data transformation. Data operations and transactions among domain microservices are avoided, and they are mostly done using the service orchestration. In general, the utility of the artifact enables us to address the concerns and ASRs. The deployment itself is based on the high-availability model and auto-scaling is enabled to ensure the performance of the application in case the load increases. Service bus deployment uses clustering and load balancing to improve the scalability by keeping the load distributed across the application nodes. The transformation project starts with phase 1 where the basic environment setup including API gateway, few microservices and service bus with integrations to existing legacy and third-party systems have been used to highlight a proof of concept. Overall, the system looks promising and further phases of the transformation project progress.
7 EVALUATION

Design science research model suggests that theoretical constructs and artifact produced during the process should be evaluated to demonstrate how well does the artifact solve the objectives i.e. appropriateness and utility of their usage. (Hevner, et al., 2004) In this thesis, the evaluation is based on the objectives and ASRs based on (Hofmeister, et al., 2007) principles of architecture validation and on the principles of SOA, MSA, and EIP. The purpose of this evaluation is to provide an understanding of the efficiency and effectiveness of an artifact. The target of the study is not only to understand the underlying problems and existing solutions available to the problems but also to contribute towards science with an artifact, which is adding value and helping other organizations to modernize their enterprise systems and architecture. However, the research and the artifact created could have some unforeseen consequences as the research done has some constraints and the problem-based research motivations.

Based on the guidelines of Design Science Research, we explained the artifact designed and it’s utility towards the problem. The solution is now taken into development and pilot done in order to understand how the concept works in real life because the complete implementation of the architectural artifact would probably take years. The idea is to understand if the proof of concept solves the problem statement and how well it solves the problem in question. When evaluating the architectural artifact created in this research project there were three iterations done to evaluate and test the pilot of the solution. Overall solution and skeleton of the architecture worked well when tested migration of services from monolithic applications to services. Modernized UI application uses a Secured API gateway to connect with different services available and integrates with other applications via a light integration service bus, which takes care of integrating the information via message queues for services and using other EIPs. However, during the iterations of the architectural design process and evaluation, few observations needed attention and iteration of the design process.

The list of these observations is mentioned below:

1. With an increase in services and components, there comes a notion of scaling and automation; however, more services and resources were an obvious result of the modernization process. This adds more costs and moving parts in the overall
enterprise architecture. Monitoring each component needs extra efforts, built-in mechanisms such as Logging as a Service and Centralized monitoring, and built-in analytics of the services and traffic.

2. Some services initially were planned based on a high-level domain understanding, but later it became evident that to design the artifact it is important to clearly specify the service – business domain contexts. The services must be built accordingly keeping in mind cohesiveness and level of interactions and communication between the services. If the services are not designed carefully, the communication costs can grow high; causing performance issues and leading to re-designing costs.

3. It is found that SOA is less expensive and interaction friendly as it does not give high communication costs if compared to MSA.
8 DISCUSSION

To understand the overall benefit of this modernization, one must understand the challenges and problems in enterprise architecture and monolithic applications that have been developed a long time ago. Often these legacy systems are composed of old technologies and typically a centralized way and single unit application. As discussed this style has led several organizations to move towards service-oriented architectures and microservices. This thesis explained the problems and challenges faced by an enterprise to maintain its enterprise architecture and difficulties in modernizing its application to serve business and customer requirements with a more agile way of delivery model. The research highlights key and important strategies, solutions and patterns available for the enterprises to modernize the landscape. It also discusses SOA and MSA in detail and their commonalities and differences; keeping in mind the application of these features to create a novel artifact “architectural strategy and design to modernize the enterprise legacy architecture”. Communications and interactions being an important part of enterprise architecture, the area is well understood and clearly states that SOA prefers orchestration whereas MSA prefers choreography. However, there has to be a balance when designing a hybrid system, which takes advantage of both SOA and MSA. It is also evident that Microservices are very helpful with scalability, maintainability, and ease of deployments. However, abundant choreography could increase the communication costs within services if not designed carefully. The challenge related to monoliths with limited capability of modern integrations is well solved with the created artifact enabling an easy migration from both sides using MSA and SOA. SOA supports easy migration and MSA helps with the splitting of monoliths into smaller course-grained services.

It enables to build real-time integrated and modern applications, which are easy to develop, deploy and maintain in the future keeping in mind scalability and high availability of the services. Mobile solutions are an important part of the artifact generated; the proposed artifact enables us to move towards mobile-first development strategy in business-critical applications without disturbing the operations. With a balanced hybrid, the flavor of SOA and MSA, performance and interoperability of applications is also well addressed. The artifact tends to do architectural simplification and keep the product portfolio simple, and flexibility to develop and maintain applications over time.
9 CONCLUSIONS

This research was conducted according to the design science guidelines (Hevner, et al., 2004) and design science process model (Peffers, et al., 2007) to develop a software artifact embedding a process that helps the designers modernize the enterprise architecture and its applications from legacy and monolith system while ensuring a good integration solution based on recurring solutions captured in the research. The research aimed to study the problem in a structured and deep-dive approach and finds the solutions available for the problem.

SOA tends to be more popular in enterprises to enable service-based architecture, however, MSA is evolving fast. Both SOA and MSA have their pros and cons, which must be kept in mind while designing a solution. In general, Microservices are a variant of SOA. However, SOA is about the orchestration of services whereas MSA, on the other hand, relies more on independent small components, which can rely on self-coordination or orchestration. Therefore, this research illustrates use both SOA and MSA to its advantages and find a trade-off that can enable enterprises to benefit from both strategies as a hybrid architecture design that can solve the challenges of modern enterprise architecture.

It was also found that Enterprise Integration Patterns (EIP) continues to serve as a key reference point for integration needs. Overall, the new architecture was evaluated by conducting a small project and results were good in terms that it is possible to solve the problems of enterprises related to legacy apps and integrations. During the architecture design iteration process and evaluation of the artifact, there were some observations made and they helped to improve the overall solution as a whole.

The research can be further extended on how the overall implementation goes and how well it fits towards to goal of digital modernization of enterprise architectures. Further research could also focus on the role of modern architecture in the Internet of Things (IoT) era, as organizations tend to move toward that direction with 5G around the corner. It could also help the organizations if the research provides a strong base for large organizations to modernize the systems for the next generation to come.
REFERENCES


